February 1987

Report No. STAN-CS-87-1149
Also numbered CSL-TR-87-315

Proceedings from the Nineteenth Annual Meeting of the Stanford Computer Forum

edited by

Katie Mac Millen, Ann Diaz-Barriga
and Carolyn Tajnai

Department of Computer Science

Stanford University
Stanford, CA 94305

LELAND STANFORD JUNIOR UNIVERSITY
ORGANIZED 1891

Proceedings

from the Nineteenth Annual Meeting of the Stanford Computer Forum

February 1987

Department of Computer Science
Prof. Nils Nilsson, Chairman
and the
Computer Systems Laboratory
Prof. John Hennessy, Director

William F. Miller, Director
Carolyn E. Tajnai, Manager
Terry Winograd, Program Chairman

Edited by Katie Mac Millen, Ann Diaz-Barriga, and Carolyn Tajnai

Abstract

Operating for almost two decades, the Stanford Computer Forum is a cooperative venture of the Computer Science Department and the Computer Systems Laboratory (a laboratory operated jointly by the Computer Science and Electrical Engineering Departments). CSD and CSL are internationally recognized for their excellence; their faculty members, research staff, and students are widely known for leadership in developing new ideas and trends in the organization, design and use of computers. They are in the forefront of applying research results to a wide range of applications.

The Forum holds an annual meeting in February to which three representatives of each member company are invited. The meeting lasts two days and features technical sessions at which timely computer research at Stanford is described by advanced graduate students and faculty members. There are opportunities for informal discussions to complement the presentations.

This report includes information on the Forum, the program, abstracts of the talks and viewgraphs used in the presentations.

Stanford Computer Forum
ERL Building, Room 448
Stanford University
Stanford, California 943054055

The Stanford Computer Forum

Advances in computer science and technology come about through the efforts of people in both the academic and industrial worlds. Great resources of scientific, professional, and managerial skills exist in both settings. Coming from the academic side, faculty members often become associated with industrial organizations as scientific consultants. These contacts benefit both the companies concerned and the faculty member, who is exposed to areas of investigation which contribute to his or her potential in teaching and research. Another strong tie between the University and industry comes from the many graduates in computer science and engineering who enter careers in industry.

Yet despite these strong natural ties, advances in the science of computing take place so fast and across such a broad front that it is becoming difficult to maintain ongoing and mutually beneficial interaction between university and industry. It is to meet this need that the Stanford Computer Forum has been established, providing a setting where academic and industrial computing talents can easily interact.

The Stanford Computer Forum, operating for almost two decades, is a cooperative venture of the Computer Systems Laboratory (jointly administered by the Departments of Computer Science and Electrical Engineering), the Computer Science Department, and a group of industrial affiliates. The Forum provides a mechanism for developing personal contacts between industrial researchers and their academic counterparts, promoting the exchange of the most advanced technical ideas between Stanford and industry. It acquaints the Stanford computer community with the special interests and concerns of industry and introduces Stanford students to industries and their problems involving computer theory and technology.

The Computer Forum is also important to Stanford as a source of significant funding. Membership contributions help foster innovation in Stanford's computer science and engineering programs. **A** healthy, vigorous industrial affiliates program is crucial to the vitality of the activities of the Computer Science Department and the Computer Systems Laboratory.

Computer Science and Computer Engineering at Stanford

The Departments of Computer Science and Electrical Engineering (including the Computer Systems Laboratory) at Stanford University are internationally recognized for their excellence. Their faculty members, research staff and students are widely known for leadership in developing new ideas and trends in the organization, design and use of computers. They are in the forefront of applying research results to a wide range of applications.

Researchers from these two organizations at Stanford are now engaged in investigations in such fields as programming languages, operating systems, integrated circuits, database systems, computer-aided design, fault-tolerant computing, computer architecture, computer communications, office automation, artificial intelligence, program verification, numerical analysis, and algorithm analysis. Over 300 graduate students contribute to these studies.

How the Forum Operates

The Stanford Computer Forum offers a variety of benefits to its industrial members.

- *Annual Meeting:* The Forum holds an annual meeting in February to which three representatives of each member company are invited. The meeting lasts two days and features technical sessions at which timely computer research at Stanford is described by advanced graduate students and faculty members. There are opportunities for informal discussions to complement the presentations.

- *Research Reports:* Three microfiche copies of all research reports issued by the Computer Systems Laboratory or the Department of Computer Science are sent either to individual members or to corporate libraries. Hardcopy reports are available for an additional charge of $200 per set a year.

- *Faculty Liaison:* A faculty member or senior member of the research staff will act as the technical liaison to each Forum company. Each year interaction in the form of a personal visit to the company, a visit from company staff to Stanford (other than the annual meeting), or any other mutually agreeable technical communication will be arranged.

- *Industrial Scholars:* In the past, companies have found it beneficial to have one of their scientists or engineers visit Stanford for an extended stay to exchange ideas and collaborate with Stanford researchers. Such opportunities are available by mutual agreement between the Forum member company and a Stanford faculty member. Due to the expenses involved with such visits, additional funding is usually necessary.

- *Computer Forum Distinguished Lecture Series:* This is a library of videotapes of the outstanding lectures presented during each academic year at the Department of Computer Science Colloquium and the Computer Systems Laboratory Seminar (the best 12 are chosen from a total of 60 lectures). Forum members may order the tapes at no charge if they return the cassettes within 30 days.

- *Student Interviews:* Forum members have an unrivaled opportunity to become familiar with the professional abilities and interests of Stanford students. Many of the presentations at the annual meeting are made by these students. Stanford assists Forum affiliates in meeting doctoral and master's students whose work the company may find significant. The Forum provides members with Ph.D, M.S. and B.S. student biographies and can arrange individual interviews.

- *Computer Guest Account:* The Forum provides a guest account on Score, the CSD DECsystem-2060 computer system. Student resumes, CSD and CSL bibliographies and faculty research interest information are maintained online for our members.

Membership in the Forum

Membership in the Stanford Computer Forum is open to corporations with strong interests in computing -- corporations that would both benefit from and contribute to this technical interchange.

The corporate membership fee is $13,500 per year. Because benefits are more fully realized through continued association, organizations are strongly encouraged to regard membership as a long-term commitment. The contribution represents an investment in a strong computer science and engineering program at Stanford, and provides an opportunity to watch that program continue its leadership in the computing community.

For Further Information, Please contact:

Manager, Stanford Computer Forum
ERL 448
Stanford University
Stanford, CA 943054055

(4 15) 723-3550

Stanford Computer Forum
Stanford University
Stanford, California 94305

Program for the Nineteenth Annual Meeting
3 — 5 February 1987

PRELIMINARY SESSIONS

CS500 SEMINAR **4:15 P.M., TUES. 3 FEB. 1987**

Skilling Auditorium

David Hartzband, Chief Scientist, AI Technology, DEC

The Provision of Induction as a Problem Solving Method in Data Model Systems

INFORMAL BUFFET SUPPER **6:00 — 9:00 P.M., TUES., 3 FEB. 1987**

Gold Lounge, Faculty Club

Conference Registration/Buffet Breakfast 8:00 — 9:00 a.m., Wed., 4 Feb. 1987

Tresidder Union, Room 281 {Oak Lounge East and West]

Plenary Session 9:00—9:45 a.m., Wed., 4 Feb. 1987

Tresidder, Room 281

Carolyn Taj nai
Manager, Computer Forum

Prof. Terry Winograd
Conference Chairman

Prof. William Miller
Director, Computer Forum; President, SRI International

Prof. Jim Gibbons
Dean, School of Engineering

Prof. Nils Nilsson
Chairman, Department of Computer Science

Prof. John Hennessy
Director, Computer Systems Laboratory

Technical Session I A 10:00 a.m. — 12:00 noon, Wed., 4 Feb. 1987

CERAS Rm. 112

Session Chairman: Prof. Thomas Binford

Statistical Approaches to Perceptual Organization
Richard Vistnes *Advisor: Prof. Thomas Binford*

Stereo Correspondence: A Hierarchical Approach
Hong Seh Lim *Advisor: Prof. Thomas Binford*

Line-Drawing Interpretation: Bilateral Symmetry
Vic Nalwa *Advisor: Prof. Thomas Binford*

Some Extensions on the Motion Specification of AL Robot Language
Chun-Sheng Cai *Advisor: Prof. Thomas Binford*

Force Control and Design of Robot Manipulators
Joel Burdick *Advisor: Prof. Thomas Binford*

Perception With Simulated Dynamics
Michael Kass *Advisor: Prof. Thomas Binford*

Luncheon 12:15 p.m., Wed., 4 Feb. 1987

Tresidder Union, Room 281

NOTES

Technical Session I B

10:00 a.m. — 12:00 noon, Wed., 4 **Feb. 1987**

Tresidder, Cypress Lounge North and South

Session Chairman: Prof. John Hennessy

MIPS-X
Prof. Mark Horowitz

Knowledge-Based Monitoring and Control: An Architecture for Distributed Systems
Bruce Hitson — *Advisor: Prof. Keith Lantz*

Log Files: An Extended File Service Exploiting Write-Once Optical Disk
Ross Finlayson — *Advisor: Prof. David Cheriton*

Internetwork Multicasting
Steve Deering — *Advisor: Prof. David Cheriton*

LISP on a Reduced-Instruction-Set Processor: Characterization and Optimization
Peter Steenkiste — *Advisor: Prof. John Hennessy*

Monitoring the Stanford Internetwork
Glenn Trewitt — *Advisor: Prof. John Hennessy*

Luncheon

12:15 p.m., Wed., 4 Feb. 1987

Tresidder Union, Room 281

NOTES

NOTES

Tours and Demonstrations 1:30 — 4:00 p.m., Wed., 4 Feb. 1987

Tour Information Appears on a Separate Sheet

Computer Systems Laboratory Seminar 4:15 — 5:00 p.m., Wed., 4 Feb. 1987

Terman Auditorium

The Transition of Computer Science Results to Business and Industry
Panel Discussion

Dr. William F. Miller, Moderator

Dr. Thomas Buckholtz
Pacific Gas & Electric Company

Dr. William Spencer
Xerox Corporation

Dr. Gordon Bell
National Science Foundation

Evening Activities 6:00 p.m., Wed., 4 Feb. 1987

Faculty Club

6:00 p.m.: Social Hour

7:00 p.m.: Dinner

Toastmaster: Prof. William F. Miller

Speaker: Dr. Abe Peled, Vice President, IBM
Topic: *On Computing and Communications*

Continental breakfast 8:00 — 9:00 a.m., Thurs., 5 Feb. 1987

Tresidder Union, Room 281

Technical Session II A 9:oo — 10:00 a.m., Thurs., 5 Feb. 1987

CERAS Rm. 112

Session Chairman: Prof. Ed McCluskey

Incremental Behavioral Simulation System
Prof. Tom Blank

A CMOS PLA Design for Built-In Self-Test
Dick Liu *Advisor: Prof. Ed McCluskey*

Temporary Failures in Digital Circuits: Experimental Results and Fault Modeling
Mario Cortes *Advisor: Prof. Ed McCluskey*

*** * * * Coffee Break * * * ***

NOTES

Continental breakfast 8:00 — 9:00 a.m., Thurs., 5 Feb. 1987

Tresidder Union, Room 281

Technical Session II B 9:oo — 10:00 a.m., Thurs., 5 Feb. 1987

Tresidder, Cypress Lounge North and South

Session Chairman: Prof. Ted Shortliffe

An Analysis of Heuristic Methods for Reasoning With Uncertainty
David Heckerman *Advisor: Prof. Ted Shortliffe*

Using a Domain Model to Drive an Interactive Knowledge Editing Tool
Mark Musen *Advisor: Prof. Ted Shortliffe*

A Decision Theoretic Approach to Heuristic Planning
Curt Langlotz *Advisor: Prof. Ted Shortliffe*

****** Coffee Break ******

NOTES

Technical Session Ill A 10:30 a.m. — 12:00 noon, Thurs., 5 Feb. 1987

CERAS Rm. 112

Session Chairman: Prof. Michael Flynn

"Post-Game Analysis" - A Heuristic Approach to Manage Concurrent Execution Environments
Jerry Yan *Advisor: Prof. Steve Lundstrom*

Performance Prediction of Concurrent Systems
Victor Mak *Advisor: Prof. Steve Lundstrom*

Beta Operations: An Efficient Implementation of a Primitive Parallel Operation
Evan Cohn *Advisor: Prof. Jeffrey Ullman*

Tradeoffs in Data-Buffer Design
J. M. Mulder *Advisor: Prof. Michael Flynn*

Stanford Packet Radio Network
B. P. Boesch *Advisor: Prof. Michael Flynn*

Luncheon 12:15 p.m., Thurs., 5 Feb. 1987

Tresidder Union, Room 281

NOTES

Technical Session Ill B 10:30 a.m. — 12:00 noon, Thurs., 5 Feb. 1987

Tresidder, Cypress Lounge North and South

Session Chairman: Prof. Vaughan Pratt

Comparisons of Composite Simplex Algorithms
Irv Lustig *Advisor: Prof. George Dantzig*

Adaptive Numerical Methods for Weather Prediction
Bill Skamarock *Advisor: Prof. Joseph Oliger*

The Proposed Center for the Study of Human-Computer Interaction
Prof. Keith Lantz

Fitting Shapes to Points
Prof. Vaughan Pratt

Luncheon 12:15 p.m., Thurs., 5 Feb. 1987

Tresidder Union, Room 281

NOTES

Technical Session IV A 1:30 — 2:45 p.m., Thurs., 5 Feb. 1987

CERAS Rm. 112

Session Chairman: Prof. Nils Nilsson

Constraints, Planning, and Learning: How to Analyze and Synthesize Hardware
Prof. Daniel Weise

A Theory of Justified Reformulations
Devika Subramanian *Advisor: Prof. Michael Genesereth*

Distributing Backward-Chaining Deductions to Multiple Processors
Vineet Singh *Advisor: Prof. Michael Genesereth*

On Default Sub-Theories
Benjamin Grosof *Advisor: Prof. Nils Nilsson*

******ShortBreak******

NOTES

Technical Session IV B 1:30 — 2:45 p.m., Thurs., 5 Feb. 1987

Tresidder, Cypress Lounge North and South

Session Chairman: Prof. Terry Winograd

The TABLOG Programming Language
Eric Muller

Parallel Execution of Lisp Programs
Joe Weening *Advisor: Prof. John McCarthy*

Shortest Paths and Visibility Problems in Simple Polygons
John Hershberger *Advisor: Prof. Leo Guibas*

Increasing the Expressive Power of Formal Systems
John Lamping *Advisor: Prof. Terry Winograd*

*** * * * Short Break * * * ***

NOTES

Technical Session V A 3:00 — 4:00 p.m., Thurs., 5 Feb. 1987

CERASRm.112

Session Chairman: Prof. David Ungar

Towards Efficient Maintenance of Integrity Constraints
XiaoLei Qian *Advisor: Prof. Gio Wiederhold*

Synchronizing Plans among Intelligent Agents via Communication
Charles Koo *Advisor: Prof. Gio Wiederhold*

High Speed Implementations of Rule-Based Systems
Prof. Anoop Gupta

Reception 4:30 — 6:00 p.m., Thurs., 5 Feb. 1987

Faculty Club, Red and Gold Lounges

An informal reception will be held at the Faculty Club to enable our visitors to become better acquainted not only with the faculty and students they have seen during the past two days, but also to meet the students and staff of the entire Computer Science Department and Computer Systems Laboratory.

NOTES

Technical Session V B 3:00 — 4:20 p.m., Thurs., 5 Feb. 1987

Tresidder, Cypress Lounge North and South

Session Chairman: Prof. Mark Linton

Call-by-Unification: How to Combine Logical and Functional Programming Efficiently
Per Bothner *Advisor: Prof. Brian Reid*

Editing Procedural Descriptions of Graphical Objects
Paul Asente *Advisor: Prof. Brian Reid*

Composing User Interfaces with Interactive Views
Craig Dunwoody *Advisor: Prof. Mark Linton*

Incrementally Linking Programs
Russell Quong *Advisor: Prof. Mark Linton*

Reception 4:30 — 6:00 p.m., Thurs., 5 Feb. 1987

Faculty Club, Red and Gold Lounges

An informal reception will be held at the Faculty Club to enable our visitors to become better acquainted not only with the faculty and students they have seen during the past two days, but also to meet the students and staff of the entire Computer Science Department and Computer Systems Laboratory.

NOTES

Stanford Computer Forum Committee, 1986/87

Ann Diaz-Barriga – Assistant to the Manager, Computer Forum

Craig Cornelius – Research Associate, KSL Alternative Representative

Helen Davis – CSPh.D. candidate; CSL student representative

Craig Dunwoody – EEPh.D. candidate; CSL student representative

Robert Engelmore – Senior Research Associate, KSL, Department of Computer Science

Barry Hayes – CSPh.D. candidate; CSD student representative

Katie Mac Millen – Publications Coordinator, Computer Forum

William F. Miller – Director, Computer Forum; Prof. of Computer Science; Pres. of SRI International

Karen Pieper – CSPh.D. candidate, CSD student representative

Eileen Schwappach – CSL Manager

Betty Scott – Administrative Officer, Department of Computer Science

Richard Shuey – Honorary Member

Carolyn Tajnai – Manager, Computer Forum; Asst. Chair, CSD, for External Relations

David Ungar – Assistant Prof. of Electrical Engineering

Terry Winograd – Associate Prof. of Computer Science

Table of Contents

Technical Session I A, Prof. Thomas Binford, Chairman

Technical Session I B, Prof. John Hennessy, Chairman

Technical Session II A, Prof. Ed McCluskey, Chairman

Technical Session II B, Prof. Ted Shortliffe, Chairman

Technical Session III A, Prof. Michael Flynn, Chairman

Technical Session III B, Prof. Vaughan Pratt, Chairman

Technical Session IV A, Prof. Nils Nilsson, Chairman

Technical Session IV B, Prof. Terry Winograd, Chairman

Technical Session V A, Prof. David Ungar, Chairman

Technical Session V B, Prof. Mark Linton, Chairman

Statistical Approaches to Perceptual Organization

Rick Vistnes

Perceptual organization in vision groups the image in ways that aid interpretation. Examples include the linking of edge elements, or other image features such as endpoints, into lines and curves; detection of parallelism and symmetry; detection of edges of textured regions. Figure-ground separation, one result of perceptual organization, is a precursor to recognition. Image organization provides useful clues for the problem of inferring 3D scene structure from the 2D image. Useful image relations have three important properties:

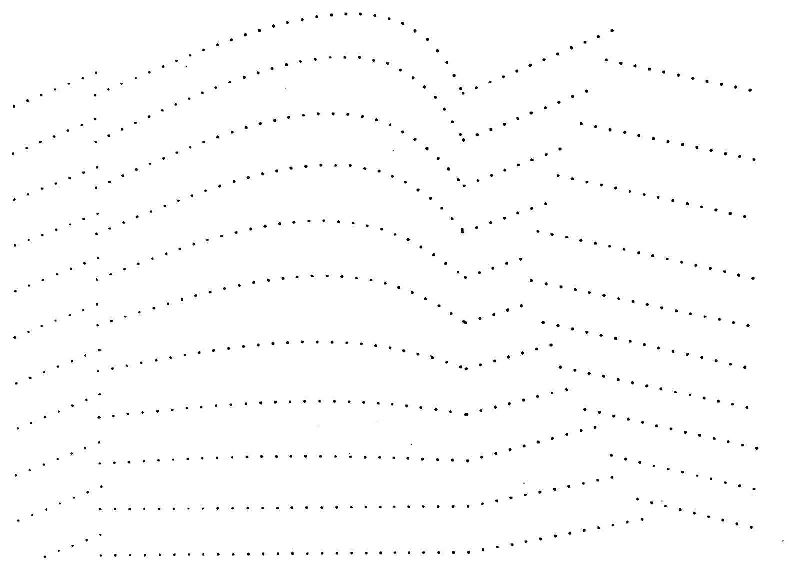
- they constrain image interpretation,
- they are unlikely to arise by chance, and
- they are useful for describing images in general.

Most image relations do not share these properties; e.g., right-angled vertices, circular arcs.

Once a useful image relation is identified, a vision system may search for it in the image. We use a statistical approach to separate those instances of the relation that could have arisen by chance from those that have a causal interpretation. In fact, finding instances of statistically significant relations in the image has been proposed as the general goal of perceptual organization. Relations detected in this manner are not subject to arbitrary detection thresholds; rather, their existence in the image is paired with an estimate, based on the properties of the surrounding image, of the probability of accidentalness.

As an example, we have studied the perception of dotted lines embedded in a random-dot background. A series of psychophysical experiments measured human detection ability. A model for dotted line detection is based on finding the statistical significance of the dot density in the center of an elongated operator compared to that in a local surround. This model generates predictions about performance; these predictions compare well with human performance, as do results of a computer simulation of dotted-line detection. These results suggest that the model is sufficient to explain human performance in this task. Implementation of a line-finding algorithm based on these ideas is underway.

Future work will apply similar statistical methods to other aspects of perceptual organization --- in particular, texture segmentation.



Example of perceptual organization. This image illustrates many human perceptual organization facilities. Dots arc grouped into curves; parallel curves are perceived. Endpoints and corners are link4 to form curves

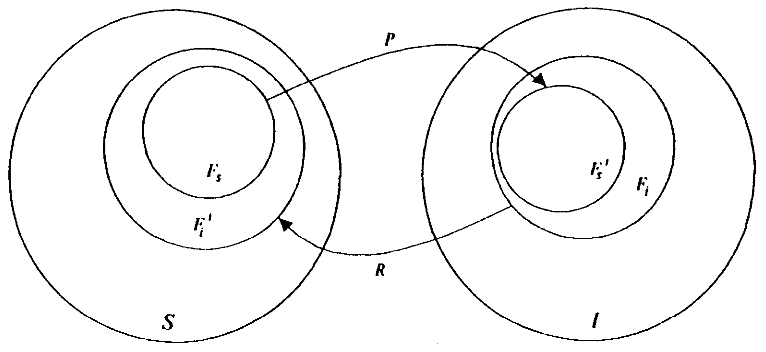
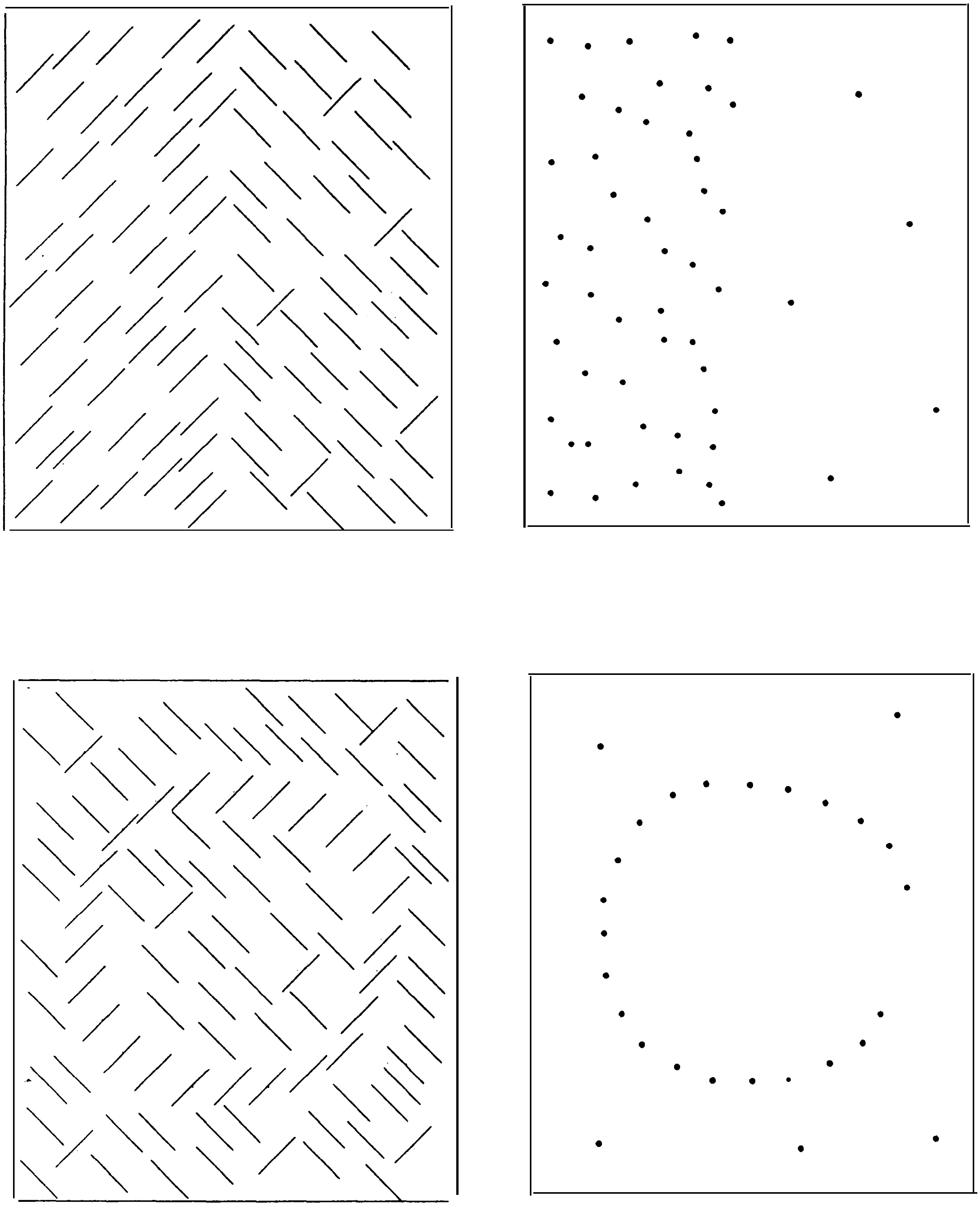
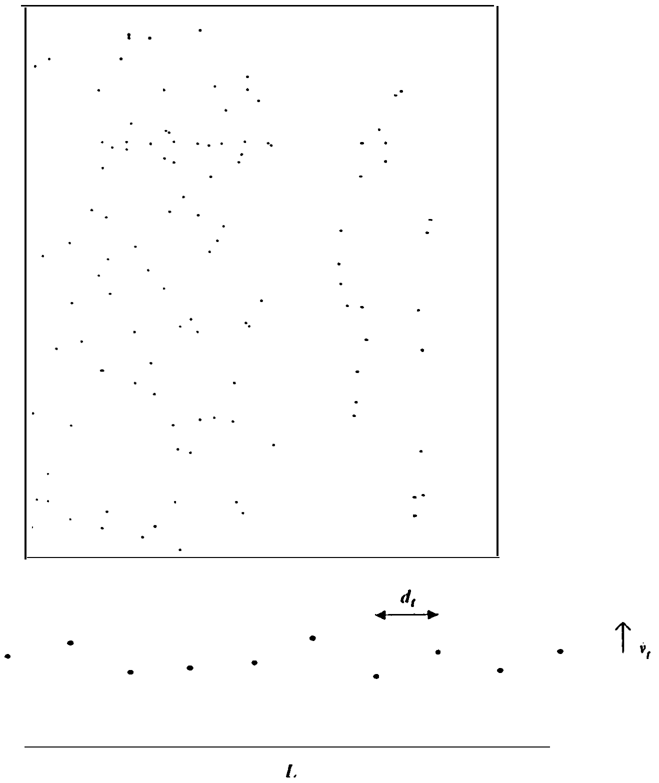


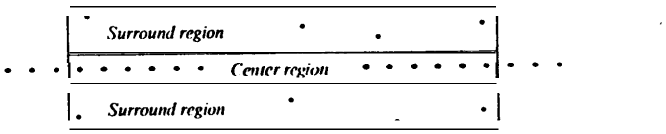
Image interpretation. The projection P from scene to image maps a set of scenes where a fart F_s is true to a set of images where fact F_s' is true. F_i is the set of all images where F_s' is true; an inference rule (harkprojection) R maps this set to a set of scenes where F_i' is true. If these two sets of scenes are nearly equal in size, then R is reliable and constrains interpretation.



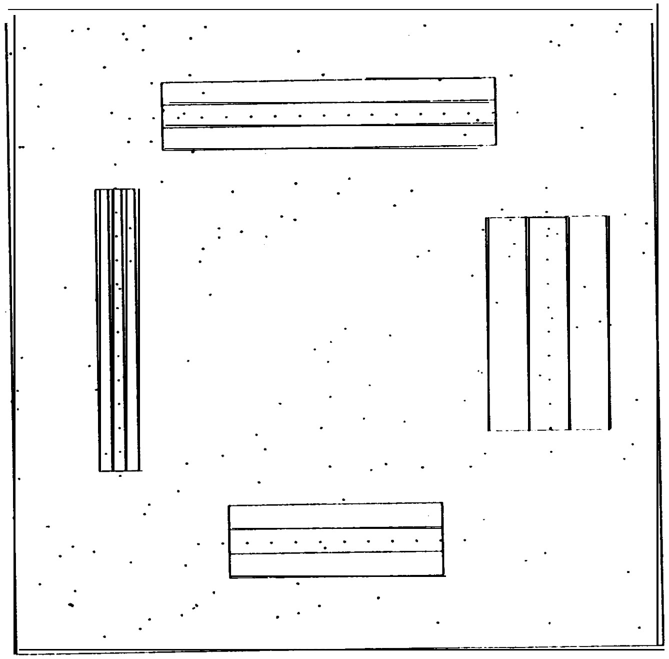
Texture segmentation is based on differences in density of "textons" in two regions. In this case, the texton cue is line segment orientation. Figures on the right were obtained from those on the left by replacing segments of a particular slope with a dot. Both 2D and 1D figures may result, as shown here.



Examples of stim   ;ed in line-detection experiments. Dotted lines appeared in o   of fou
positions; they va   n transverse variation v_t and dot spacing d_t. Detection depends   targe
dot spacing relati   surround dot spacing; as v_t increases, detection ability decrease   Result
are nearly indeper   of actual dot density.



Dotted-line detection operator used in simulations. The number of dots in th   nt N_c, and
in the surround, N_s, is counted. Under the null hypothesis that the dot dens   in   nter and
surround are equal, the probability that N_c center dots or more would be obser   is   lculated;
as this probability decreases, the probability increases of a causal interpretation.



Placement of operators in line-detection simulations. Operators at each position had various sizes and widths; each computed the significance of the number of dots in its center. The operator with highest significance was chosen.

Stereo Correspondence: a Hierarchical Approach

Hong Seh Lim

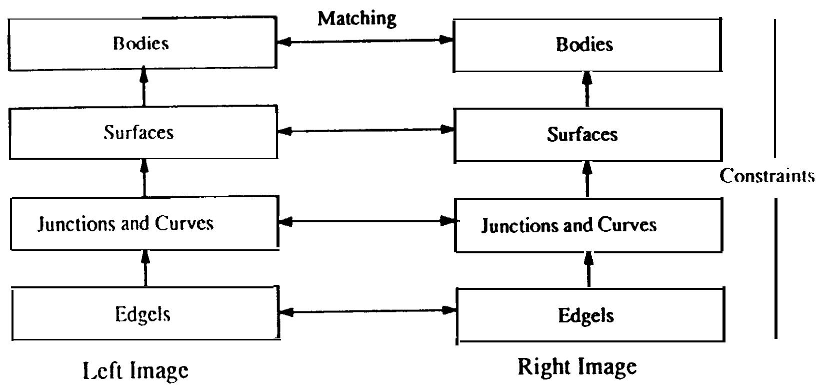
This talk describes a high-level stereo vision system for recovering depth data of a three-dimensional scene from a stereo pair of gray-scale images. The features chosen for stereo matching are edgels (i.e., short, linear edge-elements, each characterized by a direction and a position), junctions, curves, surfaces, and bodies. The system first performs a monocular interpretation on the scene in each image. It builds a hierarchical structure of the scene from the chosen features. Low-level features constitute the lower layers of this hierarchical structure, while high-level features form the higher layers. The system starts matching features between the hierarchical structures at the highest level. Results from the matching of high-level features are used to constrain and guide the matching of lower-level features. These constraints and guidance are propagated to the lowest level of the hierarchical structure. The results from the matching provides a segmented depth map of the scene.

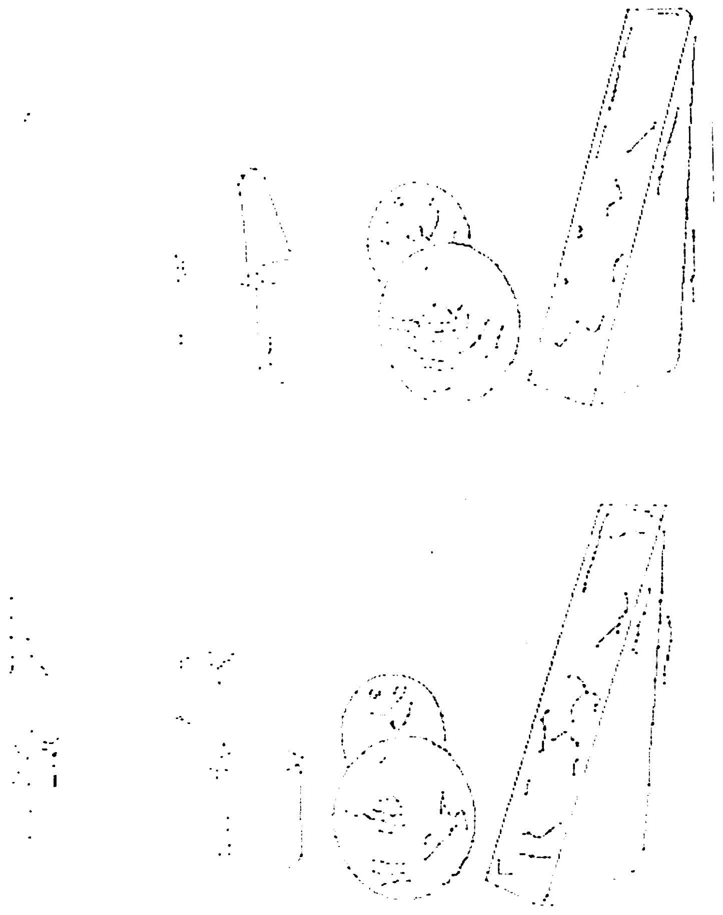
This hierarchical approach enables a more globally consistent matching result and avoids local mismatches. It reduces searching time for locating correspondence features during the matching processes. The resulting depth map is segmented and readied for surface interpolation.

Features and Constraints

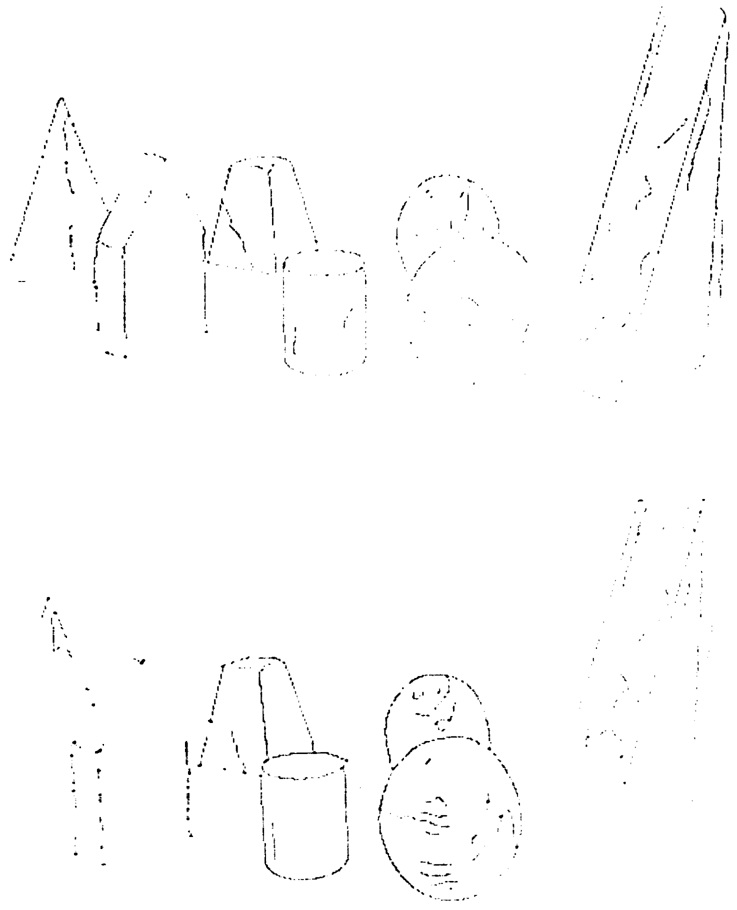
- Features
    - ○ Edgels
    - ○ Junctions and Curves
    - ○ Surfaces
    - ○ Bodies
- Constraints
    - ○ Epipolar Constraints
    - ○ Geometrical Constraints

Hierarchical Structure





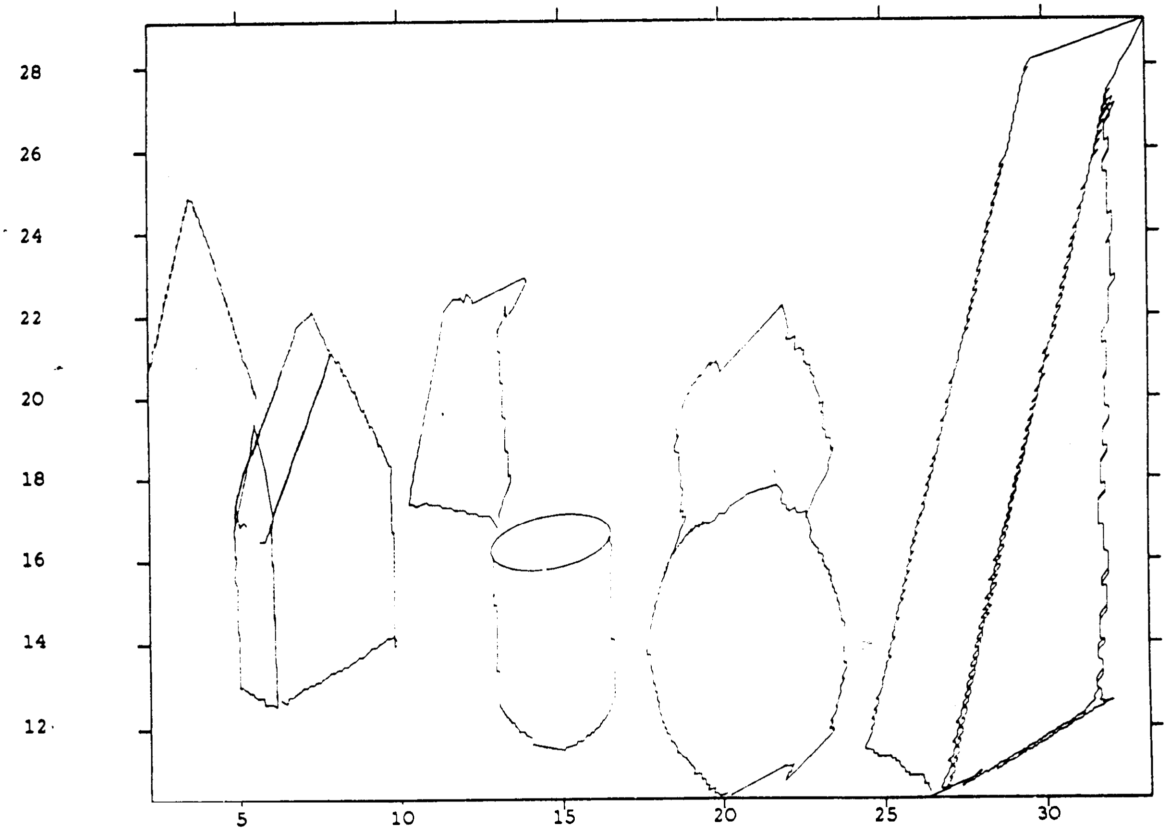
Results from Edge Detection and Edgels Aggregation



Results from Curves Extension and Junctions Detection

Important Features

- Hierarchical Matching
- Features are Matched Globally
- Consistent Matching of Features
- Matching Time is Reduced
- Results are Segmented



Segmented three dimensional data

Line-Drawing Interpretation: Bilateral Symmetry

Vic Nalwa

Symmetry manifests itself in numerous forms throughout nature and the man-made world. It is frequently encountered in line-drawings corresponding to edges in images. We seek to discover constraints on the imaged surfaces from instances of bilateral (reflective) symmetry in line-drawings obtained from orthographically projected images. A **general** viewpoint is assumed, i.e., it is assumed that the mapping of the viewed surface onto the line-drawing is stable under perturbation of the viewpoint within some open set on the Gaussian sphere. It is shown that the only planar figures which exhibit bilateral symmetry under orthographic projection with a **general** viewpoint are ellipses and straight-line segments.

We show that the orthographic projection of a surface of revolution exhibits bilateral symmetry about the projection of the axis of revolution, irrespective of the viewing direction. Further, it is shown that whenever the back-projection of the symmetry axis is invariant in space under perturbation of the viewpoint, the bilaterally symmetric line-drawing is the orthographic projection of a local surface of revolution. The axis of revolution is the back-projection of the symmetry axis. Various line-drawing configurations for which the back-projection is invariant are detailed. It is conjectured that isolated ellipses and isolated straight-line segments are the only bilaterally symmetric line-drawings whose back-projections of the symmetry axes are not invariant under perturbation of the viewpoint. Throughout, only surface regions local to the scene-events corresponding to the lines are constrained.

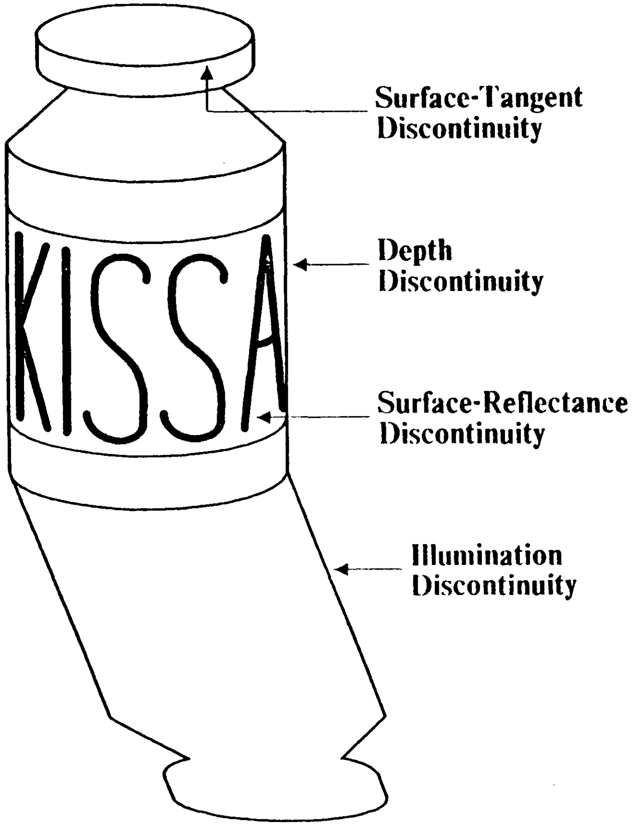

KISSA
Surface-Tangent Discontinuity
Depth Discontinuity
Surface-Reflectance Discontinuity
Illumination Discontinuity

Fig. 1. Line-Dmwing with Various Types of Edges

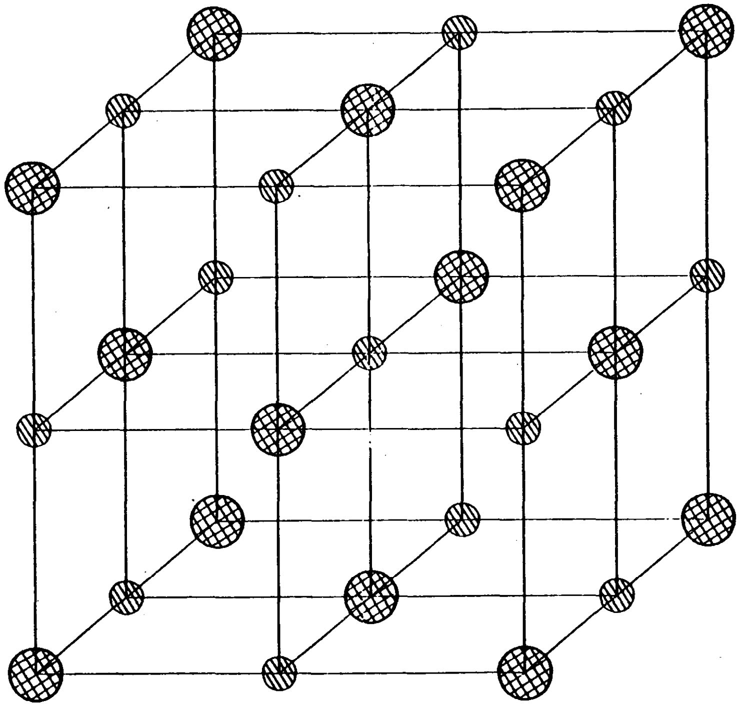


Fig. 2-a. Crystals Exhibit a Symmetric Atomic Armngement (The Sodium Chloride Crystal Structure is Shown)

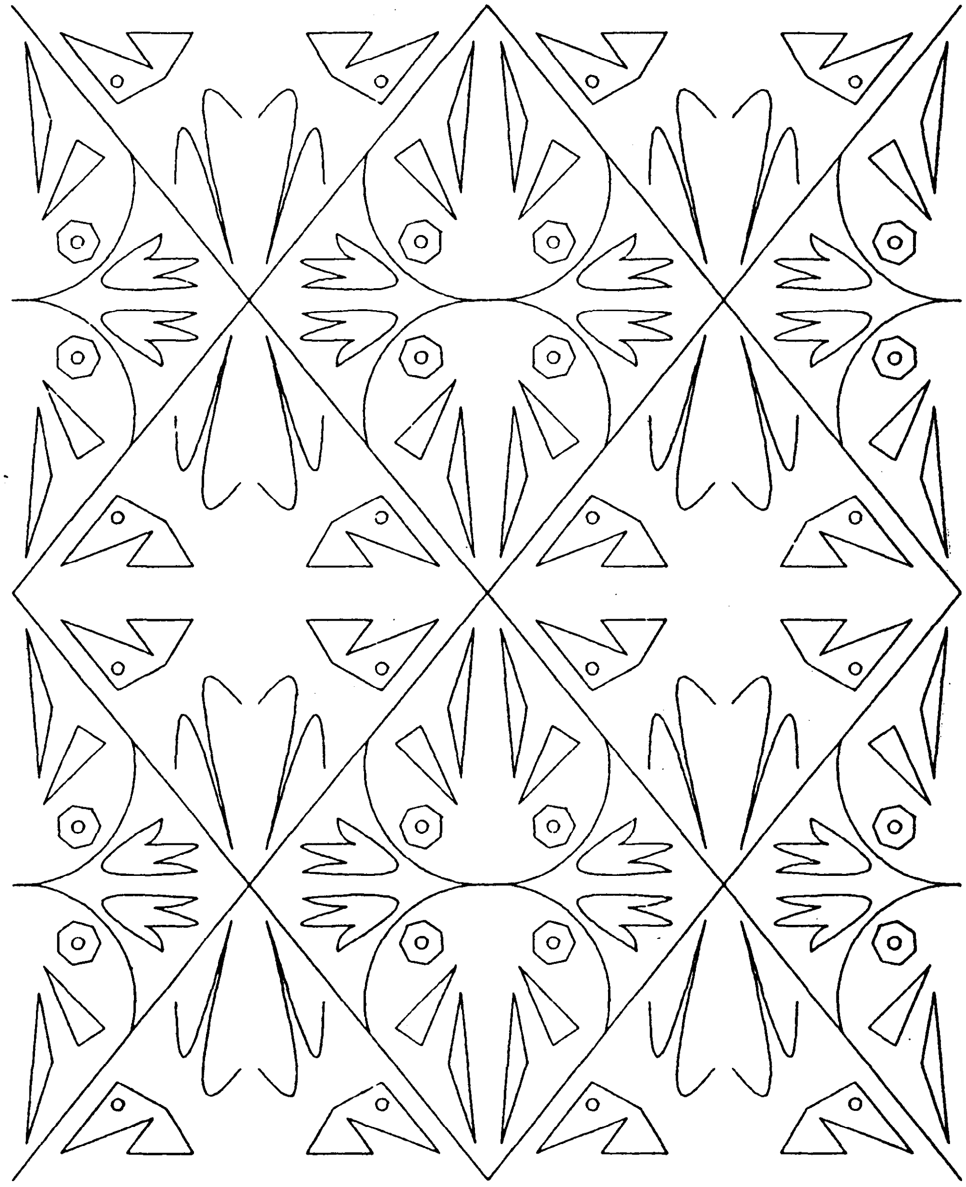


Fig. 2-b. Wallpaper Pattern Exhibiting Various Symmetries

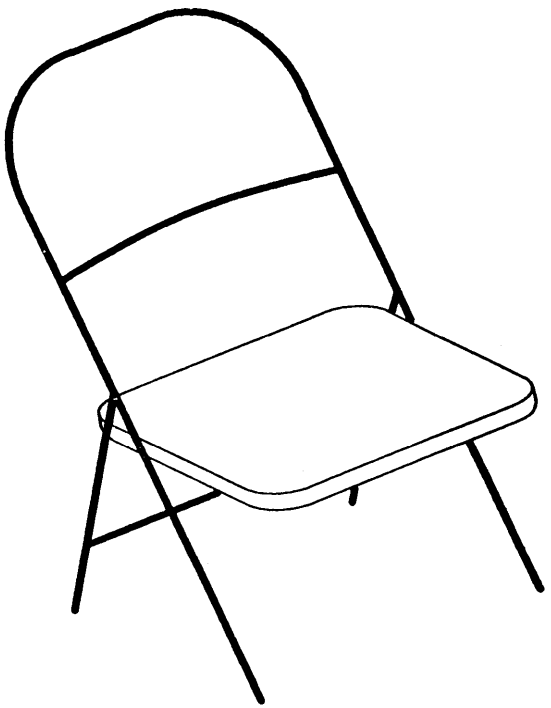


Fig. 2-c. A Chair Exhibits Reflective Symmetry in Space

Problem Addressed

Given :

A line-drawing with bilateral (reflective) symmetry.

Objective :

Constrain the 3-D scene rigorously.

Assumptions :

1. Orthographic projection.
2. General viewpoint.
3. Piecewise C ^{3} surfaces.

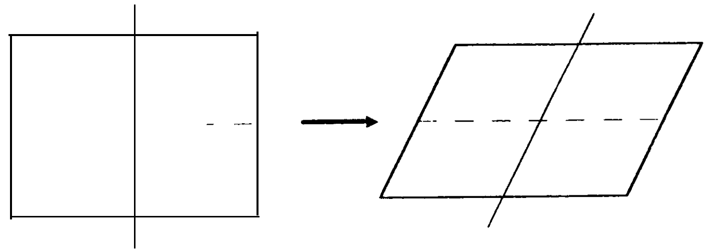


Fig. 3. Bilateral Symmetry in a Planar Figure Maps to Skewed Symmetry under Orthographic Projection

Surface of Revolution

Sufficient for Bilateral Symmetry?

Yes.

Necessary for Bilateral Symmetry?

Globally. No.

Local to the scene-edges. Yes, if under perturbation of viewpoint, the symmetry-axis is the projection of a fixed line in space.

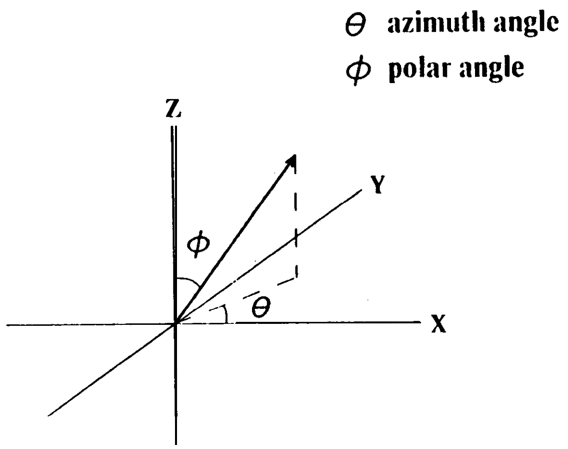
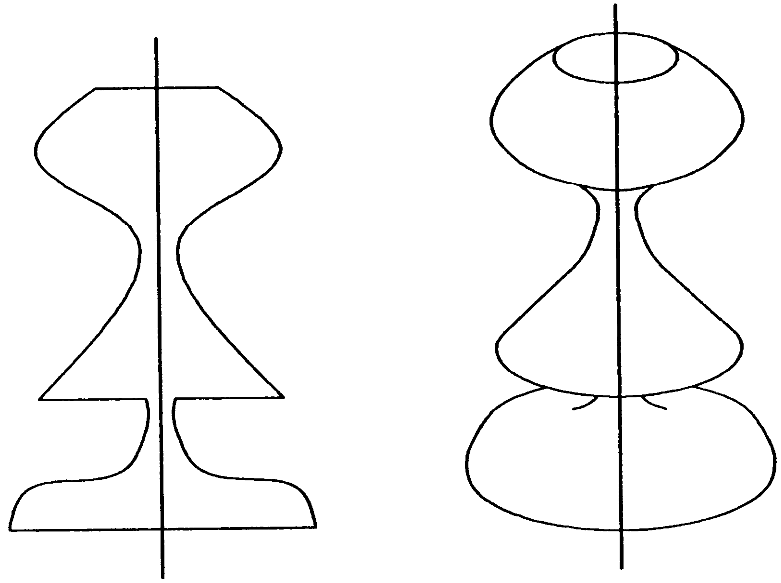


Fig. 4. The Coordinate Frame



Diametrical Cross-Section

Orthographic Projection

Fig. 5. A Surface of Revolution

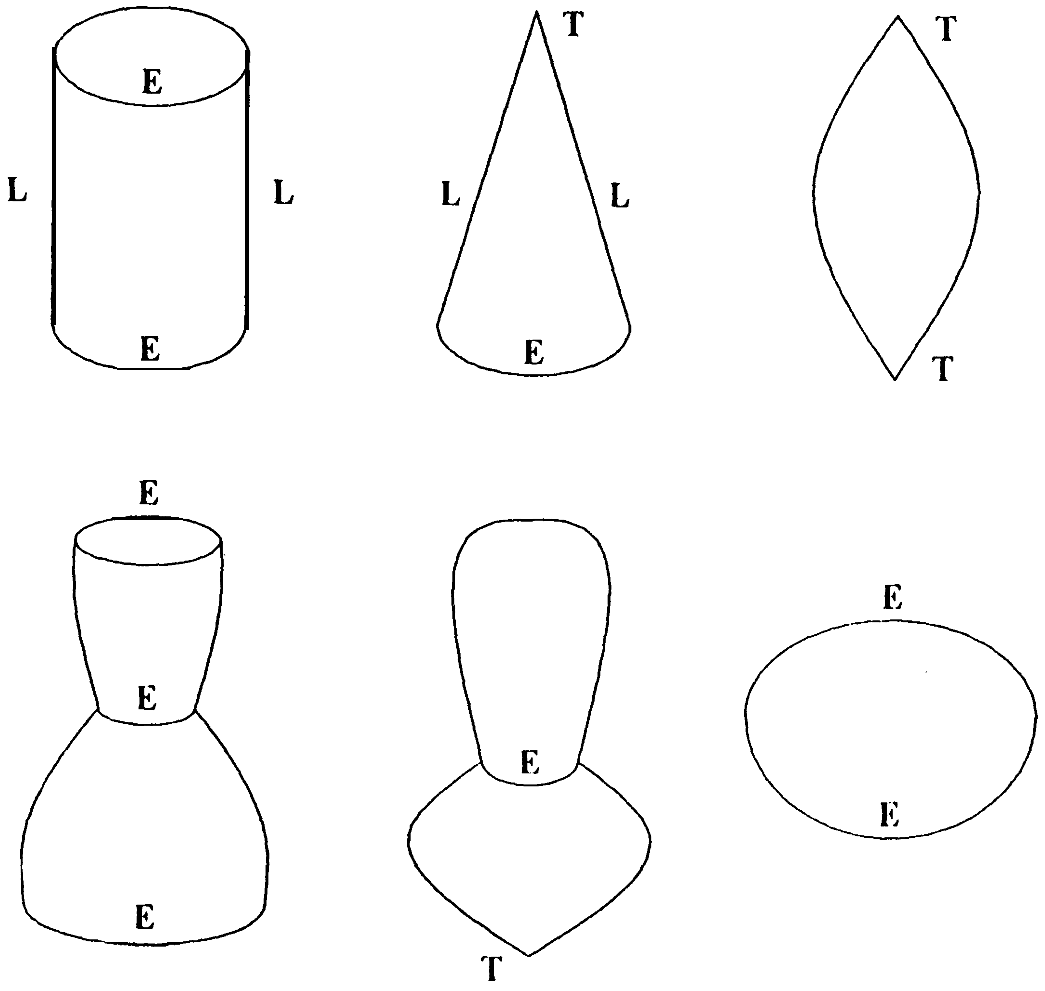


Fig. 6. **Examples of *Bila terally* Symmetric *Line-Dra* wings *from which* Local *Surfaces* of Revolution *may be* Deduced (Cues : L - Line, *E* - *Ellipse*, T - *Tip*)**

Conclusions

1. The projected surface is a local surface of revolution if, under perturbation of viewpoint, the axis of bilateral symmetry is the projection of a fixed line in space.

2. Currently, the back-projection of the symmetry-axis can be deduced to be fixed from the following cues.

A symmetric pair of straight-line segments.

Two events from among tangent-discontinuities on the symmetry-axis and elliptical segments bisected by the symmetry-axis.

3. The only planar curves which exhibit bilateral symmetry under projection are ellipses. and straight-line segments.

4. It is conjectured that the only bilaterally symmetric line-drawings without a fixed back-projection of the symmetry-axis are isolated ellipses and isolated straight-line segments.

Reference : "Line-Drawing Interpretation : Bilateral Symmetry," V.Nalwa, *Proc: IU* ***Workshop,*** Los Angeles, Feb. 1987.

Some Extensions on the Motion Specification of AL Robot Language

Chun-Sheng Cai

A new version of AL robot language editor and interpreter system has been implemented, with commands extended for operational space control, flexible grasping and 3-finger-hand operation. This talk will discuss some of its extensions on the object position and orientation trajectory specification, including the trajectory modification from the force sensory information. The talk will also discuss the motion control frame determination either in free space or with direct contact from the specified trajectory, and the tolerance, wobble and stiffness specification associated with the control frame. Finally the talk will give an easy Jacobian matrix inverse from the general control frame.

Some extensions on the motion specification of AL robot Language

1. Background

Friendly and intelligent user interface

syntax oriented editing, two window display, frame location determination

Multiple devices

Language features

operational space, flexible grasping, 3-finger-hand

2. Motion object

Simple geometry

Parametric representation

Compound geometry

3. Cartesian trajectory specification

Interpolating position path through via points

Runtime modification from force sensory information

Interpolating rotation path through via orientations

Quaternion interpolation trajectory

Reference line tracing a sequence of planes

Reference line tracing a smooth cone

Interpolation rotation path specification

Other plan trajectory

Helicoidal path, min-time path, sensor based collision free local path

Runtime conditions monitored motion

4. Motion control frame determinaton

The position control frame in free space and with direct contact

Cases of purely determined from trajectory

Cases of determined from a trajectory and a moving object geometry

kinematic and force conditions

The rotation control frame in free space and with direct contact

kinematic and force conditions

5. Motion specifications and cases of control frames coincidence

Position error tolerances

Wobble coefficients

Contact stiffness

The dependence between position and force trajectories

Case of coincidence of a position and rotation control frame

6. An simple Jacobian matrix inverse for a general control frame

Force Control and Design of Robot Manipulators

Joel Burdick

This talk will cover two areas of research--manipulator force control and the design of redundant manipulator arms--that are important to the development of robot manipulators capable of performing complex tasks. Typical industrial robot manipulators are programmed to operate in a "pick and place" mode in which the robot is commanded entirely in terms of goal positions. However, there are many important tasks in which the sensing and command of end-effector position alone is insufficient, and where the simulataneous control of both end-effector position and force is necessary. Past research in the design and implementation of control algorithms for high performance force control of robot manipulators will be reviewed, and experimental results involving generic force control operations will be presented.

Typical industrial robots are designed to have at most six joints. However, there are compelling reasons why robots should be designed with more than six joints. Some of the reasons for designing redundant manipulators (manipulators with more than six joints) will be reviewed, and in particular, the improvement of manipulator dexterity by use of redundancy will be considered.

Force Control and Design of Robot Manipulators

Joel W. Burdick

I. New Approaches to Advanced Manipulation

- a. Force Control
  - 1. Overview of Manipulator Force Control
  - 2. Implementation and Experimental results
- b. Redundancy in Manipulator Design
  - 1. Uses for Redundancy
  - 2. An Example: Dexterity

Why Force Control?

1. Force Control in Assembly

- a. Eliminate Complex Fixtures
- b. Unstructured Environments
- c. Accomplish New Tasks
- d. Inspection

II. Multi-Arm Coordination

- a. Compliant Accomodation
- b. Multiple Modes of Interaction

III. Human/Robot Interaction

- a. Teaching
- b. Safety

Some Generic Problems in Force Control

a Robustness

Stability in the presence of large variations and perturbations

b. Bandwidth

Increased bandwidth of closed loop manipulator structure

c. Impact and Bounce

High speed parts mating without bounce or chatter

d. Simultaneous Motion and Force Specification

Force Control Algorithms

I. Some Generalities

a. Force Control = Dynamics

b. Higher Bandwidth

II. Some Desirable Features

a. Parallelizeable Computations

b. Efficient Algorithms

c. Decoupling

III. A Particular Algorithm: Operational Space

a. A Unique Dynamic Model

b. Some Auxialiary Research: EMDEG

Implementation and Experiments

COSMOS Motion and Force Control System

Implements an Operational Space Control System

NYMPH Multiprocessor

a. Four 32-bit Microprocessors
b. Kinematics and Servo Rate: 200 Hz
c. Dynamics Computation Rate: 100 Hz

PUMA 560 Manipulator

Force Sensors

a. Force Sensing Wrist
b. Force Finger Sensors

Experimental Results

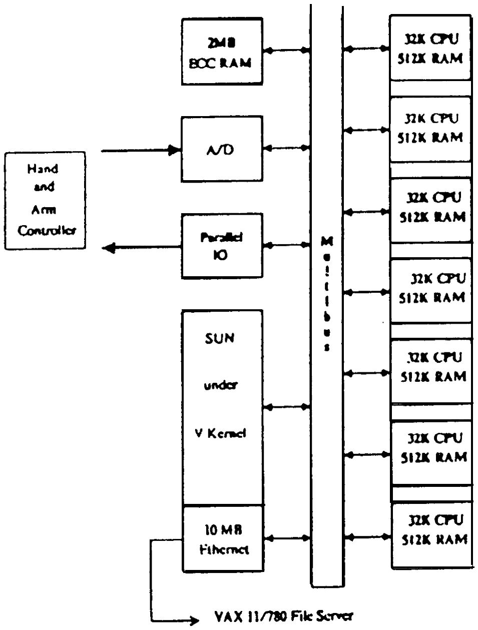
I. Generic Force Control Operations

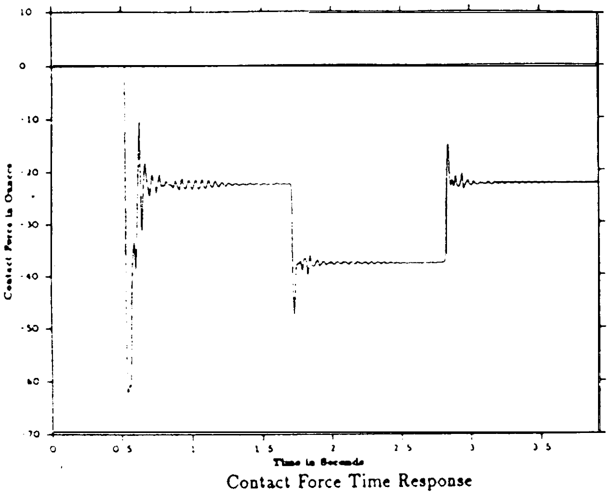
a. Static Force Control
b. Contact and Impact Stability
c. Slide
d. Tracking

II. Force Control Performance

a. Minimum Force
b. Robustness
c. Response Time

NYMPH System





Contact Force Time Response
using Force Sensing Fingers

Redundant Manipultors

I. Typical Industrial Manipulators:

1. Six Degrees of Freedom
2. In reality, this is often insufficient

II. Extra Degrees of Freedom

a. Used to attain an additional task constraint
   1. Obstacle Avoidance
   2. Internal Geometric Contstraints

b. Used to Optimize some Criterion
   1. Singularity Avoidance
   2. Inertial and or Mechanical Properties
   3 Force Controllability
   4. Dexterity
   5. . . .

III. Generalized View of Redundancy

a. Complex Mechanical Hands

b. Multiple Arm Coordination

c. Multi-Legged Walking Mechanisms

d. Non-Redundant manipulators performing symmetric tasks

IV. Example: Dexterous Workspace Optimization

a. Dextrous Workspace

b. Real robot joints can't be be with complete rotation due to mechanical constraints

c. Limited Rotation 6R robots have almost no **dex**trous workspace

1. constraints on fixturing

2. Limits complex continuous path operations

d. adding extra joints can improve dexterity

Conclusions

' Advanced Manipulation

Can be achieved through high level AI/programming techniques

Improved design of manipulators and control algorithms

Perception with Simulated Dynamics

Michael Kass

A paradigm for perception based on the notion of simulated dynamics will be presented. Physical models of the objects or features in a scene are created and subjected to forces. Some of the forces can be automatically derived from the image and act to deform or move the model into correspondence with the image data, while other forces can be exerted by a user to push the model into a particular intended local minimum. Applications of this paradigm to edge-detection, motion tracking, stereo, and solid modeling will be presented.

MIPS-X

Mark A. Horowitz

MIPS-X/MP is a Stanford research project exploring the issues in designing a very high performance multiprocessor. This talk will focus on one part of this project, the design of a new high performance processor. MIPS-X borrows heavily from the original MIPS design. Like MIPS, it is a load-store based machine, with 32 registers and uses a very simple instruction set. There are three kinds of instructions: Memory, Compute, and Branches. All instructions are fixed format, making the decode unit very simple. The machine is heavily pipelined, and the pipeline contraints (branch delay of two and load delay of one) are handled by software reorganization.

MIPS-X contains a 2K byte on-chip instruction cache and a large external cache. Trace driven simulations indicate the machine will average around 1.75 cycles/instruction. At 50 ns cycle time, this will give about 12 MIP average throughput. The first part of the talk will describe the overall architecture and organization of MIPS-X, and the second part will describe implementation issues: design method, tools used, and current status.

What is MIPS-X?

- MIPS-X - high performance 32-bit **μP** (-10 MIPS)
- MIPS-XMP - shared memory multiprocessor (-100 MIPS)

Why do this?

- Investigate high performance microprocessor design and advanced compiler issues
- **Testbed** for CAD tools
- Investigate multiprocessor issues - hard and soft

Goals for MIPS-X:

- Single-cycle execution
- Simple control
- Fixed-format, 32-bit instructions
- Simple and efficient coprocessor support
- Instruction cache
- **2μ,** 2-level metal CMOS technology
- Nonoverlapping, 2-phase, 20 MHz clock

Architecture

Load - Store Machine

Instructions:

- Memory [Offset + (Rn) ⇒ Address]
- Branch [Explicit compare included]
- Compute

Instruction mix is based on C and Pascal program analysis but we have also ported LISP.

Hardware Resources:

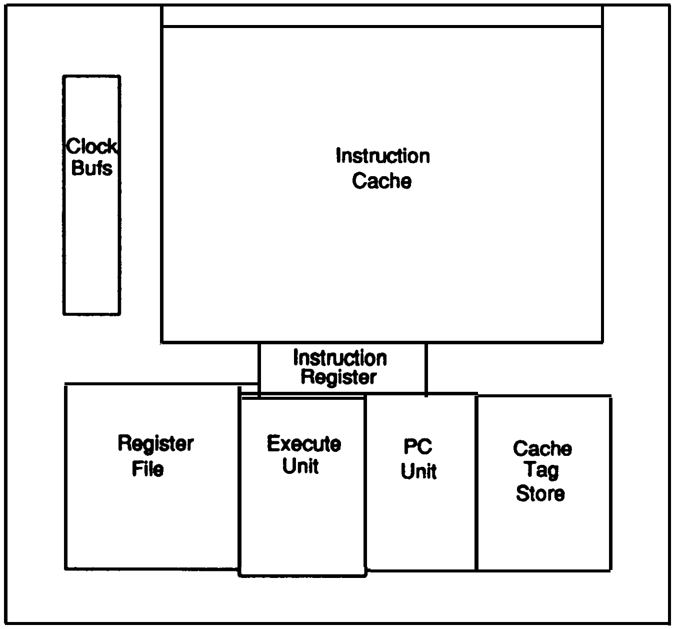
- 32 General Purpose Registers
- 32-bit Arithmetic Logic Unit
- 32-bit Funnel Shifter
- 32-bit Displacement Adder
- 32-bit **Incrementer**
- **2K-byte** Instruction Cache

Machine Timing

```
Icache - 1
Decode - 0.5
Reg    - 0.5
ALU    - 0.5
Mem    - 1.5
WB     - 0.5
```

Pipeline

IF	RF	ALU	MEM	WB		
.	IF	RF	ALU	MEM	WB	
.	.	IF	RF	ALU	MEM	WB
.	.	.	IF	RF	ALU	MEM



Divide and Conquer

7 functional parts

- Instruction Cache
- Instruction Register
- **Tagstore**
- PC Unit
- Execute Unit
- Register File
- External Interface
- **Padframe**

Very precise interface specifications were made - what signals, what timing, where they went in and out of each section.

First step: functional simulator model for each part

Full Testing

Functional simulator drives Rsim model

Rsim was used to get most of timing information - TV only used at the very end

One database and 4 **μvaxen** running NFS

1 clock cycle per minute

Run various programs

About **10K** cycles before shipping

Status and Conclusions

- Started in summer of 1984, taped out May 1986
- Silicon back in October - works at 17 **MHz**
- Die
  - 8Smmby8mm
  - 108 pins - 24 power, 84 signal
- **150k** transistors - **2/3** in cache
- Dissipates less than 1 W
- Sustained throughput including no-ops and cache misses - 12 MIPS
- Controllers - 0.2% of silicon area

Other

- Partitioning of chip worked well
- Magic was a big win - never did full DRC or extract of entire chip
- Start with a good functional model
- Functional simulator checking Rsim automatically made life much easier
- Need someone with a lot of circuit hacking experience but keep him in control! Otherwise end up with a bottom up design.
- If you didn’t simulate it, it doesn’t work!! !

Knowledge-Based Monitoring and Control: An Architecture for Distributed Systems

Bruce Hitson

Understanding the behavior of complex distributed systems is extremely difficult for a number of reasons: volume of data involved, difficulty of collection, incorrect interpretation or collection techniques, extra work required (beyond basic functionality), and inexperience of programmers, to name a few. We describe an *environment* in which the behavior of complex systems can more easily be understood. There are two major aspects to the environment: an *architecture* for monitoring and controlling a wide range of systems, and *knowledge -based* tec hniques for interpreting the values that are collected. An expert system shell containing meta-level knowledge about monitoring and control is used in conjunction with expert knowledge in specialized domains to quickly and conveniently construct *expert monitoring consultant* programs for particular applications.

This talk will focus on advanced *ENVIRONMENTS* for studying the behavior of distributed systems.

Claim: proper structuring and features for monitoring and control (architecture) and expert system techniques (knowledge-base) can be combined to produce interesting systems.

Some key aspects of environments:

- user background: non-uniform, non-monitoring; need to allow users to ask interesting questions about the behavior of (their) applications
- volume of data: potentially very large, dispersed; need to capture only "interesting" information
- statistical significance: hard to guarantee. Separate aberrations from trends (e.g., track history)
- incremental cost: don't "reinvent the wheel" for every application needing monitoring or control

Motivation for advanced monitoring and control:

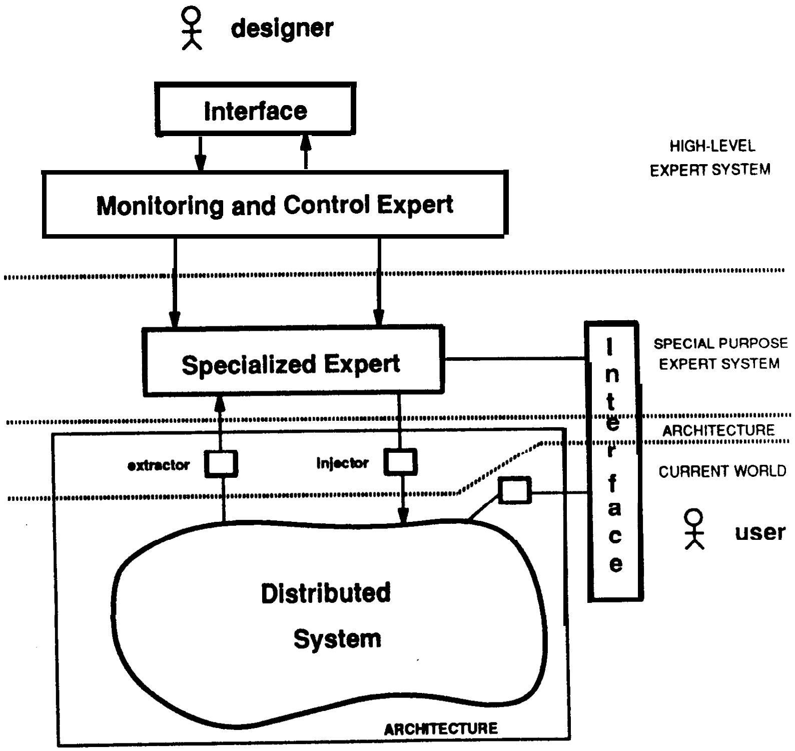
- system complexity is increasing (distributed, heterogeneous, multiple applications, more hosts)
- application complexity is increasing even faster (larger, longer running (servers), sophisticated)
- shortage of *human* monitoring and control experts; desire to capture and store monitoring knowledge for replication and teaching
- administrative considerations: existing systems have and future systems and applications will need:
  - accounting: load balancing; billing for services
  - security: detecting abnormal patterns or faults

THE SOLUTION

Provide an *environment* for studying the behavior of complex distributed systems. Two key components:

- architecture for monitoring and control
  - common (low) levels of abstraction; solid base for higher-level functionality
  - new approaches for distributed systems
- knowledge-based techniques
  - leverage: bring more effort to bear on the problem through automation
  - use domain-specific and domain-independent techniques for different knowledge types
  - encode the knowledge of monitoring and control experts and make it widely available

Multi-Level Knowledge-Based Environment



OBSERVATIONS

- current systems: non-existent or primitive monitoring. Application specific. Simple interface.
- architecture: adds structure, generality. Makes common or important concepts more apparent. Greatly improves portability. Better user interfaces more important (and easier to do).
- special-purpose expert: tailored to one particular system. Developed in standard ways. Architecture allows expert monitoring and control shells as starting point (speeds development).
- general-purpose expert: meta-level knowledge about monitoring. Provide initial knowledge for constructing special-purpose experts. Alternatively, generalize from the knowledge of specialized experts to improve meta-knowledge (learning).

Experiment: TCP/IP Expert

- monitor the behavior of TCP connections (e.g., silly window syndrome, retransmission timers)
- knowledge/issues: protocol details, realtime

Experiment: Trap Analysis Expert

- study traces of network traps (i.e., exceptional events) to identify (potential) problems.
- knowledge/issues: trap types, their relationship and significance, temporal and statistical analysis

Experiment: User Interface Expert

- track behavior of user(s) in a workstation based terminal agent, and adjust based on behavior
- knowledge/issues: usage, preferences, learning

Architecture for monitoring and control is not a radical new concept. Significant other efforts:

- BBN: (Arpa-)network monitoring and control
- Xerox: Alto monitor (and descendents)
- ISO: self-defining msgs, abstract syntax notation
- SURAN project: Smalltalk, Suns, realtime

What is novel about the proposed approach?

- monitoring and control as a *service* ;
  make taxonomy of concepts obvious and clear
- new concepts to cover monitoring in distributed systems (e.g., multicast, server model)
- simplify many concepts into fewer basic concepts
- portable and general architecture; interactive

KNOWLEDGE-BASED TECHNIQUES

YSCOPE (Hellerstein and Van Woerkom - IBM):
expert system shell plus queueing theory knowledge.

Significance: knowledge + queueing theory

ELAS (Chidanand V. Apté - Rutgers):
expert control of interactive software systems

Significance: two /eve/s + worksheet reasoning

Other work:

- RADIX (Blum and Walker - Stanford)
  Discovery of knowledge in large medical databases; similarity to large statistics databases. Interesting but I will not pursue this.

- specifier, DesigNet, configurer (BBN tools);
  traffic spec, network design, device configuration

Problems/limitations of these approaches:

- limited areas of applicability (e.g., diagnosis)
- little work at the meta-knowledge level
- limited domains => difficult to generalize; monitoring and control easier to generalize

What is novel about the proposed approach?

- emphasis on monitoring and control (not previously done; related work emphasizes performance analysis via queueing theory)
- focus on knowledge and meta-level knowledge; use of blackboard or worksheet techniques likely

DETAILS AND STATUS

- first attempt at architecture + basic user interface done. Extensions and elaboration in progress
- early knowledge-based experiments in progress
- implementation in the V-System (architecture); expert systems experiments using Prolog, KEE
- multicast techniques: many uses for efficient monitoring and control in LANs

Summary: this work is novel in that it takes a vertical slice through the components we have described (see figure: low-level, architecture, specialized monitoring expert systems, high-level expert systems), and makes this hierarchy work together in interesting ways.

Log Files: An Extended File Service Exploiting Write-Once Optical Disk

Ross Finlayson

As computer systems encounter greater demands for reliability and accountability, the use of logs becomes increasingly important. In particular, an application or subsystem may use logs for failure recovery, for security (to provide an audit trail) or for performance monitoring. This common need for logging suggests that logs, like conventional files, should be provided as a sy s tern service. Standard file systems, however, are inadequate for a general-purpose logging service. In particular, most file systems perform best on small files, with very large, continually growing files being considerably more expensive. Furthermore, standard disk files cannot make efficient use of write-once storage technology, such as optical disk. Logging, on the other hand, is naturally suited to this new technology.

In this talk, we describe a general-purpose logging service that provides access to “log files” - readable,-append-only files that are accessed in the same way as conventional files. We show how our implementation of log files makes it possible for the logging service to make efficient use of write-once optical disk. We also show how our design is able to support very large, long-lived log files that may be spread over many physical storage devices.

In addition, we argue that the falling cost of RAM, plus the availability of very high density optical disk storage, will make it feasible for applications to be built directly on top of a logging service such as ours. In this way, the application’s logged execution history is treated as being its true state, with RAM acting merely as a cache for this logged information.

Logging is a common system activity:

- reliability/recoverability
  (atomic transactions, checkpoints)

- accountability ('audit trails')

- application-specific logging, e.g.

  - monitoring

  - statistics gathering

Claim: Logs, like conventional files, should be available as a system service.

A single, *genera/-purpose* logging facility is preferable to several, specialized logging facilities.

However, standard file systems are inadequate for a general-purpose logging service, because:

- They perform best on small files; large, continually growing files are considerably more expensive.

- Whole file backup is inefficient for append-only files.

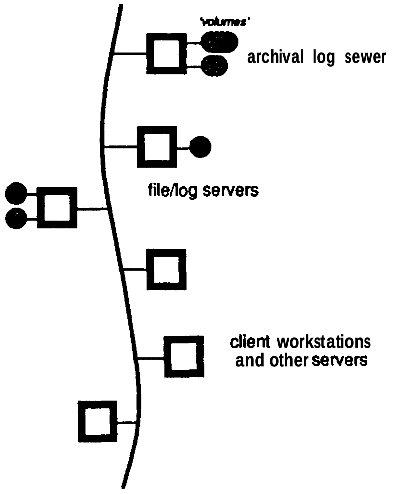
- They do not make efficient use of write-once storage, such as optical disk.

  Fortunately, write-once storage is naturally suited to logging.

Our design: A general-purpose logging service that

- can make efficient use of write-once storage (assume append-only)
- allows for *long term* logging
- is a network service

Assumed environment:



Each volume contains a sequence of *log entries* written by the clients, ordered by time.

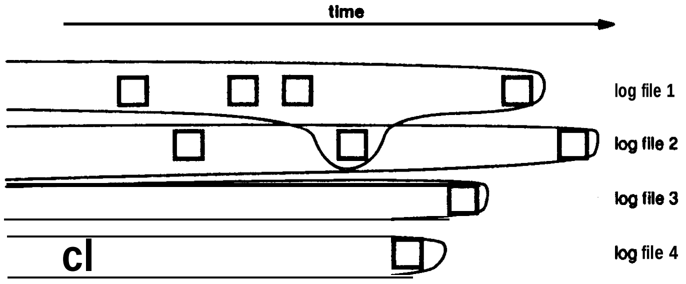
Volumes can be moved, and can fill up (and will then need to be replaced).

The sequence of successive replacements of a volume is called a *volume sequence.*

- This hides (from the clients) the presence of multiple volumes.
- It presents the abstraction of a potentially infinite sequence of log entries.
- Data can be written only to the latest volume in the sequence.

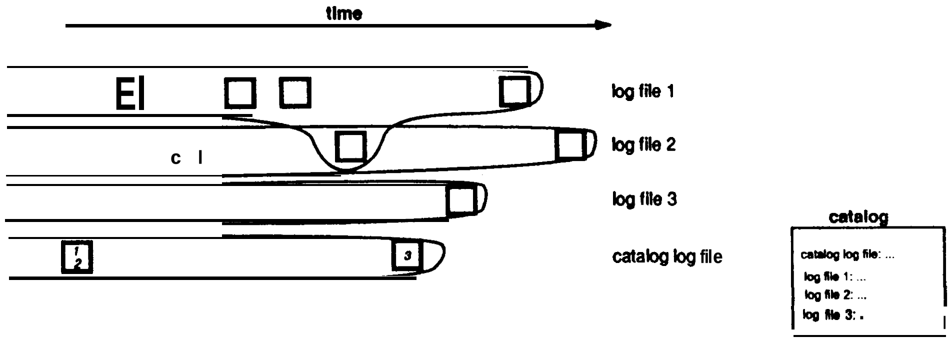
Our logging service provides access to *log files*:

- analogous to files in a regular file system, but are append-only
- *subsets* of the log entries in a volume sequence, predefined by clients
- can be read efficiently, without having to examine *every* log entry in the volume sequence
- subsets are usually (although not necessarily) disjoint.



Servers maintain a *catalog*, that describes each log file (e.g. its name and access permissions).

- Changes to the catalog are recorded in a special *catalog log file.*
- The catalog is completely defined by the catalog log file.
- A server initializes by first reading the catalog log file, to reconstruct the catalog.



Hardware trends:

- falling cost of RAM ⇒ large main memories become cost effective.

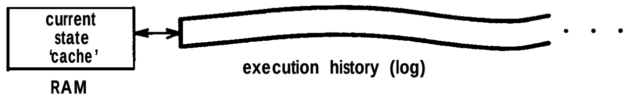
- high-density, low cost write-once storage (e.g. optical disk) ⇒ backing storage can be write-once.

⟹ The *history-based* model of system-structuring:

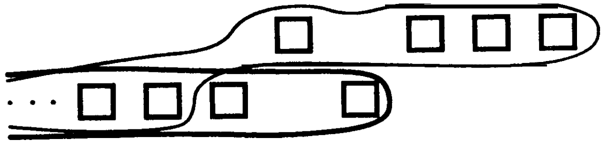
- Can build applications *on top of* a logging service.

- The system's *history* of execution is its true state.

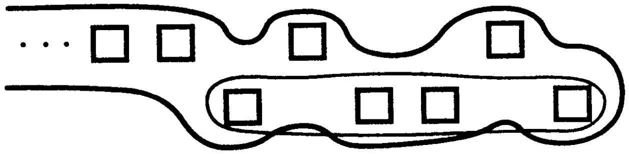
- The 'current state' is merely a cached version of the effect of this history.



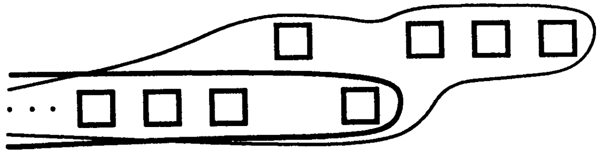
Other operations on log files:



a branch



a sublog



a superlog

Status:

Initial log server implementation:

- currently uses magnetic disk (but still assumes append-only storage)

- multiple disk partitions are used to simulate volumes (oldest volume gets overwritten)

- is integrated with the existing V-System file server

Conclusions:

- Log files provide a useful and efficient service, fitting naturally into the conventional file system abstraction.

- Log files can be implemented on write-once storage devices, allowing efficient read access and low space overhead.

- A logging service of this type can be used to build 'history-based' applications. Structuring applications in this way leads to a simplified implementation and recovery mechanism.

Internetwork Multicasting

Steve Deering

The extensive use of local networks is beginning to drive requirements for internetwork facilities that connect these local networks. In particular, the availability of multicast addressing in many local networks and its use by sophisticated distributed applications motivates providing multicast across internetworks.

In this talk we identify the properties of multicast communication that benefit distributed applications: efficient multiple-destination delivery, changing-destination delivery, and unknown-destination delivery. We propose a model of service for multicasting in a datagram internetwork that retains these properties and we show how it can be implemented as a generalization of existing unicast services. A protoype implementation within the ARPA Internet is described.

Collaborators

David Cheriton (Stanford)

Karen Lam (BBN)

DARPA End-to-End Protocols Task Force
(Bob **Braden,** chairman)

with help from Tim Mann, Jeff Mogul, and Joe **Pallas**

References

“Host Groups: A Multicast Extension for **Datagram** Internetworks”, Stanford CSD Report STAN-CS-851058
— motivation / definitions / architecture

RFC-966 — rewritten in terms of the ARPA-Internet

RFC-988 — detailed **spec** of IP modifications

Definitions

multicast	• to subset of receivers • **single** destination address
unicast	• to one receiver
broadcast	• to **all** receivers

datagram internetworks

- e.g. ARPA, Xerox, or **ISO** Internets
- unicast
- directed broadcast (Xerox)

Why multicast?

(1) efficient multi-destination delivery

- updating a replicated database
- conferencing
- parallel computing

(2) unknown-destination delivery

- querying a distributed database
- finding a name server
- disseminating routing tables

Why not broadcast?

- incurs overhead on uninterested hosts
- more overhead with each new application
- unwanted listeners
- too expensive for large internetworks
- <u>directed</u> broadcast constrained by topology

The Host Group Model

A **host group** is a set of zero or more hosts

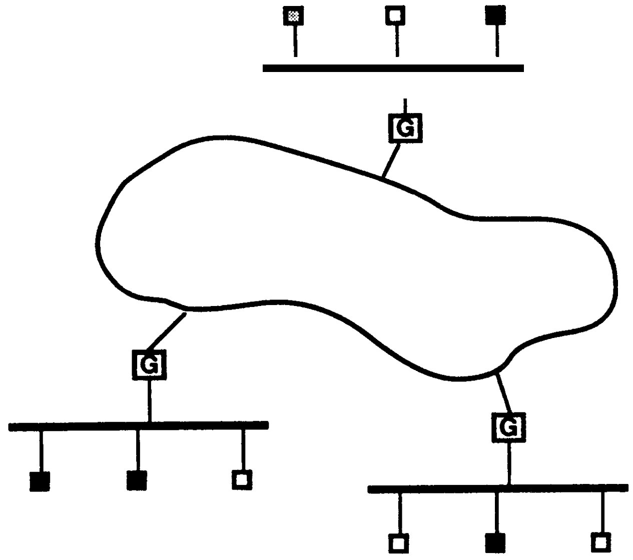
- identified by a single **IP** address
- members anywhere in the Internet
- dynamic and unbounded membership

Internet multicasting is the transmission of an **IP datagram** to a host group

- delivered to all current members
- **subject to** IP time-to-live constraint
- **and** unreliability of **datagram** delivery

Delivery Strategy

- sender transmits as a local multicast
- first gateway forwards to gateways on other member networks (the “network group”)
- remote gateways multicast to their own local members



Group Management Interface

CreateGroup() —> approval, group-adress

JoinGroup(group-address) —> approval

LeaveGroup(group-address) —> approval

Group Management Protocol

- request/response protocol between hosts and gateways
- includes periodic “keep-alive” messages to handle host and/or gateway crashes

Inter-Gateway Routing

- trivial if gateways share a broadcast channel (e.g. the Universe network)
- in general, route along shortest path tree
- gateway data structures required:
  - existing routing table
  - network membership table
    - infrequent updates
    - piggybacked on routing updates
    - loose consistency constraints
  - local host membership table
    - updated by create/join/leave requests from local hosts

Current Status

- prototype implementation for ARPA Internet
- using multicast aaents, outside of gateways
  - add extra hops to delivery path
  - no access to routing exchanges — must use static routing
- serves as **testbed** for investigating:
  - internetwork multicast routing
  - internetwork group management
  - applications of **internet** multicasting

Futures in Networking

new applications

- publishing (news, software updates)
- real-time conferencing
- distributed databases
- ...

new transport protocols

- multicast NETBLT
- VMTP
- ...

LISP on a Reduced-Instruction-Set Processor: Characterization and Optimization

Peter Steenkiste

As a result of advances in compiler technology, almost all programs are written in high-level languages, and the effectiveness of a computer architecture is determined by its suitability as a compiler target. This central role of compilers in the use of computers has led computer architects to study the implementation of high-level language programs. This thesis presents measurements for a set of Portable Standard Lisp programs that were executed on a reduced-instruction-set processor (MIPS-X), examining what instructions LISP uses at the assembly level, and how much time is spent on the most common primitive LISP operations. This information makes it possible to determine which operations are time critical and to evaluate how well architectural features address these operations.

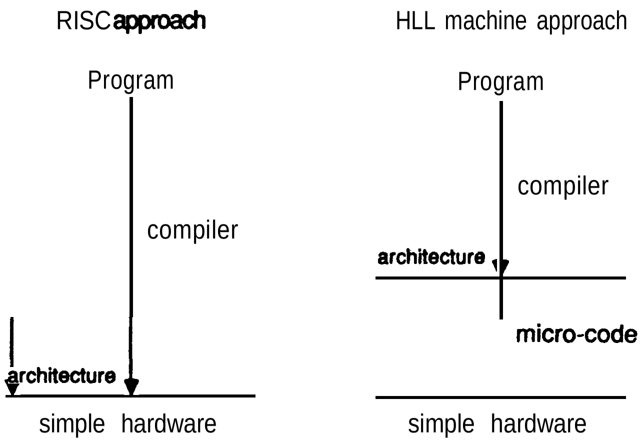
Based on these data, three areas for optimization are proposed: the implementation of the tags used for -run-time type checking, reducing the cost of procedure calls, and inter-procedural register allocation. A number of methods to implement tags, both with and without hardware support, are presented, and the performance of the different implementation strategies is compared. To reduce the cost of procedure calls, time critical LISP system functions were optimized and inlined, and procedure integration under control of the compiler was implemented. The effectiveness of these optimizations, and their effect on the miss rate in the MIPS-X on-chip instruction cache are studied. A simple register allocator uses interprocedural information to reduce the cost of saving and restoring registers across procedure calls. This register allocation scheme is evaluated, and its performance is compared with hardware register windows.

Overview

1. Global framework
2. Characterization of LISP
3. The implementation of tags for run-time type checking
4. Reducing the cost of procedure calls
5. Interprocedural register allocation

Software/Hardware Tradeoffs

Implementation of a high-level language

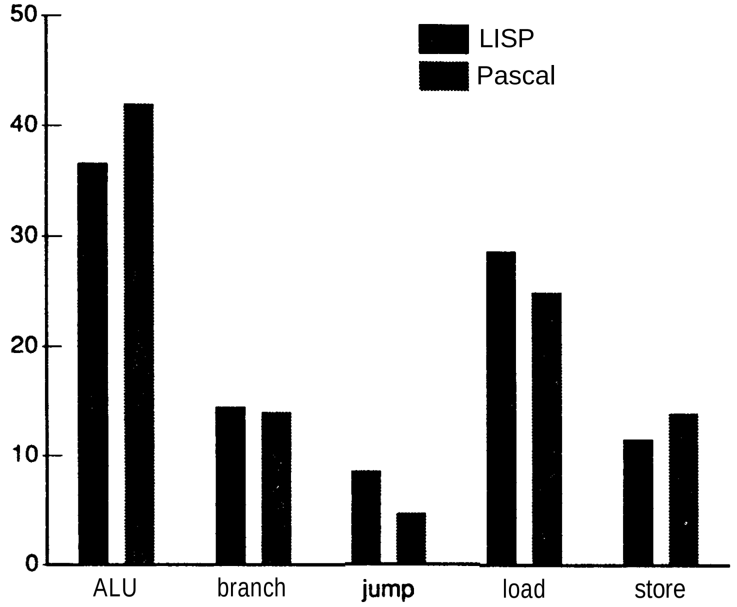


RISC approach has proven its effectiveness for Pascal, C, and FORTRAN.

What about LISP??

RISC/SPUR approach or a more traditional LISP machine

Assembly Instruction* Frequencies: LISP versus Pascal



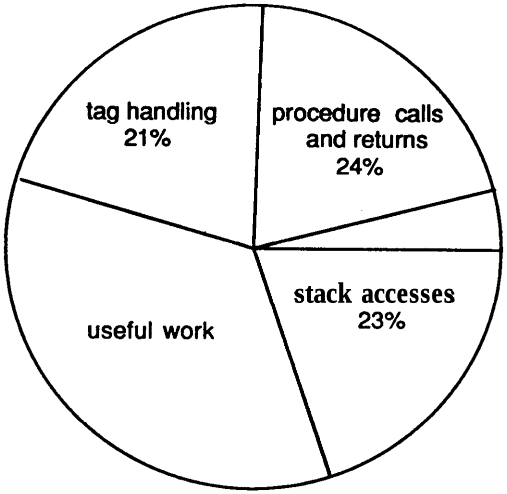
The instruction frequencies are very similar, but LISP programs execute more procedure calls (jumps).

This indicates that the performance issues for a LISP implementation on a general purpose processor are similar to the issues for other languages.

Delayed branches and software scheduling, as they are implemented in MIPS-X, are useful in reducing the cost of branches and memory accesses in LISP.

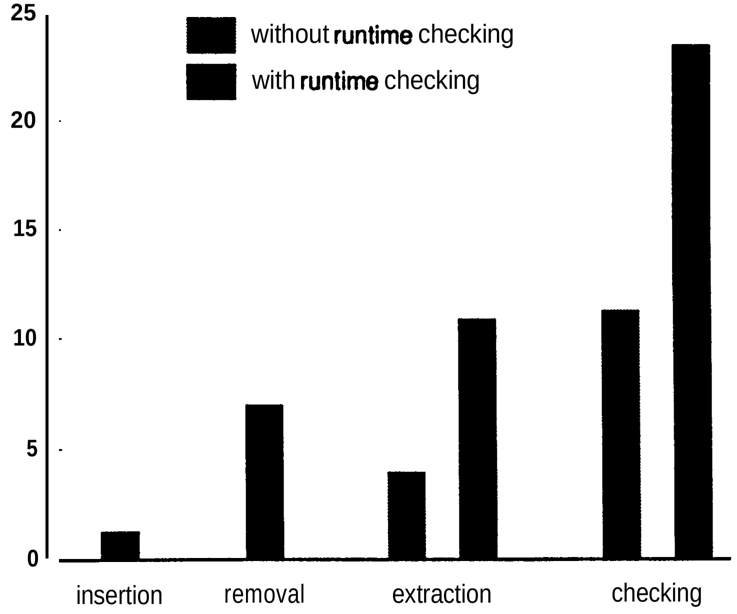
Source-Level Characterization

Source-level timing measurements show three time consuming operations:

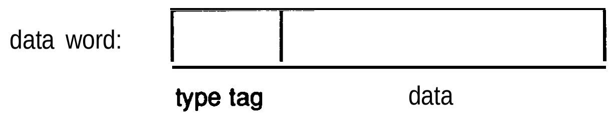


We look into optimizations in each of these three areas.

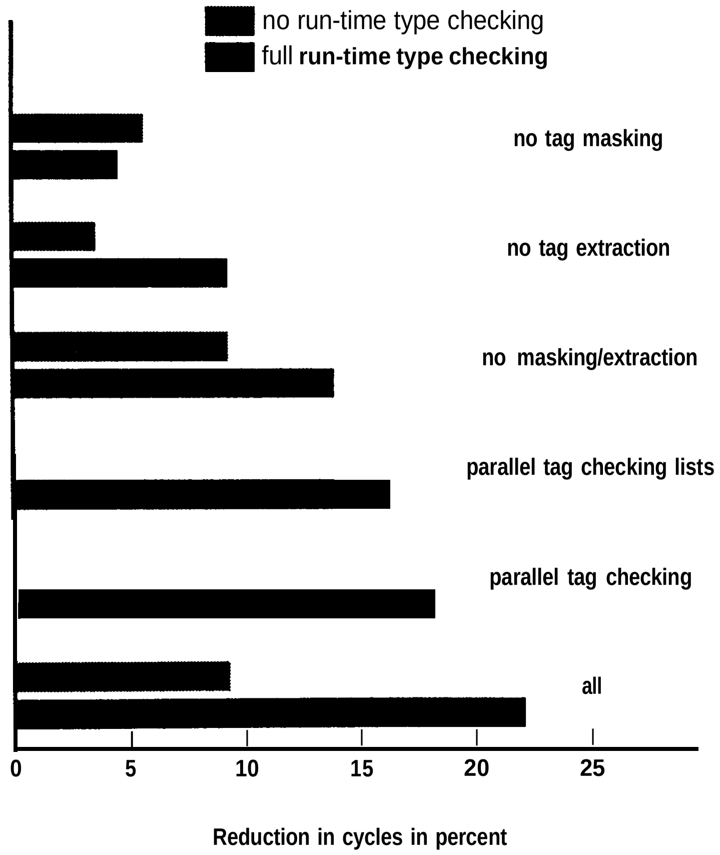
Cost of Tag Handling



A tag-information pair:



Speedup with Hardware Support for Tag Handling



Reducing the cost of Procedure Calls

Optimizing and **inlining** the most time-critical LISP system functions speeds up the LISP programs with on average 19%.

Procedure integration under control of the compiler gives an extra **speedup** of 6%.

For both optimizations, the **speedup** includes the effect of a ***decrease*** in the miss rate in the MIPS-X **on**-chip instruction cache.

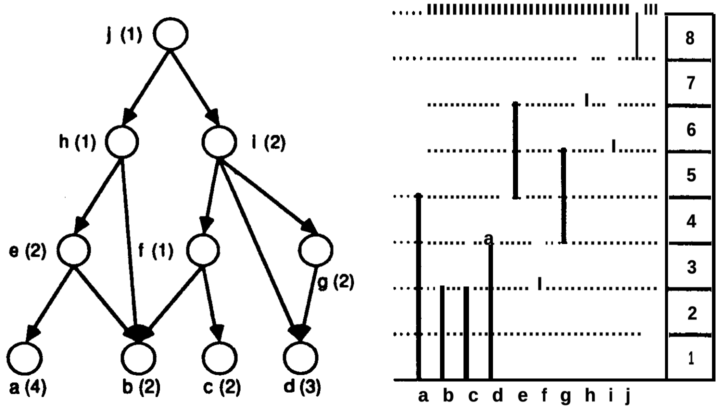
Reducing Stack Accesses

23% of all assembly instructions executed are stack accesses.

Because of the high procedure call frequency in LISP, per-procedure register allocation is ineffective.

Solution: Interprocedural Register Allocation

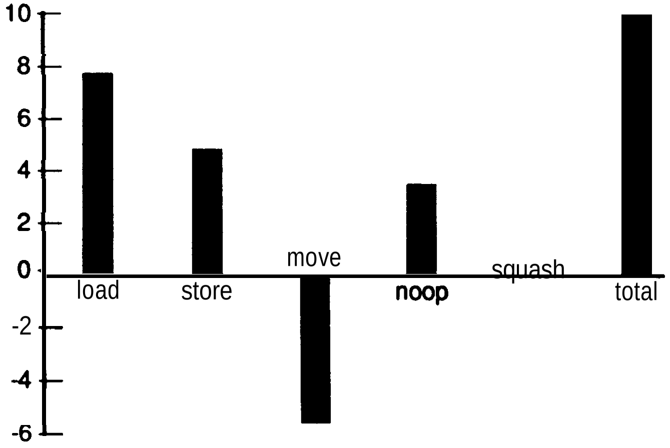
Information on the register usage of procedures is propagated in a bottom-up fashion in the program call **graph.**



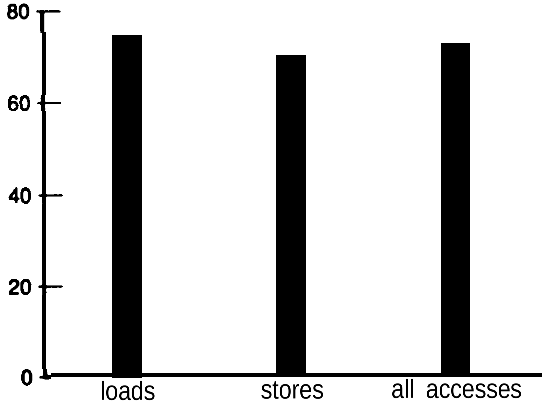
Problems: recursion, indirect procedure calls.

Interprocedural Register Allocation: Performance

Reduction in instruction frequencies:



Reduction in stack accesses:



Conclusion

- LISP executes more procedure calls than Pascal
- procedure calls, stack accessing, and tag handling are time consuming operations
- hardware support for tag handling can save between 9% and 22%
- optimization of LISP system functions and procedure integration give a **speedup** of 24%
- inter-procedural register allocation eliminates 70% of the stack accesses, with a **speedup** of 10%

Monitoring the Stanford Internetwork

Glenn Trewitt

Stanford University's networking environment consists of over 50 Ethernet segments interconnected by about 27 gateways. Over a period of three months, topological changes and traffic volumes in the network have been measured.

Changes in the topology of the network provide two different types of information. Short-term, temporary changes reflect disruptions due to gateway or network failures. Over the long term, changes show the growth of the network.

Traffic volume measurements around the network give a clear indication of network utilization and can indicate potential bottlenecks. They also provide some insight into the behavior of network users (both human and machine) over time.

The tools ‘used to make these measurements also provide timely reports of exceptional conditions in the network. Crashed or rebooted gateways or excessive error rates are automatically reported by electronic mail. This is probably the most immediate, visible benefit of these studies.

These capabilities demonstrate that remote monitoring is a practical and useful tool for viewing the behavior of the network.

Monitoring the Stanford Internetwork

Unique networking environment

- Large intemet — 60+ subnetworks
- Diverse applications — mail, telnet, NFS, . . .
- Homogenous network implementation

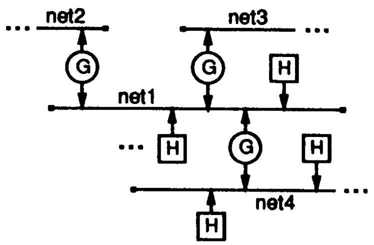
What can be learned?

- Evolution of the network
- Network utilization
- Reliability of network components

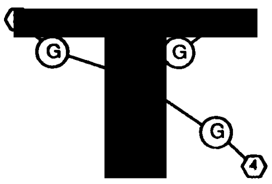
Network Topology

Collection of networks (mostly Ethernet)

Linked together by gateways (packet routers)



Actual Network Structure



Representation in Map

Each link in the map corresponds to one gateway network interface.

Topological changes represent

- Growth, over the long term
- Gateway or Network failures, over the short term

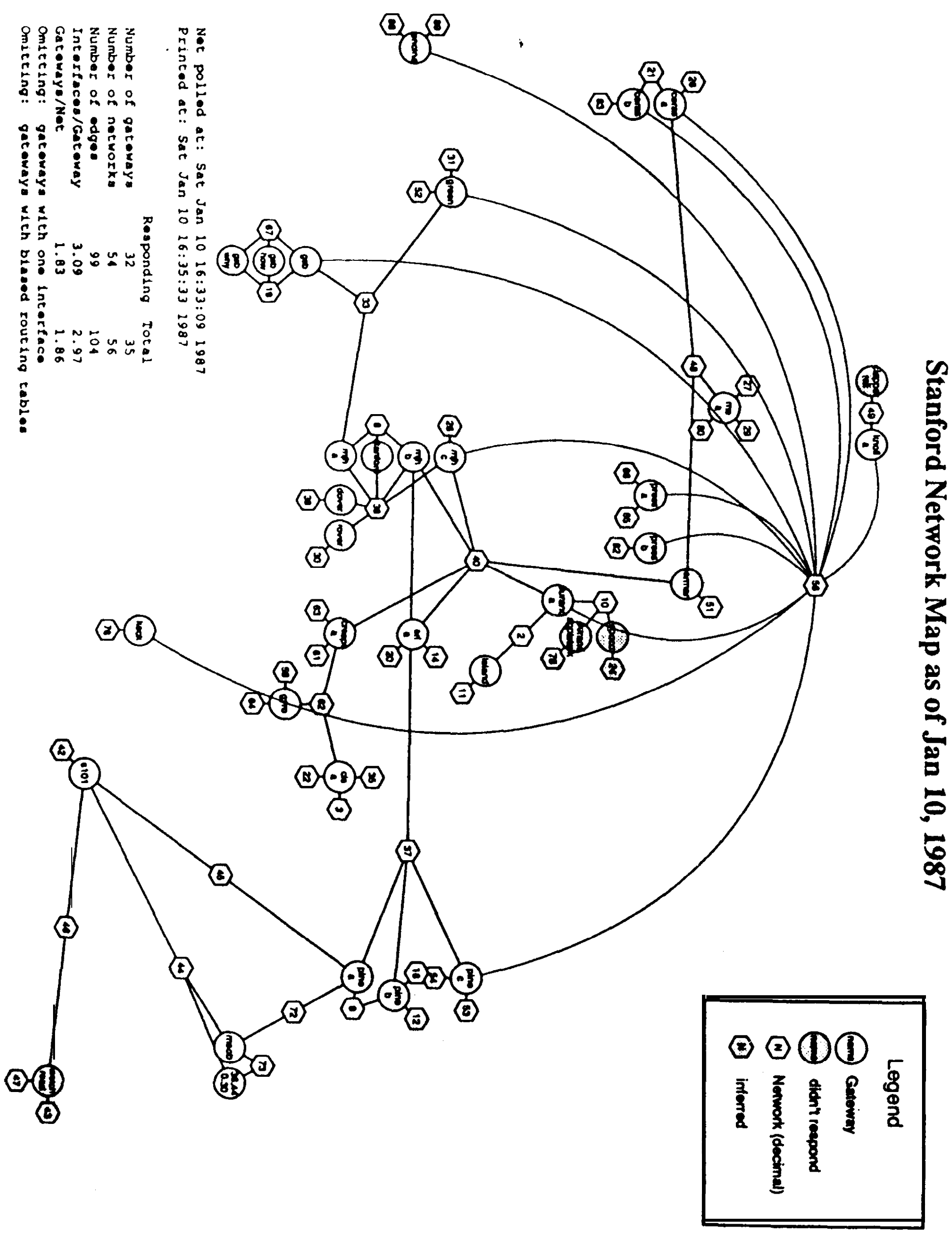
Network Growth

Between October 10, 1986 and January 10, 1987, the following changes were seen:

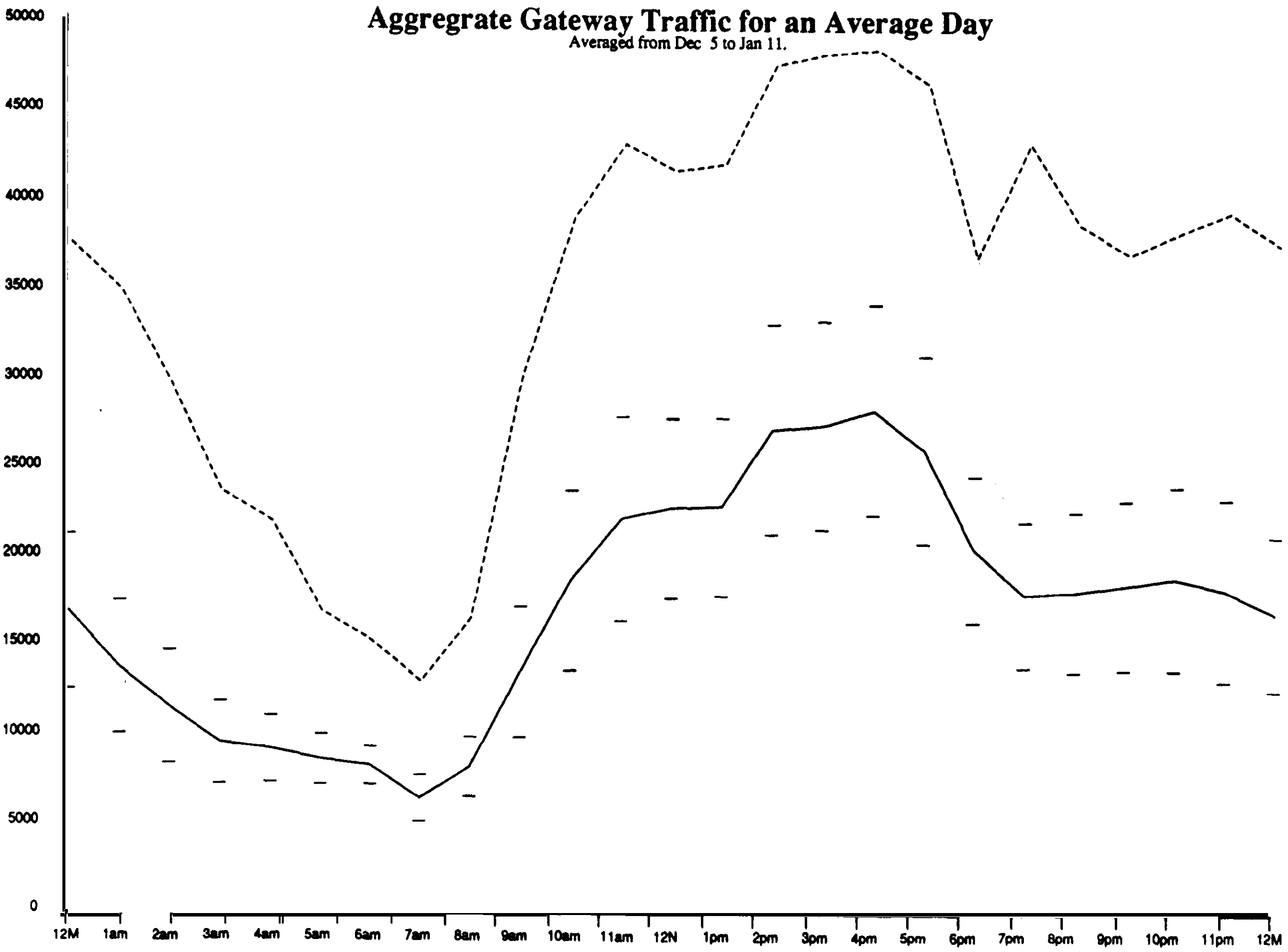
Date	Gateway	Net	Event
Oct 13	Sweet1*	83,87	+ gateway
Oct 13	Sweet2*	83,90	+ gateway
Oct 17	Durand-A	56	+ link
Oct 17	Durand-A	59	– network
Oct 29	GSB	56	+ link
Nov 20	Ceras-B	56	+ link
Oct 13	Sweet5*	83,70	+ gateway
Dec 18	Jordan-A	36,82	+ gateway
Jan 4	Jordan-A	36,82	– gateway

* AppleTalk/Ethernet gateway

Stanford Network Map as of Jan 10, 1987

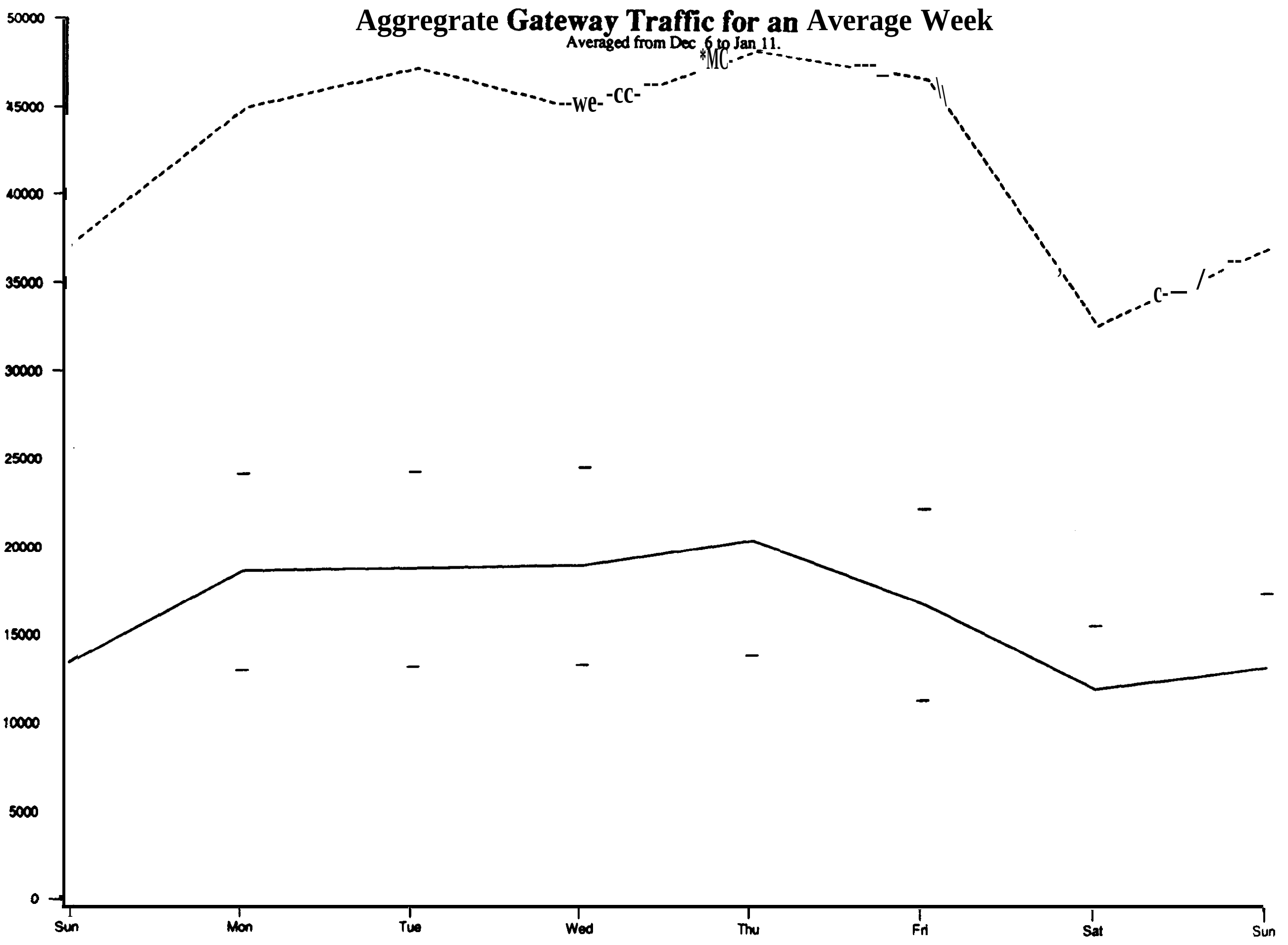


```
Net polled at: Sat Jan 10 16:33:09 1987
Printed at: Sat Jan 10 16:35:33 1987

                        Responding   Total
Number of gateways          32         35
Number of networks          54         56
Number of edges             99        104
Interfaces/Gateway        3.09       2.97
Gateways/Net              1.83       1.86
Omitting:  gateways with one interface
Omitting:  gateways with biased routing tables
```



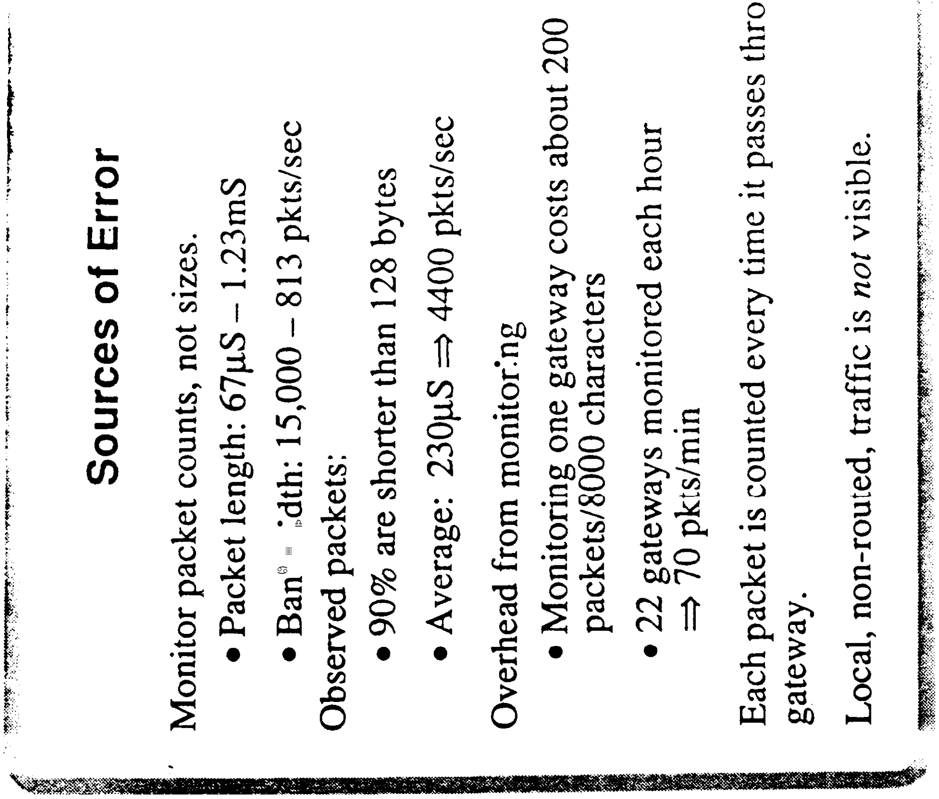
Aggregrate Gateway Traffic for an Average Day

Averaged from Dec 5 to Jan 11.



Aggregrate Gateway Traffic for an Average Week

Averaged from Dec 6 to Jan 11.



Sources of Error

Monitor packet counts, not sizes.

- Packet length: 67µS – 1.23mS
- Ban[illegible]dth: 15,000 – 813 pkts/sec

Observed packets:

- 90% are shorter than 128 bytes
- Average: 230µS ⇒ 4400 pkts/sec

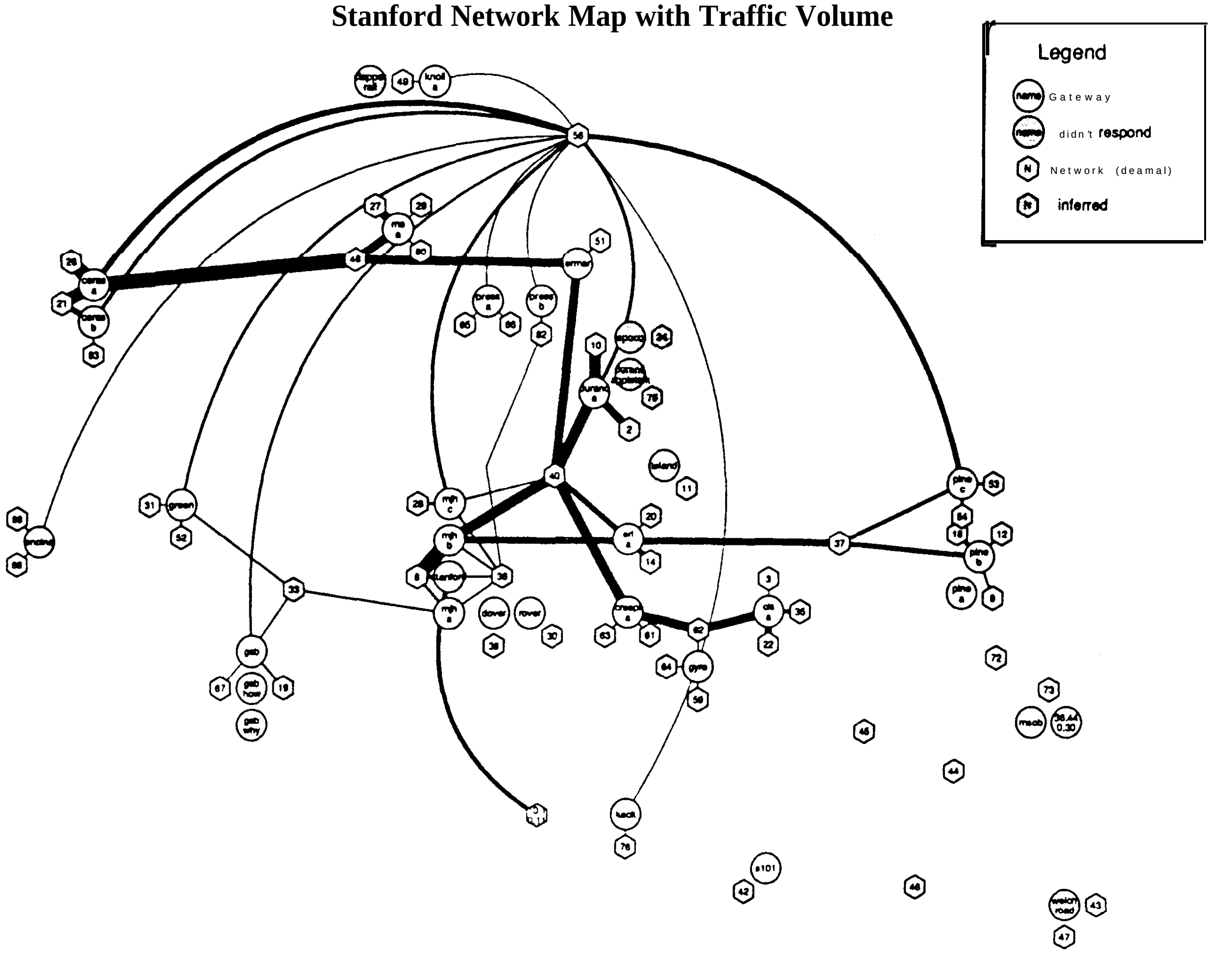
Overhead from monitor·ng

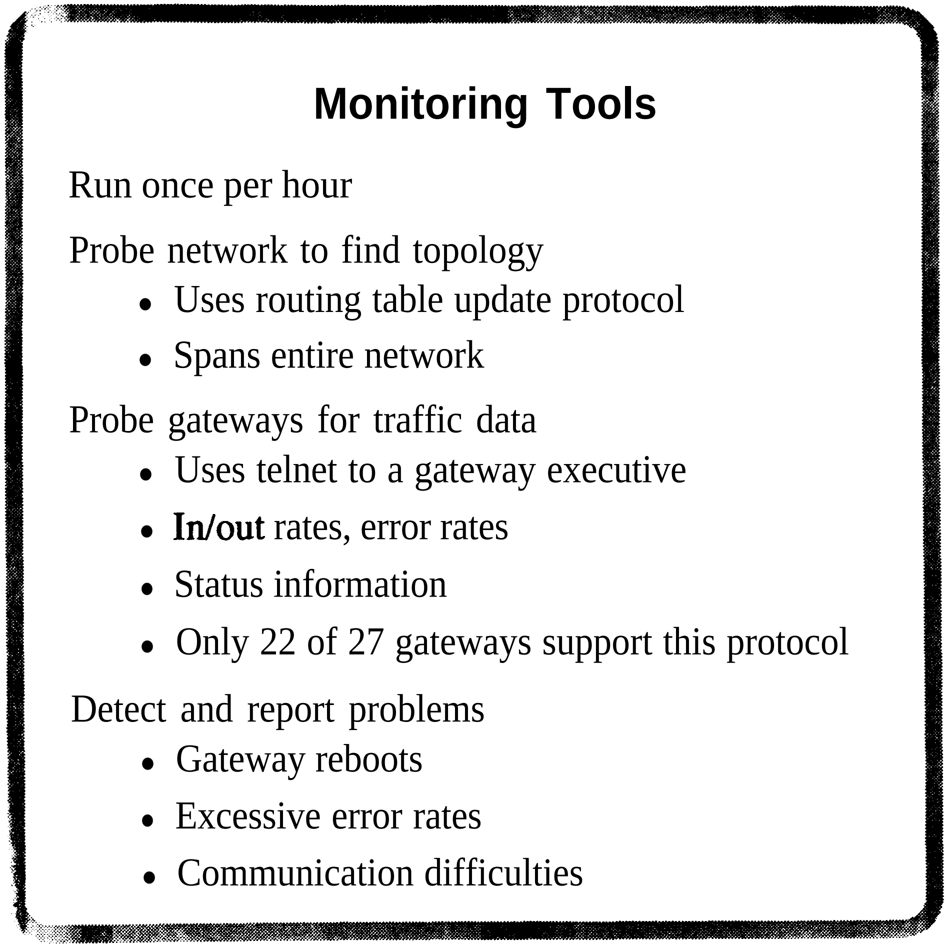
- Monitoring one gateway costs about 200 packets/8000 characters
- 22 gateways monitored each hour ⇒ 70 pkts/min

Each packet is counted every time it passes through a gateway.

Local, non-routed, traffic is *not* visible.

Stanford Network Map with Traffic Volume





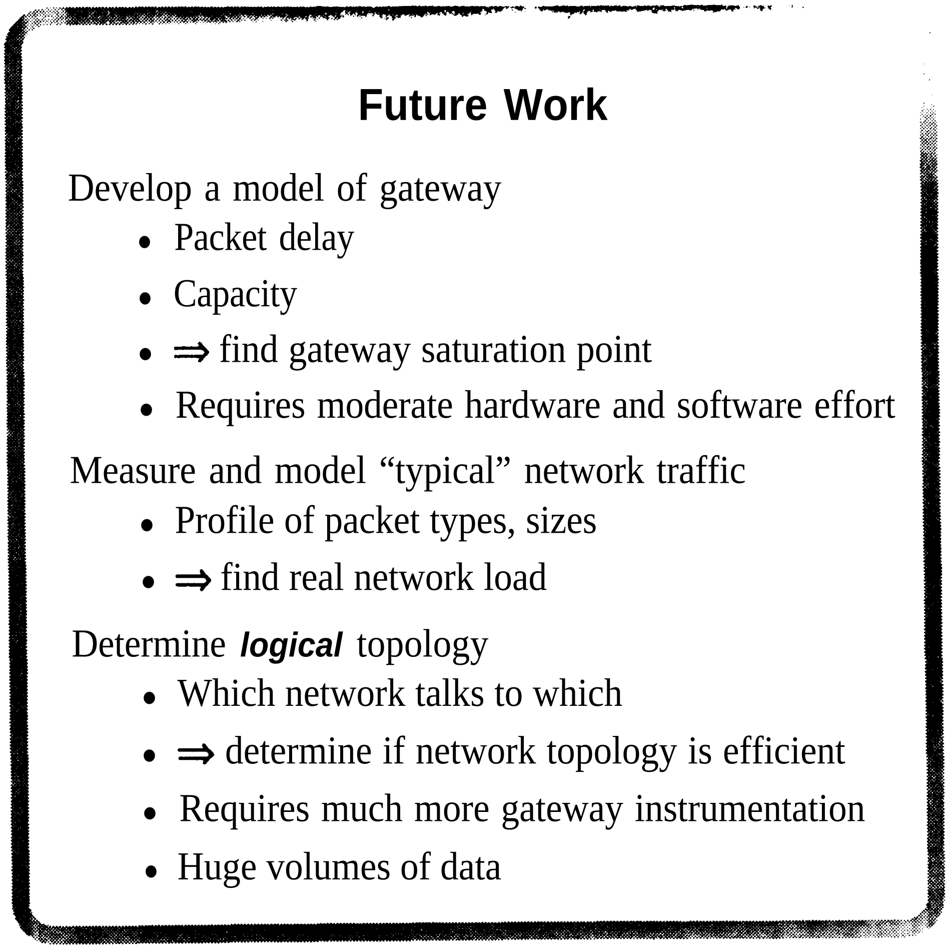
Monitoring Tools

Run once per hour

Probe network to find topology
- Uses routing table update protocol
- Spans entire network

Probe gateways for traffic data
- Uses telnet to a gateway executive
- **In/out** rates, error rates
- Status information
- Only 22 of 27 gateways support this protocol

Detect and report problems
- Gateway reboots
- Excessive error rates
- Communication difficulties



Future Work

Develop a model of gateway
- Packet delay
- Capacity
- ⇒ find gateway saturation point
- Requires moderate hardware and software effort

Measure and model “typical” network traffic
- Profile of packet types, sizes
- ⇒ find real network load

Determine ***logical*** topology
- Which network talks to which
- ⇒ determine if network topology is efficient
- Requires much more gateway instrumentation
- Huge volumes of data

Incremental Behavioral Simulation System

Tom Blank

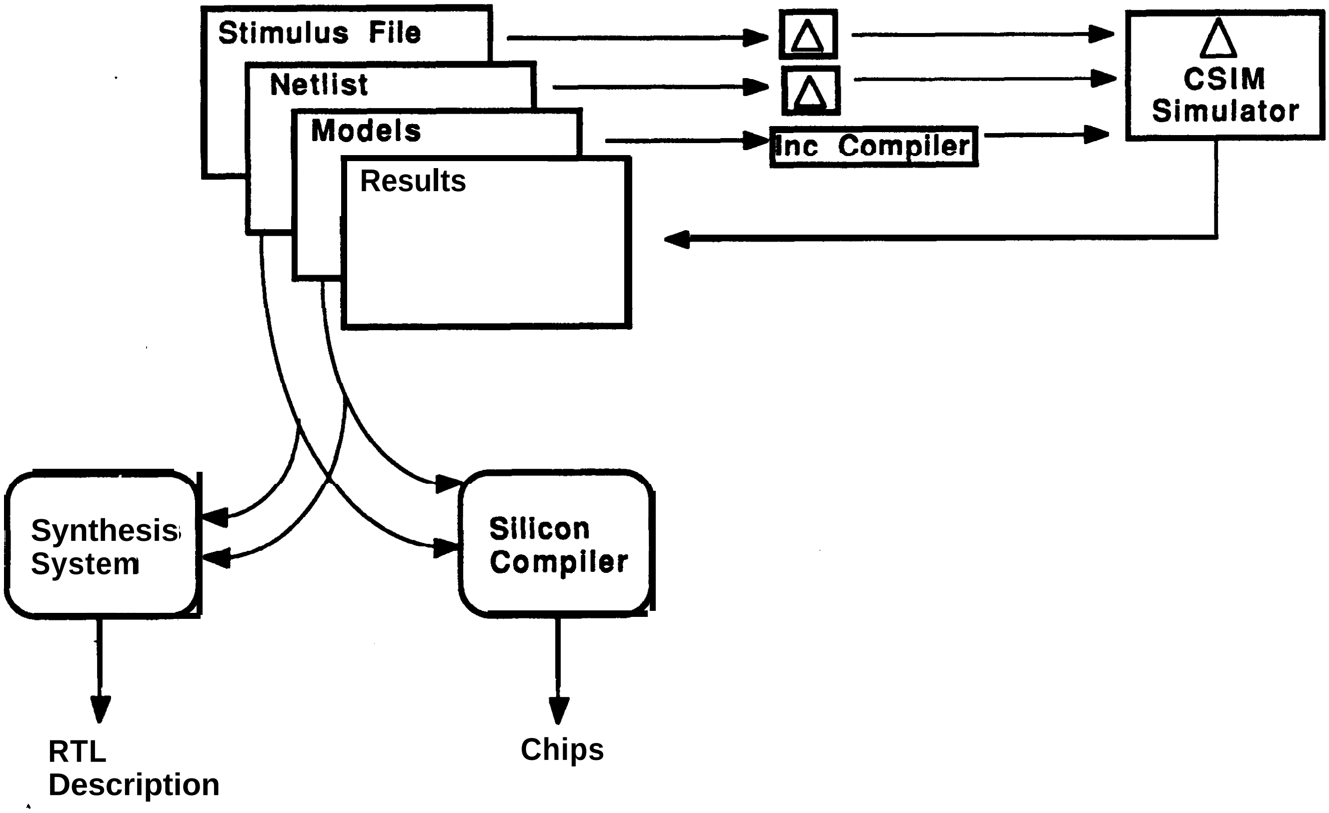
In any design process, there are two reoccuring tasks: design capture and validation. This design loop occurs at each abstraction level throughout the refinement process. A concrete electrical example is the schematic capture and simulation design loop. The key observation is that substantial "upfront" design time is spent cycling through different decisions and examining the implications. For example, simulation regression testing can require hours or even days of CPU time.

The research objective is to build an integrated collection of capture/simulate tools that minimizes the required time to go through the loop. The chosen strategy uses incremental design tools where an incremental tool only takes time proportional to the size of the change not the size of the entire design. The specific tools discussed are incremental netlist flattening, incremental simulation.

Incremental netlist flattening takes a hierarchical netlist representation and generates a flattened representation for the simulator. In incremental mode, it takes the previous flattened representation and changed schematic then alters the flattened representation to reflect the changes. The only work involved is that which is caused by the changes.

The incremental simulation idea is very different from any previous work. Simply stated, a simulation run is made collecting all the node values for all time; then a netlist change is made and the effects are propagated through the state table starting at time zero. The outputs of gates connected to a changed nodes are compared to the values stored in the history list. If the value is the same, then nothing is done; if the output is different, then a change to gate's output is scheduled, which may cause other logic gates to be evaluated.

The nice feature of this algorithm is its similarity with conventional event driven simulation; only a history list must be added to a basic simulator. One way to view this system is that the histories are a value cache that can help reduce the amount of computation required for each change. Unfortunately, keeping track of the information requires a sizable bookkeeping task. There are other methods of caching the value history that require less storage, but require more computation for each change.


Stimulus File
Netlist
Models
Results
Δ
Δ
Inc Compiler
Δ
CSIM
Simulator
Synthesis
System
Silicon
Compiler
RTL
Description
Chips

Incremental Technique

Concept: Keep past history then when designer makes a change, only do the work required to process the change.

If only one inverter is added to a design, only do one inverter's worth of work.

(Space vs. Time
Trade-off)

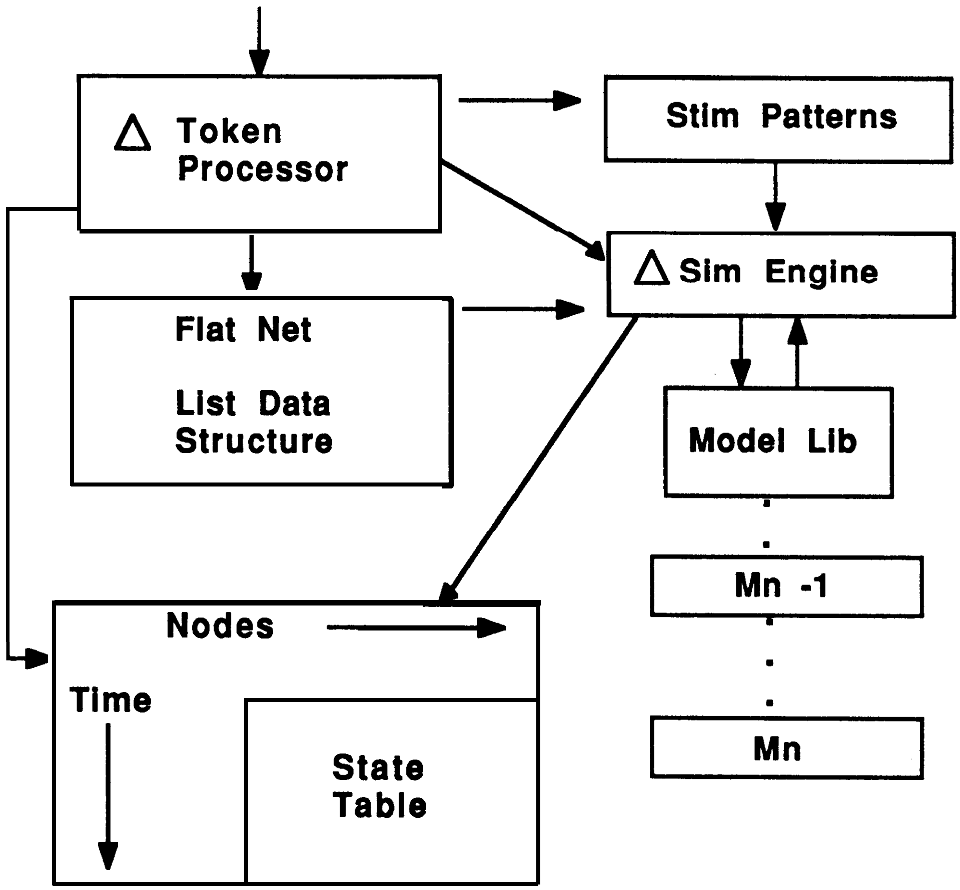
Inc Net List Compiler

Input: The new and old **netlists** for one schematic in a hierarchical design.

output: The changes required to process the A's on a flattened **netlist.**

(Analogy to Unix 'diff')

Incremental Simulator

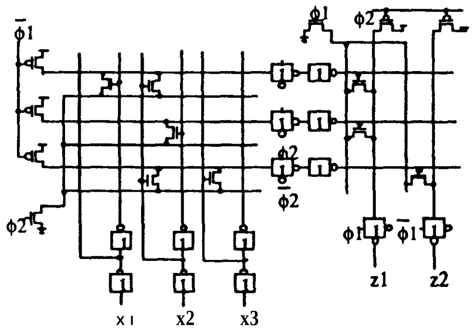


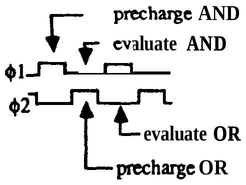
A CMOS PLA Design for Built-in Self-Test

Dick L. Liu

This talk describes a new scheme to design Built-In Self-Test (BIST) PLAs implemented with CMOS technology. These PLAs can perform function-independent self-test at normal operating speed, can detect CMOS switch-level faults, and have a lower area overhead than any other BIST schemes. A sequential parity checking technique is used to test for the AND and OR arrays of the PLA. This technique does not require any XOR cascade to evaluate parity data as the parallel checking technique used by other scheme, thus achieving an order of magnitude reduction in total testing time. As the new scheme is aimed at CMOS PLAs (other schemes are for NMOS PLAs), it accounts for switch-level stuck-open and stuck-on faults in addition to conventional stuck-at, crosspoint and bridging faults. A novel circuit design technique was used to implement the largest piece of the additional test circuitry: the test pattern generator for product lines. It makes use of a Johnson counter and a two-level decoding network to obtain a very low area overhead and to match the pitch between the PLA and the test circuitry. With these enhancements, this is an attractive BIST scheme to implement large PLAs embedded in CMOS VLSI chips.

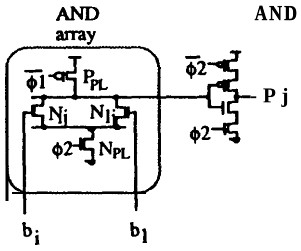
II. CMOS PLA and Switch-level Faults





Switch-level Faults

AND array consists of **precharged** NOR gates



- stuck-open fault

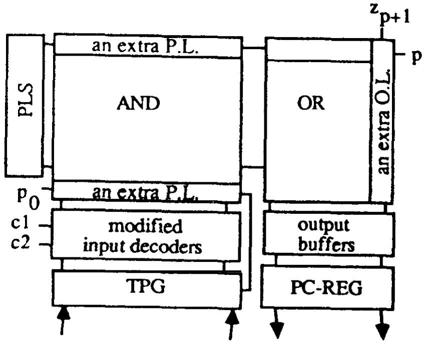
Nij' - missing crosspoint
Npl - product line **s/1**
Ppl - iwo-pattern test
{(0,...1,...0), (0,...0)}
(initializing product line to 0)

- stuck-on fault

	(i) P dominant (Gp>>Gn)	(ii) N dominant (Gn>>Gp)
Nij	product line s/O	product line **s/0**
Npl	**undetectable,but** prechmrge OK.	undetectable
Ppl	product line **s/1**	undetectable

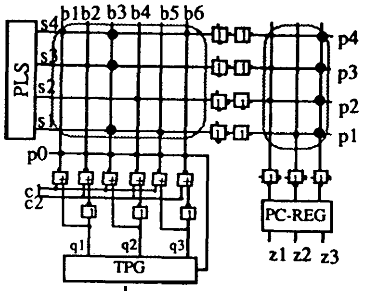
design to be P dominant

III. Augmented PLA for BIST



Additional circuitry:
- TPG:
  test pattern generator
- modified input
  decoders
- PLS:
  product line selector
- extra P.L.:
  extra product line
- extra O.L.:
  extra output line
- PC-REG:
  parity checking register

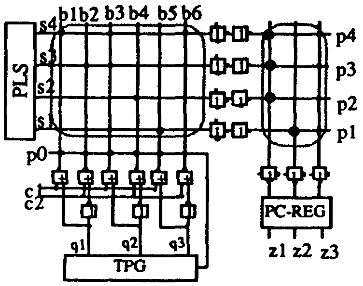
Test AND array



A-test for bit line (b3)

q1	q2	q3	c1	c2	b1	b2	b3	b4	b5	b6	S1	S2	S3	S4	p1	p2	p3	p4	z1	z2	z3
1	0	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	x	x	0
1	0	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0	0	x	x	0
1	0	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	0	x	x	0
1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0	0	0	0	x	x	0

Test OR-array

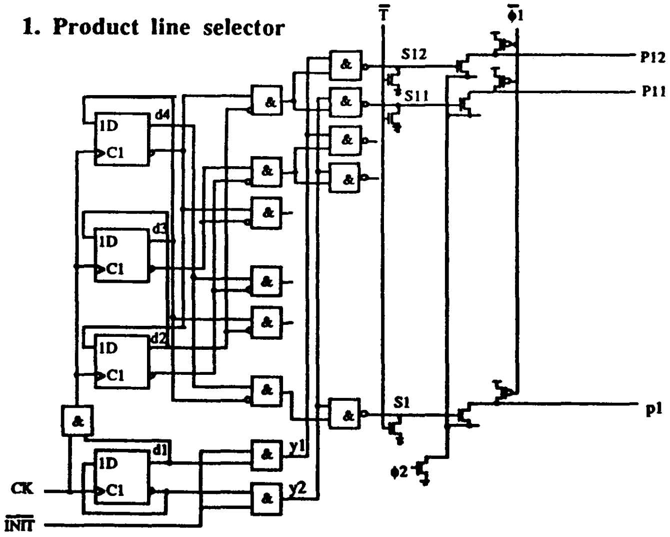


O-test for output lines

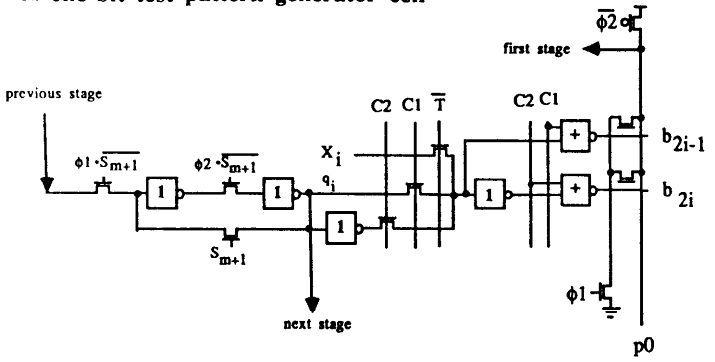
q1	q2	q3	c1	c2	b	b2	b3	b4	b5	b6	S1	S2	S3	S4	p1	p2	p3	p4	z1	z2	z3
1	1	1	0	1		0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	x
1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	·	0	0	0	0	1	x
1	1	1	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0	x
1	1	1	0	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0	1	0	x
1	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	x

IV. CMOS Implementation of Test Circuitry

1. Product line selector

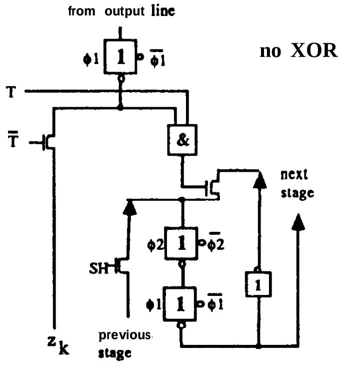


A one-bit test pattern generator cell



no extra delay is added to the input line

3. Parity-checking register (one-bit cell)



no XOR gate is used

Comparison of BIST schemes for a large PLA.

n=31, m=125, p=39

	number of test patterns	test application time	number of additional flip-flops	number of additional gates
Hassan	2 x 10^9	1x10^{10}	132	202
Daehn	227	1135	226	907
Yajima	289	74,000	320	337
Treuer	8001	664,000	95	234
New	8001	40,000	102	156

Temporary Failures in Digital Circuits: Experimental Results and Fault Modeling

Mario L. Cortes

Two types of temporary failures can be identified: transient and intermittent failures. This experimental work studies transient failures due to power supply disturbances and intermittent failures due to parameter degradation.

Failures caused by power supply disturbances can be modeled as delay faults. Noise immunity problems play only a small role in the occurrence of failures. This conclusion results from experiments where voltage sags are injected in the power supply rails of gate arrays and breadboard circuits. The susceptibility of the circuits to the occurrence of errors increases with the clock frequency. This dependency is due to increase in propagation delay with lower power supply voltages. Errors are caused by violation of timing constraints in the circuits. It was also found that supply disturbances can cause metastability.

Intermittent Failures are studied by stressing (temperature, supply voltage and extra loading) good parts. The behavior of the chips under stress is similar to that of a marginal chip under normal operating conditions. The experiments show that most intermittent failures are pattern-sensitive for both sequential and combinational circuits. The stuck-at fault model is shown to be inappropriate to describe intermittent failures. A case is presented shere a single intermittent failure is not detected by a test set with 100% single stuck-at fault coverage.

TEMPORARY FAILURES

TRANSIENT FAILURES (Non Recurrent)

- Power Supply Disturbances
- Electromagnetic Interference
- Alpha Particles

INTERMITTENT FAILURES (Recurrent)

- Marginal Behavior (Parameter Degradation
- Timing
- Metal-Related (Open/Short)
- Poor Design

INTERMITTENT FAILURES

INDUCING INTERMITTENT FAILURES IN GOOD CHIPS

- Good Chips Under Stress
- Marginal Chips Under Normal Condition

EXPERIMENTS

- Sequential Circuits: Temperature And Voltage Stress
- Combinational Circuits: Extra Loading

PATTERN SENSITIVITY

IMPLICATIONS ON FAULT MODELING

- Stuck-at Fault Model not Appropriate

I.F. IN MARGINAL CHIPS (PARAMETER DEGRADATION)

- Leakage Paths (Partial Short)
- Increased Resistance (Open)
- Output Voltage Out-of-specification
- Timing
- Parameter Variation

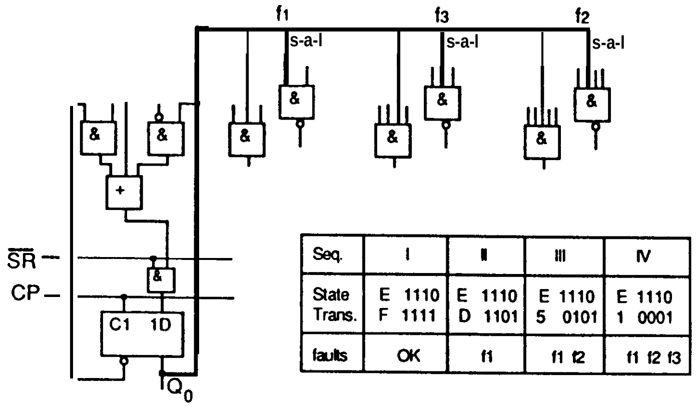
I.F. UNDER STRESS

- STRESSES: Temperature, Voltage and Loading
- **TYPE** OF FAILURE: Noise-Immunity Degradation

OUTPUT SEQUENCES vs Vcc RANGE
(vendor A, 119 °C)

	Sequence	**Vcc Range (V)**
I:	····**0123456789ABCDEF01**····	3.50 - 5.48
II:	····**9ABCD** ED E...ED **EF0**····	5.48 - 5.57
III:	' ····**89ABCD E5 6789**····	5.57 - 5.70
IV:	····**89ABCD** EI **23456789**····	5.70 - 6.50

STUCK-AT FAULT TEST SETS



Seq.	I	II	III	IV
State Trans.	E 1110 F 1111	E 1110 D 1101	E 1110 5 0101	E 1110 1 0001
faults	OK	f1	f1 f2	f1 f2 f3

ERROR PATTERN FOR VENDOR D: ...**0123456789AB CF 01...**

STUCK-AT TEST SET
- Complete Test Set for the Combinational Part
- 100 % Single Stuck-at Fault Coverage
- Test Does Not Detect Incorrect Transition: C to F
- Pattern Sensitivity

EXHAUSTIVE TEST
- No Fault Model; 100% Coverage

PSEUDO-EXHAUSTIVE TEST
- Contains State C
- Detects the **Intermittent** Failure
- In General: 100% Coverage if no Coupling Between Segments

SUMMARY AND CONCLUSIONS

GOOD PARTS UNDER STRESS
MARGINAL PARTS UNDER NORMAL CONDITIONS

STRESSES:
- Temperature
- Power supply voltage
- Extra loading

PA-I-TERN**SENSITIVITIY**
- Experimental results: LSTTL and CMOS
- Other technologies

FAULT MODELS
- Stuck-at
- Exhaustive and Pseudo-Exhaustive Testing

TECHNIQUE FOR INJECTING FAILURES

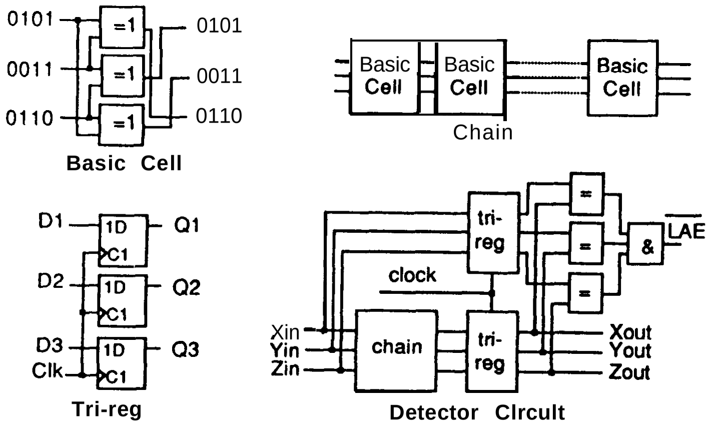
TRANSIENT ERRORS DUE TO POWER SUPPLY DISTURBANCES

EXPERIMENTS:
- CMOS Gate Array
- CMOS and **LSTTL** Breadboards **(74HCXX,** 74LSXX)

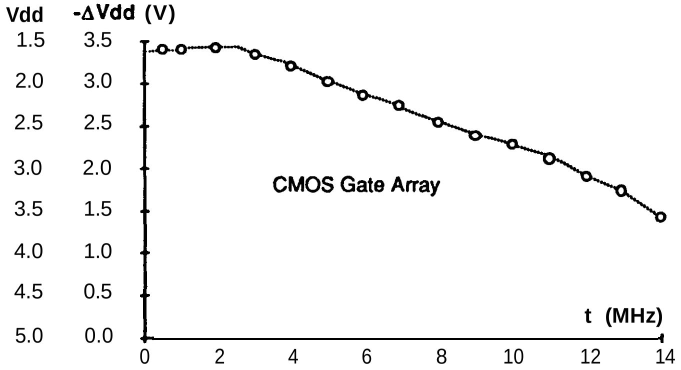
DELAY EFFECTS: DOMINANT FACTOR
- Gate Array: Experiments and Logic Simulation
- Breadboard: Direct Observation

SUPPLY DISTURBANCES AND **METASTABILITY**

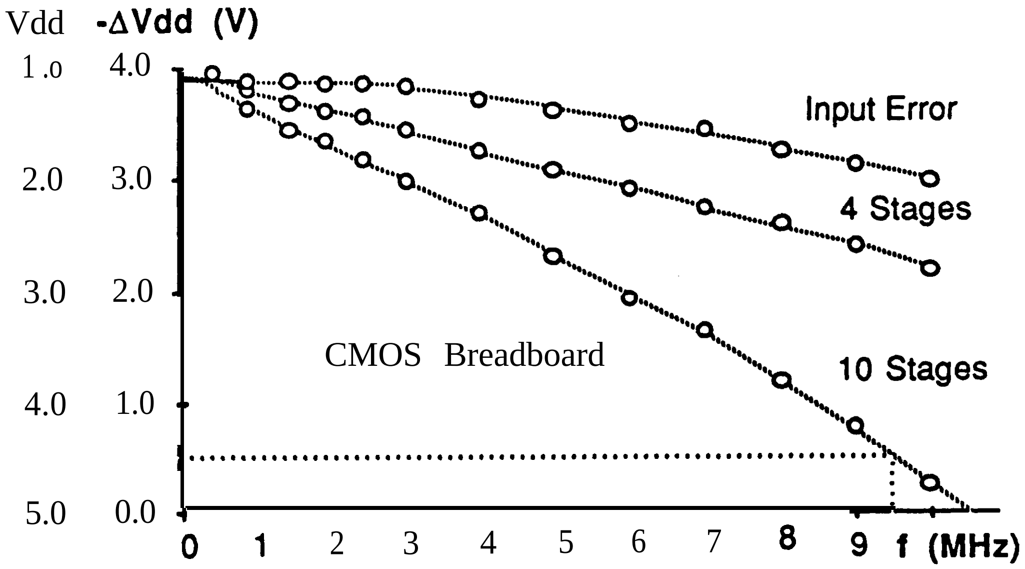
CIRCUIT UNDER TEST (CUT)



TOLERANCE TO SUPPLY DISTURBANCES vs FREQUENCY



TOLERANCE TO SUPPLY DISTURBANCES VS FREQUENCY



TRANSIENT ERRORS DUE TO POWER SUPPLY DISTURBANCES

SUMMARY AND CONCLUSIONS

TOLERANCE TO DISTURBANCES vs CLOCK FREQUENCY

DELAY FAULTS

TIGHT TIMING ⇒ POOR TOLERANCE TO DISTURBANCES

SUPPLY DISTURBANCES ⇒ METASTABILITY

OTHER FAMILIES: E-D NMOS, ALSTTL [Wagner 85]

An Analysis of Heuristic Methods for Reasoning with Uncertainty

David Heckerman

Over the last two years, I have been working to better characterize heuristic methods for reasoning under uncertainty such as the MYCIN certainty factor (CF) model, the PROSPECTOR scoring scheme, and the INTERNIST-l scoring scheme. Developing a deeper understanding of these approaches will allow new systems to be developed that incorporate the strengths of these approaches but avoid many of their weaknesses.

In this talk, I will focus on the CF model, a method for managing uncertainty that has seen widespread use in rule-based expert systems. MYCIN's knowledge is stored as an interconnected network of rules of the form "IF E THEN H" where E is a piece of evidence for hypothesis H. A certainty factor is attached to each rule representing the *change* in belief of H given E. Uncertainty is propagated through the network of rules by applying a set of *combination functions* to the CF's attached to the rules. The developers of the model justified the combination functions by showing that they satisfied a set of intuitive properties such as commutativity and associativity.

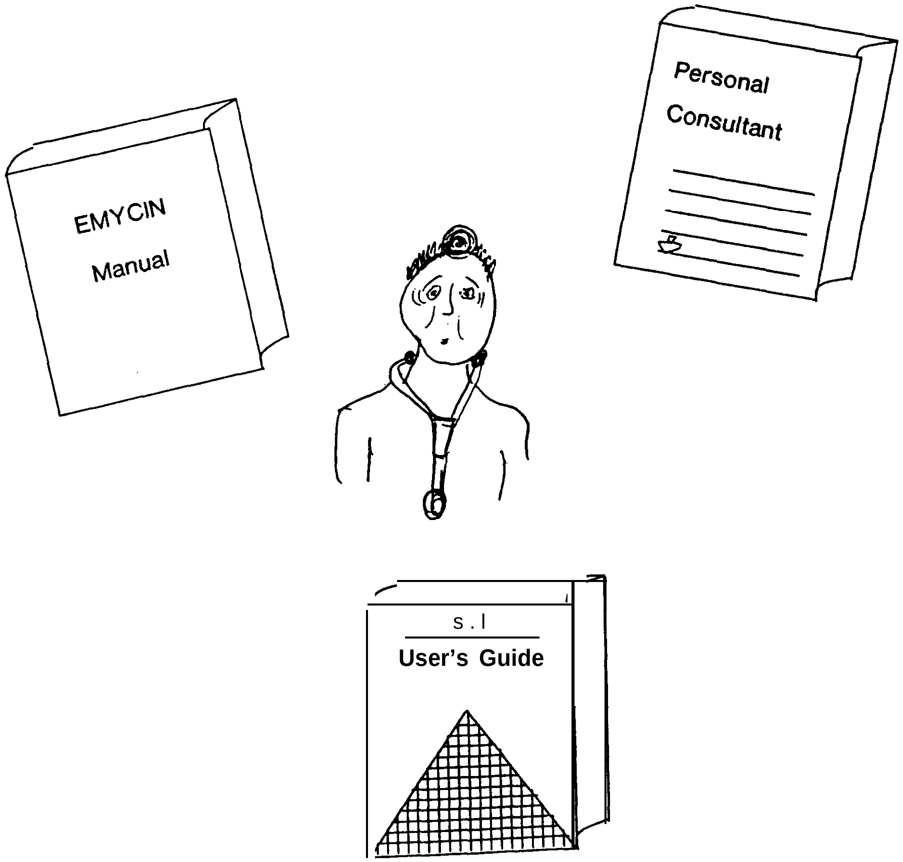
.One of the components of the CF model is an interpretation of certainty factors in terms of probabilistic quantities. Such an interpretation is important as it can be used to translate expert knowledge into CF's and to communicate system recommendations and explanations to users. In analyzing the model, I have shown that the probabilistic interpretation ascribed to certainty factors is inconsistent with the combination functions used to manipulate them. Such an inconsistency is undesirable because it can lead to the distortion of knowledge as it is passed through the system from expert to user. Fortunately, however, a minor reformulation of the original probabilistic interpretation can be shown to be consistent with the intuitive properties used to justify the combination functions. In this reformulation, the certainty factor is recognized as a monotonic transformation of a probabilistic quantity called the likelihood ratio.

Using the reformulation, implicit assumptions in the CF model can be better characterized. For example, one assumption of the model is that the certainty factor attached to a rule does not depend on the truth status of other variables in the knowledge base. This assumption was made so that rules would retain a *modular* character. Through the reformulation, this assumption can be shown to be equivalent to strong assumptions of conditional independence.

The imposition of these assumptions is a limitation of the CF model. If an assumption is invalid, a system implementor has no choice but to make the assumption and hope that the system performs in a satisfactory manner. Recognizing this limitation, however, suggests an improvement to the CF model. In particular, it would be useful to give the system implementor the ability to explicitly represent assertions of probabilistic independence made by the expert and to give the inference mechanism the ability to take advantage of these assertions when they are made. Such a methodology has been developed and will be discussed.

Benefits of analysis

- . Prediction of performance
- . Improvement of performance



A MYCIN rule

IF <evidence> THEN <hypothesis>, CF

Example:

IF the organism is gram-positive
and grows in clumps

THEN (the increase in belief that)
the organism is Staphylococcus

is .7.

Notation:

$E \xrightarrow{CF(H,E)} H$

a ttempts to formalize the model

Probabilistic interpretation:

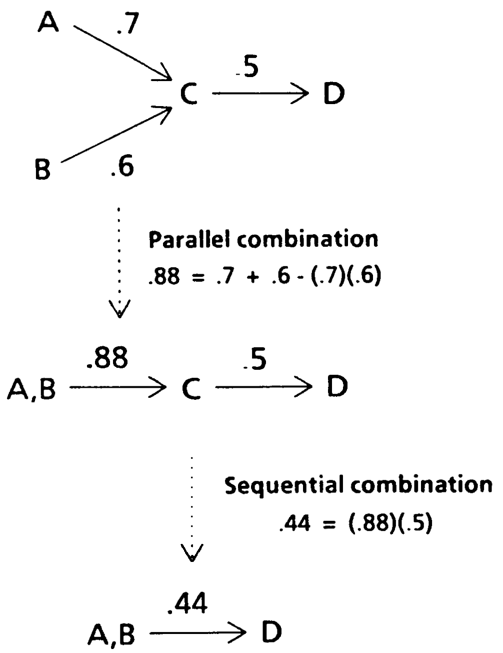
$$CF(H,E) = \begin{cases} \dfrac{p(H|E) - p(H)}{1 - p(H)} & p(H|E) > p(H) \\ \\ \dfrac{p(H|E) - p(H)}{p(H)} & p(H) > p(H|E) \end{cases}$$

Properties:

Commutativi y: $CF(H, E_1E_2) = CF(H, E_2E_1)$

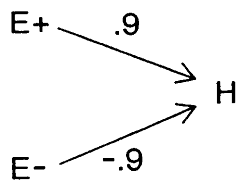
Associativity: $CF(H, (E_1E_2)E_3) = CF(H, E_1(E_2E_3)^{\supset}$

The propagation of uncertainty in an inference network

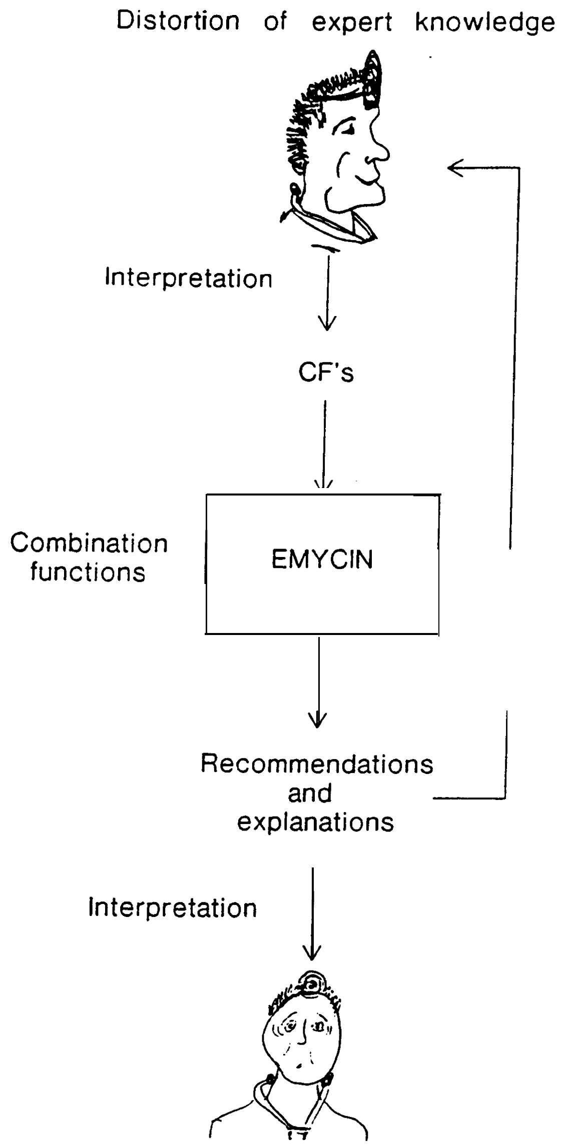


Original Interpretation ==><== Properties

Consider



```
Let p(H) = .5

First, update belief in H given evidence ≤+:
   .9 = (p(H|E+) - p(H)) / (1 - p(H))
   p(H) = .5  ==>  p(H|E+) = .95

Now update with evidence E-:
   -.9 = (p(H|E+E-) - p(H|E+)) / p(H|E+)
   p(H|E+) = .95  ==>  p(H|E+E·) = .10

Updating in the reverse order:
-.9 = (p(H|E-) - .5) / .5              ==>    p(H|E-) = .05
 .9 = (p(H|E-E+) - .05) / (1 - .05)    ==>    p(H|E-E+) = .90

Therefore p(H|E+E-) <> p(H|E E+)
```


Distortion of expert knowledge
Interpretation
CF's
Combination functions
EMYCIN
Recommendations and explanations
Interpretation

Resolving the inconsistency

Relax the probabilistic interpretation to

CF(H,E) = f(p(H|E), p(H))

Look for quantities of the above form satisfying the properties of the combination functions.

The likelihood ratio

$O(H|E) = \lambda(H,E)\ O(H)$

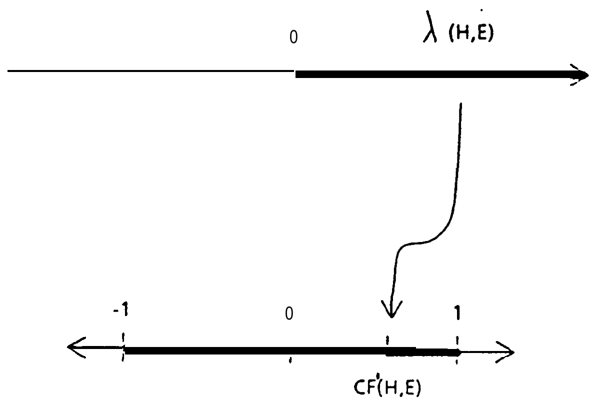
where

$O(H) = p(H) / (1 - p(H))$

$O(H|E) = p(H|E) / (1 - p(H|E))$

$\lambda(H,E) = p(E|H) / p(E|\sim H)$

A mapping of the likelihood ratio



$$CF'(H,E) = \begin{cases} \lambda(H,E) - 1/\lambda(H,E) & \lambda \geq 1 \\ \lambda(H,E) - 1 & \lambda < 1 \end{cases}$$

$$= \begin{cases} \dfrac{p(H|E) - p(H)}{p(H|E)(1 - p(H))} & p(H|E) > p(H) \\[2ex] \dfrac{p(H|E) - p(H)}{p(H)(1 - p(H|E))} & P(H) > p(H|E) \end{cases}$$

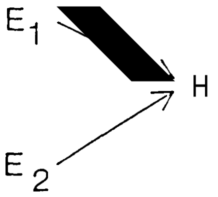
The combination function for this quantity is exactly the **MYCIN** parallel combination function.

Limitations of the CF model

Modularity assumption:

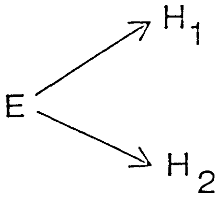
The CF attached to a rule is independent of beliefs associated with variables elsewhere in the network.

Implications of the modularity assumption



$p(E_1 | H, E_2) = p(E_1 | H)$

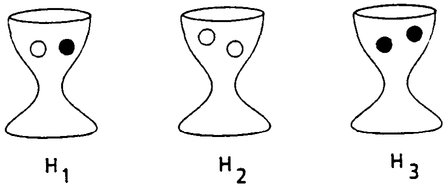
$p(E_1 | {\sim}H, E_2) = p(E_1 | {\sim}H)$



IMPOSSIBLE

Restriction on parallel combination imposed by modularity

Suppose there are 3 urns.



Before any balls have been drawn:

$$\text{Black ball} \xrightarrow{CF = 0} H_1$$

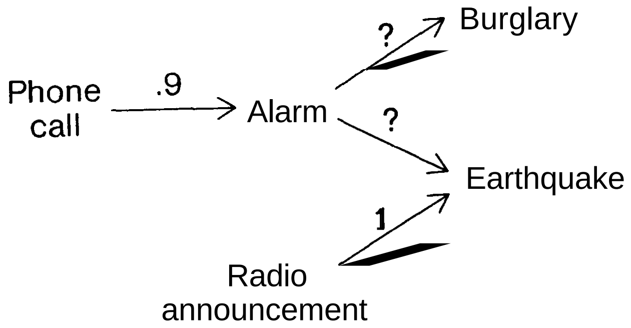
After a while ball has been drawn:

$$\text{Black ball} \xrightarrow{CF = 1} H_1$$

The lack of conditional independence makes it impossible to attach a single CF to the rule.

Restriction on divergence Imposed by modularity

"Mr. Holmes received a telephone call from his neighbor notifying him that she heard a burglar alarm sound from the direction of his home. As he was preparing to rush home, Mr. Holmes recalled that last time the alarm had been triggered by an earthquake. On his way driving home, he heard a radio newscast reporting an earthquake 200 miles away."



When rules diverge from a common piece of evidence, it is impossible to assign a single CF to those rules.

Relaxing the modularity assumption: Influence Diagrams

Inference nets: Assumptions of conditional independence are imposed by the methodology

Influence diagrams: Assumptions of conditional independence are imposed by the user

Summary

An analysis of the CF model reveals inconsistencies between the original interpretation of the numbers and their use.

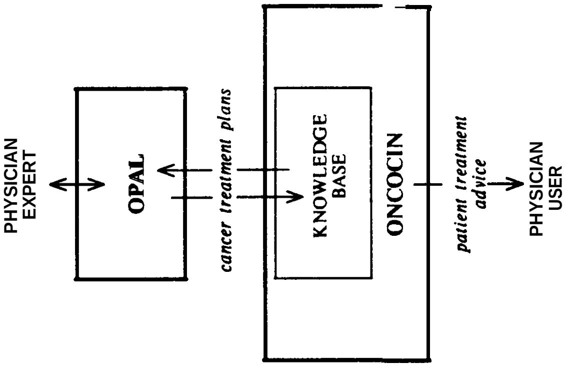
When the inconsistencies are removed, further analysis reveals implicit assumptions of independence made by the model.

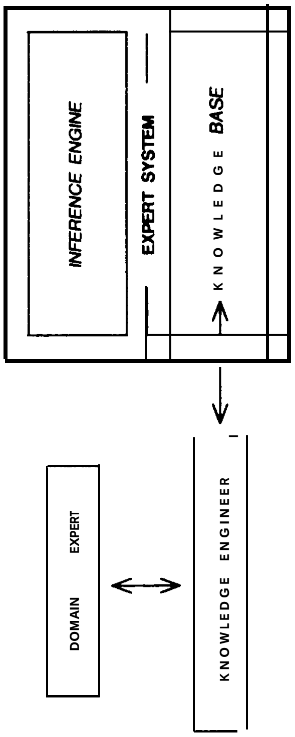
Influence diagrams retain the power of inference nets while providing control over the assumptions of independence.

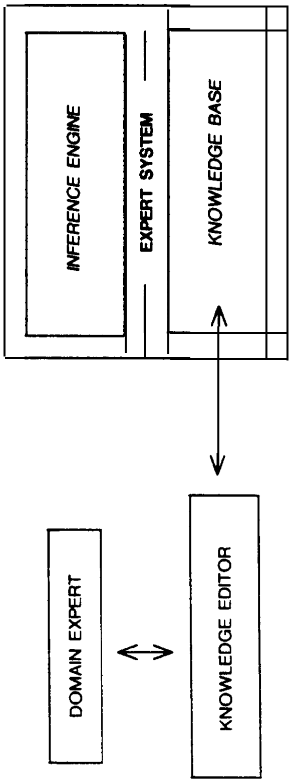
Using a Domain Model to Drive an Interactive Knowledge Editing Tool

Mark A. Musen

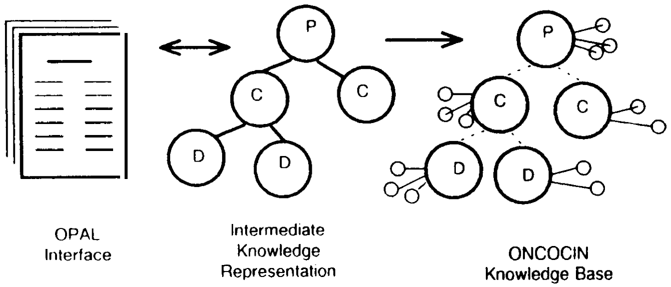
The manner in which programs that acquire knowledge for expert systems display the contents of a knowledge base affects how users enter new knowledge. Previous knowledge acquisition tools have incorporated semantics that allow knowledge to be entered and edited in terms of either the structural representation of the knowledge or the problem-solving method in which the knowledge is ultimately used. A more effective paradigm may be to use the semantics of the application domain itself to govern access to an expert system's knowledge base. This approach has been explored in a program called OPAL, which allows medical specialists working alone to enter and review cancer treatment plans for use by an expert system called ONCOCIN. Knowledge acquisition tools based on strong domain models should be useful in application areas whose structure is well understood and for which there is a need for repetitive knowledge entry.


PHYSICIAN EXPERT
OPAL
cancer treatment plans
KNOWLEDGE BASE
ONCOCIN
patient treatment advice
PHYSICIAN USER


DOMAIN EXPERT
KNOWLEDGE ENGINEER
INFERENCE ENGINE
EXPERT SYSTEM
KNOWLEDGE BASE


DOMAIN EXPERT
KNOWLEDGE EDITOR
INFERENCE ENGINE
EXPERT SYSTEM
KNOWLEDGE BASE

Knowledge Flow in OPAL



Conceptual Data Model

- Semantic assumptions about data
- Framework for interacting with program

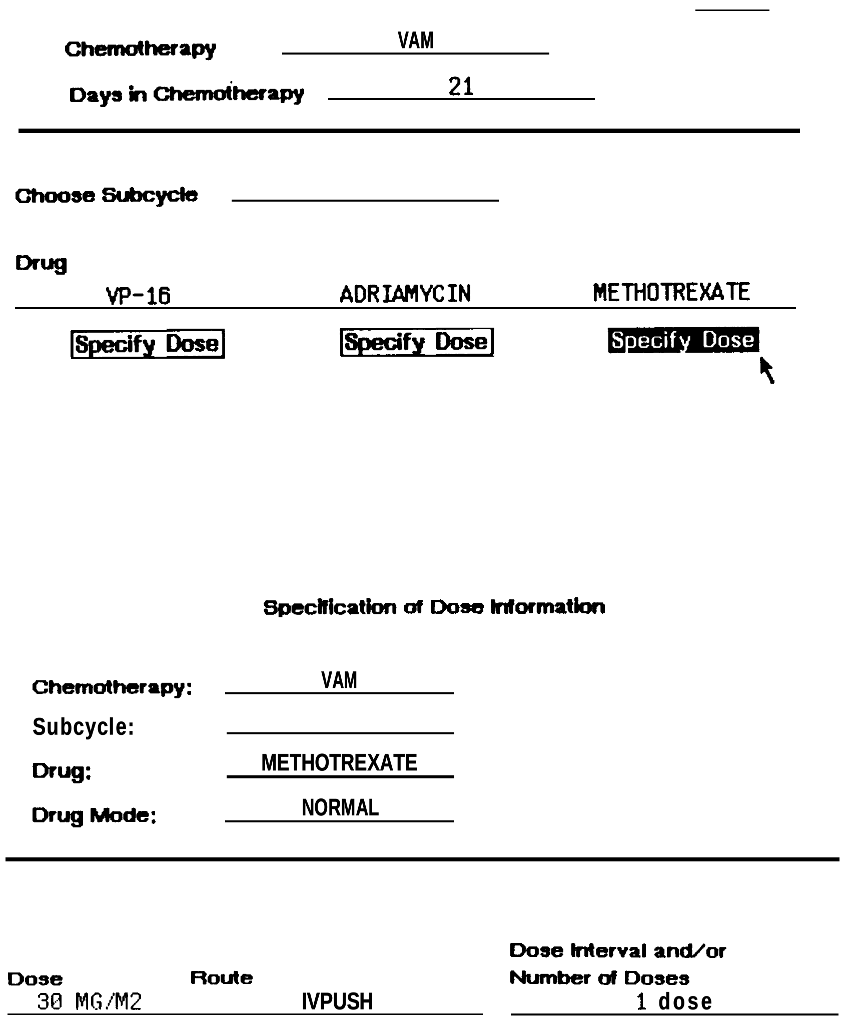
Data Models for Knowledge Editors

1. Knowledge representation
2. Problem-solving strategy
3. Domain model

The Domain Model in OPAL

- Entities and Relations
- Domain Actions
- Domain Predicates
- Procedural Specifications

Entities and Relations



Chemotherapy VAM

Days in Chemotherapy 21

Choose Subcycle

Drug

VP-16	ADRIAMYCIN	METHOTREXATE
Specify Dose	Specify Dose	Specify Dose

Specification of Dose Information

Chemotherapy: VAM

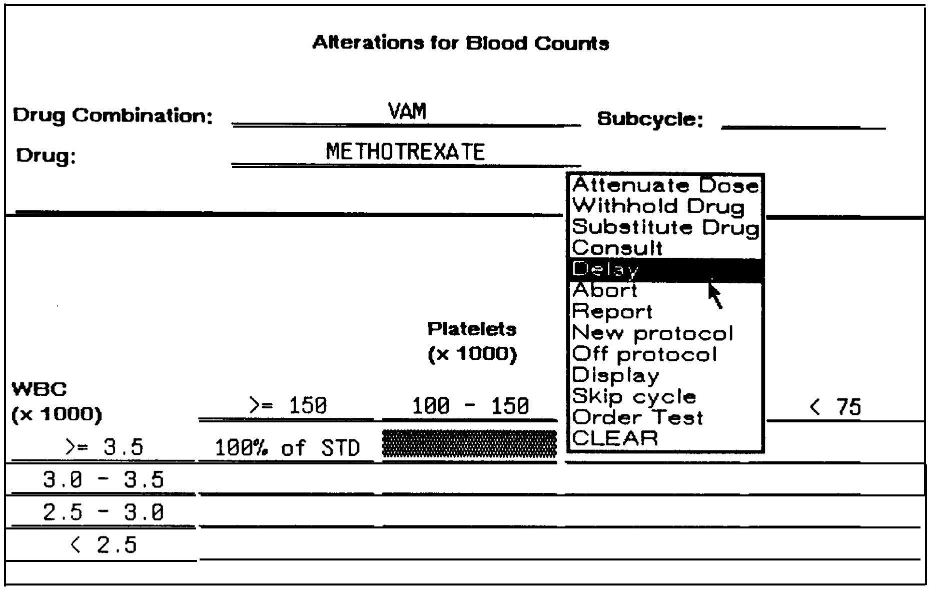
Subcycle:

Drug: METHOTREXATE

Drug Mode: NORMAL

Dose	Route	Dose Interval and/or Number of Doses
30 MG/M2	IVPUSH	1 dose

Domain Actions

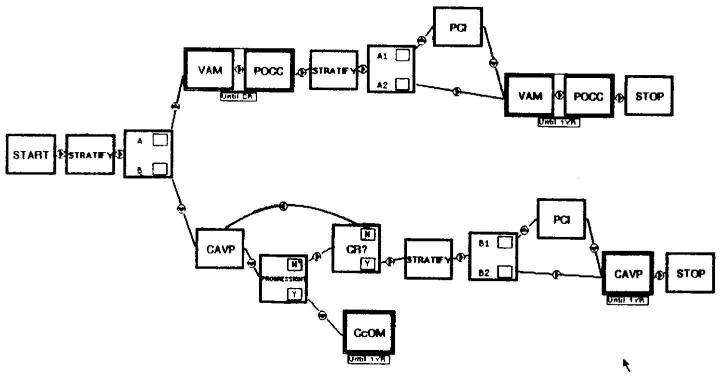


Alterations for Blood Counts

Drug Combination: VAM **Subcycle:**

Drug: METHOTREXATE

WBC (x 1000)	**Platelets (x 1000)** >= 150	100 - 150		< 75
>= 3.5	100% of STD	▨		
3.0 - 3.5				
2.5 - 3.0				
< 2.5				



Using Domain Model as the Data Model:

- Allows program to approximate conceptual model of experts
- Facilitates rapid development of related knowledge bases

Using Domain Model as the Data Model:

- Restricts ability to define new concepts
- May not capture all necessary knowledge
- Assumes domain model will be used repetitively

Some Sample Domains:

- Clinical trials
- Process control
- Troubleshooting electronic instruments
- Geological exploration

Knowledge-Base Congeners

- Derived from common knowledge-level analysis of an application area
- Individual instantiations of a generic domain template
- Use common inference engine

Summary

- Knowledge-acquisition tools project a “data model” of the knowledge base
- OPAL’s data model is derived from a knowledge-level analysis of the domain
- The model permits rapid specification of knowledge bases by non-programmers

A Decision-Theoretic Approach to Heuristic Planning

Curt Langlotz

Many important planning problems are characterized by uncertainty about the current situation and by uncertainty about the consequences of future action. These problems also inevitably involve tradeoffs between the costs and benefits associated with possible actions. Decision theory is an extensively studied methodology for reasoning under these conditions, but has not been explicitly and satisfactorily integrated with artificial intelligence approaches to planning. Likewise, many perceived practical limitations of decision theory, such as problem solving results that are difficult to explain and computational needs that are difficult to satisfy, can be overcome through the use of artificial intelligence techniques. We are building a system which combines the decision-theoretic and artificial intelligence approaches to planning. The combination of these approaches allows better explanation of planning decisions than either one alone, and may have advantages for system performance. In addition, the explicit representation of probabilities and utilities allows flexibility in the construction of a planning system. This means that assumptions made by such systems, which may be critical for their performance, are more easily modified than in a system that does not explicitly represent uncertainties and tradeoffs.

A Decision-Theoretic Approach to Heuristic Planning

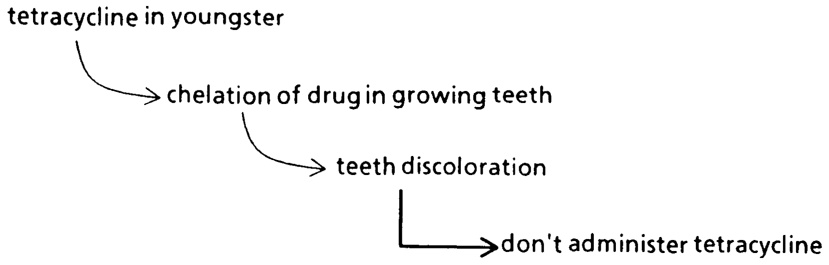
Curtis P. Langlotz
Edward H. Shortliffe
Lawrence M. Fagan

Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University

MYCIN's Tetracycline Rule

```
RULE055
-------
If: 1)The therapy under consideration is tetracycline
    2) The age (in years) of the patient is less than 8
Then: There is strongly suggestive evidence (.8) that
        tetracycline is not an appropriate therapy for use
        against the organism
```

Support Knowledge



tetracycline in youngster

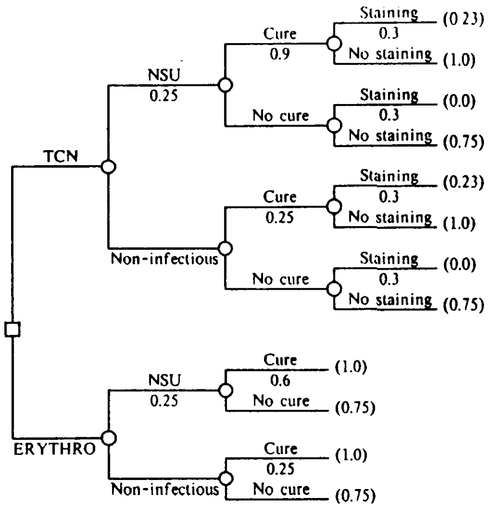
chelation of drug in growing teeth

teeth discoloration

don't administer tetracycline

Questions Relevant to Decision-Making

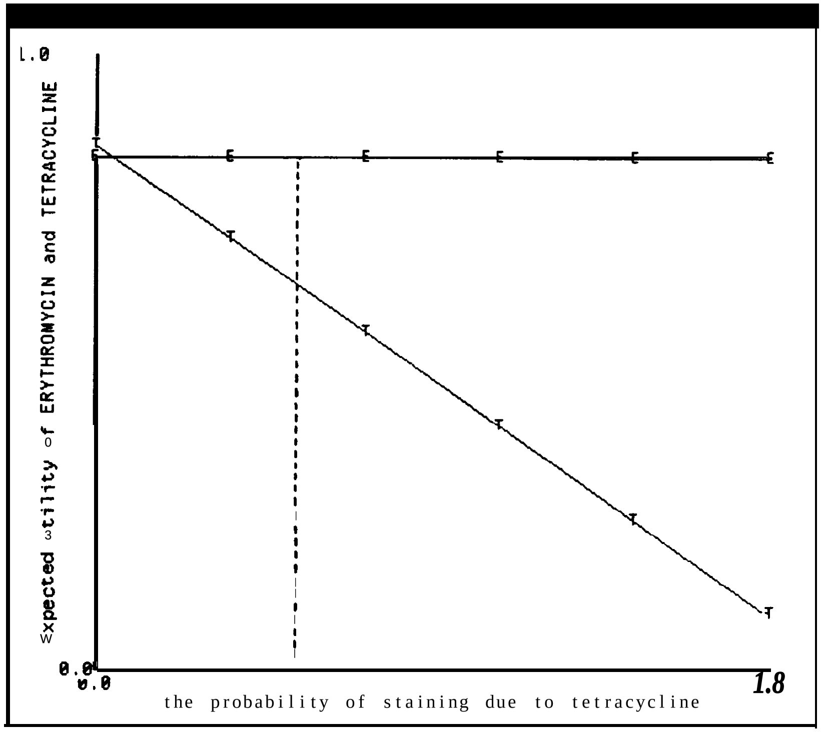
- What if the likelihood of dental staining were 1 /1 OO? 1/1000?
- What if the infecting organism were resistant to all drugs except tetracycline?
- What if tetracycline merely caused gastrointestinal distress?



Results from the Decision Tree

```
The expected utility of giving erythromycin is: 0.83
The expected utility of giving tetracycline is: 0.63
```

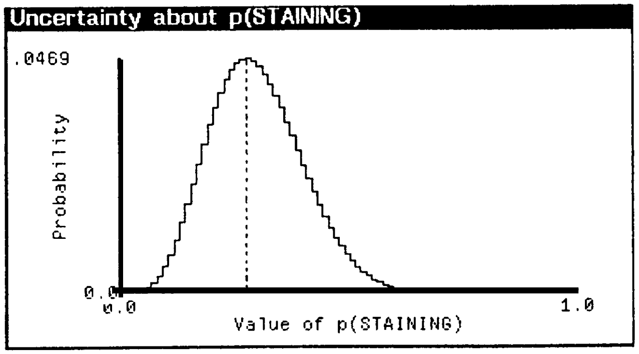
One-way Sensitivity Analysis

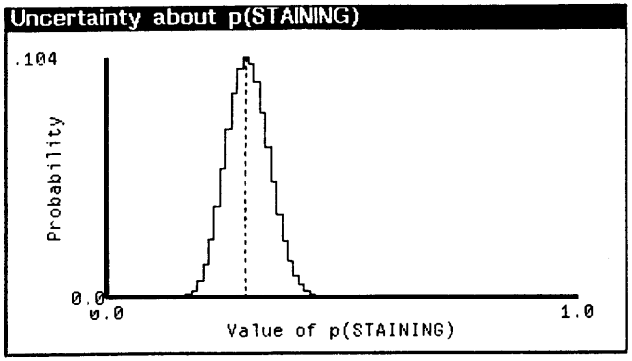


Objections to Decision Theory

- Input unrealistically precise
- Results unrealistically precise
- Combinatoric explosion
- Results difficult to justify

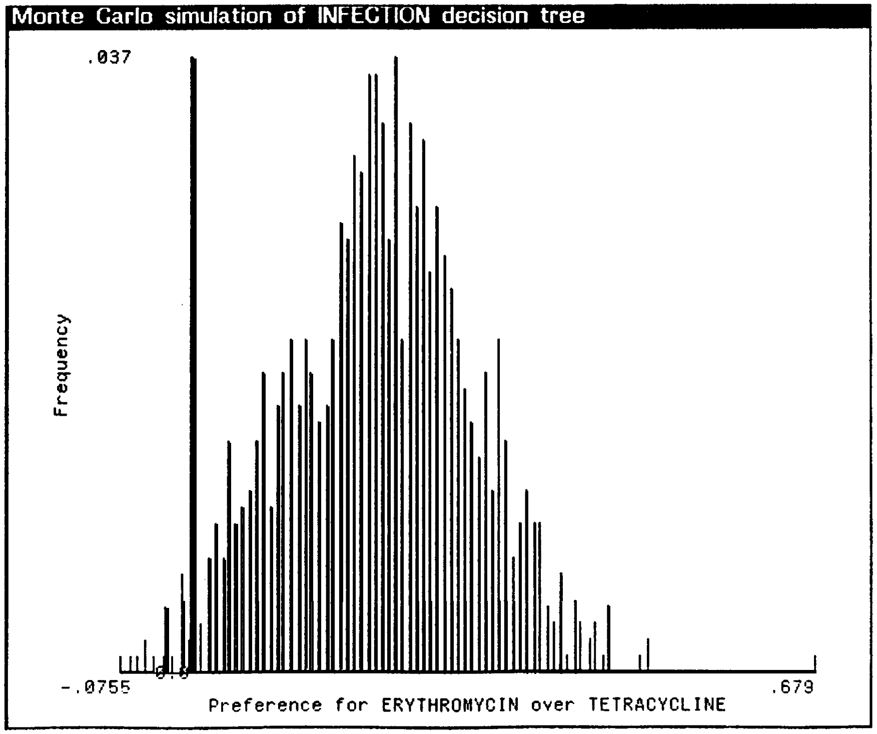
Representing Uncertainty About Quantities

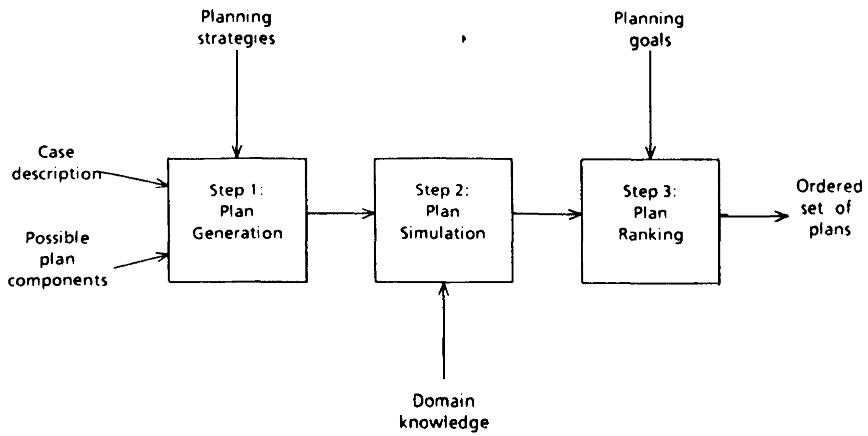




Monte Carlo Simulation

- Repeatedly:
  - ∘ Assign a value to each quantity
  - ∘ Compute expected utilities
  - ∘ Determine the preferred choice
- Count the frequency of results

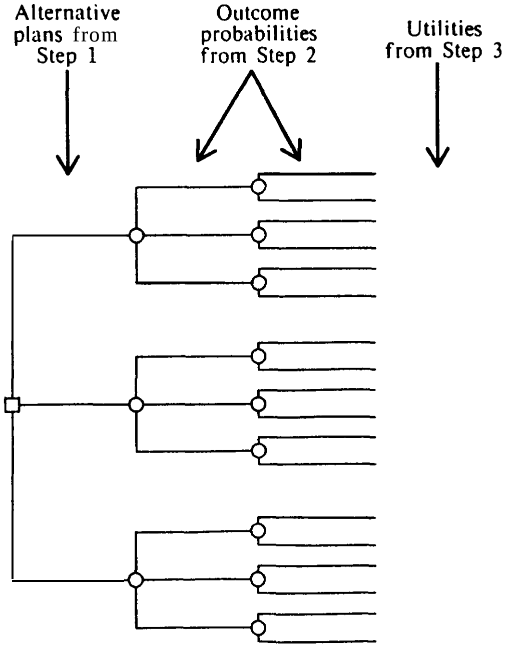




Explaining Decision-Theoretic Support ‘Knowledge

```
((SIGNIFICANT PSTAIN) (> PTCN/CURE PERYTHRO/CURE))

(OUTWEIGHS (> PTCN/CURE PERYTHRO/CURE) (SIGNIFICANT PSTAIN))
```

The fact that the probability of curing NSU with tetracycline is greater than the probability of curing NSU with erythromycin is outweighed by the fact that the probability of staining due to tetracycline is significant.


Alternative plans from Step 1
Outcome probabilities from Step 2
Utilities from Step 3

Summary

. Additional data collection necessary

. Assumptions made explicit

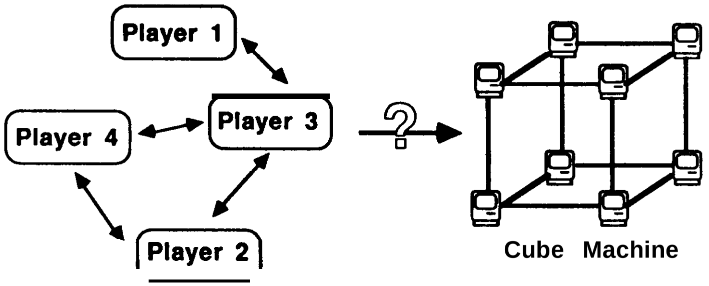
. Appealing explanations generated

Post-Game Analysis A *Heuristic* Approach to Manage Concurrent Execution Environments

Jerry C. Yan

In many distributed programming paradigms, computations are represented by collections of communicating computing agents (say ***"processes").*** When mounted onto distributed hardware, the speed-up obtainable is critically dependent on where these agents are placed initially and how resources are managed during run-time in response to changing program behavior. A "heuristic-based" frame-work that makes use of program execution history is proposed to improve static placement decisions incrementally as well as modifying run-time allocation policies. Initial application of simple heuristics on simple problems demonstrated promising results.

The "Mapping Problem"



Map a single computation to minimize execution time

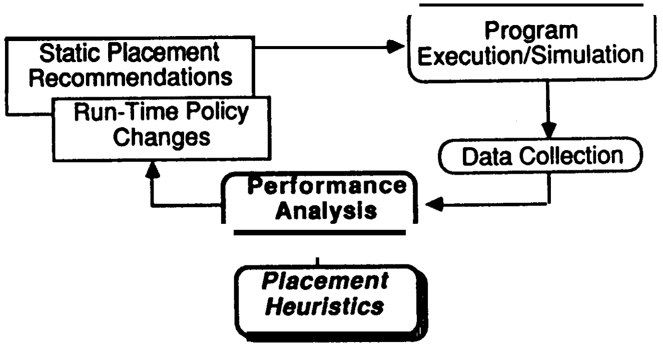
What? Where?

Why?

How? — Difficult!
- NP Complete
- Model/Predict dynamic program behavior
- Run-time information management
- Fast turn-around experimentation environment

"Axe!"

" Post-Game" Analysis



	Data Available	Actions Available
Pre-Game	program text/programmer	static placement
Mid-Game	partial world view	dynamic placement/migration
Post-Game	complete world view	all of the above

"Forecast"	difficult, unpredictable
"Adjustments"	costly, based on incorrect/partial info
"Analysis"	at least "statistically correct" information

Whv bother with **"post-game"?**
- programs with ***data-independent*** behavior
- post-game — a special case for mid-game analysis
- ***"hind-sight is better than fore-sight"***

“Opportunistic” **Generate and Test**

Elements of the Problem Solving Paradigm

- World view — current configuration
- Perspectives — world view ***from** an angle*
- Points of focus — partial perspectives
- Actions — specific/abstract/chained
- Heuristics — **knowledge** based evaluation → actions

Problem Solving Process

- Data abstraction → perspectives
- Iterative application of heuristics → actions
- Combine abstract actions → specific actions
- Rank/Resolve actions → placement/policy changes

Heuristics

- Represent different/independent lines of reasoning
- Predict impact of purturbations on execution time

Actions

- Static placement changes
- Dynamic placement policy changes

“Period” / “Power”

- Predicting purturbations to different extent
- Predicting effects of multiple actions

" Perspective" Analysis

Perspectives and **Concerns**

focus	**Concern**
player	“happy” at current site?
site	“happy” with resident players?
	utilization — effectively distributed?
player-player	move together/apart closer/further?
player-site	prioritize target-selection

Corresponding Heuristics

IF	**(<pred> p_1)**	“emigrate”
IF	**(<pred> s_1)**	**"emigrate"/"immigrate"**
IF	**(<pred> p_1 p_2)**	" move”
IF	**(<pred> p_1 s_1)**	“move”

Actual **Parameters**

- player descriptors
  - $\mathbf{cpu}_{used}$, $\mathbf{cpu}_{wait}$
  - (n $\mathbf{send}_{rem}$ + $\mathbf{n\_recv}_{rem}$) / ($\mathbf{n\_send}_{loc}$ + $\mathbf{n\_recv}_{loc}$)
- site descriptors
  - $\mathbf{cpu}_{user}$, $\mathbf{cpu}_{os}$, $\mathbf{cpu}_{contention}$
  - $\mathbf{memory}_{used}$
- interaction matrices
  - (p-p, p-s): amt send, amt_**recv**
  - (p-p): msg wait = $\mathbf{time}_{transmit}$ + $\mathbf{time}_{synchronization}$

An Initial Experiment

Four Single-Heuristic-Policies

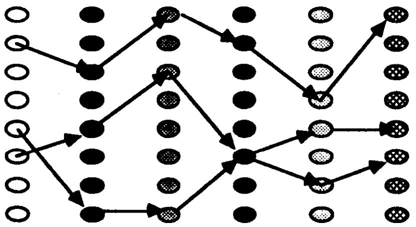
1 **If (and** ***(most-crowded*** **$site_a$)**
(most unhappy **$player_x$ $site_a$)**
(most vacant **$site_b$))**
then ***(move*** **$player_x$ $site_b$)**

2 **If (and** ***(most-crowded*** **$site_a$)**
(most_happy **$player_x$ $site_a$)**
(most vacant **$site_b$))**
then ***(move*** **$player_x$ $site_b$)**

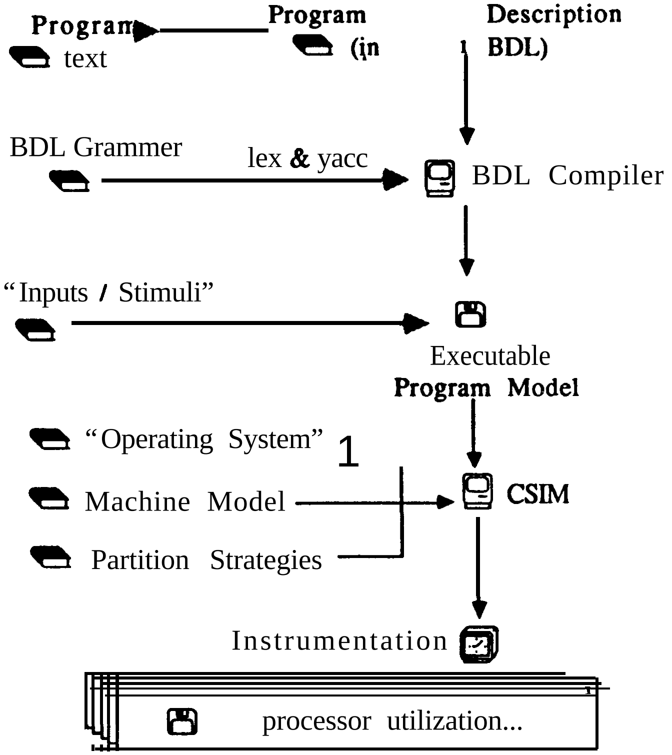
3 **If (and** ***(most-crowded*** **$site_a$)**
(most_unhappy **$player_x$ $site_a$)**
(likes_most **$player_x$ $site_b$))**
then ***(move*** **$player_x$ $site_b$)**

4 **If (and** ***(most-crowded*** **$site_a$)**
(most_happy **$player_x$ $site_a$)**
(likes most **$player_x$ $site_b$))**
then ***(move*** **$player_x$ $site_b$)**

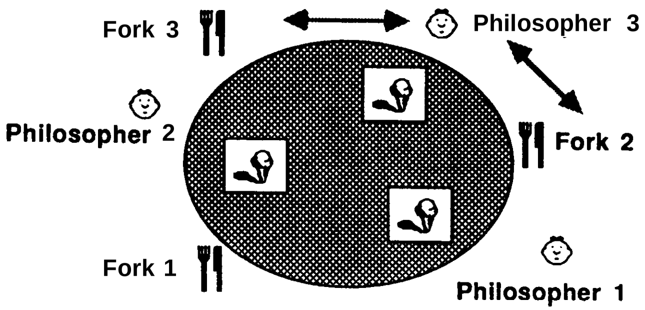
The Program — A "pipe-line" of communicating players



" Axe" (ala " BeatCal")

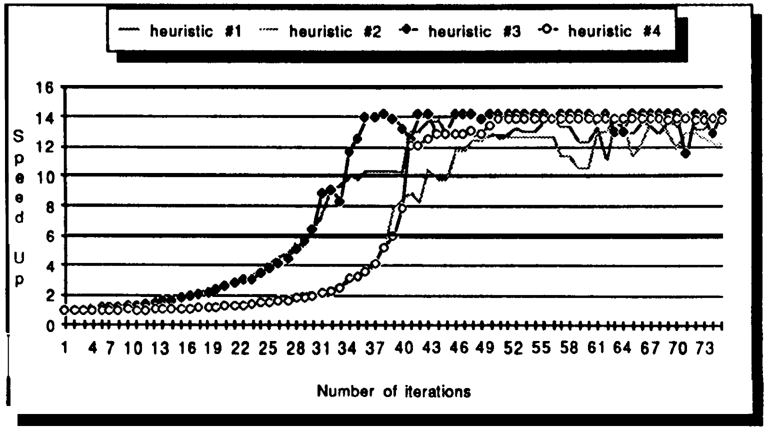


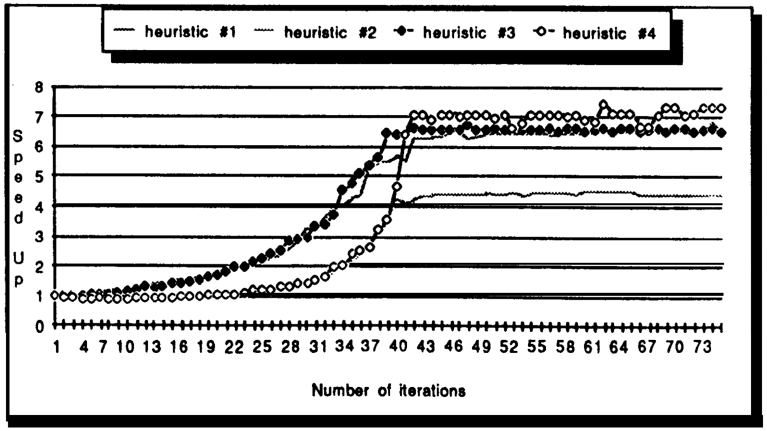
The Philosophers Who Love Ice-cream



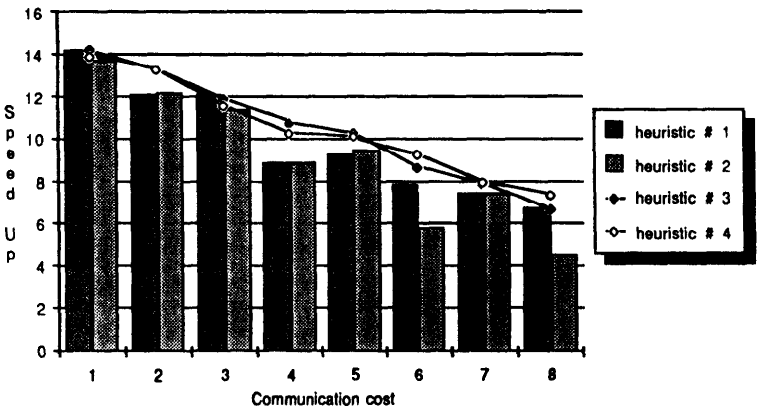
```
(DefPlayer Philosopher                    ; Structure specification
  (init start eat ack)                    ; message names
  (Left fork Right_fork)                  ; acquaintances
  o-
  (init                                   ; initialization
    (record Left-fork Right-fork))        ; get neighbors' addresses
  (start_eat                              ; execution
    (post Left_fork pick self)            ; try to get left fork
    (wait ack)                            ; waits for acknowledgement
    (post Right_fork pick self)           ; try the other fork
    (wait ack)                            ; wait for acknowledgement
    (run 23)                              ; he eats!
    (post Left fork release)              ; releases left fork
    (post Right_fork release)))           ; releases other fork

(Global Joe Philosopher)                  ; "Global" declaration
```

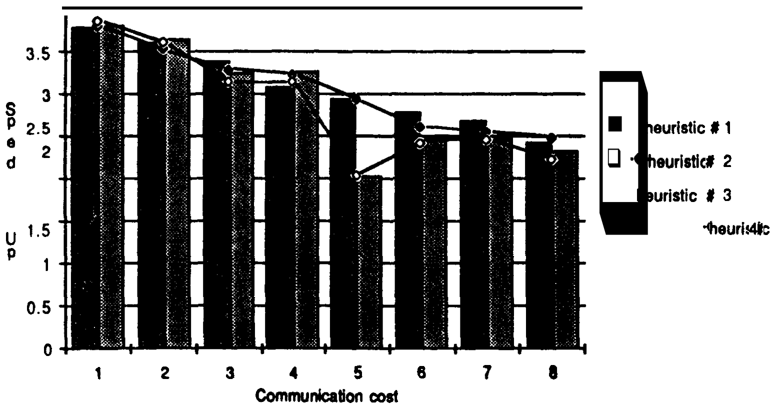
Results II: Progress Charts



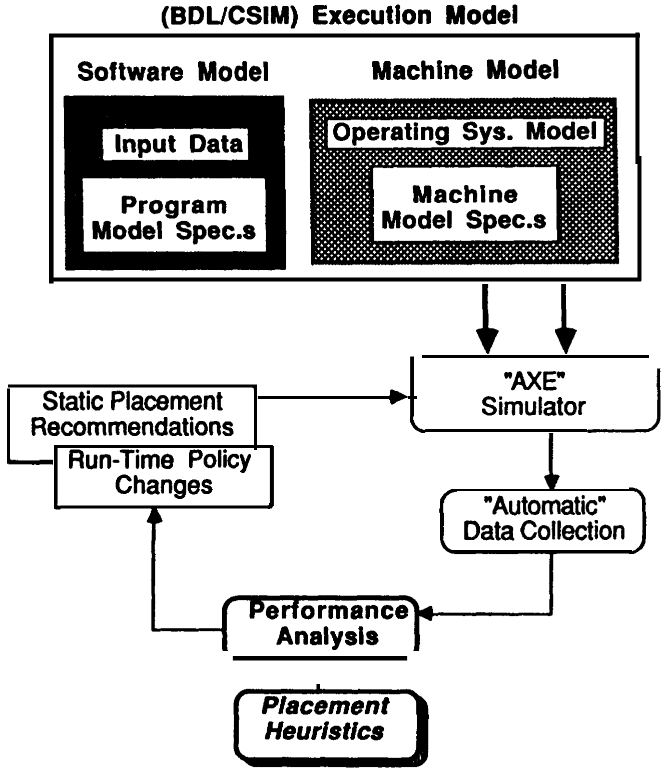


Results III: Variation of BAS





Summary and Future Work

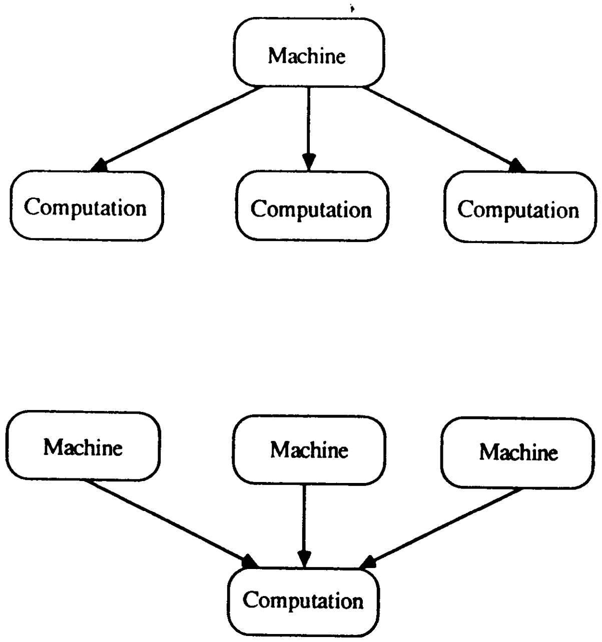


Performance Prediction of Concurrent System

Victor W. Mak

Parallel processing is very promising in delivering a lot of computing power to a wide variety of problems. Unlike conventional von Neumann architectures, we do not have the knowledge and experience to program these new architectures to take full advantage of their concurrency features. Finding the right computation structure on the right machine is still something we know very little of. A tool that can predict performance of computations on parallel machines is definitely needed to further our understanding of concurrent systems.

Earlier works in this area either dealt with a very simple computation structure or resulted in methods with combinatorial complexity. Simulation is also not feasible due to the enormous amount of time required. An approximate method using queueing network modeling and a probabilistic approach is proposed. This method is very efficient in predicting performance of series-parallel-reducible task structures running on parallel machines. In this presentation, the basic ideas of the method will be explained. Numerical results for three test cases are presented and compared to those of simulations. Although the method is only an approximation, it is found to be very accurate and computationally efficient, and is well suited to be a tool to get first-order performance estimates of concurrent systems.



Needs ☞ A Tool to Predict Performance of Computations on Parallel Machines

Problems

☞ Simulation:

very time consuming

☞ Markov Modeling:

state explosion

☞ Petri Net:

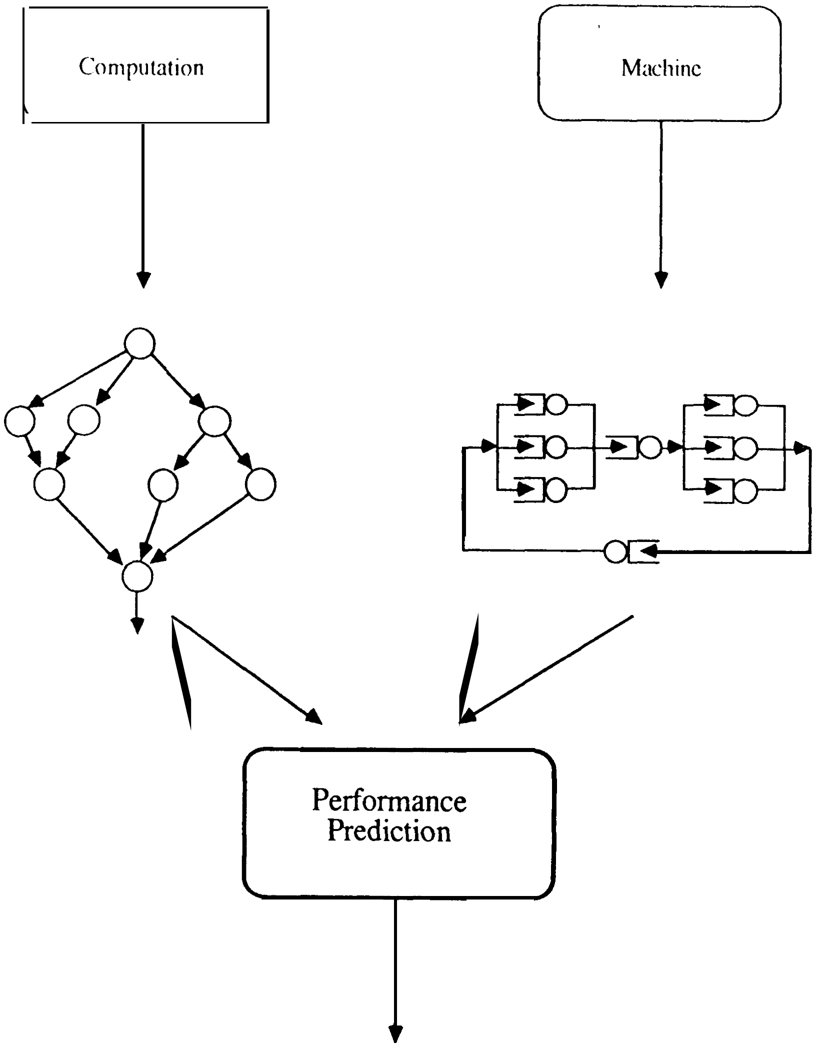
isomorphic to Markov Chain

☞ Queueing Network Modeling:

cannot handle concurrency

Proposed Method:

Extend Queueing Network Modeling to handle concurrency



Related Work

☞ Thomasian & Bay 8/83, 12/86

- Generate a Markov Chain of different task combinations
- Use Queueing Network to solve for state transition rates

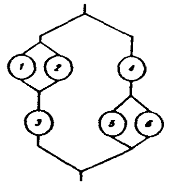


Fig. 1. Six-task system.

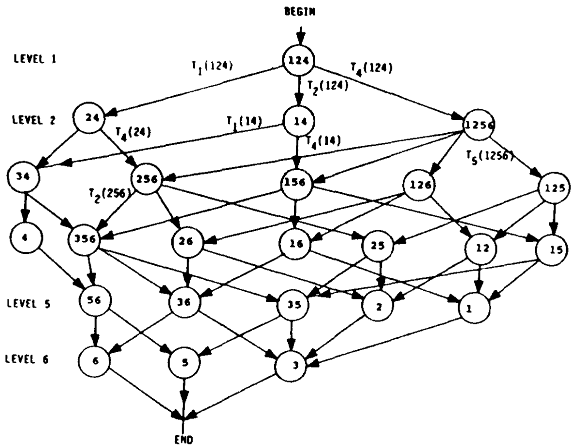
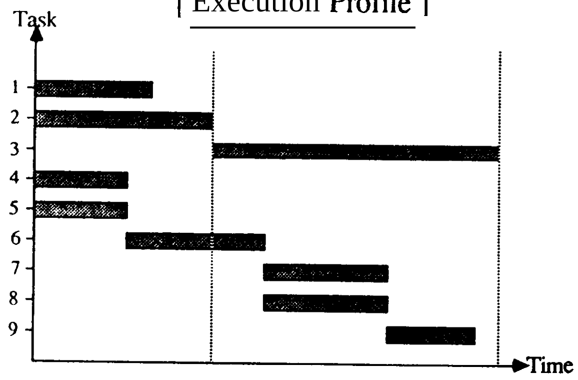
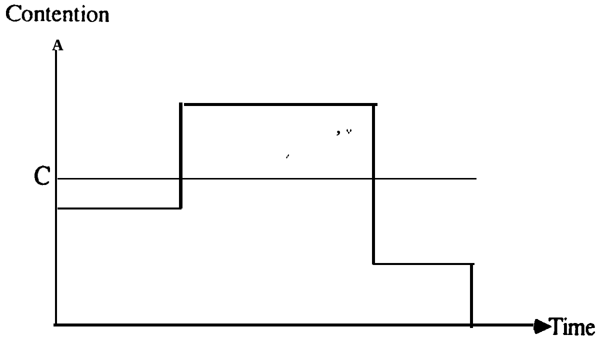


Fig. 2. Markov chain for six-task system.

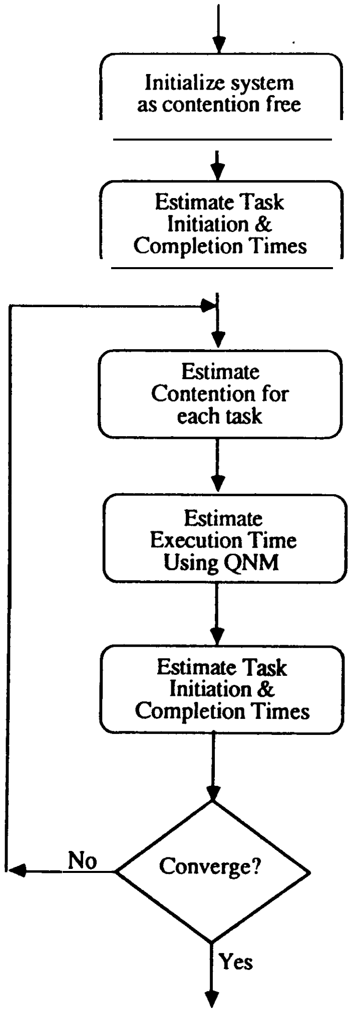
Execution Profile

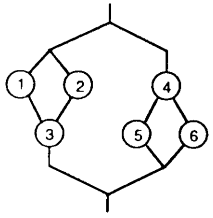


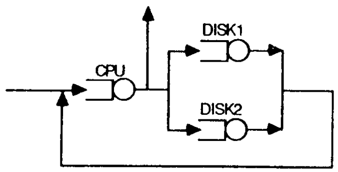
☞ Contention as seen by a task changes as task enters and leaves the system



☞ Approximate contention as C during a task's execution interval

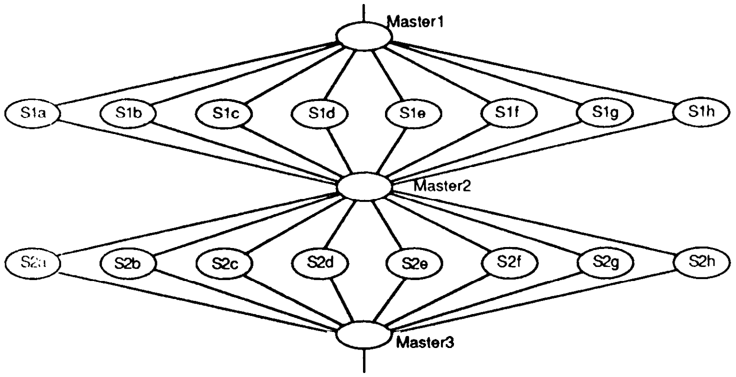

Initialize system as contention free
Estimate Task Initiation & Completion Times
Estimate Contention for each task
Estimate Execution Time Using QNM
Estimate Task Initiation & Completion Times
Converge?
No
Yes

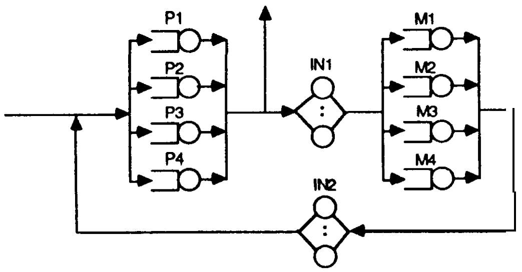




Results of 6-Task System

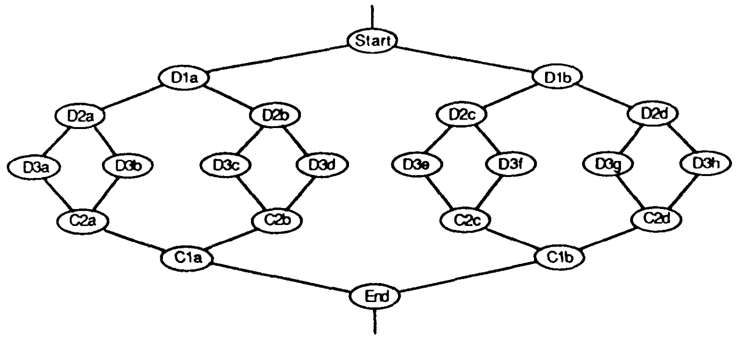
	Operations l	Simulation	Error
C	6.235	6.140	1.55%
E_1	1.825	1.795	1.67%
E_2	1.825	1.876	-2.72%
E_3	2.233	2.222	0.50%
E_4	2.470	2.469	0.04%
E_5	1.716	1.705	0.65%
E_6	1.716	1.720	-0.23%
I_1, I_2, I_4	0.000	0.000	0.00%
I_3	2.738	2.643	3.59%
I_5, I_6	2.470	2.469	0.04%





Results of Master-Slave System

	Operational	Simulation	Error
C	10.452	10.732	-2.61%
$E_{Master1}$	1.070	1.059	1.04%
E_{S1a}	1.667	1.642	1.52%
E_{S1b}	1.430	1.390	2.88%
E_{S1c}	1.667	1.646	1.28%
E_{S1d}	1.430	1.401	2.07%
E_{S1e}	1.687	1.643	1.46%
E_{S1f}	1.430	1.427	0.21%
E_{S1g}	1.667	1.638	1.77%
E_{S1h}	1.430	1.425	0.35%
$E_{Master2}$	1.070	1.068	0.19%
E_{S2a}	1.677	1.659	1.08%
E_{S2b}	1.677	1.642	2.13%
E_{S2c}	1.677	1.632	2.76%
E_{S2d}	1.677	1.666	0.66%
E_{S2e}	1.421	1.432	-0.77%
E_{S2f}	1.421	1.398	1.65%
E_{S2g}	1.421	1.387	2.45%
E_{S2h}	1.421	1.430	-0.63%
$E_{Master3}$	1.070	1.075	-0.47%
$I_{Master1}$	0.000	0.000	0.00%
I_{S1}	1.070	1.059	1.04%
$I_{Master2}$	4.712	4.814	-2.12%
I_{S2}	5.702	6.882	-1.70%
$I_{Master3}$	9.382	9.657	-2.05%



Results of Divide-And-Conquer System

	Operational	Simulation	Error
C	12.002	11.994	0.07%
E_{Start}	1.070	1.083	-1.20%
E_{D1a}	1.407	1.401	0.43%
E_{D1b}	1.407	1.363	3.23%
E_{D2a}	1.504	1.503	0.07%
E_{D2b}	1.504	1.494	0.67%
E_{D2c}	1.504	1.479	1.69%
E_{D2d}	1.504	1.482	1.48%
E_{D3a}	1.653	1.665	-0.72%
E_{D3b}	1.653	1.640	0.79%
E_{D3c}	1.653	1.625	1.72%
E_{D3d}	1.653	1.640	0.79%
E_{D3e}	1.653	1.659	-0.36%
E_{D3f}	1.653	1.653	0.00%
E_{D3g}	1.653	1.642	0.67%
E_{D3h}	1.653	1.660	-0.42%
E_{C2a}	0.972	0.940	3.40%
E_{C2b}	0.972	0.955	1.78%
E_{C2c}	0.972	0.944	2.97%
E_{C2d}	0.972	0.936	3.85%
E_{C1a}	0.909	0.891	2.02%
E_{C1b}	0.909	0.900	1.00%
E_{End}	1.070	1.069	0.09%
I_{Start}	0.000	0.000	0.00%
I_{D1a}, I_{D1b}	1.070	1.083	-1.20%
I_{D2a}, I_{D2b}	2.477	2.484	-0.28%
I_{D2c}, I_{D2d}	2.477	2.446	1.27%
$\mathrm{ID3a}$, I_{D3b}	3.980	3.988	-0.20%
I_{D3c}, I_{D3d}	3.980	3.978	0.05%
I_{D3e}, I_{D3f}	3.980	3.925	1.40%
I_{D3g}, I_{D3h}	3.980	3.928	1.32%
I_{C2a}	6.460	6.373	1.37%
I_{C2b}	6.460	6.330	2.05%
I_{C2c}	6.460	6.312	2.34%
I_{C2d}	6.460	6.300	2.54%
I_{C1a}	8.566	8.550	0.19%
I_{C1b}	8.566	8.522	0.52%
I_{End}	10.932	10.925	0.06%

Future Directions

- ☞ Improve the Approximation of Equivalent Contention

- ☞ Extend the method to non-series-parallel computation structures

- ☞ Application: Evaluate different parallel architectures and idenitify bottleneck in the system

- ☞ Application: Evaluate different Task Allocation Schemes

Beta Operations: Efficient Implementation of a Primitive Parallel Operation

Evan R. Cohn

We will consider the primitive parallel operation of the Connection Machine, the Beta Operation. Let the input size of the problem be N and output size M. We will show how to perform the Beta Operation on an N-node hypercube in $O(\log \mathrm{N} + \log^2 M)$ time. For a $\sqrt{N} \times$ @mesh-of-trees, we require $O(\log \mathrm{N} + \sqrt{M})$ time. We show that the AT^2 upper bound thus obtained for the mesh-of-trees is close to optimal.

Beta Operations: Efficient Implementation of a Primitive Parallel Operation

Evan Cohn
and
Ramsey Haddad

Stanford University

Preliminary Definitions

Given a binary function, F and and array of values c = $[c_0, \ldots, c_m]$, let us define the F-reduction *of* C as the natural reduction, except with no specified evaluation order. That is

- $F/[c_0] = c_0$

- $F/[c_0, \ldots, c_m] =$
  $F(F/[c_{\pi_0}, \ldots, c_{\pi_i}], F/[c_{\pi_{i+1}}, \ldots, c_{\pi_m}])$
  for some □ _< i < m, and some permutation π.

$n = \log N$
$q = \log |G|$

The Beta Operation:

Input:

A function, F, and N pairs, (g_0, v_0), $, (g_{N-1}, v_{N-1})$

Let s_i be the array of v-values from (g, v) pairs with $g = g_i$.

Output:

$y_i = F/s_i$, for all non–empty s_i's

How fast can we perform this operation on a hypercube ?

If $G = 1$ a simple implementation takes time $\log N$.

If $G = N$ we have a permutation routing and the best time known is $\log^2 N$.

The below implementation takes time $O(\log N + \log^2 |G|)$.

Efficient Hypercube Implementation

There are two phases.

We break the cube into subcubes and apply the following generic step in both phases.

Sort. We sort the (g, v)-pairs in the subcube by g-value.

Reduce. For each distinct g, we combine the pairs with that g into a single (g, v)-pair, by applying the function F to the associated v-values.

Compact. We route the resulting (g, v)-pairs ($\leq |G|$ of them) into the lowest numbered processors of the subcube, retaining their sorted-by-g order.

PHASE 1

Break the N processors into $N/|G|^3$ hypercubes of $|G|^3$ nodes each such that hypercube j has binary representation:

$$\overbrace{*\cdots*}^{3q}\overbrace{j}^{n-3q}.$$

For each hypercube we perform the generic step.

PHASE 2

Steps i = 0 through i = $n/q - 4$:

Break the $N/|G|^i$ lowest numbered processors into $N/|G|^{i+4}$ hypercubes of $|G|^4$ nodes each, such that hypercube j has binary representation:

$$\overbrace{0\cdots0}^{iq}\overbrace{*\cdots*}^{4q}\overbrace{j}^{n-(i+4)q}.$$

Note that with this choice of breaking up the cube, each hypercube has only $|G|^2$ (g, v) pairs.

How fast can we perform this operation on the MOT ?

If G = **1** a simple implementation takes time log **N.**

If G = **N** we have a permutation routing and the optimal time is $\sqrt{N}$.

The below implementation has three phases and takes time $O(\log \mathbf{N} + \sqrt{|G|})$.

Efficient Mesh-of-Trees Implementation

PHASE 1

Break the **N** processors into $N/4|G|$ sub-MOT's with side $\sqrt{4|G|}$. Perform the Beta Operation on these sub-MOT's.

PHASE 2

Steps i = **1** through $3q/2 - 1$**:**

In step i we divide the MOT into $N/(4^i|G|)$ sub-MOT's with sides of length $\sqrt{4^i|G|}$. We then conceptually clump **4** contiguous sub-MOT's into a single square sub-MOT with twice the side length. Route the (g, v) pairs to the bottom rows. Apply the generic step.

PHASE 3

Steps i = 1 through $\log(n/2q$ – 1)

In each step i we will increase the side of the sub-MOT from $|G|^{2^{i-1}+1}$ to $|G|^{2^{i}+1}$ and apply the generic step.

The last step will leave us a single MOT with side $\sqrt{N}$. Notice that during this phase each sub-MOT will have a number of processors equal to the square of the number of non-trivial (g, v)-**pairs**.

Summary

- On hypercube or shuffle-exchange, implementation takes $O(\log$ [illegible] + $\log^2 |G|)$.
- On MOT, implementation takes $O(\log N + \sqrt{|G|})$.
- These time bounds hold even if $|G|$ is unknown beforehand.
- Any when- and where-determinate circuit that can perform Beta Operations for any $|G|$ requires AT^2 = $\Omega(N|G|$ log [illegible] where [illegible] is the number of input pairs.
- Variants of Beta Operation take same time and space.

Tradeoffs in Data-Buffer Design

Hans Mulder

Recent technology advances require low traffic ratios for on-chip buffers because of two reasons: the on-chip computational speed increases faster than the pin-bandwidth, and bus congestion for multi-microprocessor systems can seriously limit performance. The processor architecture, compile-time software, and buffering techniques determine the bandwidth requirements for a particular processor, and can be optimized to reduce these requirements. The research presented here mainly concerns the traffic effects of different data-buffering techniques; specifically a comparison between data buffers, and the area distribution between instruction and data caches.

In this talk, I will show a performance comparison as function of occupied area for four different data buffers, namely: a single register set, multiple overlapping register windows as found in RISC, a copy-back associative cache with 16-byte blocks, and a single register set and cache combination. Because caches have significant tag and status bit overhead, the area occupied by a cache data bit is 1.5 to 3 times more than that of a register bit for block sizes from 16 to 4 bytes. This comparison shows a clear division in three sections: small buffers (≤ 32 words) which necessarily are single registers sets, large buffers (> 128 words) which have to be two-level, and a sort-of-grey area in between were the buffer choice depends on many factors not discussed here.

An interesting utility function emerges when the data-cache traffic ratio is combined with the instruction-cache traffic ratio described in Mitchell's Thesis (Stanford, 1986). This utility or traffic-ratio function is shown for a fixed format (32-bit) STACK and a bit-encoded DCA architecture. Together they comprise the two extremes on the traffic scale; all other architectures fit in between. The total traffic-ratio function shows that for a total area of 256 register-equivalent words, the data cache should get 25 % of the area in case of the STACK architecture and 75 % in case of the DCA architecture.

Introduction

Memory access constraints limit performance

On-chip computational speed increases faster than pin bandwidth

Bus congestion for multi-processor systems

What affects the processor's traffic requirements

Processor architecture

Compile-time software

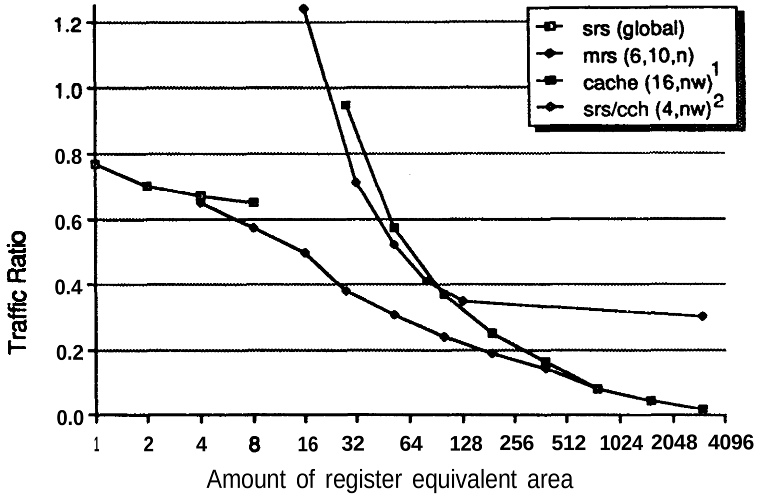
Instruction and data buffering

Research summary

Comparitive analysis of data buffering techniques

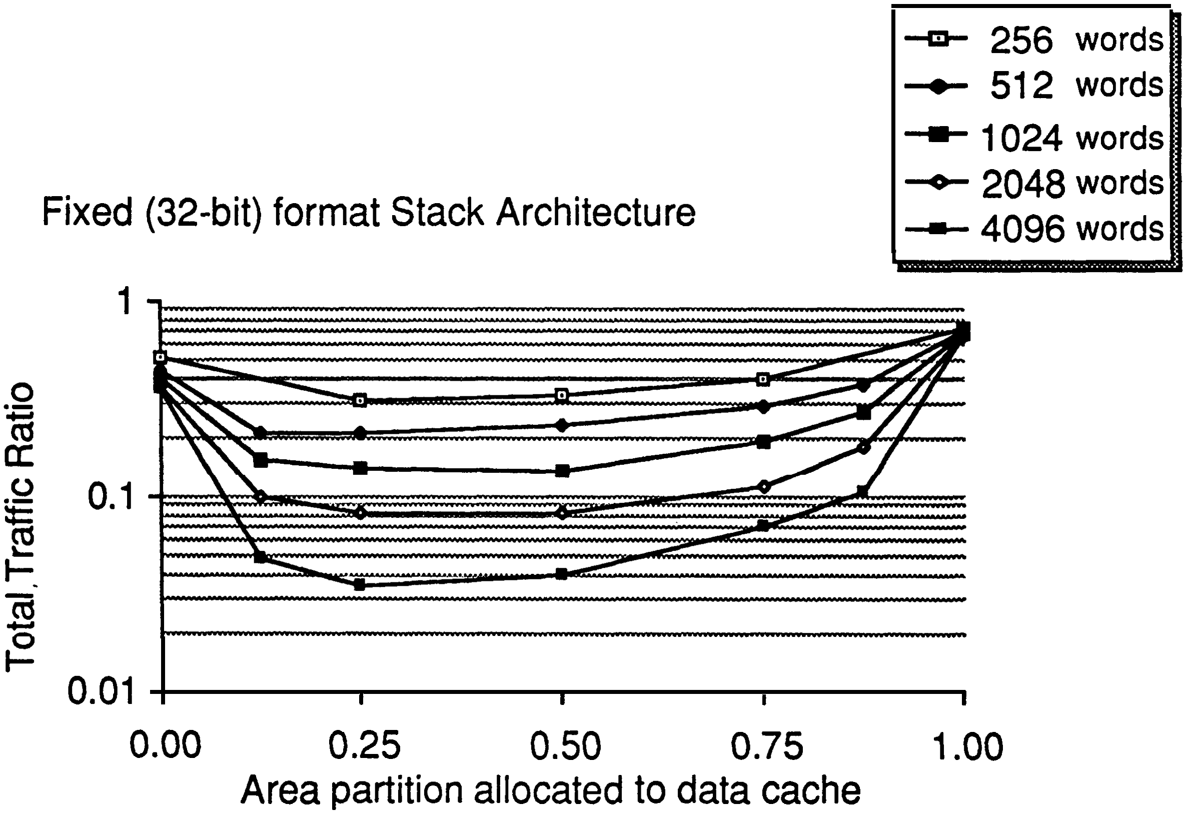
Tradeoffs in buffer design

' Data Buffer Comparison



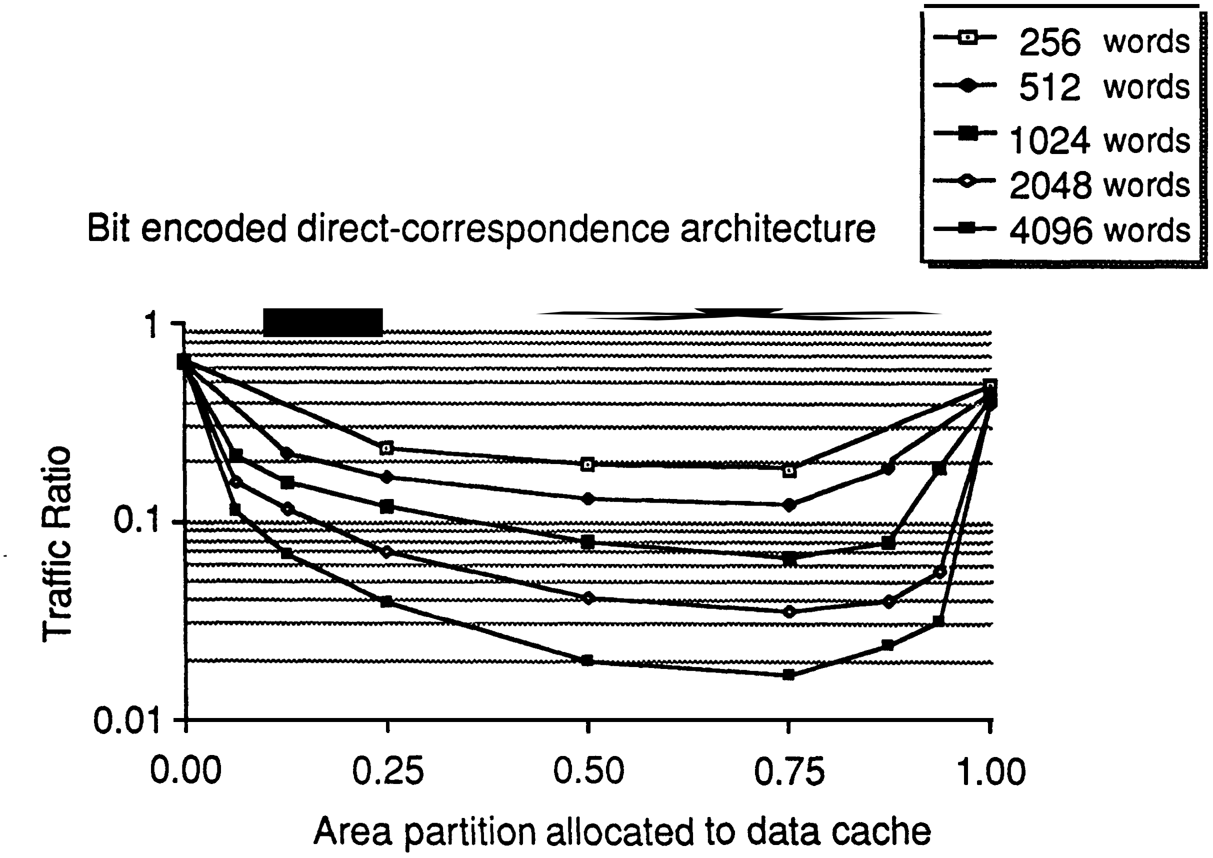
1. fully associative. 16-byte blocks. copy back. total area = 1.5 . data area.
2. fully associative. 4-byte blocks. copy back. total area = 3 . data area.

Instruction’ versus Data[2] Cache



1. 2-way set assoc. 16-byte blocks. total area = 1.2 * data area.
2. fully assoc. 16-byte blocks. copy back. total area = 1.5 * data area.

Instruction[1] versus Data[2] Cache



Bit encoded direct-correspondence architecture

[1] 2-way set assoc. 16-byte blocks. total area = 1.2 • data area.

[2] fully assoc. 16-byte blocks. copy back. total area = 1.5 * data area.

Stanford Packet Radio Network

Brian Boesch

The Stanford Packet Radio project is an experiment in extending a relatively high performance connection from the university network to the researchers' homes. The system is intended to serve the dual role of providing a service and of performing as a test bed for implementation and evaluation of protocols and user interfaces for distributing workload between hosts and workstations while minimizing the requirements for high speed communication. The emphasis is on simplicity and low cost while still providing a significant performance enhancement over conventional dial up communication.

In addition, preliminary research into the performance of layered protocols is discussed in the context of the Stanford Packet Radio net. When protocols are layered, interaction between the layers can cause significant performance degradation. The use of Transmission Control Protocol (the Internet transport layer protocol) with High Level Data link Control(HDLC) is a particular example of this adverse interaction. Rather than addition of a discrete link layer, a case is made for using specialized gateways to avoid the inefficiencies of layering TCP over HDLC.

Introduction

Project Goals

- Provide High Speed Remote Access to SU computer Network
- Provide a Basis for Future Research
- Modest Hardware development
- Research Tool

Motivation for Packet Radio

Why distribute to workstations?

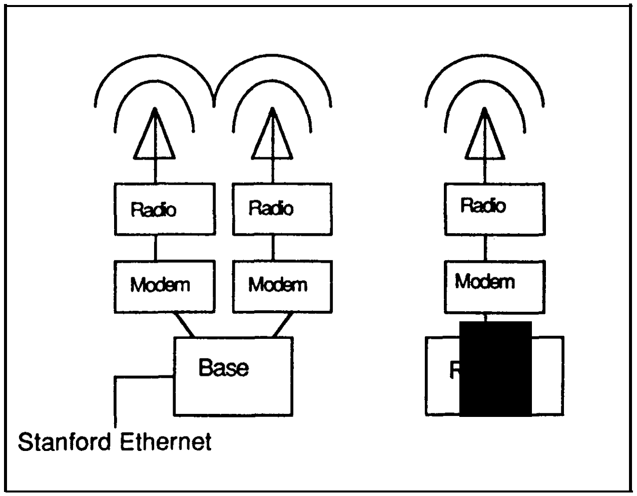
- User give up power of high speed network when home.
- Powerful workstations exist but are "lobotomized" by terminal emulation programs
- Host computers provide: large computation, storage, and shared data
- Terminal intensive tasks can be better handled by workstation.

Why Radio ?

- Radio is relatively low per station cost/ low cost to resite
- Availability of bandwidth
- Shared high speed channel matches bursty traffic better than dedicated low speed channel

Block Diagram of System

* 56 Kbps data rate
* Star configuration
* Base station is full duplex
* Remotes operate half duplex
* Use Internet protocols



Protocol Layering

ISO concept of layered protocols very appealing

- Simplicity - Each layer is effectively an independent entity
- Network design - Gateways and exterior networks need not know higher level protocols
- Interchangeability - Selection of protocol for each layer made relatively independently of other layers

Define terms:

- "active layer" - attempts to ensure correct reception of it's data
- "passive layer" - transforms and transports data but does not ensure correct delivery

Layers

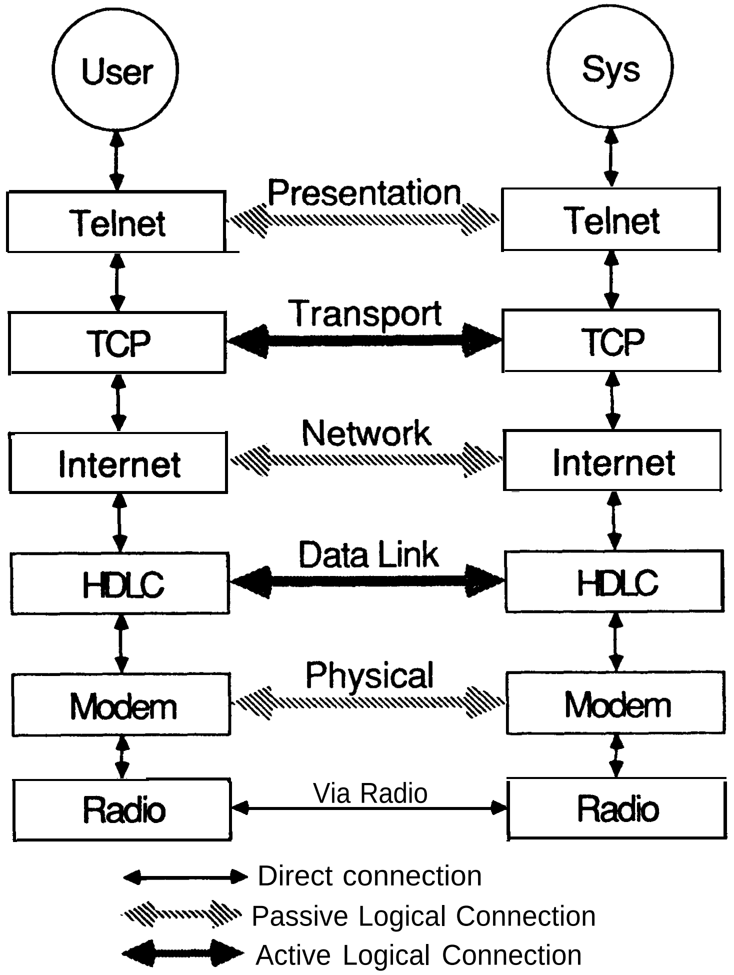
Active protocol layers make assumptions about their environment:

- Error rate
  Lower Layers
- Delays in transit
- Cost of transmission
- Congestion control
- Ordering
  Higher Layers
- Duplicate data

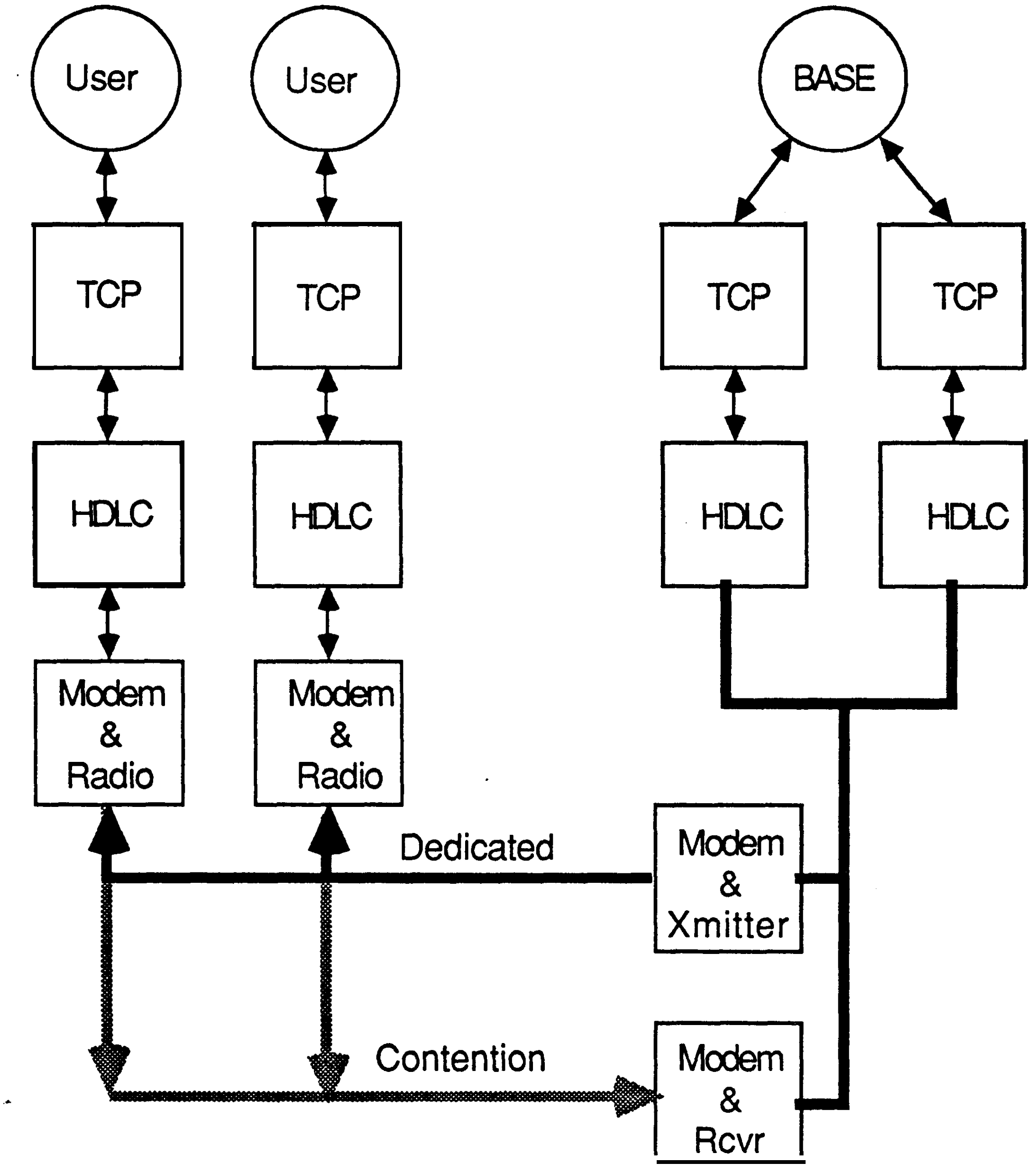
Passive layers

- Move and route data
- Control congestion by discarding data

User-Telnet Model



Simulated Architecture



Preliminary Findings

Assumption failures:

- HDLC Holds packets to ensure correct ordering
- TCP assumes delays primarily to net not to transmission speed (Delay is independent of packet size)
- HDLC assumes all packets are significant and must be successfully transmitted

Performance degradation:

- HDLC's attempts to order packets cause delays
- TCP may retransmit long packets assuming they were lost
- Significant increase in link traffic

Specialized Gat

A gateway that "understands" TC

- Combine packets
- Discard redundant data
- Resize packets without sim them
- Retransmit data over link at intervals than normal for hig
- Minimize effects of incorre other layers

Comparisons of Composite Simplex Algorithms

Irvin J. Lustig

For almost forty years, the simplex method has been the method of choice for solving linear programs. The method consists of first finding a feasible solution to the problem (Phase I), followed by finding the optimum (Phase II). Many algorithms have been proposed which try to combine the two phases into a single composite phase. This presentation will discuss the procedure of comparing the merits of some of these composite algorithms.

Computational aspects of the Weighted Objective, Self-Dual Parametric, and Markowitz Criteria algorithms are presented. No prior knowledge of the simplex method will be assumed. A framework for comparing the efficiencies of the different algorithms and their many possible variants will be presented. A large amount of computational experiments for each algorithm have been made. These are used to compare the algorithms in various ways. Although one would expect that one of the many possible composite algorithms would turn out to be a clear winner, computational experience suggests that no algorithm is best for all problems. Curiously enough, even a clear loser can be the best on some problem.

Linear Programs Input As:

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \text{ A } b \\ &\quad l \le x \le u \end{aligned}$$

where

A is I, 2, or =

This form is equivalent to:

$$\begin{aligned} &\min c^T x \\ &\text{s.t.} \quad [A\ I]x = 0. \\ &\quad l \le x \le u \end{aligned}$$

A basis is readily available.

Algorithms Considered are Limited To:

Composite Algorithms: Compromise Phase I./Phase II procedure

Simplex Method Variants: Choose a Pivot Sequence

Linear Time Ratio **Tests**

```
procedure generic_simplex;
    Comment Let $B$ be a basis for $[A\ I]$.
    while not finished do
       begin
       $\{s, q\} \leftarrow \{incoming\_column, exiting\_column\}$;
       Comment $q$ is in column $r$ of $B$.
       $pivot(B, s, q, r)$;
       $finished \leftarrow optimal(B)$ or $unbounded$ or $infeasible$;
       end
```

Algorithm Testing

- Non-Random Problems
- MINOS as core of program
  - Same Input and Output Routines
  - Time M5SOLV and subordinate routines
  - Algorithms are subroutines of M5SOLV
- MicroVax II - MicroVMS 4.3, VAX Fortran 4.1
- Run all tests in Batch
- Use Same Tolerances

Each Algorithm Run on 12 Problems.

For each problem, run

- Unscaled, Unscaled Crash Basis
- Scaled, Scaled Crash Basis
- Unscaled, Scaled Crash Basis
- Scaled, Unscaled Crash Basis

Problem Name	Problem Size		
	Rows	Cols	NonZeros
ADLITTLE	56	97	465
AFIRO	28	32	88
BANDM	306	472	2659
BEACONFD	174	262	3476
BRANDY	221	249	2150
CAPRI	272	353	1786
E226	226	282	3038
ETAMACRO	401	688	2489
ISRAEL	175	142	2358
SHARE1B	118	225	1182
SHARE2B	99	79	802
STAIR	357	467	3857

```
procedure primalsimplex;
  finished ← false;
  while not finished do
    begin
    optimal ← false;
    if needs-factoritation( B) then factorize(B);
    if infeasible-basis(B) then $\hat{c}$ ← d else $\hat{c}$ ← c;
    FindPI: Solve $\pi^T B = \hat{c}_{kb}$ for $\pi$;
    PriceOut: Find $s$ such that $x_s$ has minimum reduced cost $\bar{c}_s$;

    if $\bar{c}_s > 0$ then optimal ← true else
      begin
      Represent: Solve $Bp = [A\ I]_{\cdot s}$ for $p$;
      RatioTest: Find $r$ such that $x_{kb(r)}$ blocks the change in $x_s$;

      if $q = 0$ then unbounded ← true else
        begin
        q t k b r ;
        Update: Update value of $x$;

        pivot$(B, s, q, r)$;
        end;
      end;
    infeasible ← infeasible-basis(B) and optimal;
    finished ← optimal or unbounded or infeasible;
    end;
```

```
procedure weighted_objective;
  finished ← false;
  while not finished do
    begin
    optimal ← false;
    if needs-factorization(B) then factorize(B);
    if infeasible_basis(B) then $\hat{c}$ ← d + UC else $\hat{c}$ ← c;
    FindPI: solve $\pi^T B = \hat{c}_{kb}$ for $\pi$;
    PriceOut: Find s such that $x_s$ has minimum reduced cost $\bar{c}_s$;
    if $\bar{c}_s > 0$ then optimal ← true else
      begin
      Represent: Solve $Bp = [A\ I]_{\cdot s}$ for p;
      RatioTest: Find r such that $x_{kb(r)}$ blocks the change in $x_s$;
      if q = 0 then unbounded ← true else
        begin
        a ← kb(r):
        Update: Update value of x;
        pivot(B, s, q, r);
        end;
      end;
    infeasible ← infeasible_basis(B) and optimal;
    finished ← optimal or unbounded or infeasible;
    end;
```

```
procedure Markowitz_Criterion;
  finished ← false;
  while not finished do
    begin
    optimal ← false;
    if needs_factorization( B) then factorize(B);
    i fnfeasible_basis(B) then
      FindPI: Solve $\sigma^T B = d_{kb}$ for $\sigma$;
    else $\bar{d}$ ← 1;
    FindPI: Solve $\pi^T B = c_{kb}$ for $\pi$;
    PriceOut: Find s such that $x_s$ has minimum reduced cost $\bar{c}_s/\bar{d}_s$;
    if $\bar{c}_s > 0$ then optimal ← true else
      begin
      Represent: Solve $Bp = [A\ I]_{\cdot s}$ for p;
      RatioTest: Find r such that $x_{kb(r)}$ blocks the change in $x_s$;
      if q = 0 then unbounded ← true else
        begin
        q ← kb(r);
        Update: Update value of x;
        pivot(B, s, q, r);
        end;
      end;
    infeasible ← infeasible_basis( B) and optimal;
    finished ← optimal or unbounded or infeasible;
    end;
```

```
procedure $Self\_Dual(\bar{f}, \bar{d})$;
  finished ← false;
  while not finished do begin
    optimal ← false;
    if needs_factorization(B) then factorize(B);
    RatioTest: Find $s$ or $r$ such that $x_{kb(r)}$ or $\bar{c}_s$ blocks the change in $\theta$
    if $blocks(\bar{c}_s)$ then begin
       Represent: Solve $Bp = [A\ I]_{\cdot s}$ for $p$;
       RatioTest: Find $r$ such that $x_{kb(r)} + \theta \bar{f}_r$ blocks the change in $x_s$;
       if $r = 0$ then unbounded ← true else begin
          FindPI: Solve $\gamma^T B = e_r$ for $\gamma$;
          PriceOut: Compute $\bar{\gamma} = A_{r\cdot} - \gamma[A\ I]$;
          end
       end
    else if $blocks(x_{kb(r)})$ then begin
       FindPI: Solve $\gamma^T B = e_r$ for $\gamma$;
       PriceOut: Compute $\bar{\gamma} = A_{r\cdot} - \gamma[A\ I]$;
       RatioTest: Find $s$ such that $\bar{c}_s + \theta \bar{d}_s$ blocks the change in $\pi_r$;
       if $s = 0$ then infeasible ← true else
          Represent: Solve $Bp = [A\ I]_{\cdot s}$ for $p$;
       end
    finished ← optimal or unbounded or infeasible;
    if not finished then begin
       $q \leftarrow kb(r)$;
       Update: Update values of $x, \bar{c}, \bar{d}, \bar{f}$;
       pivot(B, s, q, r);
       end;
    end;
```

Scoring System:

For a given starting basis, give 00 points to Phase I/II algorithm. (PHU, PHSS)

For other algorithms, compute "factor" as compared to base method. (e.g., 200 = 2 times slower than best time)

Best score is least.

Compare for Iterations and CPU Time.

Total points across scaling, problems, and bases.

Total CPU Factors

Alg. Variant	Score	Total	
FDS	1566		
FD SS	1628	6356	Self-Dual (No Artificials)
FD J	1516		
FD JS	1647		
FN S	1555		
FN SS	1732	6410	Self-Dual (No Artificials) (Scale Free)
FN J	1490		
FN JS	1633		
M/ S	1881		
M/ SS	2095	7797	Markowitz
M/ U	1760		
MAUS	2061		
PHS	1097		
PHSS	1200	4725	Phase I/II
PHU	1200		
PHUS	1228		
SDS	1538		
SDSS	1618	6370	Self-Dual
SDU	1602		
SDUS	1612		
SNS	1594		
SNSS	1676	6503	Self-Dual (Scale Free)
SNU	1627		
SNUS	1606		
WTS	1131		
WTSS	1205	4762	Weighted
WTU	1192		
WTUS	1233		

Total Iteration Factors

Alg. Variant	Score	Total	
FDS	1227		
FDSS	1287	4970	**Self-Dual (No Artificials)**
FDU	1180		
FDUS	1276		
FNS	1241		
FNSS	1364	5100	**Self-Dual (No Artificials) (Scale-Free)**
FNU	1190		
FNUS	1305		
MAS	1637		
M/ SS	1798	6721	Markowitz
M/ U	1503		
M/ US	1782		
PH S	1111		
PH SS	1200	4758	Phase **I/II**
PH J	1200		
PH JS	1247		
SD S	1232		
SD SS	1281	5069	Self-Dual
SD J	1275		
SD JS	1282		
SN S	1300		
SN SS	1359	5270	Self-Dual (Scale Free)
SNU	1314		
SNUS	1296		
WTS	1104		
4644S	[redacted]72		**Weighted**
J	1160		
JS	1208		

Adaptive Numerical Methods for Weather Prediction

William (Bill) Skamarock

Weather prediction requires integrating equations which describe atmospheric motion over an extremely large spatial domain but phenomena of interest are localized and transitory. The important and difficult part of this problem is resolving and correctly predicting the movement of localized, fine structure, i.e., fronts, cyclones and other disturbances. Numerical procedures relying on fine meshes covering the entire domain are inefficient and in the case of weather prediction often prohibitively expensive. Adaptive solution procedures appear ideally suited for numerical weather prediction given the locality of weather phenomena. We have applied the adaptive grid technique developed by Berger and Oliger to the regional weather prediction problem.

The Berger and Oliger technique entails integrating the equations first on a coarse grid and then on finer grids which have been placed based on the estimated error in the coarse grid solution. By correctly coupling the integrations on the various grids, periodically re-estimating the error and recreating the finer grids, uniformily accurate solutions are economically produced.

Adaptively we consider this a two-dimensional problem because the atmosphere's vertical length scales are much smaller than its horizontal length scales. Thus, we have coupled the software used to implement this technique in two dimensions with a module which solves the equations descibing the three-dimensional time dependent motion of the atmosphere.

The adaptive model has been used to solve for the motion of a two-dimensional cyclone being advected by an easterly current and also to solve for a disturbance which develops as a result of perturbing an unstable jet, both of these in a east-west periodic channel. The simulation of the cyclone demonstrates the use of fine grids to track a disturbance. The largest errors in the numerical solution are associated with the cyclone. A single fine grid follows the high error, and hence the cyclone, across the region. The disturbance which develops as a result of perturbing an unstable jet is a classic weather pattern observed in the mid-latitudes. By adaptively solving for the development of this disturbance we show that the adaptive model is capable of solving for realistic, three-dimensional atmospheric flows. This simulation also demonstrates the feasibility of using multiple, rotated, overlapping fine grids.

The success of these simulations strongly supports the concept that adaptive refinement should occur only where dictated by the error in the numerical solution and that this has been sufficient to improve the accuracy and overall resolution of the entire solution. Using this concept has produced the first adaptive solution of atmospheric flows and the first detailed, quantitative results concerning error in the numerical solutions.

ADAPTIVE NUMERICAL METHODS
FOR WEATHER PREDICTION

Bill Skamarock
Joseph Oliger
Robert Street

Support:
Office of Naval Research
Naval Environmental Prediction Research Facility

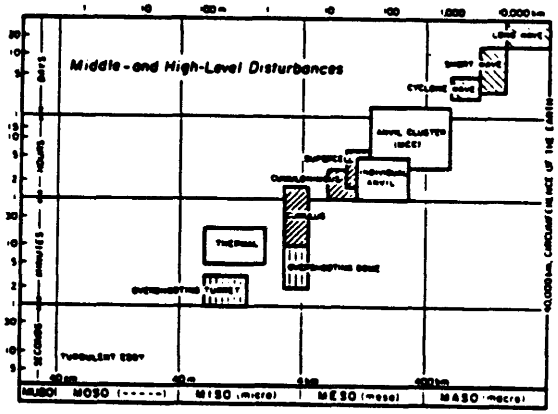
1) What is the problem?

2) Adaptive weather model.

3) Adaptive simulations of
- cyclone
- jet stream disturbance

1) The Problem:

predicting weather?



- Many *scales of motion* need predicting

- Present methods integrate equations describing motion of the atmosphere

  Initial Value Boundary Problem

- A major problem: *resolution*

Hurricane Anita

Radar reflectivity

1 September 1977

Domain:
300 × 400 km.

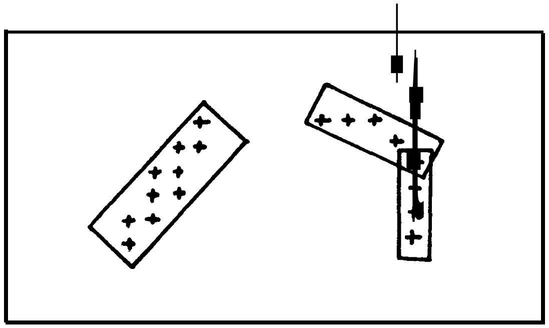
2) Adaptive weather model
   - disturbances are localized , hence *local refinement* techniques are ideal.

Adaptive weather model couples:
- software system of Berger & Oliger
- solver for primitive equations of meteorology.

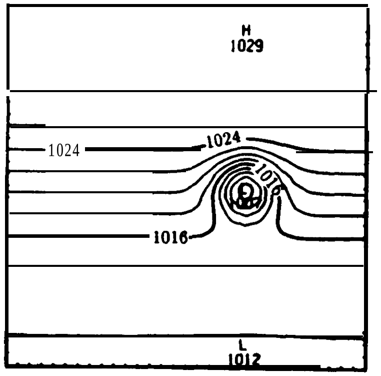
Berger and Oliger Adaptive Grid Solution Technique

Start with solution on coarse grid at time t.

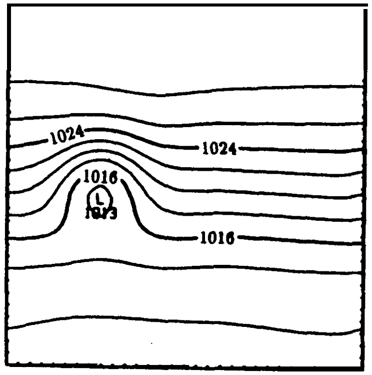
1) Estimate error in numerical solution at gridpoints.
2) Flag points where the error is too large.



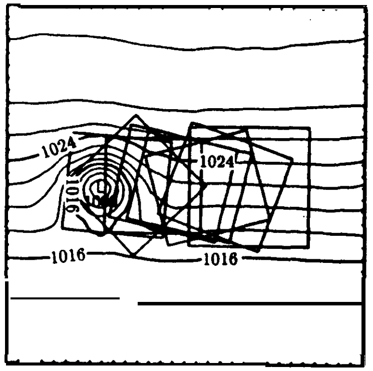
3) Fit rotated rectangles around flagged points, these are the fine grids.
4) Initialize solution on the fine grids with coarse grid solution and integrate to time t + dt.
5) *Update* coarse grid solution with fine grid solutions.
6) Repeat process.



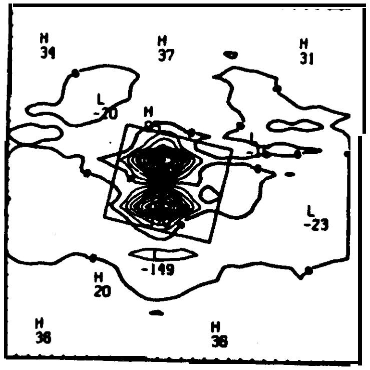
Surface pressure (millibars) at t=0.



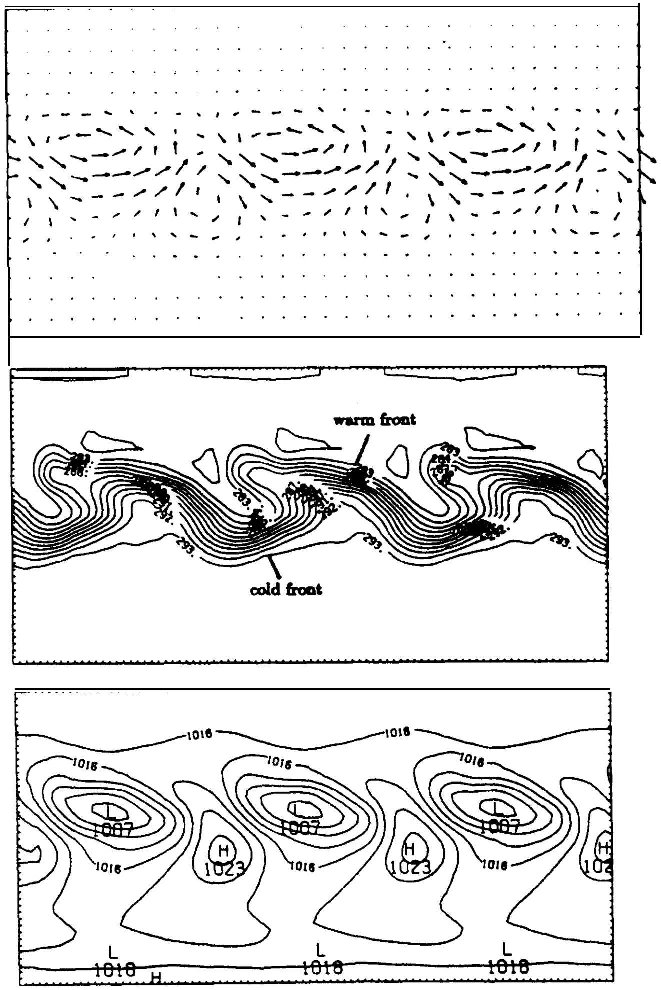
Surface pressure after 3 days.
Coarse grid run.



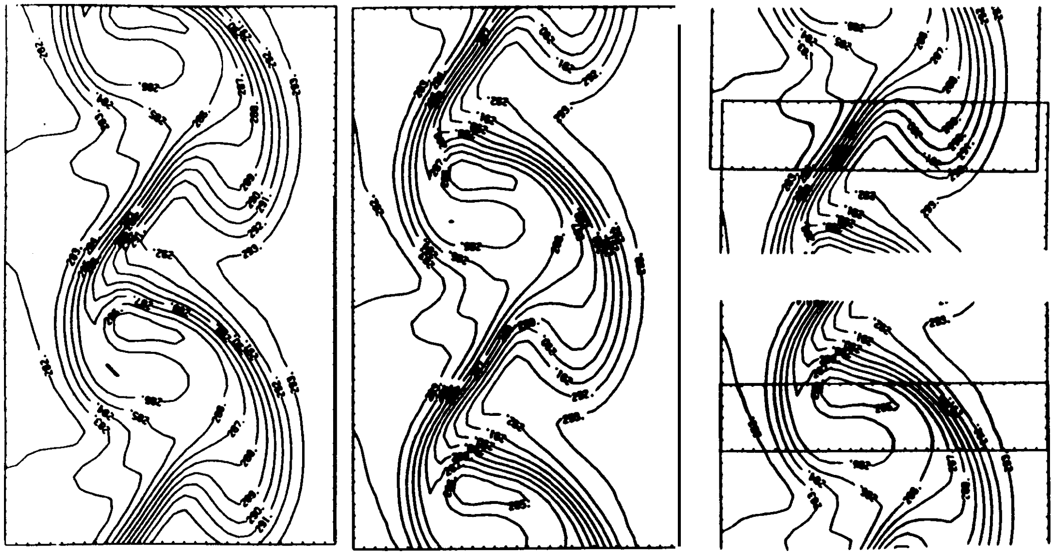
Surface pressure after 3 days.
Adaptive grid run.



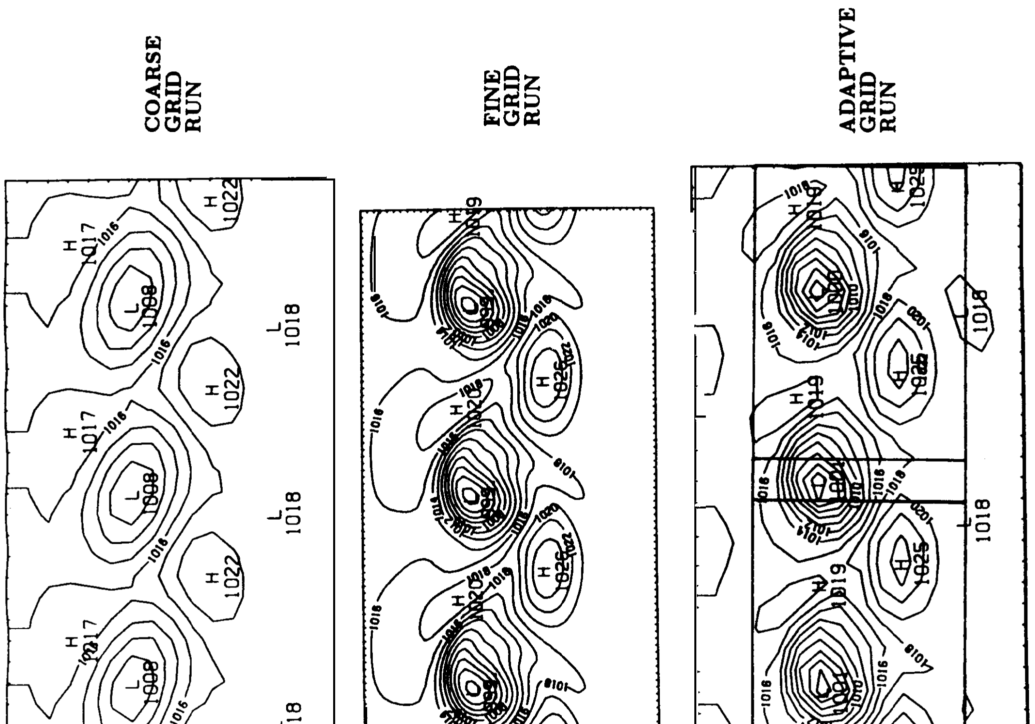
Error estimate, *u* velocity.
(dimensionless $\times 10^4$)



Wind vectors at ≈ 900mb., temperature (K) at the same level and surface pressure (mb) at t=0. for unstable jet simulation.



Temperature at ≈ 900mb for fine grids shown in previous plot.



Surface pressure in millibars at *T* = 72. hours.

The Proposed Center for the Study of Human-Computer Interaction

Keith A. Lantz

Historically, advances in many areas of computer research have been hindered by a lack of interdisciplinary research. This problem is particularly acute in the area of human-computer interface design, where the relevant disciplines include many outside those of computer science and electrical engineering -- including graphics design, psychology, and communications. All too much of the work reported in the literature has been done in relative isolation, with a consequent tendency to be relevant only to the specific domains in which it was performed.

Therefore, we propose to address human-computer interaction in the context of an interdisciplinary ''center'' (or ''program'') whose fundamental goal is to ascertain basic principles of human-computer interface design, and the environments to which those principles do or do not apply. The success of this endeavor will depend on the participation of application domain experts, graphics designers, psychologists, and other specialists -- in addition to computer scientists and electrical engineers. This approach stands in stark contrast to one whose primary goal is to develop a specific application, which may, at some point, require a user interface. On the other hand, developing basic principles to begin with requires experimentation in a variety of environments, which should in fact yield a number of interesting applications!

The center is intended to encourage investigation of any issue that impacts on the design, implementation, or evaluation of human-computer interfaces. Representative topics include the effective use of multiple communications media, computer-supported collaborative work (including real-time multimedia conferencing), software architecture, dialogue specification languages, global data interchange facilities, cognitive modeling (of the user), and human factors experiments (in support of all other areas).

Fitting Shapes to Points

Vaughan Pratt

The emphasis in the literature on least-squares fitting of curves and surfaces to a set of points has been on parametrically presented polynomial curves and surfaces. The few papers that do treat algebraically presented figures treat only curves in the plane. We give several algorithms for fitting an algebraic shape (line, circle, conic, cubic, plane, sphere, quadric, etc.) to a set of points, namely exact fit, simple fit, pseudoeuclidean fit, spherical fit, and blend fit, all but the last two of which are applicable to any algebraic shape.

Exact fit generalizes to all algebraic shapes the well-known determinant method for exact planar fit. Simple fit is an easily described, fast, but nonrobust reduction of the general overconstrained case to the exact case. Pseudoeuclidean fit is a slow but relatively robust method based on a good surrogate for the Euclidean distance metric. Spherical fit takes advantage of a special property of spheres to permit fast robust fitting of circles and spheres; no prior fast circle fitters have been robust, and there have been no previous sphere fitters. Blend fit finds the best fit of a Middleditch-Sears blending surface to a set of points while blending a set of base surfaces, via a nonpolynomial generalization of planar fit.

These methods all require $(am+bn)n^2$ operations for fitting a surface of order n to m points, with $a=2$ and $b=1/3$ typically, except for spherical fit where b is larger, and pseudoeuclidean fit, a slow iterative method. All save simple fit achieve a robustness previously attained by comparably fast algorithms only for planes. All admit incremental batched addition and deletion of points at cost an^2 per point and (except for pseudoeuclidean fit) bn^3 per batch.

1 Problem Domain

- Fit a given shape to m points
- Criterion: minimize total distance D of points to surface
- $D^2 = \Sigma_i d^2(p_i, S)$
- Whence we want least-squares true fit

2 State of the Art

- Least-squares fitting mainly done for parametric surfaces
- A little done for algebraic curves
- Nothing for nonplanar surfaces (spheres, quadrics,. . .)

3 Algebraic Shapes

- Algebraic shape: a class of surfaces of form

$$a_1 q_1(x_1, \ldots, x_d) + \ldots + a_n q_n(x_1, \ldots, x_d) = 0$$

E.g. circle, conic, cubic, plane, sphere, quadric, etc.

- Specify in terms of a basis:

Line: x, y, 1

Circle: $x^2 + y^2$, x, y, 1

Conic: x^2, xy, y^2, x, y, 1

Sphere: $x^2 + y^2 + z^2$, x, y, z, 1

Quadric: x^2, xy, y^2, yz, z^2, zx, x, y, z, 1

4 Exact Fit

- Circle case gives the paradigm:

$$\begin{vmatrix} 1 & x_1 & y_1 & x_1^2 + y_1^2 \\ 1 & x_2 & y_2 & x_2^2 + y_2^2 \\ 1 & x_3 & y_3 & x_3^2 + y_3^2 \\ 1 & x & Y & x^2 + y^2 \end{vmatrix} = 0$$

5 Simple Fit

$$A = \begin{vmatrix} 1 & x_1 & y_1 & x_1^2 + y_1^2 \\ 1 & x_2 & Y2 & x_2^2 + y_2^2 \\ 1 & x_3 & y_3 & x_3^2 + y_3^2 \\ 1 & x_4 & y_4 & x_4^2 + y_4^2 \\ 1 & x_5 & y_5 & x_5^2 + y_5^2 \end{vmatrix}$$

- 1. Perform Cholesky decomposition of $A'A$

  Find n x n upper triangular U s.t. $U'U = A'A$.

  Of size independent of m

- 2. Replace last row of U by last row of exact fit matrix

  namely the basis row

  $1 \quad x \quad y \quad x^2 \; + \; y^2$

  Call result U^+

- Proceed as for exact fit: $|U^+| = 0$
- Automatically normalizes last coefficient to 1

6 Variations

- $A'A \rightarrow U'DU$ (squarerootless Cholesky)
- Permute n dimensions of $A'A$ to order diag(U).

  Purpose: removes singularities
- Compute $A'A$ incrementally as

$$\prod_{\text{row } Z \in A} Z'Z$$

Purpose: Exploration

- Costs: n^2 per point for $Z'Z$, total mn^2

  $n^3/6$ for Cholesky decomposition

  n^2 to compute determinant

7 Spherical Fit

- Circle: $A(x^2 + y^2) + Dx + Ey + F = 0$

  Prior methods: None robust

  Best previous: Bookstein, $A = 1$

  Problem with Bookstein: lines are singularities
- Our circle invariant: $D^2 + E^2 - 4AF$
- Sphere: $A(x^2 + y^2 + z^2) + Dx + Ey + Fz + G = 0$

  Prior methods: None at all
- Our sphere invariant: $D^2 + E^2 + F^2 - 4AG$
- Generalizes immediately to higher dimensions

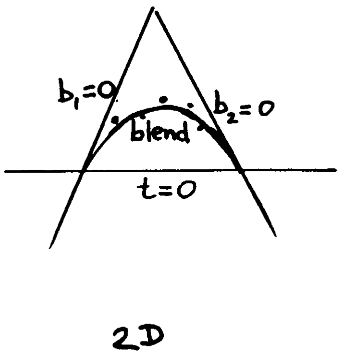
8 Middleditch-Sears Blending Surfaces

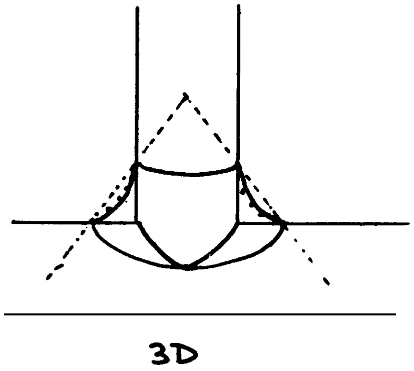
- Concept: Surface smoothly connecting adjoining surfaces
- Applications: chamfers, fillets, sculpture
- Components:

  Base surfaces $b_i = 0$

  Truncating surface t = □
- Blending surface: $At^m + B \prod_i b_i = 0$

Effect:





9 Fitting a Blending Surface

- Idea: Replace $At^m + B \prod_i b_i = 0$

  By $At + B \sqrt[m]{\prod_i b_i} = 0$
- This gives same class of surfaces
- But now the linear combinations in t are “exposed”
- This allows the blend to be fitted as a **line**

Constraints, Planning, and Learning: How to Analyze and Synthesize Hardware

Daniel Weise

Special purpose hardware will soon prove to be faster and cheaper than general purpose hardware for many tasks. I am researching the problem of automatically constructing hardware from behavioral descriptions. Among the problems are choosing an architecture that gives the best speed for a given cost, implementing that architecture in some technology, proving the design is correct (verification), and avoiding rewriting the system every time the technology changes. The tools and techniques employed come from compiler theory, verification theory, and artificial intelligence. Maintaining and relating information from the different design levels requires a constraint system. Learning is required for a robust, technology independent system.

Goal: Automatic Hardware Design

- Two Subproblems: Choosing an architecture, and implementing the architecture.
- Eliminating Design Methodolgies: The interplay between abstraction and constraints.
- Technology Independence: Domain exploration and learning.

Two Subproblems

- Choosing an architecture.
  - Input: Functional Language
  - Output: RTL (functional units plus inteconnect).
  - Employs cost model. Limited hardware budget.
  - Gain: dedicated communication, parallelism.
- Implementing the architecture.
- Strongly coupled problems: makes the problem hard. Resource constraints at implementation level affect choices at architecture level.

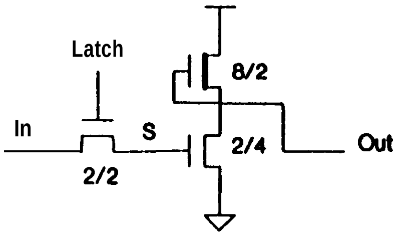
Synthesis Without Methodology

- Design methodologies trade design complexity for bigger and slower designs.
- Methodology ensures abstractions in abstraction space correspond to abstraction in physical space.
- We must articulate the constraints to be observed when composing a particular device with other devices.

Structure, Function, and Constraints

- Constraints ensure a design (or plan) will exhibit its intended abstract behavior.
- For some domains (such as hardware), constraints can be automatically derived.
- Constraints can be used for planning (synthesis) and verification (analysis).

Example: Inverting Latch



```
(df((INVERTING-LATCH s) latch in)
  (stream-cons
      ; this is the output
      (if latch (not in) (not s))
      ; this is the next state
      (INVERTING-LATCH(if latch in s))))
```

Constraints:

- latch ⇒ overpowers(in, s)
- latch ⇒ fallsfirst(latch,in)
- control(latch)
- high-voltage(latch)

Constraints and Planning

- MOLGEN style planning.
- Each decision posts constraints.
- Other decisions must obey constraints (or undo earlier decisions).
- “Backtracking” occurs when a design is overconstrained.

Verification

These ideas are in Silica Pithecus, a MOS circuit verifier

- Input is schematic, logical constraints, and intended digital behavior.
- Output is yes or no plus constraints.
- Formal Proof
- Accurate circuit model:
  - Transistors are bidirectional and have mutiple operating regions
  - Nodes are modeled as capacitors.
- Bugs Caught: Charge Sharing, Races, Hazards, Threshold Drops, Logic Problems
- Works quickly, interactively, and incrementally.

Technology Independence

- TTL, ECL, different flavors of MOS (2 level metal, 2 level poly, etc.)
- Expert systems are too fragile.
- Prime examples: Weaver, DAA
- Will use a graduated training set.
- Domain exploration for learning new idioms and methods.
- Hardware synthesis is an excellent domain for method acquisition.

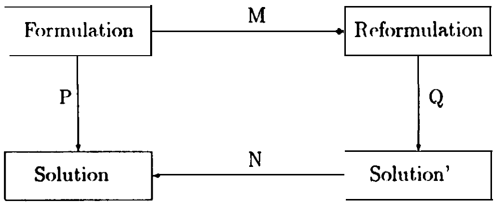
A Theory of Justified Reformulations

Devika Subramanian

Present day expert systems are limited forever by the conceptualizations of the world provided to them by their designers. This thesis research explores issues in the construction of adaptive systems that can automatically extend and modify their conceptualizations to achieve computational efficiency or descriptional elegance.

In this talk, we examine a special case of this problem: reconceptualizing or reformulating a knowledge base in terms of new objects, functions and relations, in order to make the computation of a pre-specified class of queries more efficient, for a given problem solver. Automatic reformulation is not possible until a reformulator can *justify* a shift in conceptualization. We thus argue for a meta-theoretical analysis of this problem and present a class of justifications called irrelevance justifications. We formalize the notion of irrelevance and propose the use of the *irrelevance principle* for generating abstraction reformulations.

A Ball and Stick Model



P,Q : problem solving methods
M,N : formulation to formulation maps

The problem of reformulation is to uncover principles of shifting from one conceptualization of the world to another in a domain-independent way.

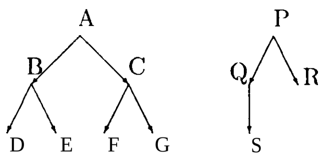
- Goal-directed Reformulations
- Correct Reformulations
- Good Reformulations

The Importance of Reformulation

- Practical
  - Problem-Solving Technique
  - Construction of Adaptive Systems
- Theoretical
  - Role of Representation in Problem Solving
  - Designing Good Representations, Automated Vocabulary Engineering

The Founding Fathers Example

I:



Father (x,y) ⟹ *Ancestor* (x,y)

Ancestor (x,z) A *Ancestor* (z,y) ⟹ *Ancestor* (x,y)

Ancestor (z,x) A *Ancestor* (z,y) ⟹ *SameFamily* (x,

The Reformulation Problem

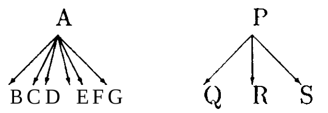
Given :

- Formulation F1 above.
- Problem Solver is a Depth First Backward Chainer.

Find a reformulation F2 such that:

- F2 solves all the *SameFamily* queries that F1 does.
- F2 consumes no more space than F1.
- Answers to *SameFamily* queries in F2 are found in constant time.

A Reformulation F2 that is correct and good

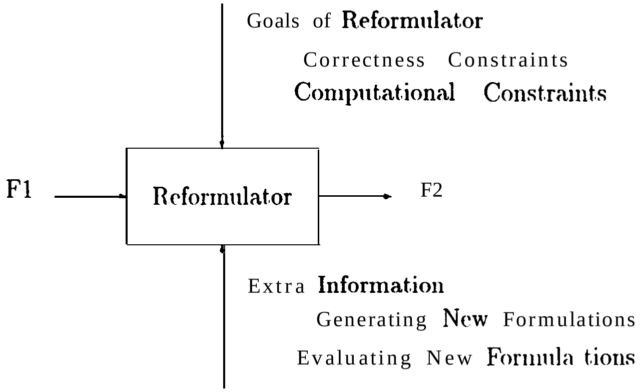


FoundingFather (z,x) A *FoundingFather* (z,y)
⟹ *SameFamily* (x,y)

A Reformulation F2 that is correct but not good

SameFamily(A,A)	*SameFamily*(P,P)
•	•
•	•
•	•
SameFamily(G,G)	*SameFamily*(S,S)
49 facts of this form	16 facts of this form

On the Path to Generation



Goals of Reformulator

Correctness Constraints

Computational Constraints

Extra Information

Generating New Formulations

Evaluating New Formula tions

Problems

- What knowledge does the reformulator need?
- What are the reasoning requirements of a reformulator?

Approach

- Need to *explain* reformulations before we can automatically generate them.
- Justification-based reformulations.

Why is F2 a Reformulation of F1?

Correctness Proof

Theorem: F1 *solves* ***SameFamily*** (x,y) if and only if F2 solves ***SameFamily*** (x,y).

Proof: Notice that ***FoundingFather*** ⊂ *Ancestor*. The conceptual shift is affected by throwing away the ***Father*** relation and parts of the ***Ancestor*** relation that were computationally irrelevant to the *SameFamily* query.

Goodness Proof

Theorem: F2 is more efficient than F1 for the *SameFamily* query for a Depth First. Backward Chainer.

Proof:

- Space Requirements
- Time Requirements:

  In F1: Depth of family tree

  In F2: Constant time.

Irrelevance

Fact f is Irrelevant, to Fact g modulo M written as: I(f,g,M)

- M solves for g.
- There is a weakening M' of M that establishes g while being non-commit, tal on f.

Logical Irrelevance and Computational Irrelevance

Example

Ancestor (x,y) *A*

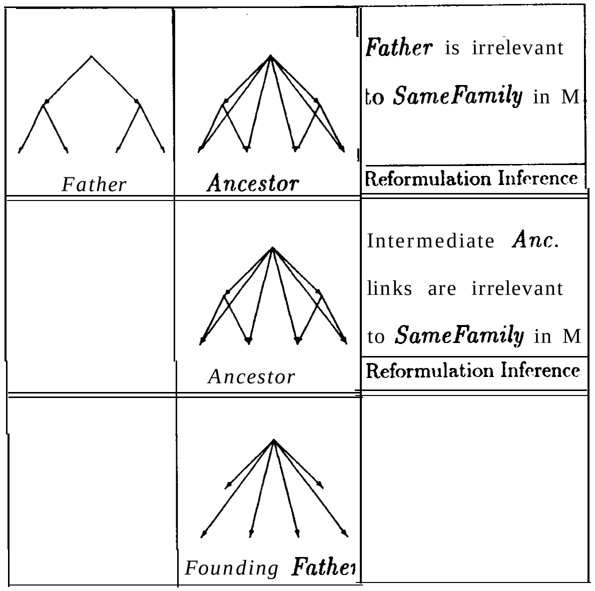
Derives-Only(*Father* (x,y), ***Ancestor*** (x,y), M)

⟹ I(***Father*** (x,y), ***SameFamily*** (m,n), M)

If all you can derive from *Father* (x,y) is the corresponding *Ancestor* (x,y) fact,,

Then if *Ancestor* (x,y) is explicitly in M, *Father* (x,y) is redundant wi th respect to the query.

The Relevance of Irrelevance



Father is irrelevant to ***SameFamily*** in M

Reformulation Inference

Intermediate ***Anc.*** links are irrelevant to ***SameFamily*** in M

Reformulation Inference

The Derivation of F2 from F1

The Irrelevance Principle

Minimize a problem formulation by throwing away all facts and distinctions that are irrelevant, or redundant with respect to the computation being preserved.

The use of this principle is justified by appealing to properties of the problem solver: e.g. it works better with less clutter.

This principle captures some abstraction reformulations.

Conclusions and Future Work

- Presented a justification-based approach to reformulation.
- Introduced reasoning about irrelevance as a meta-theoretic reasoning tool for automatic optimisation of formulations.
- Irrelevance Reasoning **has** applications in Database View Design and Digital Circuit Diagnosis.
- Reasoning *about* computational structures.

Distributing Backward-Chaining Deductions to Multiple Processors

Vineet Singh

Effective allocation of resources in a multiprocessor can make a dramatic improvement in the speedup obtained. Optimal allocation is ruled out because even simplistic computation and multiprocessor models make the problem NP-complete or worse. I present an effective, sub-optimal allocation strategy for backward-chaining deduction on a practical multiprocessor.

The target class of multiprocessors for this dissertation satisfies the following properties: (1) there are a finite number of processors; (2) each processor has a finite amount of local memory; (3) there is no global memory; (4) processors are connected with some scaleable interconnection topology; (5) processors can communicate only by sending messages to each other; (6) message delay is some non-zero monotonic function of the amount of data in the message and the distance between source and destination; (7) each processor can perform backward-chaining deductions based on the subset of the logic program that it contains.

Sequential execution models for backward-chaining deduction, used in sequential logic programming languages, are not suitable for multiprocessors. I present a parallel execution model called *PM*. For the same multiprocessor class, *PM* can exploit the most parallelism among execution models that use control based on dataflow principles.

The proposed allocation strategy needs some restrictions that *PM* does not require. First, the type of backward-chaining deduction is restricted. In particular, no recursive clauses are allowed, unit clauses must be ground, and certain probabilistic uniformity and independence assumptions must apply. Second, the type of allocation is restricted to compile-time allocation. Third, a partitioning of the database is assumed to be given.

Allocation is based on a formally defined cost function. This cost function quantifies intuitive notions of undesirable allocations. I will sketch out algorithms for efficient computation and recomputation of this cost function.

The allocator consists of an initial allocation phase followed by a local optimization phase. In the initial allocation phase, database partitions are allocated to processors one at a time using a ***greedy*** algorithm. The local optimization phase consists of a sequence of cost-reducing reallocations of partitions to neighboring processors.

P M has been implemented on an instrumented, event-driven simulator of the FAIM-1 multiprocessor. Implementation of the allocation strategy is under way. Speedups obtained by the allocator will be contrasted with an unreachable upper bound and with speedups obtained using carefully done human allocations.

Goals

Speedup

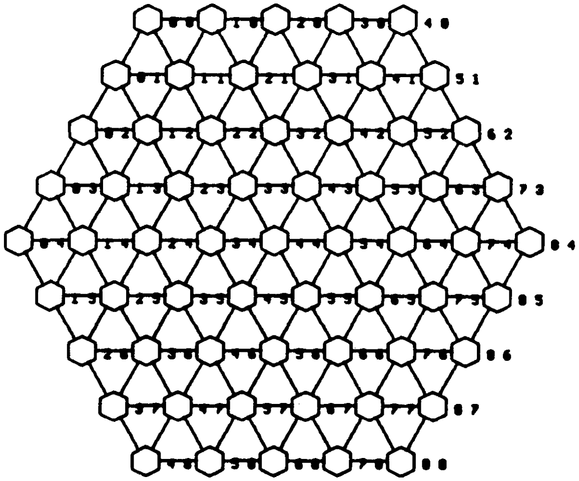
Parallel execution model

Resource allocation

- Processors
- Memory
- Communication bandwidth

Multiprocessor

- Finite number of processors
- Finite local memory
- Interconnection topology
- Message-passing
- Non-zero message delays
- Each processor can do backward-chaining



PM: a parallel execution model

Horn clauses

Dataflow-like control

Parallelism exploited

- Or-parallelism
- And-parallelism
- Pipelining

Allocator I/O

Input

- Logic program
- Partitioning of logic program
- Bag of goal patterns
- Domain information
- Set of processors
- Memory per processor
- Communication delay function
- Processing speed

output

- Allocation:
  Many-to-one mapping from partitions to processors

Restrictions

- No global minimization
- Compile-time allocation
- No recursive clauses
- Ground unit clauses
- No variable bindings to functionals
- Uniform distributions
- Independent terms and literals

Allocation Algorithms

Greedy allocation

Local search

Cost Function

Cost = Communication cost + Processor multiplexing cost

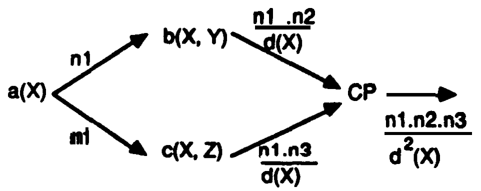
Efficient to compute and recompute

Communication Cost

$$CC = \sum_{\forall i} delay(message_i)$$

Before allocation time

- **Data communicated**
- **Number of messages**



nl a(uc)
n2 b(uc1, uc2)
n3 c(uc1, uc2)

Duplicate answers allowed

Complexity

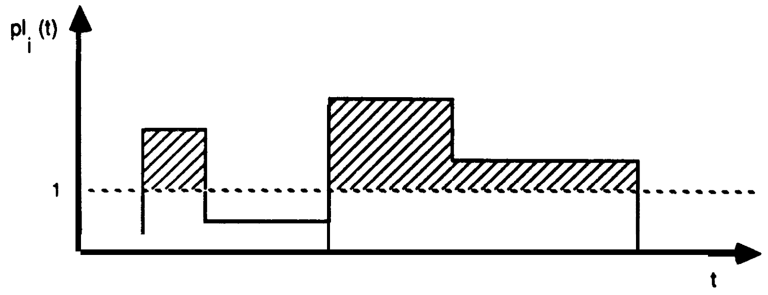
Processor Multiplexing Cost

Assume

- Zero communication delays
- Infinite number of virtual processors at each actual processor

$$PMC = \sum_{\forall i} \int_0^\infty epl_i(t)dt$$

$epl_i(t)$ = ***excess processor load*** of ith processor at time t



PMC= shaded area

Complexity

References

Singh, V. ***Distributing Backward-Chaining Deductions to Multiple Processors.*** PhD thesis, Stanford University, June 1986 (expected).

Singh, V. and Genesereth, M.R. ***PM*** A Parallel Execution Model for Backward-Chaining Deductions. ***Future Computing Systems,*** Oxford University Press, 1(3), 1987 (to appear). Also available as KSL-85-18, Knowledge Systems Laboratory, Computer Science Department, Stanford University.

Singh, V. and Genesereth, M.R. A Variable Supply Model for Distributing Deductions. In ***Proceedings of IJCAI-85,*** Morgan Kaufmann Publishers Inc., August 1985.

On Default Sub-Theories

Benjamin Grosof

Non-monotonicity is ubiquitous when programs are viewed declaratively, e.g. especially in AI. Prioritized defaults are a natural and powerful way to express preferred beliefs and non-monotonic reasoning. However, in general, non-monotonic inference is holistic, i.e. totally context-dependent: a computational nightmare. We invent the notions of default module and default sub-theory, relative to a logic of prioritized defaults based on circumscription. These enable us to characterize special-case, hierarchical decompositions of (prioritized-) default theories which are advantageous inferentially in two ways: partial monotonicities and modularities emerge; and primitive default sub-theories can be implemented more straightforwardly. Conversely, the frank embrace of preferences and context-dependencies among beliefs facilitates and clarifies the specification of a large non-monotonic "knowledge" base by a program designer. Treating default theories as strongly-decomposed into simpler-form default sub-theories thus provides a step toward practically automating a wide variety of applications of non-monotonic reasoning.

The Ubiquity of Non-Monotonicity

Logical Non-Monotonicity (NM'icity) $\triangleq$

as axioms accumulate, conclusions may be retracted.

Major Programming Paradigms:

- "logic" programming: least-fixed-point semantics
- object-oriented, e.g. frames: inheritance with exceptions

Representational Conventions, e.g.:

- Closed World Assumption
- Frame Assumptions

Empiricism and Action: acting in the presence of

- incomplete knowledge
- changing environment and knowledge

Defaults and Preferred Beliefs

Informally, a default " :> $D(\ x)$" means

"if D(x) is consistent, then conclude D(x) ".

Perspective: *preferences* among beliefs resolve ***conflicts***

- "hard" preferred to any default
- some defaults are preferred over other defaults

The generality and natural-ness of this formulation of NMR is important especially when specifying an agent with ***working hypotheses.***

The Challenge of Inference

Violated: the ability to keep intermediate results.

NM'icity is non-modularity, i.e. context-dependence.

General ***case: caveat emptor***

holism: a computational nightmare

Strategy:

- decompose ⟶ default "sub-theories"
- find relatively ***easy special cases***
  - with ***partial*** monotonicities and modularities

Circumscriptive Default Logic

Can be viewed as a meta-language that
"Compiles" to ***circumscription:*** via ***reformulation theorems.***

- ***base*** axioms

  $\boldsymbol{bird(Tweety)}$
  $ostrich(\boldsymbol{Joe})$
  $\forall x.ostrich(x) \Rightarrow bird(x)$

- ***default*** axioms

  $(D_2) \quad : > \; bird(x) \Rightarrow flies(x)$
  $(D_1) : \quad > \quad ostrich(x) \Rightarrow \neg flies(x)$

- ***prioritization*** axioms

  $D_1 \succ D_2$

- ***fixture*** axioms

  $\mathcal{FIX}(bird(x))$

Default Modules and Sub-Theories

Defn #1: ***Module*** M = a set of CDL axioms

Default Theory T = NM closure $\mathbf{C}(M)$

Defn #1.1: ***Sub-Module*** = subset of module

$$M = \wedge_i \ SM_i$$

PROBLEM: (Sub-) modules' meanings are context-*de*pendent.

$$T \neq \mathrm{Th} \ (\wedge_i \ \mathrm{C}(SM_i))$$

Defn #2: a module is a prioritization of sub-modules

$$M = (\overline{SM} : R)$$

...then use ***composition methods***

(De)Composing

To form sub-modules:

- Retain all base and fixture axioms.
- ***Partition defaults*** into groups.
- Each group keeps its *intra*-group priorities.
- *Inter*-group priorities are reformulated as priorities between ***modules***, rather than between defaults. Together with ***conflicts*** between the groups, they determine how to:
- Import and export "***declarations***" of defaults between modules. Imported declarations become fixture axioms.
- (Optionally, ***omit*** axioms as possible.)

Methods to compose sub-theories:

- ***Parallel:*** conjoin several NM closures
- ***Sequenced:*** conjoin previous' NM closure to current base, then do NM closure

An Example

$$M \quad = \quad (SM_1 \succ SM_2)$$

SM$_1$	SM$_2$
$\mathcal{FIX}(bird(x))$	$\mathcal{FIX}(bird(x))$
$bird(\textbf{Tweety})$	$bird(\textbf{Tweet y})$
$ostrich(\textbf{Joe})$	$ostrich(\textbf{Joe})$
$\forall x.ostrich(x)\Rightarrow bird(x)$	$\forall x.ostrich(x)\Rightarrow bird(x)$
$:> \ \textbf{ostrich(x)+-flies(x)}$	$:> \ \textbf{bird(x)+ f lies(x)}$

$$\boxed{+ \quad \mathcal{FIX}(ostrich(x)\Rightarrow\neg flies(x))}$$

$$\textbf{PAR} \ :: \ T \ = \ \mathrm{Th}(\mathrm{C}(SM_1) \wedge \mathrm{C}(SM_2^+))$$

$$\textbf{S\&Q} \ :: \ T \ = \ \mathrm{C}(\mathrm{C}(SM_1) \wedge SM_2)$$

Why Care?

By syntactically characterizing modules/theories with decompositions

- that are simple, e.g. few declarations are exchanged
- whose primitive modules/theories are implementable via extant NM mechanisms, e.g. Negation as Failure

We can

- design relatively efficient, correct NM theorem-provers which automatically decompose, and exploit modularities.
- implement richer classes
- understand trade-offs between expressive power and efficiency.
- facilitate and clarify the specification of large NM "KB"s .

Potential Applications

- inductive theory formation
  - rule and concept learning
  - diagnosis
  - sensory interpretation
    * vision
    * motion
    * signal analysis
    * speech recognition
- causality, action, and change
- heuristics and flexible policies
- retractible decisions: planning, design
- "Bayesian" and "evidential" reasoning
- "KB"/DB updating
- integrate default inheritance with rest of reasoning

Conclusion

The logical notion of modular sub-theories in (prioritized-) default theories provides a link from highly mathematical research in general-purpose non-monotonic logics to practicality, w.r.t.:

- inferential feasibility and efficiency
- expressive naturalness and structure

The TABLOG Programming Language

Eric Muller

TABLOG [1] is a logic-programming language based on quantifier-free first-order logic that includes all the standard logical connectives, such as equality, negation and equivalence. Programs are nonclausal: they do not need to be in Horn clause form or any other normal form. They can compute either functions (as in LISP) or relations (as in PROLOG).

Two deduction rules are used for the execution of programs: *nonclausal resolution* (which corresponds to a case analysis) and *equality replacement* (which corresponds to replacement of equals by equals). PROLOG programs are typically provided with cut annotations to allow their efficient execution. Such annotations are not necessary in TABLOG, since implicit cuts are introduced during the computation. Lazy evaluation provides an elegant way to manipulate infinite data structures.

A powerful mechanism has been introduced supporting a hierarchical structure for TABLOG programs and permitting the reuse of code. A compiler for a virtual TABLOG machine, written in TABLOG itself, is under development. It is expected that TABLOG programs will be executed as efficiently as their PROLOG counterparts, despite the additional features available to the programmer.

[1] Y. Malachi, Z. Manna, and R. Waldinger, ''TABLOG -- A new approach to logic programming'' in D. Degroot and G. Lindstrom, editors, *Logic Programming: Relations, Functions, and Equations,* Prentice-Hall, 1985.

TABLOG

A logic-programming language:

- based on quantifier-free first-order logic
- includes all connectives (equality, negation, equivalence, . . .)
- programs may be relations or functions
- no cut annotation required

LOGIC PROGRAMMING

	Logical System	+	Proof Rules
PROLOG:	Horn Clauses	+	SLD-Resolution
TABLOG:	quantifier-free first-order logic	+	nonclausal resolution & equality rule

- model world as theory
- express problem as goal $P(\overline{x})$

- prove sentence $\exists\overline{x}$ P(z)
- extract answer from the proof

THE PROOF RULES

- nonclausal resolution rule

assertions	*goals*
$\mathcal{F}[\mathcal{P}]$	
	$\mathcal{G}[\mathcal{P}']$
	$\neg\mathcal{F}[\mathit{false}]$ $\wedge$ $\mathcal{G}[\mathit{true}]$

where $\theta P = \theta P'$

- equality rule

assertions	*goals*
$\mathcal{F}[s = t]$	
	$\mathcal{G}[s']$
	$\neg\mathcal{F}[\mathit{false}]$ $\wedge$ $\mathcal{G}[t]$

where $\theta s = \theta s'$

A SIMPLE EXAMPLE

ASSERTIONS

AI: alpinist(tony) A alpinist(mike);

A2: alpinist(x) → skier(x) v climber(x)

A3: ¬likes(x,snow) → ¬skier(x);

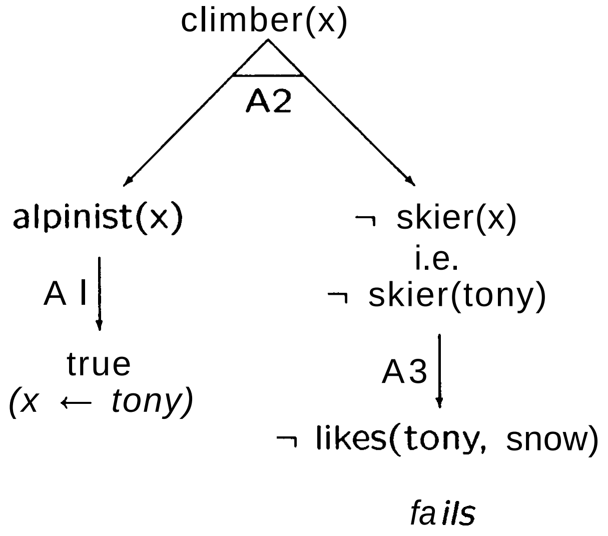
A4: likes(x,rain) → ¬climber(x);

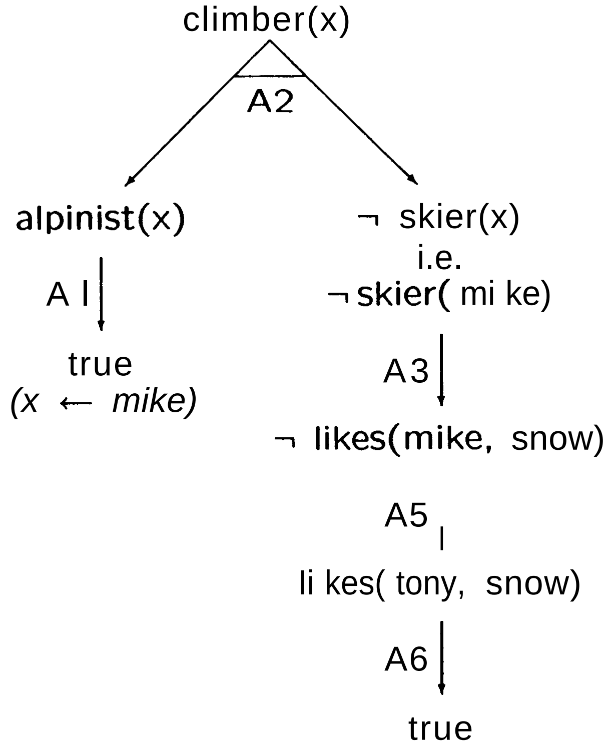
A5: likes(mike,x) ↔ ¬likes(tony,x);

A6: likes(tony,rain) A likes(tony,snow);

GOAL

climber(x);





solution : x ← mike

COMPUTE FIRST n PRIME
~UMᵃ ≤RS

```
ASSERTI○NS
primes(n) = truncate(n, sift(integers(2)));

truncate(0, u) = [];
truncate(succ(n), i•u) = i•truncate(n, u);

integers(i) = i•integers(succ(i));

sift(i•u) = i•sift(filter(i  u));

filter(p, n•u) = if p divides n
                 then filter(p, u)
                 else n•filter(p, ω⁼;

GOAL x = primes (50);

x = [2, 3, 5, 7, 11, .. , 229]
```

PROLOG VERSION

```
primes(N, U) :- truncate(N, int(2), U).

trωncate(0, U, []) :- !.
truncate(N, U, [P|V]) :
   getone(U, P, U1)  N1 is N-1,
   truncate(N1, filter(P, U1), V).

getone(int(N), N, int(N1)) :  N1 is ~ + 1.
getone(filter(P,U), M, W) :-
   getone(U, N, V),
   filter(P, N, V, M, W).

filter(P, N, V, N, filter(P, V)) :-
   divides(P, N), !.

filter(P, N, V, M, W) :-
   getone(V, N1, V1),
   filter(P, N1, V1, M, W).
```

PROGRESS

Past

- Designed by Malachi, Manna & Waldinger
- Interpreter in LISP

Present

- Introduction of types and modules
- Compiler for a virtual TABLOG machine

Future

- Concurrent TABLOG
- Programming environment

Parallel Execution of Lisp Programs

Joe Weening

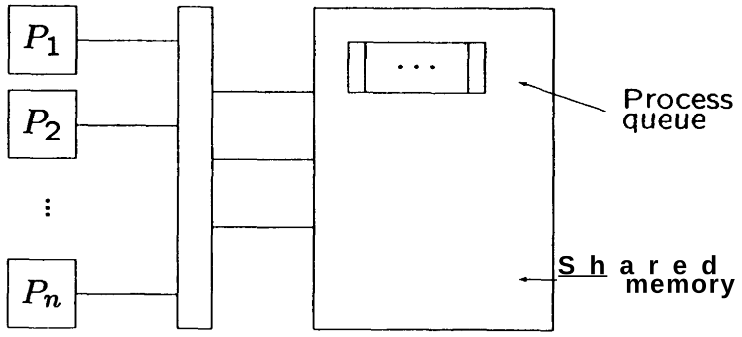
Lisp programs are well-suited for execution on a shared-memory multiprocessor using a queue-based ‘execution model. Function calls and variable bindings provide opportunities for process creation at many levels in a Lisp program. The Qlisp language lets a programmer use this model with the ability to decide at runtime whether computations are to be done in parallel.

We investigate several strategies for deciding when to create parallel processes. These include analysis of the size of subcomputations, the state of the multiprocessor, and a top-down approach based on the structure of the program being executed.

Some properties of Lisp:

- Primarily a functional language
- Manipulates symbolic data
- Uses pointers to represent complex data structures in memory
- Garbage-collected heap storage

Queue-based parallel processing



- Running processes decide when to create new processes
- Must avoid having too few or too many processes in queue

Qlisp language (Gabriel & McCarthy)

- Extension of Common Lisp

- Assumes shared memory and queue-based execution model

- Allows parallelism at any level

Qlisp project at Stanford

- Compile Qlisp for existing shared-memory multiprocessor (Alliant FX/8)

- Develop applications in symbolic computation (e.g. MACSYMA)

Qlisp programs with increasing amount of parallelism

```
(defun fun0 (x y z)
  (let ((a (glom x y))
        (b (bum z)))
    (cons a b)))

(defun fun1 (x y z)
  (qlet (> x 5) ((a (glom x y))
                 (b (bum z)))
    (cons a b)))

(defun fun2 (x y z)
  (qlet 'eager ((a (glom x y))
                (b (bum z)))
    (cons a b)))
```

“Futures” used to implement eager parallelism (cf. Halstead’s Multilisp)

Example:

```
(let ((u (fun2 x y z)))
  (list (car u)
        (car (cdr u))))
```

- `(car u)` won’t cause any waiting

- `(car (cdr u))` might wait for process computing `(bum z)`

Other Qlisp primitives

- QLAMBDA — provides mutual exclusion and “process closures” for parallel object-oriented programming

- QCATCH/CATCH/THROW — used for “or parallelism;” can kill processes whose results are no longer needed

Process creation depending on size of computation

Let $t =$ estimated time of a subcomputation

If $t > T$ then create new process

Advantages

- Limits the overhead of process creation

Disadvantages

- Can create processes needlessly
- Estimates may be hard to compute

Process creation depending on state of machine

Let $n =$ current number of processes in queue

If $n < N$ then create new process

Advantages

- Simple to implement

Disadvantages

- May have bad interaction with scheduling

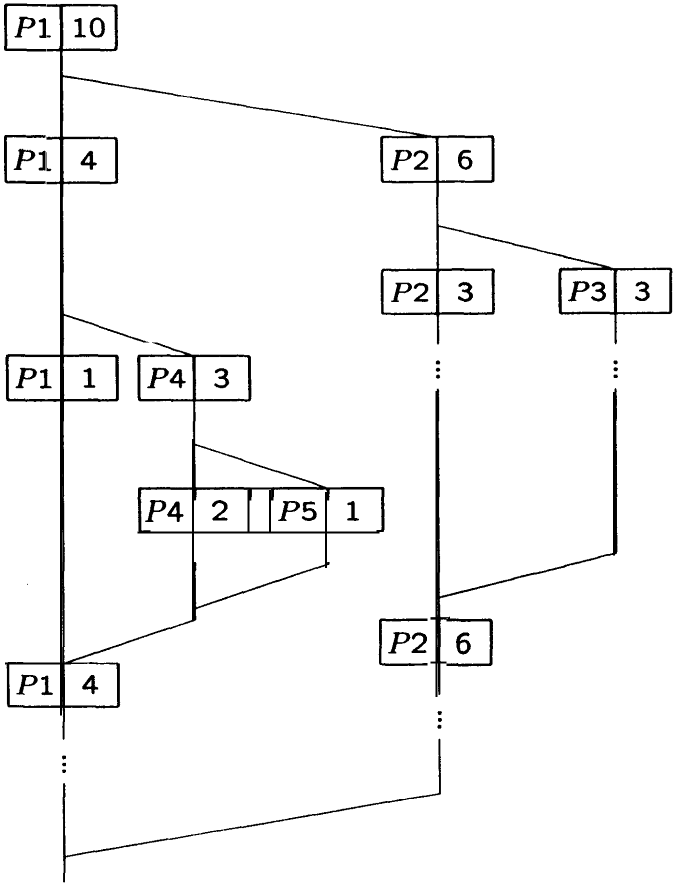
Top-down approach to process creation

- Each process has a number of “virtual processors” assigned

- Virtual processors are delegated to child processes, returned to parent when child terminates

- When number of virtual processors drops to 1, new processes are no longer created

Allocation of virtual processors based on time estimate for subcomputation



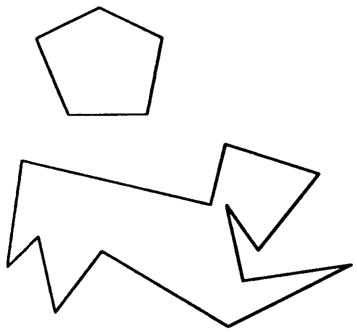
Shortest Paths and Visibility Problems in Simple Polygons

John Hershberger

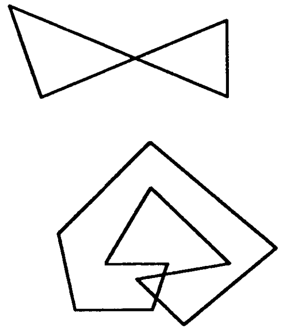
This talk describes solutions to several shortest path and visibility problems involving a simple planar polygon *P* with *n* sides. (A simple polygon is a closed polygonal path that does not intersect itself.) We consider the following problems: (i) Preprocessing *P* so that for any query point *q* inside *P*, we can find the length of the shortest path from *q* to a fixed source vertex *s* in logarithmic time; (ii) Computing the subpolygon of *P* visible from a particular vertex or segment on the polygon boundary; (iii) Preprocessing *P* so that for any query point *x* in *P*, we can find in logarithmic time the portion of some fixed polygon edge *e* that is visible from *x*; (iv) Preprocessing *P* so that for any query ray *r* emerging from e, we can find in logarithmic time the first intersection of *r* with the boundary of *P;* (v) Computing all pairs of mutually visible polygon vertices in constant time apiece.

All these algorithms are based on a single geometric structure, the *shortest path tree,* which is the collection of all shortest paths inside *P* from a given source vertex *s* to the other vertices of *P*. The shortest path tree can be computed in *O(n)* time once a triangulation of the polygon is known. (A triangulation is a set of diagonals that splits the polygon interior into *n-1* triangles.) Since the current bound for triangulation is *O(n* log log *n)* [Tarjan-Van Wyk, 1986], the shortest path tree can be built in *O(n* log log *n)* time.

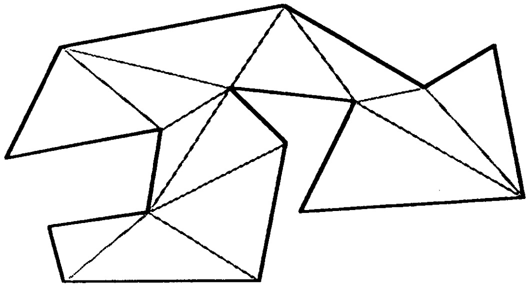
Simple Polygons



Non-Simple Polygons



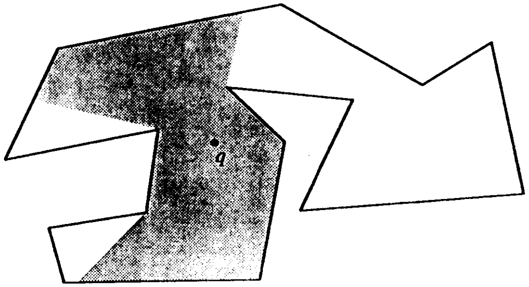
Simple polygons can be triangulated in $O(n \log\log n)$ time [TV86].



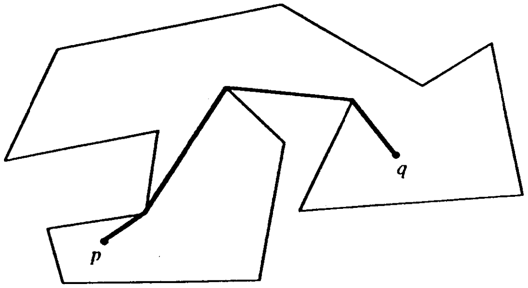
The algorithms that follow use triangulation as a first step, then do a linear amount of additional work.

Visibility and shortest paths inside a simple polygon: the polygon boundary is an opaque, impenetrable wall.

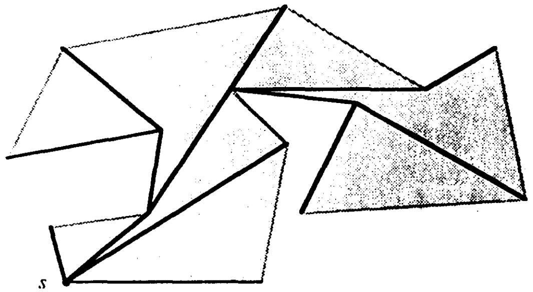
The *visibility polygon* of a point q inside the polygor is the part of the polygon interior visible from q.



The shortest path between points p and q inside the polygon.

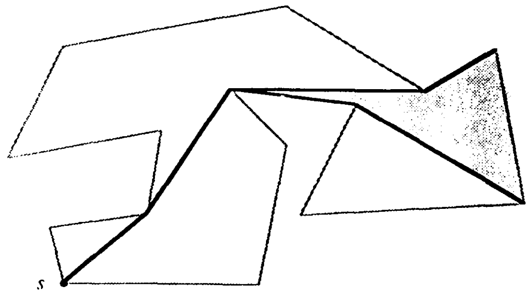


An important tool: the *shortest path tree*.



The shortest path tree can be built in linear time once a triangulation is known.

The shortest path tree divides the polygon interior into regionscalled*funnels*.

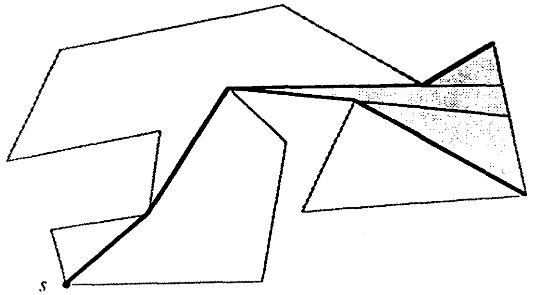


The single-source shortest path query problem:

> Given a source vertex s, preprocess the polygon so that the distance from s to a query point q can be found in $O(\log n)$ time.

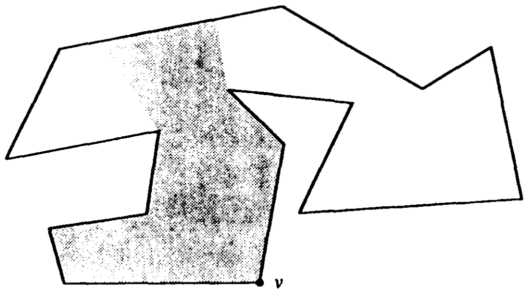
Solution: the *shortest path map*.

Produce by partitioning funnels of the shortest path tree.

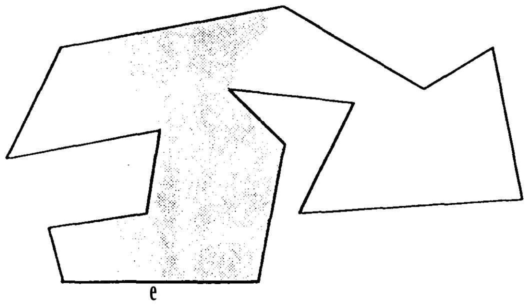


Visibility Problems

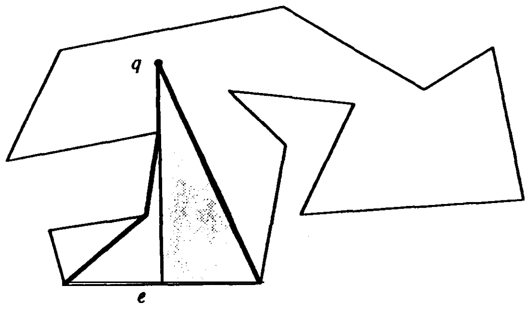
1. Visibility polygon of a vertex v: find in linear time from the shortest path map.



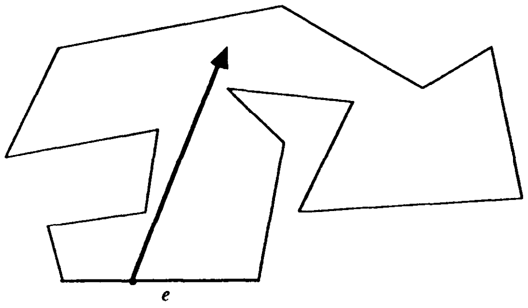
2 . Visibility polygon of a segment e: find in linear time using the shortest path trees of the segment endpoints.



3. Find the part of e visible from a query point q in $O(\log n)$ time.

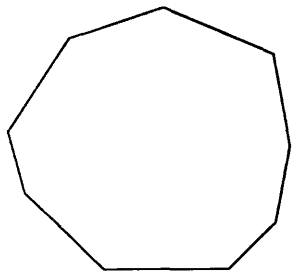


4. In $O(\log n)$ time find where a query ray from e first hits the polygon.

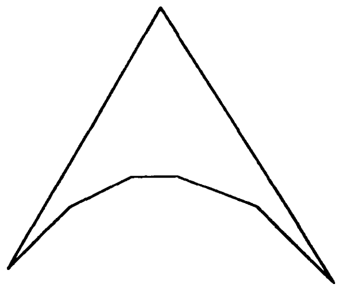


The *visibility* graph of a simple polygon records all vertex-vertex visibilities.

The number of visibilities ranges from $O(n)$ to $\Omega(n^2)$.



$\binom{n}{2}$ visibilities



$2n - 3$ visibilities

Find the visibility graph in time proportional to its size by finding shortest path maps for all vertices.

- Build the first shortest path map as described earlier.

- Move the source of the shortest path map around the polygon boundary, transforming the shortest path map of each vertex into that of its neighbor.

Each transformation changes just a few edges of the shortest path map. The total number of changes is roughly equal to the number of vertex-vertex visibilities. Each change takes constant time.

Conclusion

Visibility and shortest paths are closely related.

The shortest path tree is a useful tool for solving a variety of visibility problems inside simple polygons.

Increasing the Expressive Power of Formal Systems

John Lamping

Current programming languages restrict users to limited combinations of the concepts they support, resulting in limitations on their expressive power. This is especially noticeable when it would be useful to have language constructs available in data objects. For example, current programming languages generally don't provide data objects that incorporate free variables. This is true even of systems that provide data ranging over linguistic structures.

We show that a combination of two closure conditions will eliminate many of the restrictions: Any construct available in language expressions must be available in data objects. And if an operation of the language can be naturally decomposed into several parts then each part must be available independently. These two conditions act synergistically to preclude restrictions on how concepts can be combined.

The language design principle embodied in the two conditions is most useful in languages that contain their own meta-theory. It can also provide a convenient way of specifying partial evaluation. And it suggests ways of decomposing procedural reflection. The independence provided by the conditions also enhances the amount of parallelism available in a program.

We have designed a functional programming language which satisfies the two conditions. As a result it is able to straightforwardly express concepts like variables, environments, closures, dependencies, and expressions. And it can freely incorporate these concepts into other data objects. Yet the language has only 4 primitive constructors. We have designed an interpreter for this language and are in the process of implementing it. We have also designed the outline of a logic programming language which also satisfies the two conditions.

Describing a Logic Circuit

```
FUNCTION mux(a, b, c: BOOLEAN):
           BOOLEAN;
        BEGIN
          mux := (a AND c)
                  OR
                  (b AND NOT c)
        END;
```

Describing an Arbitrary Logic Circuit

```
⋮
TYPE signal= RECORD
             CASE tag:class of
                  simple: ↑input;
                  compound: ↑gate
             END;
TYPE gatekind = (ANDGATE, ORGATE, NOTGATE);
TYPE gate = RECORD
            kind: gatekind;
            firstinput: ↑signal;
            secondinput: ↑signal
            END;
TYPE env = • • •
⋮
FUNCTION simulate(circuit:signal, inputs:env):
                  BOOLEAN;
            BEGIN
              ⋮
            END;
⋮
mux := • • •
```

Wouldn't This be Nicer

```
s1 := [a] AND [c];
s2 := [b] AND NOT [c];
          :
   later in the' execution
          :
mux := s1 OR s2;
```

Problem

Programming languages only allow limited combinations of the concepts they support.

Solution

Design languages following two closure principles:

1. Constructs available in expressions are also available in data objects.

2. Naturally separable operations are available independently.

Principle 1

Any construct available in expressions of the language is also available in data objects.

Related to "language as data"

```
Examples:  s1 := [a];
           s1 := [a] and [c];
```

Principle 2

If an operation of the language can be naturally decomposed into several parts then each part is available independently.

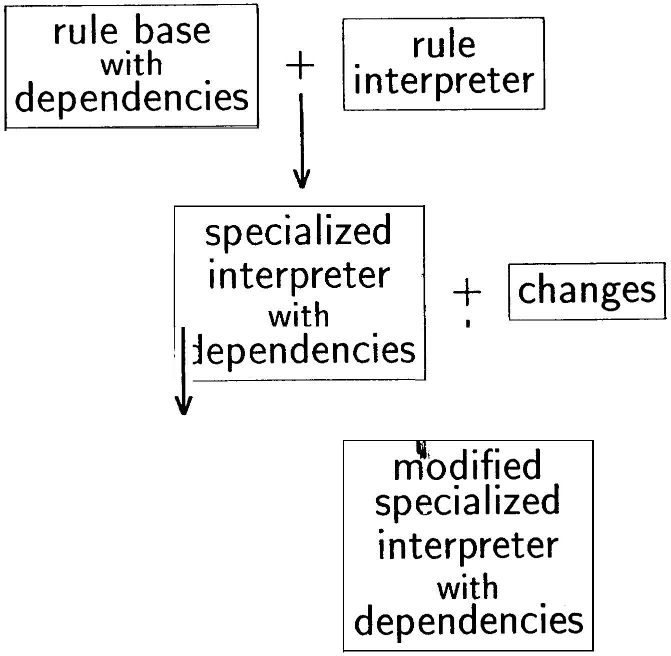
Elaborating "independently" :

Operations can be applied in any order.

Operations are applicable to many data objects.

```
Examples:   mux := s1 or s2;
            sel := mux WHERE c=TRUE;
```

An Illustration



I t's Possible

Functional language

- Design
  - Four basic constructs
  - Control for free
- Implementation
  - Reduction interpreter
  - Sharing algorithm
  - In progress

Logic language design

Areas of Applicability

M-eta-theoretic systems

Extensibility

Partial evaluation

Reflection

Parallelism

Towards Efficient Maintenance of Integrity Constraints

Xiaolei Qian

The need for integration of knowledge and data into knowledge base systems is widely recognized. Such a knowledge base system should provide uniform management of both data and knowledge. One of the important means of specifying the semantics about data is via integrity constraints. The constraint management component of a knowledge base system needs to provide means for both high-level constraint specification and efficient run-time constraint enforcement.

Experience has shown that the conventional database approaches to integrity constraint enforcement are not successful because they have to compromise between these two requirements. They either require the checking code to be written in arbitrary procedures which get triggered at run time, or they provide a single code generation algorithm which usually generates sub-optimal code. The former makes it extremely difficult to verify the correctness of the database states, while the latter makes the validation task too expensive to be used effectively. `

In this talk, we demonstrate the feasibility and power of a knowledge-based approach to the efficiency problem of constraint validation. We propose a reformulation approach which exploits knowledge about the application domain and database organization to reformulate integrity constraints into semantically equivalent ones in the sense that they enforce the same condition, and which are cheaper to enforce given the existing database configuration. A transformational mechanism is then used to generate efficient checking code. Since these reformulation and transformation processes are fully automated and are adapted to application and database evolution, we will be able to build a highly automated constraint manager which provides efficient maintenance of integrity of large integrated knowledge base systems.

Integrity Constraint Validation

The role of integrity constraints in DBMS

- Expressing semantics
- Representing knowledge
- Corrcctncss criteria for database extensions

Functions of Constraint Manager

- Constraint Specification
  - High-level, declarative Language
  - What instead of How
- Constraint Verification
  - Usc theorem proving techniques
  - Guarantce consistcncy of constraints
- Constraint Validation
  - Validatcd whenever state changes
  - Ensurc validity of database contcnts

Solution to the Validation Problem

- Procedural Approach (Triggers)
  - Not flexible, cannot use existing access paths
  - Error-prone, no guarantee of correctness
  - Low-level Spec., hard to maintain & understand
- Syntax-directed Approach
  - No state-dependent & domain-specific knowledge
  - Optimization for all application & configuration
- Query Modification
  - Slowing down query processor
  - Impossible to reformulate constraints
- Knowledge-basecl Approach
  - Rcformulate with configuration knowledge
  - Wcak cquivalcnce of transformation

Example

Knowledge:

$K1.(\forall e \in EMP\text{-}SET)$
$(MANAGER(e) = NAME(M(e)))$

$K2.(\forall e1, e2 \in EMP\text{-}SET)$
$(NAME(e1) = NAME(e2) \rightarrow e1 = e2)$

$K3.(\forall e \in EMP\text{-}SET)$
$(MANAGER(e) \neq NULL)$

$K4.(\forall m \in MGR\text{-}SET)$
$(MSAL(m) = MAX\{SALARY(e) | M(e) = m\})$

The following constraint takes time $O(n^2)$:

$$(\forall e \in EMP\text{-}SET)(\forall m \in MGR\text{-}SET)$$
$$(MANAGER(e) = NAME(m) \rightarrow SALARY(m) \geq SALARY(e))$$

Applying knowledge K1 and K2:

$$(\forall e \in EMP\text{-}SET)(\forall m \in MGR\text{-}SET)$$
$$(M(e) = m \rightarrow SALARY(m) \geq SALARY(e))$$

By equivalent transformation and K3:

$$(\forall e \in EMP\text{-}SET)$$
$$(SALARY(M(e)) \geq SALARY(e))$$

which takes time $O(n)$.

If, instead of applying K3, we apply K4:

$$(\forall m \in EMP\text{-}SET)$$
$$(SALARY(m) \geq MSAL(m)).$$

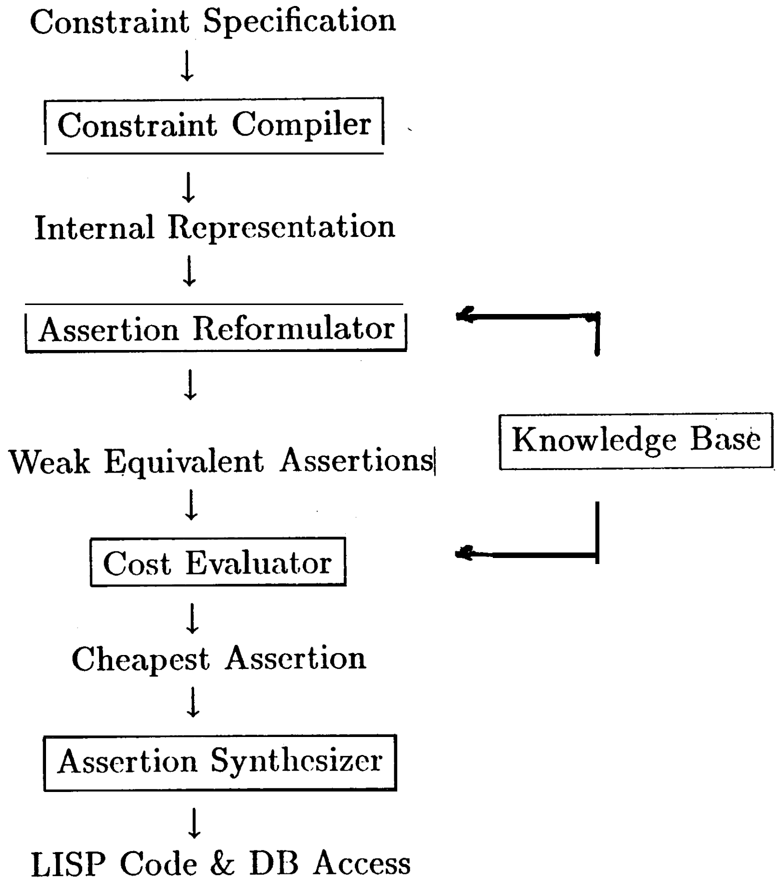
which takes time $O(n')$.

Knowledge-based Approach

Characteristics of Knowledge-based Approach

- High-level constraint specification
- Efficient implementation through reformulation
- A knowledge base of
  - o domain-independent equivalence-preserving transformation rules
  - o domain- & implementation-specific knowledge
- Transformation rules for
  - o constraint reformulation in o cheaper ones
  - o constraint synthesis into LISP and DB access
- Restricted notion of equivalence
  - o transformation rules are equivalence-preserving, but
  - o knowledge is used that may change over time

Organization of Constraint Manager



Knowledge in Constraint Reformulation

- Application semantics
    - ○ Cardinality of relationships
    - ○ Value relationships between domains
    - ○ Current state of the application

- Database structure & binding
    - ○ Map certain constraints to available structure
    - ○ Virtual vs actural domains supported by DB
    - o Existing bindings

- Physical *organization & access paths
    - ○ Materialized links & mappings (index)
    - 0 Physical clustering & locality

- Database utilization through monitoring
    - ○ Frequency of usage of different ‘operations
    - ○ Frequency of access of various links & relations

- Performance & cost information
    - o Relation sizes and image sizes
    - o Cost of different implementation

- Algebraic properties of operations
    - ○ functions and mappings
    - ○ set-, arithmetic-, & relational-operators
    - ○ Logic axioms
    - 0 correctness-preserving transformations

Conclusion

- Convensional approach does not work well because
  - o Syntax-driven and stat e-independent
  - 0 Cannot reformulate constraints
  - o Expensive or no guarantee of correctness

- Knowledge-based approach is PROMISING because
  - o state-dependent nature of constraints
  - o richness of constraint specification
  - o essential demand for automatic, efficient validation

Synchronizing Plans among Intelligent Agents via Communication

Charlie Koo

In a society where a group of agents cooperate to achieve certain goals, agents perform their tasks based on certain plans. Some tasks may interact with tasks done by other agents. One way to coordinate the tasks is to let a master planner generate a plan and distribute tasks to individual agents accordingly. However, there are two difficulties. Firstly, the master planner needs to know all the expertise that each agent has. The amount of knowledge sharply increases with the number of specialties. Secondly, the master-planning process will be computationally more expensive than if each agent plans for itself, since the planning space for the former is much larger. Thus, distributed planning is motivated.

The objective of this thesis research is to devise a model for *synchronizing* and *monitoring* plans independently made by nonhostile intelligent agents via *communication.* The proposed model allows agents to plan autonomously and then synchronize their plans via a commitment-based communication vehicle.

The communication vehicle includes a language, a set of communication operators and a set of commitment tracking operators. The tracking operators provide means to monitor the progress of plan execution, to prevent delays, and to modify plans with less effort when delays happen. A deadlock detection scheme using this communication model is also described.

Outline

Introduction

A model for multi-agent performance systems

Commitment-based communication

Assumptions about the agents

A case study

Conclusions

Introduction

Motivations for distributed problem-solving

- Resources of agents are limited.
- Knowledge integration is difficult.

Objectives of this research

- To formalize a model for synchronizing plans generated by individual agents.
- To design a communication model which allows agents to exchange commitments for the purpose of cooperation.
- To provide a means for agents to monitoring the execution of individual plans to avoid replanning as much as possible.

Target Domains

- Distributed manufacturing
- Decentralized project management
- Distributed robot planning

A case study

To illustrate the following characteristics:

- Agents plan autonomously.
- Agents can synchronize their activities among themselves via communication.
- Parallelism will be preserved.
- Communication reliability - deadlock detection and introduction of asymmetry.

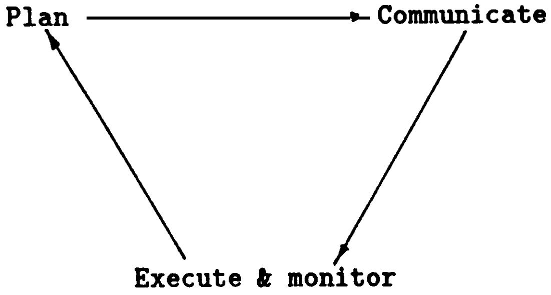
The example case

. Tasks:

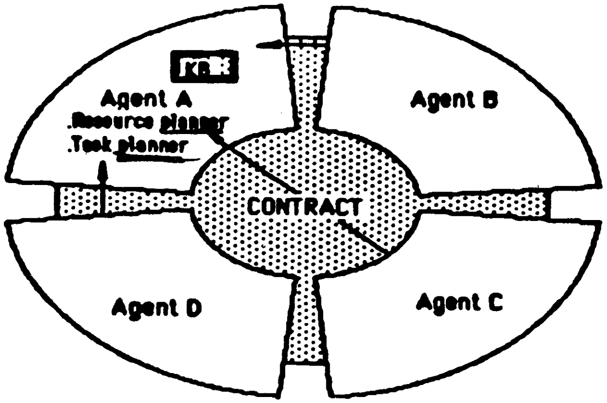
- Agents available:
  - **2** carpenters
  - **1** plumber
  - **1** technician
  - **1** painter

Two views to look at ,the problem:

Personal view:



Global view:



Distributed multi-agent performance system

Four components:

1. Individual planner(s) :

2. A communication language

3. A task distribution scheme

4. Execution monitoring mechanism

The commitment-based communication language

Motivations:

- Exchanging intentions is not enough for cooperative work. We need a commitment-binding mechanism between agents.
- Need a commitment-tracking mechanism

The form of a “contract” (A simplified version) :

- Asker=
- Doer=
- What=
- When=
- Qualification=
- Justification=

Execution monitoring

It involves a set of primitive actions. The operators are listed as follows:

1. **Summarize** all the commitments
2. **Trace-up** qualification links
3. **Trace-down** qualification links
4. **Inquire** information
5. **Signal** other agents

Assumptions about the agents

1. Agents communicate via our communication language only.

2. Each agent has all the necessary knowledge to construct plans to achieve its own goals.

3. Between all the agents, the knowledge necessary for synchronizing the plan is complete.

4. Agents are nonhostile but not necessarily benevolent.

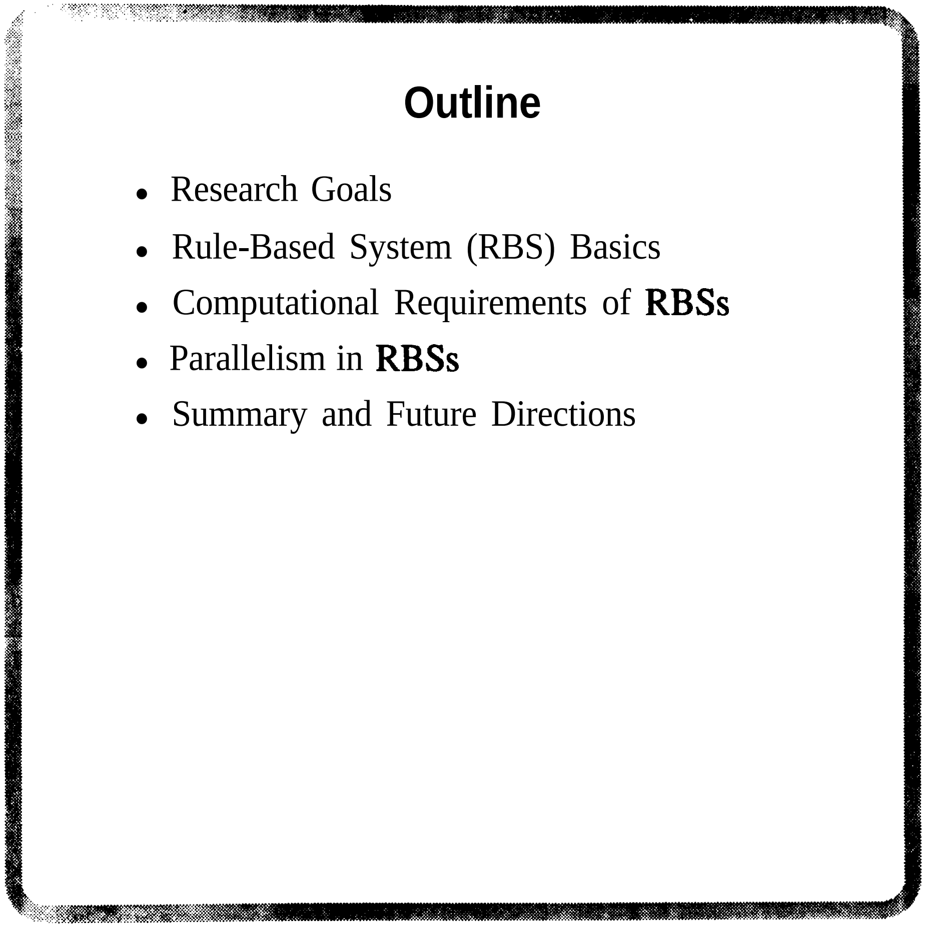
Conclusion

- Proposed a distributed planning-execution model
- Proposed a commitment-based communication protocol
- Identified a set of communication operators for binding commitments
- Identified a set of commitment-tracking operators
- Proposed a deadlock detection mechanism

High-Speed Implementations of Rule-Based Systems

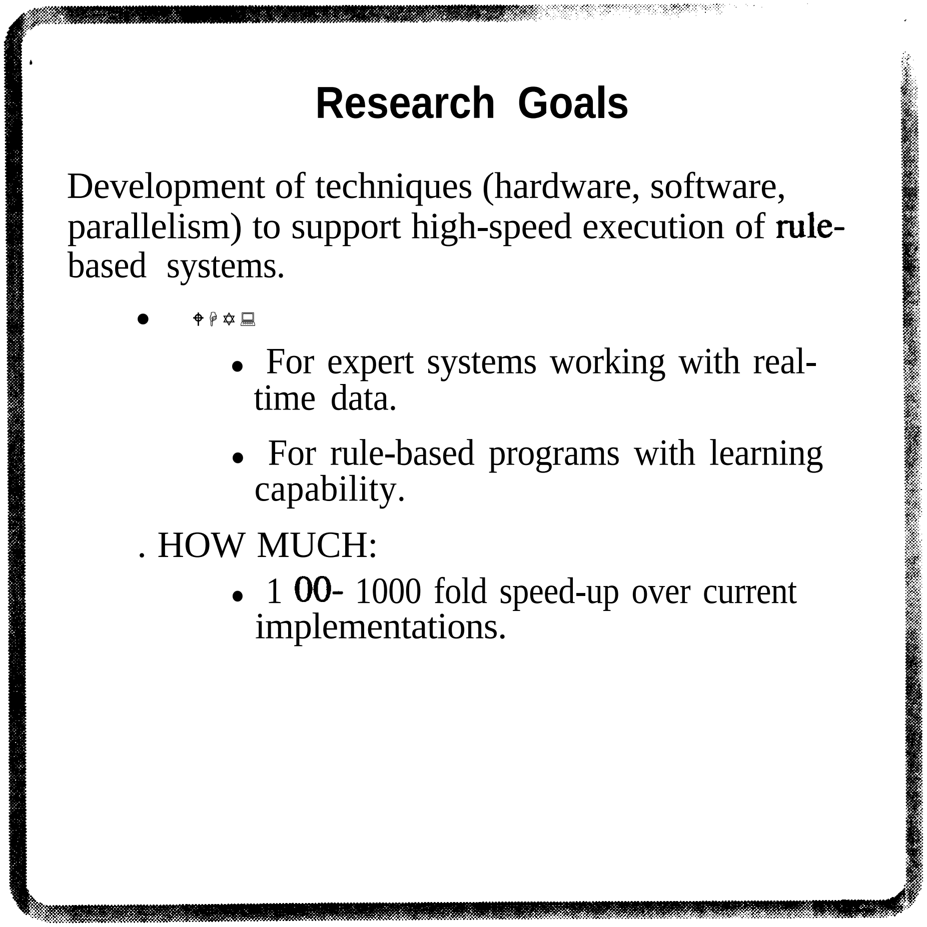
Anoop Gupta

Rule-based systems are widely used in Artificial Intelligence for modeling intelligent behavior and building expert systems. Most rule-based programs, however, are extremely computation intensive and run quite slowly. The slow speed of execution has prohibited the use of rule-based systems in domains requiring high performance and real-time response. In this talk we explore the role of parallelism in the high-speed execution of rule-based systems. We show that, contrary to early expectations in the field, only limited speed-up (10-20 fold) can be obtained from the use of parallelism. To get this 10-20 fold speed-up, it is necessary to exploit parallelism at a very fine granularity and shared-memory multiprocessors are necessary. We also discuss some of the research issues currently being explored.



Outline

- Research Goals
- Rule-Based System (RBS) Basics
- Computational Requirements of **RBSs**
- Parallelism in **RBSs**
- Summary and Future Directions

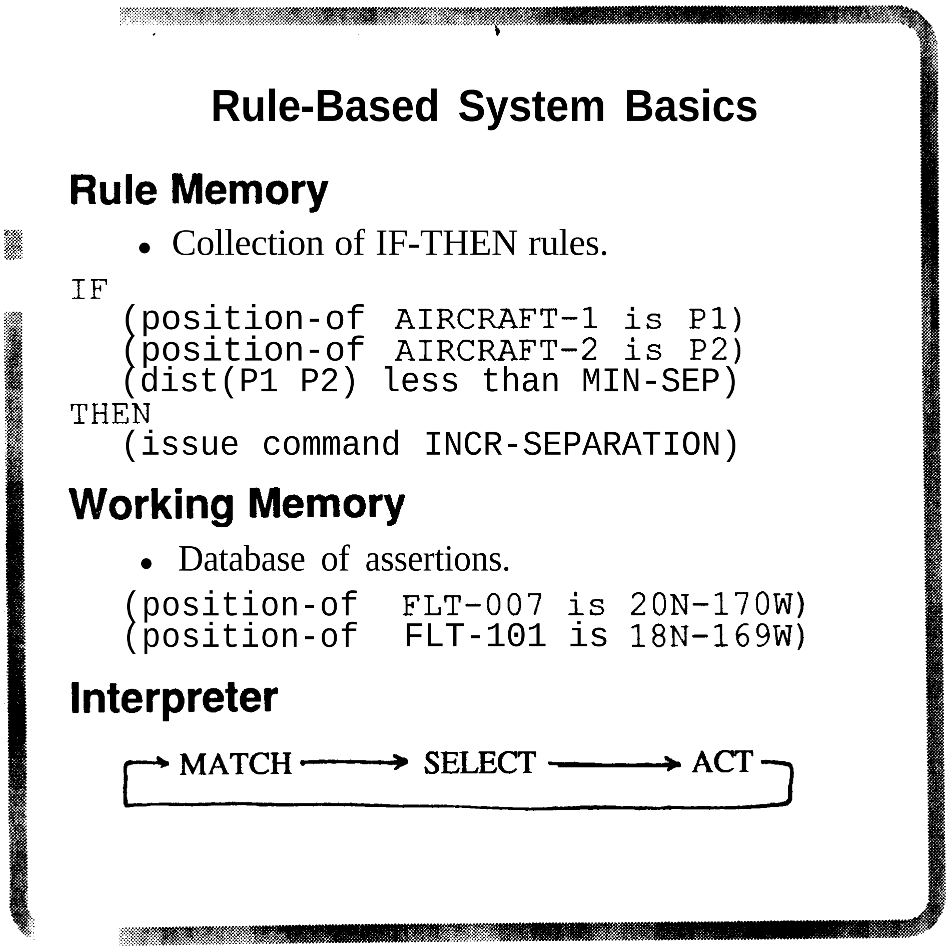


Research Goals

Development of techniques (hardware, software, parallelism) to support high-speed execution of **rule-based** systems.

- ⌖👇✡💻
    - For expert systems working with real-time data.
    - For rule-based programs with learning capability.

. HOW MUCH:
- 1 **00**- 1000 fold speed-up over current implementations.



Rule-Based System Basics

Rule Memory

- Collection of IF-THEN rules.

```
IF
   (position-of AIRCRAFT-1 is P1)
   (position-of AIRCRAFT-2 is P2)
   (dist(P1 P2) less than MIN-SEP)
THEN
   (issue command INCR-SEPARATION)
```

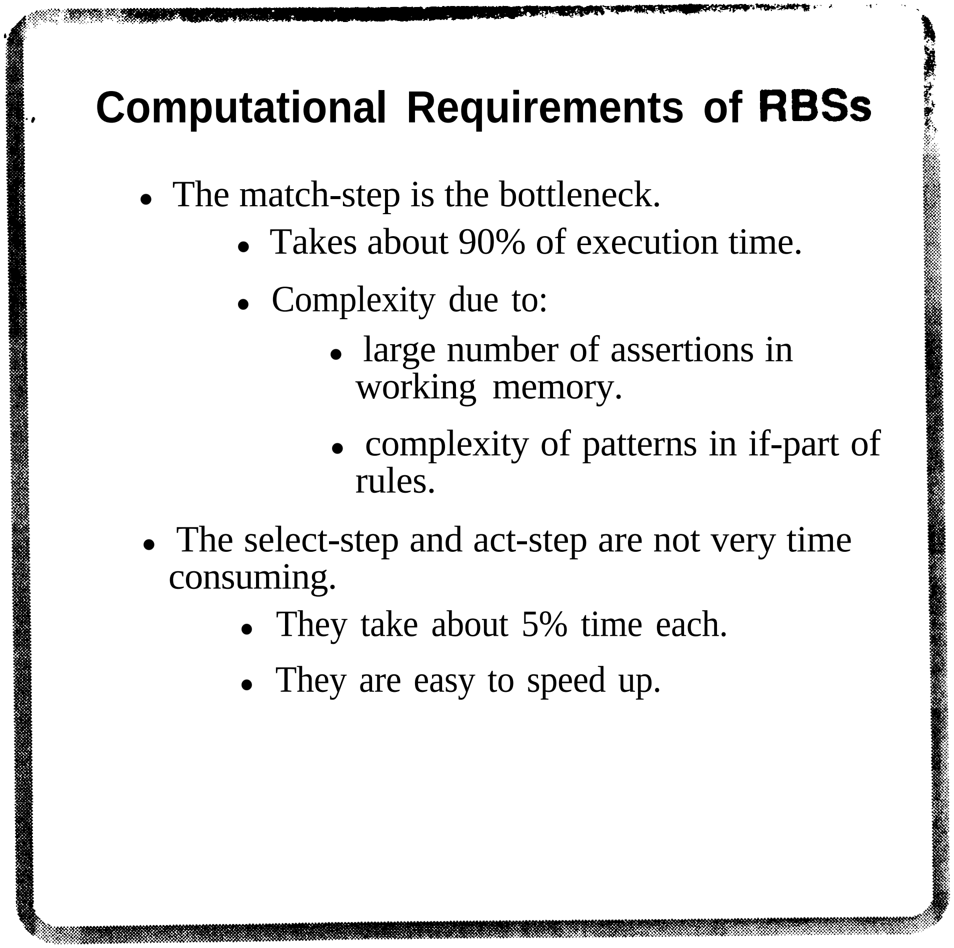
Working Memory

- Database of assertions.

```
   (position-of  FLT-007 is 20N-170W)
   (position-of  FLT-101 is 18N-169W)
```

Interpreter

→ MATCH ——→ SELECT ——→ ACT ↩



Computational Requirements of RBSs

- The match-step is the bottleneck.
  - Takes about 90% of execution time.
  - Complexity due to:
    - large number of assertions in working memory.
    - complexity of patterns in if-part of rules.
- The select-step and act-step are not very time consuming.
  - They take about 5% time each.
  - They are easy to speed up.

Solutions to the Speed Problem

- Faster Technology
- Custom Processor Architectures
  - Comparison of CISC, RISC, Custom Arch, . .
- Compilation and Algorithmic Improvements
  - Compilation gives factor of 10
  - New algorithms gives factor of 4
- Parallelism
  - On surface, large amount of parallelism.

Parallelism in Production Systems

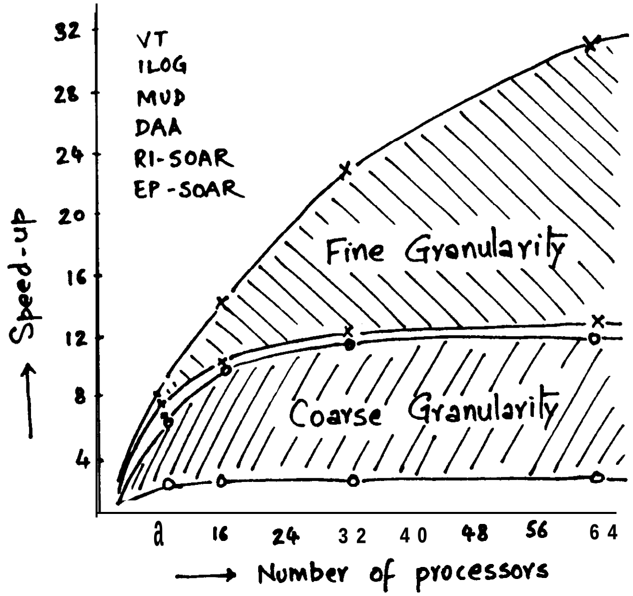
Issues

- What is the appropriate granularity?
- How much speed-up can we expect?
- Is reprogramming of existing applications necessary?
- What are suitable parallel machines?

Granularity and Speed-Up

- Coarse Granularity: (Rule-level parallelism)
  - Match for individual rules is performed in parallel.
  - Average speed-up is 7-fold.
  - Speed-up is low due to large variation in processing times.
- Fine Granularity: (Intra-Rule parallelism)
  - Processing for individual rule may be done on multiple processors.
  - Average speed-up is 20-fold.
  - Length of **subtasks** is 20- 100 machine instructions.
  - Requires lots of shared data between subtasks.

, Granularity and Speed-Up (Graph)

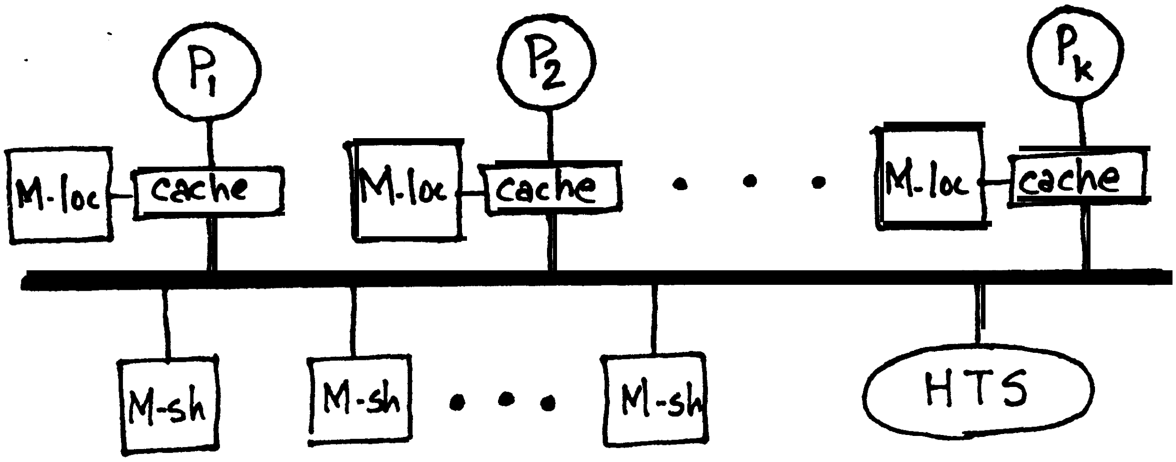


Is Reprogramming Necessary?

- Results shown so far correspond to programs to which no changes were made.
- Expect another factor of 3-5 (?) speed-up if programs are rewritten with parallelism in mind.
    - Reason: Semantic information available to the programmer.
- Currently, one of the important research areas.

Suitable Parallel Architectures?

- Shared-memory architectures are desirable.
    - Fine-grain algorithm requires shared data.
    - Dynamic load balancing.
- Examples: VAX-1 1/784, Encore Multimax, Sequent, ...
- Expected performance on a 32 processor machine with 2-MIPS processors is 3800 rule-firings/sec.



Summary and Future Directions

- To obtain 2-3 orders of magnitude speed-up, a multipronged approach is necessary:
  - Technology (10-fold)
  - Algorithms (4-fold)
  - Parallelism (20-fold)
- Parallelism must be exploited at fine granularity to get significant speed-up.
- Shared-memory multiprocessors are the right machines, at least for the near future.
- To get the next major factor in speed-up, one must look at task-level parallelism.

Call-by-unification: How to Combine Logical and Functional Programming Efficiently

Per Bothner

There are a number of ''new'' programming language paradigms intended to more cleanly and concisely express algorithms. The Q language and compiler is an attempt to integrate some of these ideas in a unified language, and to see how they can be implemented efficiently.

In this talk we concentrate on adding logical variables (and constraints) to a language with lambda expressions, using unification to bind formal and actual parameters. A simple optimization reduces this to call-by-value in the normal case. We also express constraints in terms of functional dependencies betwen variables, and examine how to fit assignment into this framework.

New languages to save the world

Fashionable ways to improve programming languages (and thereby programmer productivity) include:

- Functional programming (Backus' FP; ML)
- Logic programming (Prolog)
- Object-oriented programming (Smalltalk-80)
- Array/list programming (APL, QNial; Lisp)

Can we make languages unifying these ideas:

- with simplicity and coherence;
- while still being able to do low-level things;
- and execute efficiently on conventional hardware?

Here, we concentrate on how to combine the advantages of logic programming (systematic searching for solutions, programs may be run "backwards") with those of function values and expressions (fewer explicit temporararies, higher-order functionals).

Influences include Fresh [Smolka] and Icon [Griswold].

Towards logic programming

We allow an expression to return multiple results.

Zero results is failure (corresponds to `false`).

Simple expressions return one value: $Eval\llbracket \mathsf{x} \rrbracket = x$.

$$Eval\llbracket \mathsf{x} > 0 \rrbracket = \begin{cases} \{x\}, & \text{if } x > 0 \\ \{\}, & \text{otherwise} \end{cases}$$

$$Eval\llbracket \mathsf{e}_1 \mid \mathsf{e}_2 \rrbracket = Eval\llbracket \mathsf{e}_1 \rrbracket \cup Eval\llbracket \mathsf{e}_1 \rrbracket$$

A function with multiple-valued arguments returns the Cartesian product of applying to each combination of individual values.
E.g. $Eval\llbracket (10 \mid 20) + (1 \mid 2) \rrbracket = \{11, 12, 21, 22\}$.

There are no explicit multiple results: They are implemented by backtracking and handling exceptions.

Logical variables

A variable is an abstract data structure, which can be *instantiated* with a value, or be uninstantiated.

Axiom of instantiation: A program cannot distinguish between an instantiated variable and the value it is instantiated to (within the "pure" language).

Undeclared identifiers implicitly are variables.

Value unification

a=b tries to make two objects (*not* terms) equal.

If neither is instantiated, link their values together.
If both are instantiated, compare their values (by recursive unification if records or lists).
If one instantiated, instantiate the other to its value.

Important: Instantiation is undone on backtracking.

Call-by-unification

The lambda expression λp.e is a "function literal." We extend it so that the formal parameter p can be an arbitrary expression. Instead of just an identifier, p is a general pattern which is matched against the actual parameter.

After defining f = λp.e, the meaning of
y = f(x) is:

- Evaluate x, giving x.
- Initialize environment of f (and allocate new uninstantiated variables).
- Evaluate p (in the saved closure of f), giving p.
- Unify x=p, usually binding some of the newly-allocated variables.
- Evaluate e, giving e.
- Return e (and then unify it with y).

Examples

Standard functional definition of factorial:

```
fact = λx. if x=0 then 1 else x*fact(x-1)
```

Let us define an overloaded function definition:

```
def f x is a and f y is b
```

to mean:

```
f = λα.((α=x) & a) | ((α=y) & b)
```

```
def father ("John jr"|"Susan") is "John sr"
def mother ("John jr"|"Susan") is "Laura"
and mother "Laura" is "Alice"
and mother "John sr" is "Martha"
def parent x is (father x | mother x)
def grandmother x is mother(parent x)
```

$Eval[\![$`m = grandmother x & (m,x)`$]\!]$ =
{("Alice", "John jr"),("Alice", "Susan"),
("Martha","John jr"),("Martha","Susan")}

Optimization

If the first reference to x is x = e, then the unification reduces to just storing a reference to e in x. If at compile-time we can find these initial unifications, we can replace expensive logic variables and unification by cheap memory references.

We do this by traversing the expression tree in execution order. The state of a variable is initially *unused.* If an *unused* variable is on one side of a unification, it becomes *set*, otherwise it becomes *general.*

The problem is that dynamic control flow is different from static program order. This can be solved by conceptually saving these state bits when the control flow splits, and *merging* these bits when the flow merges.

At the end, only *general* variables need be logical, so functional-style programs pay no extra cost.

Types

Types are coercions. E.g.:

$$Eval[\![\texttt{Float(x)}]\!] = \begin{cases} \{x\}, & \text{if } x \text{ is floating-point} \\ \{x * 1.0\}, & \text{if } x \text{ is fix-point} \\ \{\}, & \text{otherwise} \end{cases}$$

We call "~" the "inverse operator" because it is defined by:

F-x = a ≡ x = F a.

We can combine these ideas to do type-checking of parameters:

```
cotan = λFloat~x. l/tan(x)
```

Dereferencing

$$Eval[\![\texttt{Value(x)}]\!] = \begin{cases} \{x\}, \text{if } x \text{ is not a variable} \\ \{\}, \text{if } x \text{ not instantiated} \\ \{Contents(x)\}, \text{otherwise} \end{cases}$$

Dependencies and constraints

If F(x_1, . ..) fails (in F's parameter section), where some x_i are variables, we create a *calculated* variable $\langle F, x_1, \ldots\rangle$, such that:
Value($\langle F, x_1, \ldots\rangle$) $= F$(Value(x_1), . . .).

Partly because of the Axiom of Instantiation, we cannot calculate this value "at need", but must do so whenever one of the x_i changes (and save the result). Thus there must be a pointer so x_i can notify $\langle F, x_1, \ldots\rangle$ so that it must re-calculate itself.

If F is a predicate, we have a constraint: e.g. `x < y`.

Dependencies must be destroyed on back-tracking (since false constraints must be removed).

It is useful (but tricky) to allow backwards calculation: (e.g. given 3 = sqrt(x), instantiate x=9). The same applies for solving systems of linear equations.

Assignments

It is occasionally difficult to totally avoid side-effects, so it is useful to consider variables which may be instantiated by either unification, assignment or both.

`a = New(T)` is a new assignable variable whose values are coerced to type T. Thus `a:=b` assigns $T(b)$ to a.

Assignments are not undone on backtracking, but the value can be changed by future assignments (limited by unification and other constraints).

Since variables linked by unification are equivalent, if one was created with `New`, any can be assigned, assuming coercion functions (T) do not conflict.

Note in `a:=b`, b is coerced to a ground value (a value with no variable references in it). Otherwise, if b were calculated (a dependent variable), we would get a dangling reference after backtracking removed it.

Other contributions

Single, linear contiguous stack which supports backtracking and closures.

A *block* is a record constructor. A *class* is a block (or rather a function whose body is block). Thus almost no extra mechanism is needed for class definitions.

The standard interface is a read-eval-print loop. There is no interpreter (the compiler always generates machine code), yet the response time is unnoticable.

The system supports modules, dynamic and static linking, and is written in a mixture of C and itself.

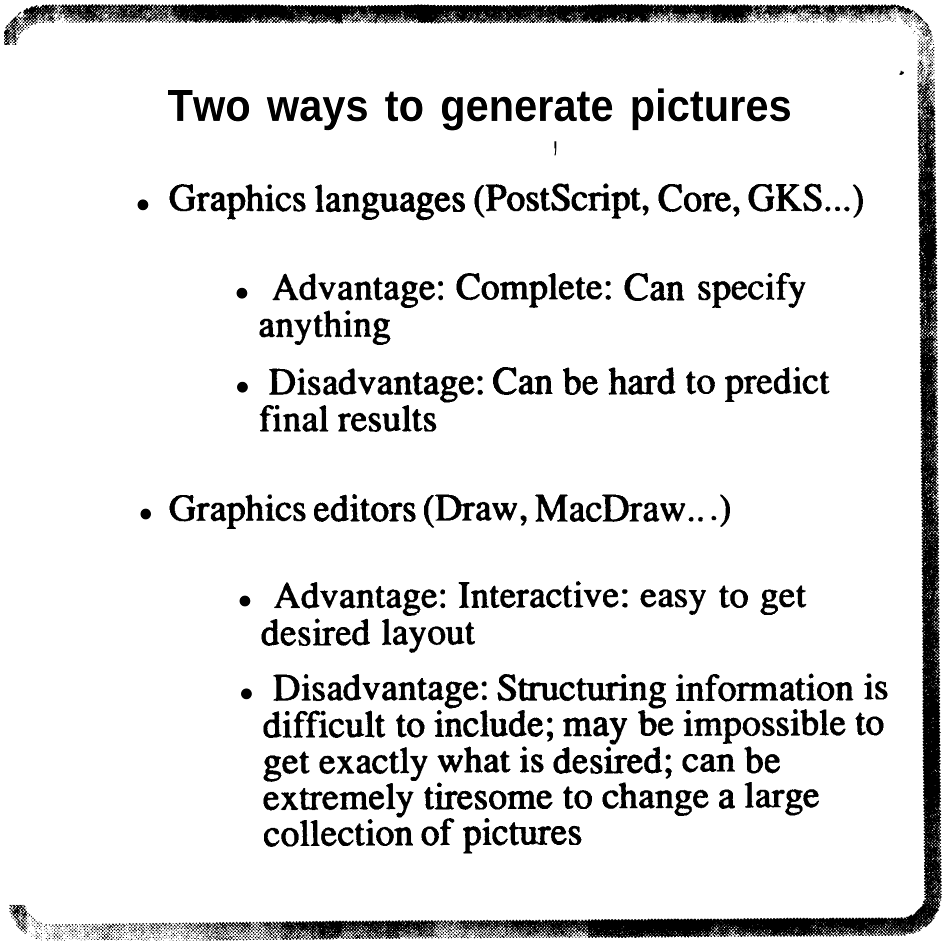
The evaluation rule is to *Apply* pairs of sub-expressions, where the application rule depends on both types. One benefit is coercion not only of parameters, but also of functions: e.g. a scalar function can be coerced to one that works on sequences (as in APL).

Editing Procedural Descriptions of Graphical Objects

Paul Asente

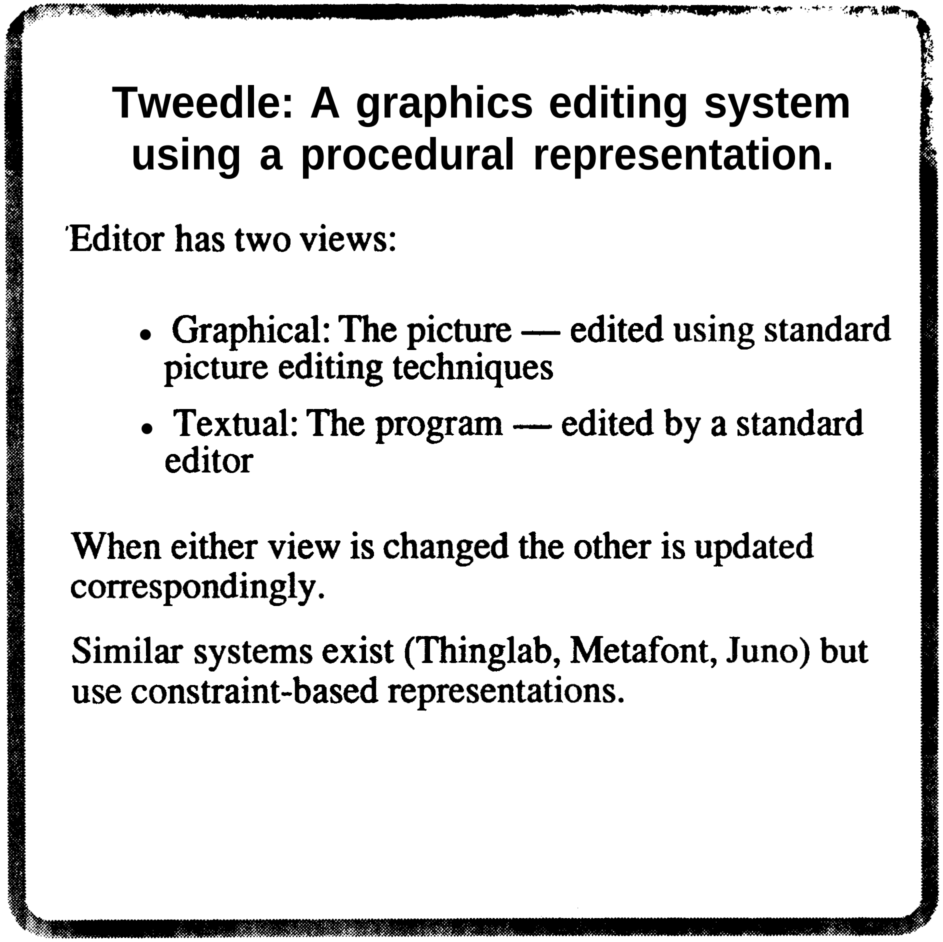
Traditionally, computer-generated images were created by running programs written in a programming language that supported graphics, either directly or through procedure calls. More recently, interactive graphics editors have appeared. These two approaches have different and complementary advantages and disadvantages. Tweedle is a graphics editor that attempts to attain the advantages of each approach while avoiding the disadvantages.

In Tweedle the user may modify either the picture itself or the program creating the picture; the other representation is automatically kept up-to-date. We will discuss the methods used by the editor to keep consistency when one representation is changed.



Two ways to generate pictures

- Graphics languages (PostScript, Core, GKS...)
  - Advantage: Complete: Can specify anything
  - Disadvantage: Can be hard to predict final results
- Graphics editors (Draw, MacDraw...)
  - Advantage: Interactive: easy to get desired layout
  - Disadvantage: Structuring information is difficult to include; may be impossible to get exactly what is desired; can be extremely tiresome to change a large collection of pictures



Tweedle: A graphics editing system using a procedural representation.

Editor has two views:

- Graphical: The picture — edited using standard picture editing techniques
- Textual: The program — edited by a standard editor

When either view is changed the other is updated correspondingly.

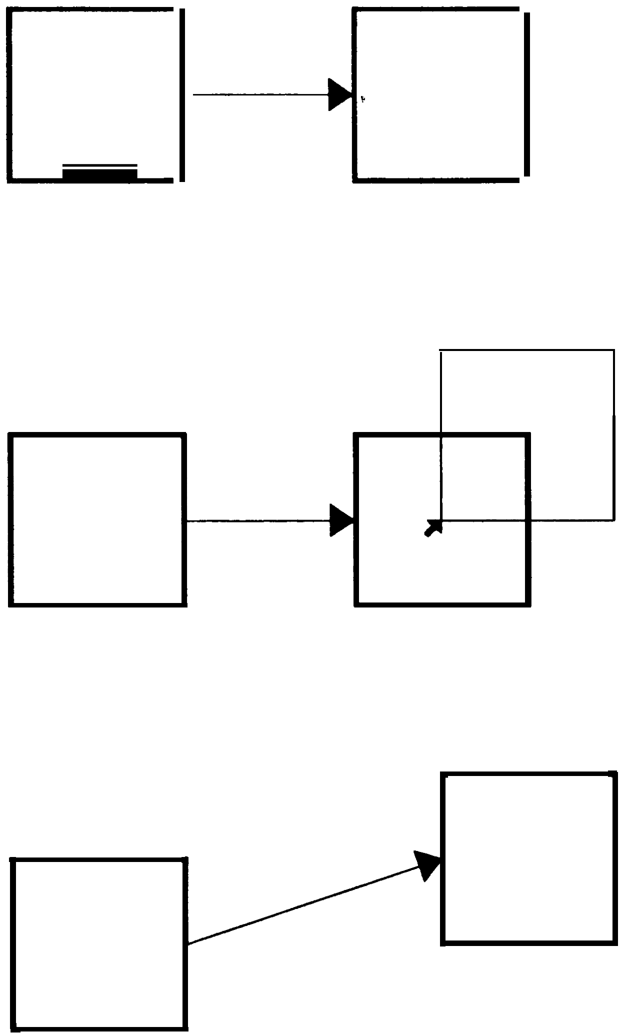
Similar systems exist (Thinglab, Metafont, Juno) but use constraint-based representations.

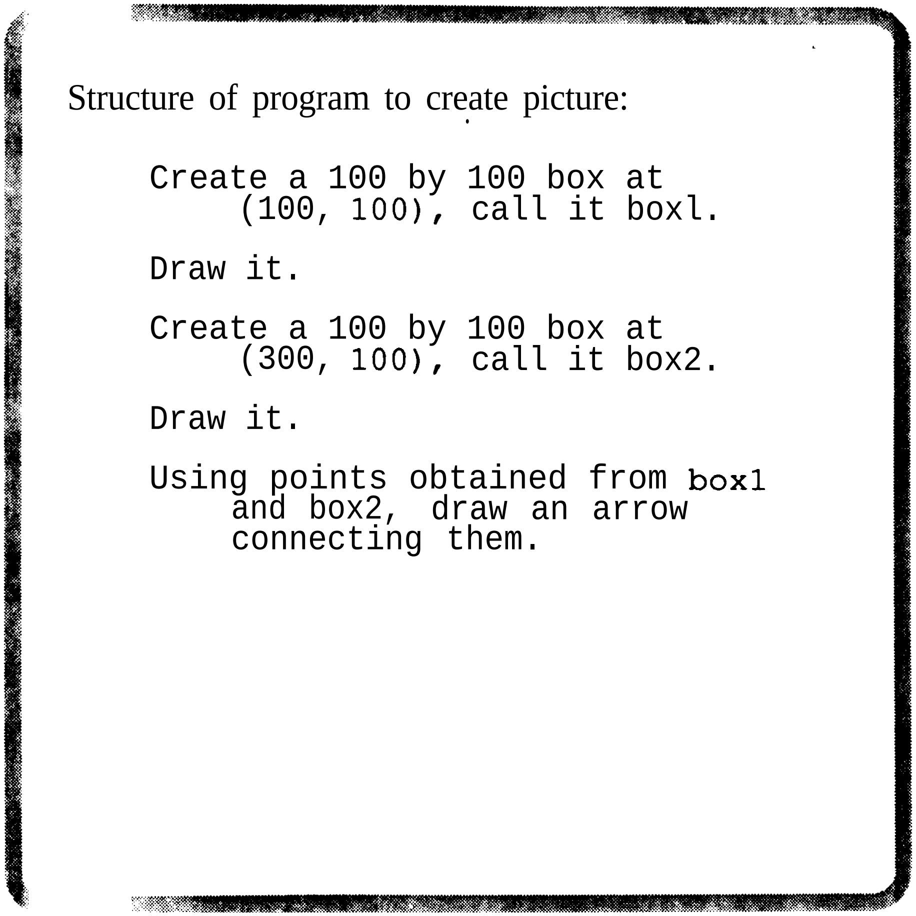
Program Interface

The user is presented with two windows, one showing the program text and one showing the picture.

There are also menus and dialog windows.

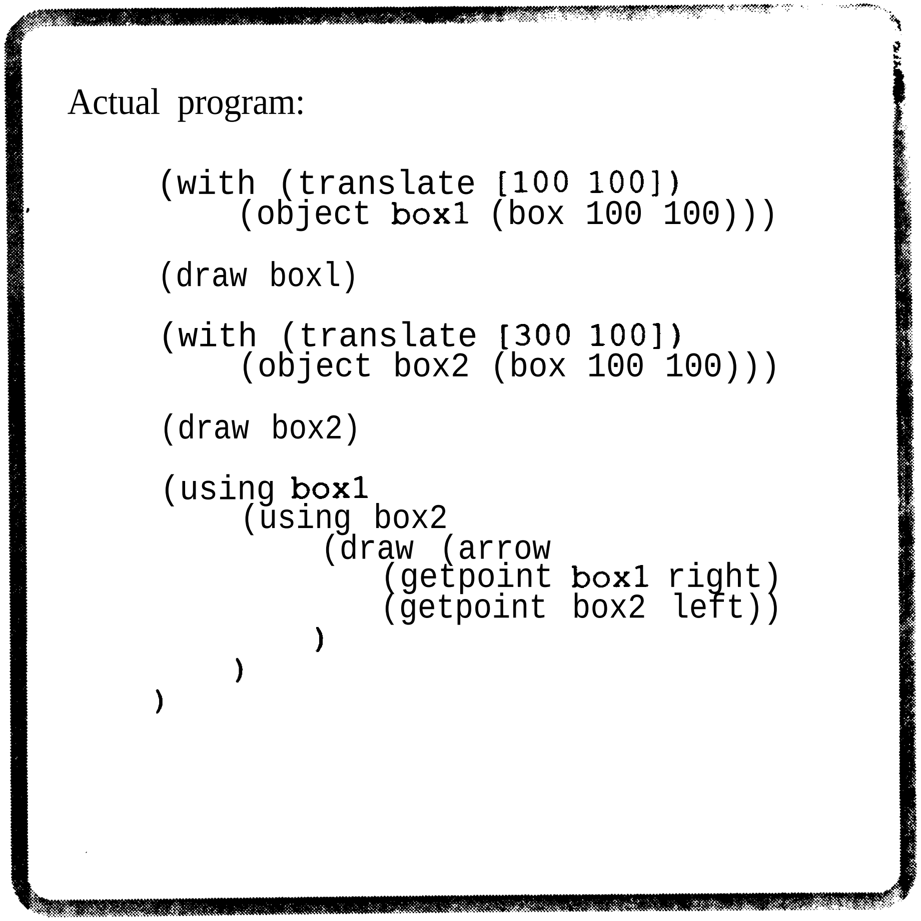
We will trace the actions the editor performs for a simple graphical request, moving a box.





Structure of program to create picture:

```
Create a 100 by 100 box at
     (100, 100), call it boxl.

Draw it.

Create a 100 by 100 box at
     (300, 100), call it box2.

Draw it.

Using points obtained from box1
    and box2, draw an arrow
    connecting them.
```



Actual program:

```
(with (translate [100 100])
    (object box1 (box 100 100)))

(draw boxl)

(with (translate [300 100])
    (object box2 (box 100 100)))

(draw box2)

(using box1
    (using box2
        (draw (arrow
           (getpoint box1 right)
           (getpoint box2 left))
        )
    )
)
```

User selects the right box, chooses ''translate'' from a menu, and chooses a new location for the box. This is what happens next:

1. Calculate the offset of the cursor motion. Say it is (50, 50).
2. Find a variable that refers to the right box. Here it is **"box2"**.
3. Create a statement that does the appropriate move:
   ```
   (transform box2
   (translate [50 50])).
   ```
4. Insert it at the end of the program.
5. Invoke the incremental parser on the new code.
6. Execute the new code.

Executing the new code involves several step:

1. Change the stored location of **box2**.
2. Notice that the arrow depends upon the location of **box2** and re-execute the invocation of ''arrow''.
3. Redraw the display to reflect the new situation.

In particular, we don't use the ''back door'': the picture is updated by the re-execution of the program, not directly from the cursor motion.

Going from changes to the program text is similar, but starts by analyzing the program to find differences and proceeding at the incremental parsing step.

Conclusions

- It works. You really can use a procedural representation in this type of a system.
- It's fast enough to be usable, even in the current prototype version.

Composing User Interfaces With Interactive Views

Craig Dunwoody

High-quality user interface software is critical to the usefulness of an interactive computing system. Unfortunately, such software is difficult to build using current tools. We are developing InterViews, a software system that facilitates the construction of powerful, flexible and efficient user interfaces.

InterViews is based on the idea of creating *Interactive Views* of an object to be manipulated (the *Subject).* A Subject provides an abstract data type interface to an object of some type, independent of interaction details. A corresponding View presents the Subject's state to the user and translates user input into operations on the Subject. A Subject may have one or more attached Views. Views are parameterized to allow for user tailoring. A user interacts with a View's parameter settings using a meta-View (a View of a View).

InterViews defines a set of common Subject and View classes that can be composed to create more sophisticated interfaces. For example, a Text Subject and View (implementing editable text) can be used to implement higher-level objects such as a Typescript (text-based command-response interaction) and a Document (mixed text, drawings, and other elements). The availability of these composable building blocks simplifies the task of building new applications and helps insure interface consistency across applications.

A prototype implementation of InterViews is now in everyday use by about fifteen researchers.

Goals

- Ease of building interfaces
- Interface quality and consistency
- Extensibility
- Customizability
- Efficiency

Perspective.

Graphics Packages

- GKS, PHIGS, QuickDraw

Window Systems

- NeWS, X

Toolkits

- SunVIEW, XToolkit

Frameworks

- Smalltalk MVC, MacApp

User Interface Management Systems

- BLOX, Domain/Dialogue

. Subjects and Interactive Views

Subjects

- Functional interface, no interaction details
- Text: 1-D character sequence, markers

Interactive Views

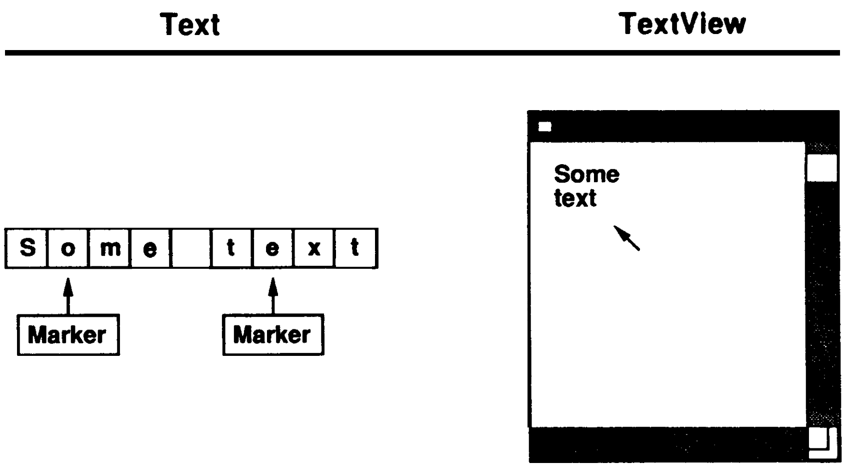
- Present Subject's state to user
- Translate user input into operations
- Text: 2-D layout, selection, editing

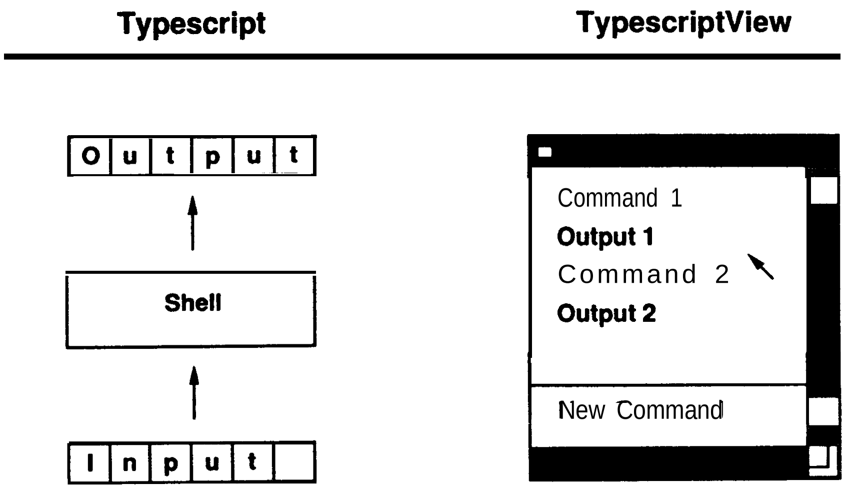
Meta-Views

- Interact with View parameter settings

Efficient implementation

- Buffering, caching, localization





The Interviews Environment

Imaging, Layout, and Windowing

- Canvas
- Boxes and glue, Frame, Viewer

Event Handling

- Dispatcher, Cursor

Selection

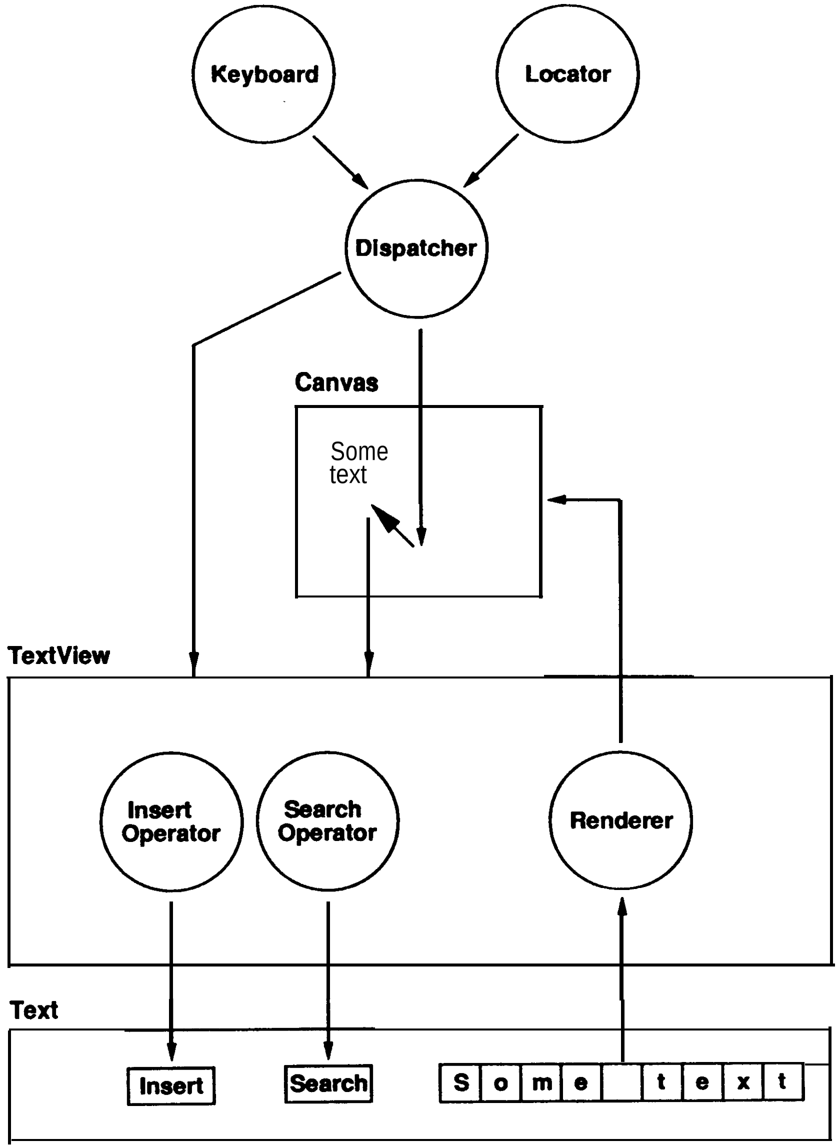
- Tagged rendering, Selection service

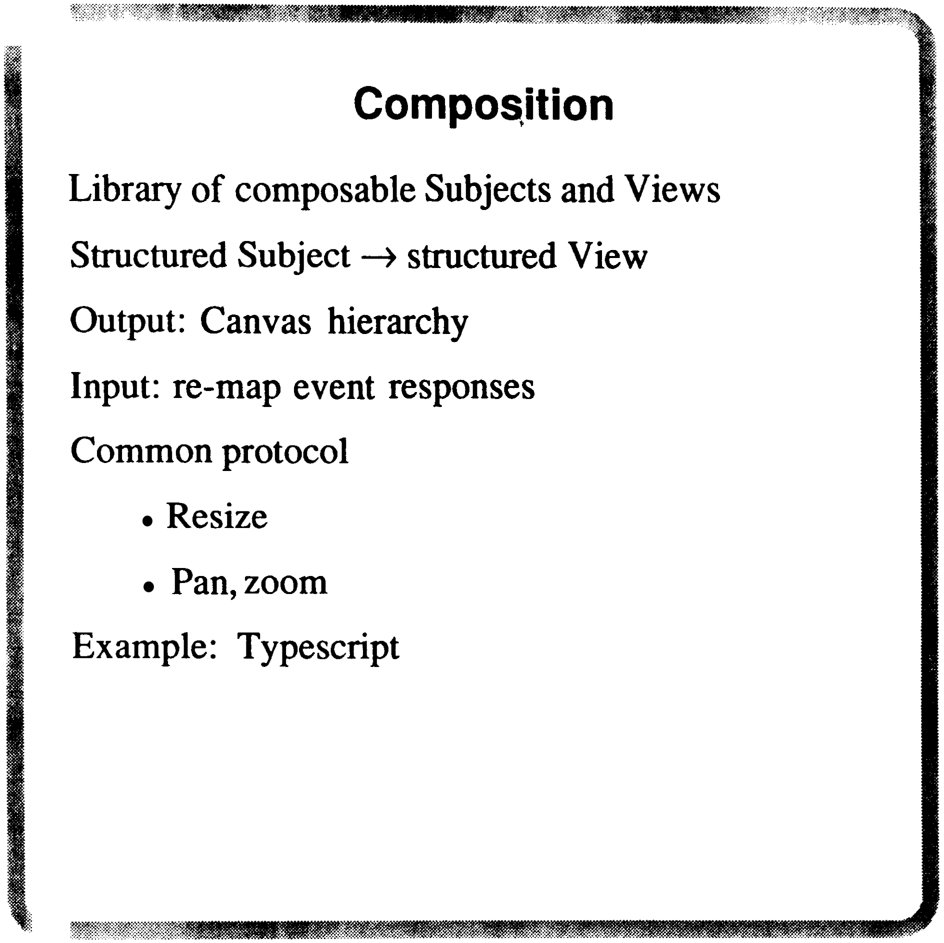
Operation Invocation

- Operators

Instantiation

- Prototypes, copying





Composition

Library of composable Subjects and Views

Structured Subject → structured View

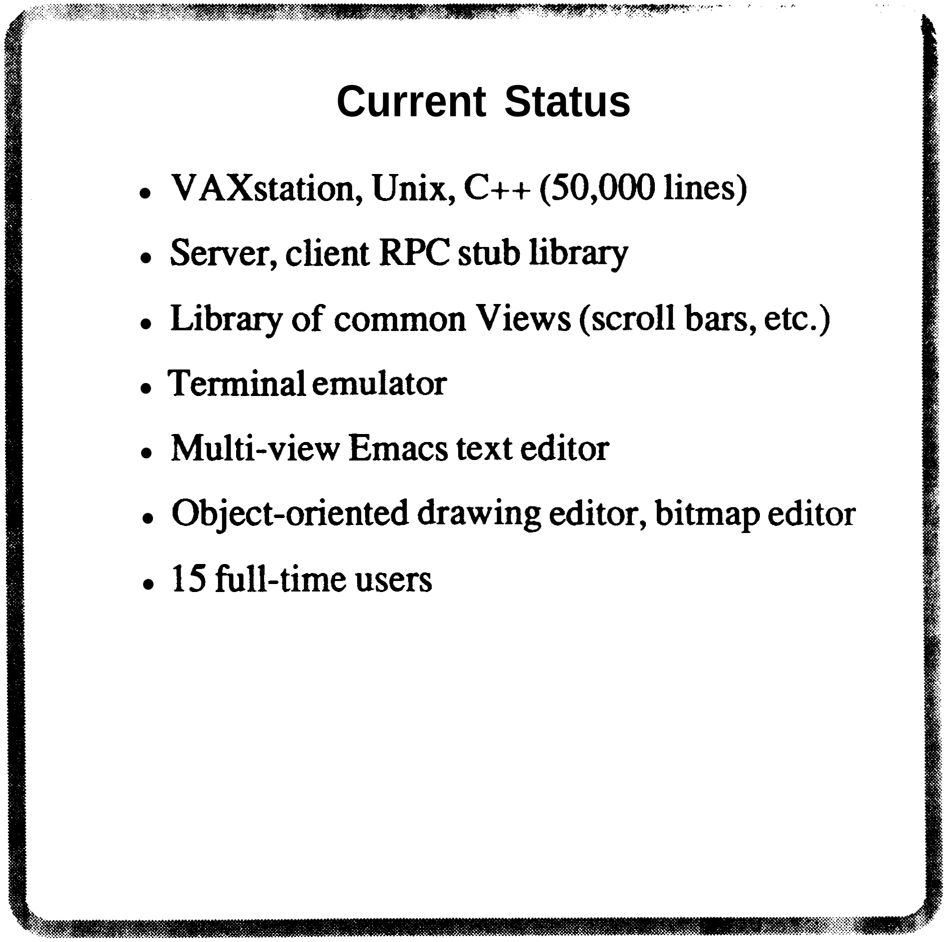
Output: Canvas hierarchy

Input: re-map event responses

Common protocol

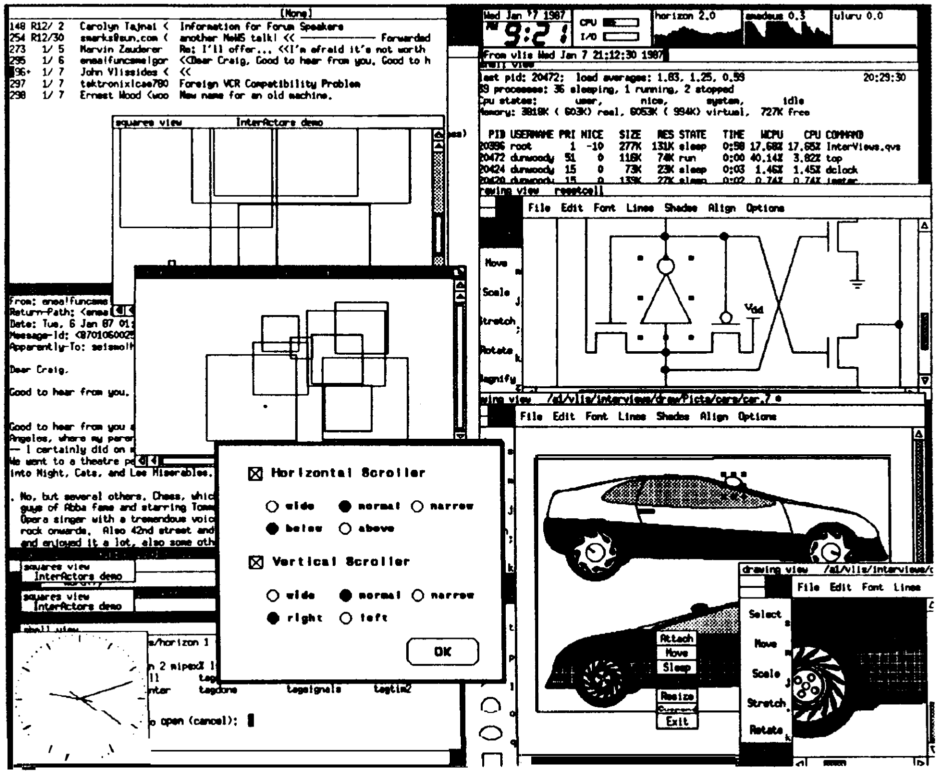
- Resize
- Pan, zoom

Example: Typescript



Current Status

- VAXstation, Unix, C++ (50,000 lines)
- Server, client RPC stub library
- Library of common Views (scroll bars, etc.)
- Terminal emulator
- Multi-view Emacs text editor
- Object-oriented drawing editor, bitmap editor
- 15 full-time users



```
                        [None]
148 R12/ 2   Carolyn Tajnai <   Information for Forum Speakers
254 R12/30   smarks@sun.com (   another NeWS talk! << ------------ Forwarded
273   1/ 5   Marvin Zauderer    Re: I'll offer... <<I'm afraid it's not worth
295   1/ 6   ansa!funcsmalgor   <<Dear Craig, Good to hear from you. Good to h
296+  1/ 7   John Vlissides <   <<
297   1/ 7   tektronix!cae780   Foreign VCR Compatibility Problem
298   1/ 7   Ernest Wood <woo   New name for an old machine.
```

```
Wed Jan  7 1987   9:21
From vlis Wed Jan 7 21:12:30 1987
last pid: 20472;  load averages: 1.83, 1.25, 0.59                    20:29:30
39 processes: 36 sleeping, 1 running, 2 stopped
Cpu states:        user,       nice,       system,       idle
Memory: 3818K ( 603K) real, 6053K ( 994K) virtual,  727K free

  PID USERNAME PRI NICE   SIZE   RES STATE   TIME   WCPU    CPU COMMAND
20396 root       1  -10   277K  131K sleep   0:58 17.68% 17.65% InterViews.qvs
20472 durwoody  51    0   116K   74K run     0:00 40.14%  3.82% top
20424 durwoody  15    0    73K   23K sleep   0:03  1.46%  1.45% dclock
```

```
From: ansa!funcsma
Return-Path: <ansa
Date: Tue, 6 Jan 87 01:
Message-Id: <8701060025
Apparently-To: seismo!h

Dear Craig,

Good to hear from you.

Good to hear from you
Angeles, where my paren
— I certainly did on
We went to a theatre p
into Night, Cats, and Les Miserables.

. No, but several others, Chess, whic
  guys of Abba fame and starring Tom
  Opera singer with a tremendous voic
  rock onwards.  Also 42nd street and
  and enjoyed it a lot, also some oth
```

```
s/horizon 1
n 2 mipex% 1
ll        tag
nter      tagdone      tagsignals      tagtim2

o open (cancel):
```

☒ Horizontal Scroller

○ wide ● normal ○ narrow

● below ○ above

☒ Vertical Scroller

○ wide ● normal ○ narrow

● right ○ left

OK

Summary

Ease of building interfaces

- View library, composition

Interface quality and consistency

- View library

Extensibility

- Common View protocol

Customizability

- Parameterized Views, meta-Views

Efficiency

- Buffering, caching, localization

Incrementally Linking Programs

Russell W. Quong

Minimizing program link time improves program turnaround time, thus increasing productivity. We have implemented an incremental linker, **inclink**, which substantially reduces link time.

The design of **inclink** is based on a quantitative profile of how programs are compiled and linked during development and maintenance. We instrumented the UNIX utility **make** to record program compilations and links. Using the collected data, we have determined distributions of how much time programmers spend waiting for compiling and linking, how many modules are compiled each time a program is linked, and how the sizes of compiled modules change. Our measurements show that most programs are relinked after only one or two modules are recompiled regardless of program size, and that over 90% of all recompilations yield object code that is less than 100 bytes larger in size.

Inclink keeps all symbol and relocation information in memory in dependency/use lists and uses a fully incremental algorithm that minimizes access to the file system. Relink time is proportional to the size of the change, but not the size of the program. In practice, **Inclink** is 4 to 40 times faster than the normal Unix linker, **ld**.

Inclink, an incremental program linker

Motivation

- Reduce program turnaround time.
- More computing resources - trade space for time.

Design

- Measure how programs are compiled and linked.
- Incremental representation - dependency/use lists.

Results

- Program size vs. Link time.
- Significantly faster relinks.
- Server for other tools (e.g. debugger)

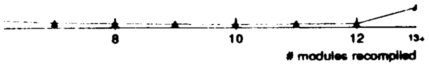
Measurements

- How many modules are there in the average program?
- How large is the average module?
- How many modules are compiled each time a program is linked?
- How much does a module's object code size change?
- How long does the average program take to compile and link?
- How much time passes between make sessions for the same program?
- How do turnaround time, link time, object code size and number of modules correlate with each other?

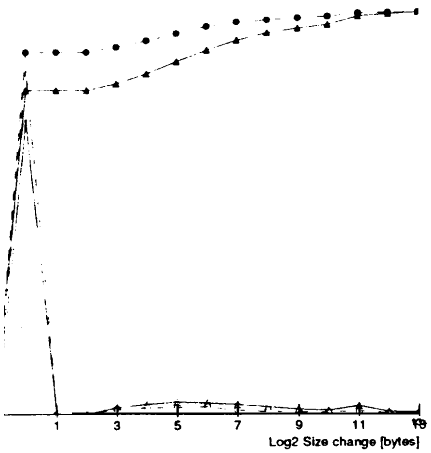
Summary of commands and data

```
# of users: 94
# Programs (dirs) : 818
# of makes : 13117
# commands = 53545, # ignored = 3182
Avg time per make (sec) : 66
```

Command	Runs	Files	Time: User	Sys	Total	Per Run
cc:					187202	17
CC:	10610 5714	10610 5714	161415	25787	177623	31
			152254	25368		31
f77d:	2[illegible]	26[illegible]	19311	2621	82989	71
pc:			161480	21508	08659	
ld:	13880 1486	159453 1486	99964	12: 8695	63[illegible]34	73 13



ıodules recompiled per make



:hange in file size

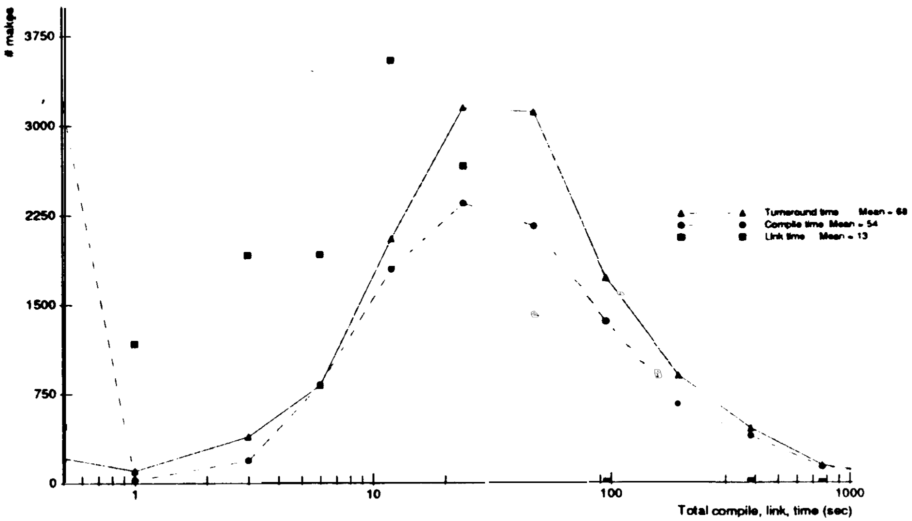


Figure 3: Total turnaround time

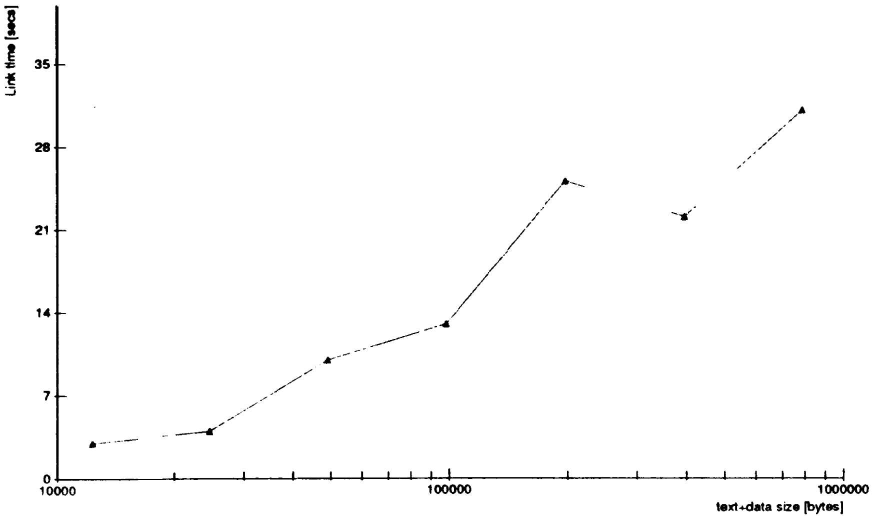
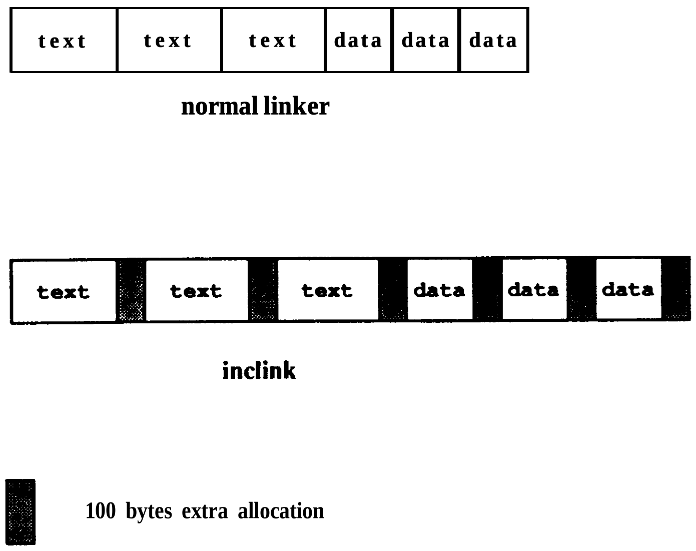
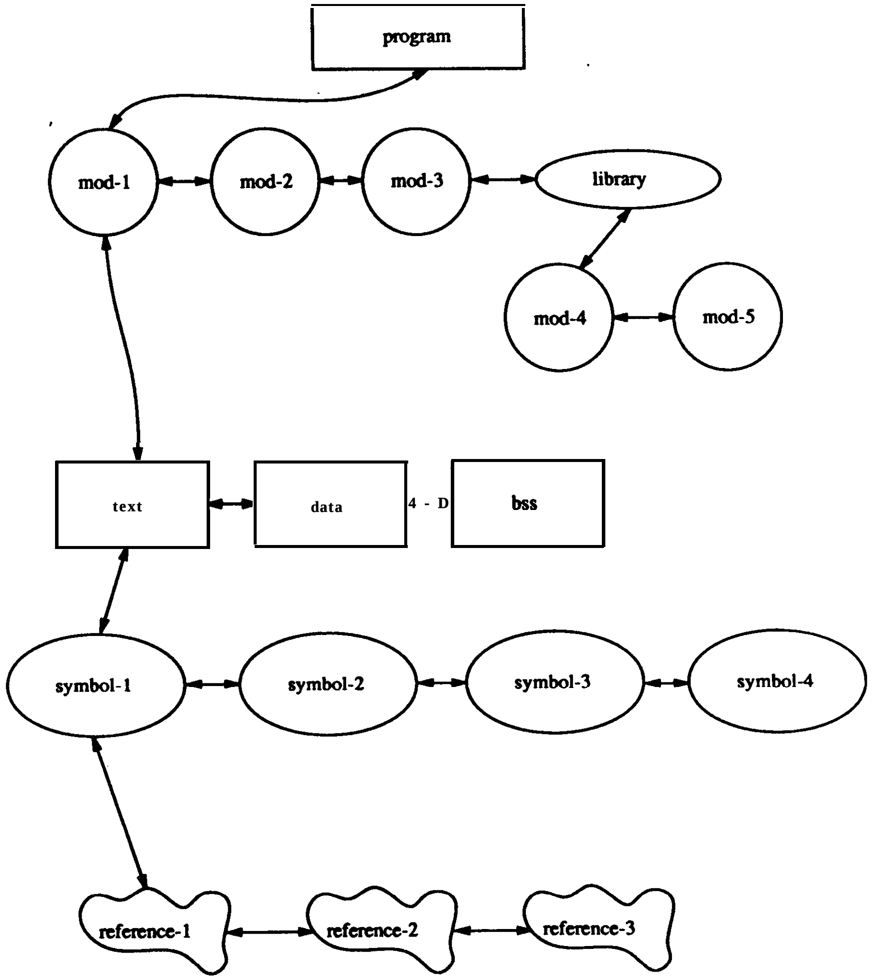


Figure 4: Link time vs. final executable object size

Memory allocation of segments





Data Structures of **inclink**

Results: inclink vs. ld

Link times (in seconds)

program (# mod, text size)	*ld*	*inclink*		
		1st time	1 mod A	3 mod A
simple (2 mod, 1K)	0.3	2.6	0.32	na
inclink (1.5 mod/lib, 40K)	7.9	8.8	0.97	1.89
idraw (35 mod/lib, 170K)	36.5	32.3	0.81	2.40

Inclink speed up factor (Id = 1.00)

program (# mod, text. size)	*ld*	*inclink*		
		1st time	1 mod Δ	3 mod A
simple (2 mod, 1K)	(1.0)	(0.1)	(0.9)	na
inclink (15 mod/lib, 40K)	(1.0)	(0.9)	(8.1)	(4.2)
idraw (35 mod/lib, 170K)	(1.0)	(1.1)	(45.1)	(15.2)

na = not applicable (only 2 modules).
Preliminary data.

Breakdown of inclink running time.

All times in seconds.

inclink (15 modules/libraries, 40K text)

phase	startup	1 mod Δ	3 mod Δ
read modules	7.51	0.72	1.44
relocation	0.50	0.06	0.17
write obj file	0.69	0.15	0.24
total	8.80	0.98	1.89

total time [system time/user time].

idraw (35 modules/libraries, 170K text)

phase	startup		1 mod Δ		3 mod Δ	
read modules	28.60	8.3/20.2	0.62	0.1/0.5	2.00	0.8/1.2
relocation	1.76	0.0/ 1.7	0.02	0.0/0.0	0.16	0.0/0.2
write obj file	1.79	1.6/ 0.2	0.13	0.1/0.0	0.22	0.2/0.0
total	32.34	10.1/22.2	0.81	0.2/0.7	2.44	1.0/1.4

Preliminary data.

Conclusions + Future Work

- Program changes usually small.
  - Independent of program size.
  - Small change in code size.
  - Few modules changed.
- Incremental linking
  - Link time ~ size of change
  - 10X - 40X faster for large programs
- Integrate into rest of programming envronment
- Dynamic loading.