100

The First 99 Reports

Compiled by James Mason

June 5, 1993

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Systems Research Center

The charter of SRC is to advance both the state of knowledge and the state of the art in computer systems. From our establishment in 1984, we have performed basic and applied research to support Digital's business objectives. Our current work includes exploring distributed personal computing on multiple platforms, networking, programming technology, system modelling and management techniques, and selected applications.

Our strategy is to test the technical and practical value of our ideas by building hardware and software prototypes and using them as daily tools. Interesting systems are too complex to be evaluated solely in the abstract; extended use allows us to investigate their properties in depth. This experience is useful in the short term in refining our designs, and invaluable in the long term in advancing our knowledge. Most of the major advances in information systems have come through this strategy, including personal computing, distributed systems, and the Internet.

We also perform complementary work of a more mathematical flavor. Some of it is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. Other work explores new ground motivated by problems that arise in our systems research.

We have a strong commitment to communicating our results; exposing and testing our ideas in the research and development communities leads to improved understanding. Our research report series supplements publication in professional journals and conferences. We seek users for our prototype systems among those with whom we have common interests, and we encourage collaboration with university researchers.

Robert W. Taylor, Director

The First 99 Reports

Compiled by James Mason

June 5, 1993

Abstract

This 100th issue in the SRC Research Report series contains indexed abstracts of the previous ninety-nine, with book and journal source information. It also documents what software SRC makes freely available for research and educational use.

Contents

Preface

We at the Systems Research Center value opportunities to involve our colleagues both inside and outside Digital in the work we do. Communicating our ideas and results freely is an important part of fostering cooperation with this wider technical community. We publish our results in technical journals and conference proceedings, but our most identifiable contribution to the computer science literature is the SRC research report series. Our reports are readily available to universities, libraries, computer science laboratories, and individual researchers throughout the world.

SRC Report 100 contains the abstracts and title-page cartoons of the previous ninety-nine, together with an index, and gives information about software that we make available for others to use and experiment with.

The chief influence shaping the SRC research reports has been a determined little Englishwoman named Cynthia Hibbard, who has served as editor for the series almost since its beginning.

On each report Cynthia has played a slightly different role, but always as a champion for the interests of the reader. Sometimes she has helped her authors get their thoughts organized so that the reader can follow them. Sometimes she has helped her authors rewrite their sentences and paragraphs so that the reader can understand them. All too often she has struggled against recalcitrant typesetting programs so that the reader can in fact read.

Cynthia has a clear vision of what she calls the "story line" of each report, and she works to free the story line as Michelangelo worked to free a statue from the stone. She chips away at bluster, muddle, and reticence, one notch at a time. We are lucky to have her as a colleague.

Many of the cartoons included here were drawn by our colleague Jorge Stolfi. What Jorge started in a light-hearted way with SRC Report 4 quickly became a tradition. Other authors solicited cartoons from Jorge, and, encouraged by his example, closet cartoonists at SRC have honed their drawing skills to provide decoration for some of their own research reports.

Coincidentally, this 100th report in our series comes as SRC nears the end of its first decade. There are still many challenges and opportunities ahead for systems research. We look forward to maintaining productive relations with our colleagues, and continuing to advance the state of knowledge in this exciting field.

1 Abstracts of SRC Research Reports 1-99

- SRC Research Report 1

  A Kernel Language for Modules and Abstract Data Types
  R. Burstall and B. Lampson
  September 1, 1984. 51 pages.

  A small set of constructs can simulate a wide variety of apparently distinct features in modern programming languages. Using a kernel language called Pebble based on the typed lambda-calculus with bindings, declarations, and types as first-class values, we show how to build modules, interfaces and implementations, abstract data types, generic types, recursive types, and unions. Pebble has a concise operational semantics given by inference rules.

  Also in: *Information and Computation,* February 1988, Volume 76, Numbers 2 and 3.

- SRC Research Report 2

  Optimal Point Location in a Monotone Subdivision
  Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi
  October 25, 1984. 33 pages.

  Point location, often known in graphics as *hit detection,* is one of the fundamental problems of computational geometry. In a point location query we want to identify which of a given collection of geometric objects contains a particular point. Let S denote a subdivision of the Euclidean plane into monotone regions by a straight-line graph of m edges. In this paper we exhibit a substantial refinement of the technique of Lee and Preparata for locating a point in S based on separating chains. The new data structure, called a *layered dag*, can be built in O(m) time, uses O(m) storage, and makes possible point location in O(log m) time. Unlike previous structures that attain these optimal bounds, the layered dag can be implemented in a simple and practical way, and is extensible to subdivisions with edges more general than straight-line segments.

  Also in: *SIAM Journal on Computing,* Volume 15, Number 2, pp 317-340, May 1 1986.

- SRC Research Report 3

  On Extending Modula-2 for Building Large, Integrated Systems
  Paul Rovner, Roy Levin, John Wick
  January 1, 1985. 46 pages.

  This paper addresses some of the problems of using Modula-2 to develop large programs, that is, programs with more detail than can be managed effectively by one person. The primary weaknesses of Modula-2 for building large systems of concurrent applications that share data structures and code are discussed. A collection of language changes and extensions to strengthen Modula-2 for such applications are described. Experience using the extended language for a large software project suggests that it is good for the cooperative development of large, experimental programs that share memory.

- SRC Research Report 4

  Eliminating go to's while Preserving Program Structure.
  Lyle Ramshaw
  July 15, 1985. 27 pages.

  Suppose that we want to eliminate the local *go to* statements in a PASCAL program by replacing them with multilevel loop exit statements. There is a standard technique for doing so that succeeds if and only if the flow graph of the PASCAL program is reducible. This technique assumes that we don't allow ourselves either to introduce new variables or to replicate code, but that we do allow ourselves to reorder the atomic tests and actions within the text of the program and to rewrite the connecting control structures from scratch. In this paper, we shall investigate the extent to which *go tos* can be replaced with exits while preserving as much as possible of the program's original structure. On the negative side, we shall find that there are programs whose flow graphs are reducible but whose *go tos* cannot be eliminated without reordering their tests and actions. That is, programs with *go tos* can have their atomic elements in some weird static order, an order that doesn't correspond in any structured way to the dynamic flow of control. We shall analyze this situation by augmenting our flow graphs with edges that encode the static order of the atomic elements and then showing that the augmented flow graphs of programs with exits are always reducible. On the positive side, given a program with *go tos* whose augmented flow graph is reducible, we shall show that we can replace its *go tos* with exits while preserving essentially all of its structure. In fact, we can simply delete the *go to* statements and the labels they jump to and insert various exit statements and labeled Repeat-Endloop pairs for them to jump out of, without changing the rest of the program text in any way.

Also in: *Journal of the ACM,* October 1988.

- SRC Research Report 5 (Superseded by Reports 58, 60, 72, 74, 82)

  Larch in Five Easy Pieces.
  J. V. Guttag, J. J. Horning, and J. M. Wing,
  July 24, 1985. 125 pages.

  The Larch Project is developing tools and techniques intended to aid in the productive use of formal specifications. A major part of the Larch Project is a family of specification languages. Each Larch specification has one component written in a language derived from a programming language and another component written in a language independent of any programming language. We call the former *Larch interface languages* and the latter the *Larch Shared Language.* We have gathered together five documents about the Larch family of languages: an overview, an informal description of the Shared Language, a reference manual for the Shared Language, a handbook of specifications written in the Shared Language, and a report on using Larch/CLU, which is one of the interface languages.

- SRC Research Report 6

  A Caching File System for a Programmer's Workstation.
  Michael D. Schroeder, David K. Gifford, and Roger M. Needham
  October 19, 1985. 23 pages.

  This paper describes a workstation file system that supports a group of cooperating programmers by allowing them both to manage local naming environments and to share consistent versions of collections of software. The file system has access to the workstation's local disk and to remote file servers, and provides a hierarchical name space that includes the files on both. Local names can refer to local files or be attached to remote files. Remote files, which also may be referred to directly, are immutable and cached on the local disk. The file system is part of the Cedar experimental programming environment at Xerox PARC and has been in use since late 1983.

Also in: *Communication of the ACM,* March 1988, Volume 31, Number 3.

- SRC Research Report 7

  A Fast Mutual Exclusion Algorithm
  Leslie Lamport
  November 30, 1985. 15 pages.

  A new solution to the mutual exclusion problem is presented that, in the absence of contention, requires only seven memory accesses. It assumes atomic reads and atomic writes to shared registers.

  Also in: *ACM Transactions on Computer Systems,* February 1987, Volume 5, Number 1, pp 1-11.

- SRC Research Report 8

  On Interprocess Communication
  Leslie Lamport
  December 25, 1985. 50 pages.

  A formalism, not based upon atomic actions, for specifying and reasoning about concurrent systems is defined. It is used to specify several classes of interprocess communication mechanisms and to prove the correctness of algorithms for implementing them.

  Also in: *Distributed Computing,* 1986, Number 1, pp 77-101.

- SRC Research Report 9

  Topologically Sweeping an Arrangement
  Herbert Edelsbrunner and Leonidas J. Guibas
  April 1, 1986. 31 pages.

  Sweeping a collection of figures in the Euclidean plane with a straight line is one of the novel algorithmic paradigms that have emerged in the field of computational geometry. In this paper we demonstrate the advantages of sweeping with a topological line that is not necessarily straight. We show how an arrangement of n lines in the plane can be swept over in $O(n^2)$ time and $O(n)$ space by such a line. In the process each element (i.e. vertex, edge, or region) is visited once in a consistent ordering. Our technique makes use of novel data structures which exhibit interesting amortized complexity behavior; the result is an algorithm that improves upon all its predecessors either in the space or the time bounds, as well as being eminently practical. Numerous applications of the technique to problems in computational geometry are given—many through the use of duality transforms. Examples include solving visibility problems, detecting degeneracies in configurations, computing the extremal shadows of convex polytopes, and others. Even though our basic technique solves a planar problem, its applications include several problems in higher dimensions.

Also in: *Journal of Computer and System Sciences,* February 1989, Volume 38, Number 1, pp 165-194.

- SRC Research Report 10

  A Polymorphic lambda-calculus with Type:Type
  Luca Cardelli
  May 1, 1986. 27 pages.

  Type theory has been used in organizing and clarifying programming language features. As such features become more complex we will need more advanced and powerful type systems like Martin-Loef's intuitionistic theory of types.

  This paper investigates the use of such powerful type systems from a programming language perspective. To satisfy language design needs, these type systems must be extended so that their ordinary semantic theories are no longer applicable. A semantics is developed that justifies the extensions of Martin-Loef's type system with recursion and the Type:Type property.

- SRC Research Report 11

  Control Predicates Are Better Than Dummy Variables For Reasoning About Program Control
  Leslie Lamport
  May 5, 1986. 19 pages.

When explicit control predicates rather than dummy variables are used, the Owicki-Gries method for proving safety properties of concurrent programs can be strengthened, making it easier to construct the required program annotations.

Also in: *ACM Transactions on Programming Languages and Systems,* April 1988, Volume 10, Number 2, pp 267-281.

- SRC Research Report 12

  Fractional Cascading
  Bernard Chazelle and Leonidas J. Guibas
  June 23, 1986. 58 pages.

  In computational geometry many search problems and range queries can be solved by performing an iterative search for the same key in separate ordered lists. In Part I of this report we show that, if these ordered lists can be put in a one-to-one correspondence with the nodes of a graph of degree d so that the iterative search always proceeds along edges of that graph, then we can do much better than the obvious sequence of binary searches. Without expanding the storage by more than a constant factor, we can build a data-structure, called a *fractional cascading structure*, in which all original searches after the first can be carried out at only log d extra cost per search. Several results related to the dynamization of this structure are also presented. Part II gives numerous applications of this technique to geometric problems. Examples include intersecting a polygonal path with a line, slanted range search, orthogonal range search, computing locus functions, and others. Some results on the optimality of fractional cascading, and certain extensions of the technique for retrieving additional information are also included.

- SRC Research Report 13

  Retiming Synchronous Circuitry
  Charles E. Leiserson and James B. Saxe
  August 20, 1986. 42 pages.

  This paper shows how the technique of *retiming* can be used to transform a given synchronous circuit into a more efficient circuit under a variety of different cost criteria. We model a circuit as a graph, and we give an $O(|V||E|lg|V|)$ algorithm for determining an equivalent circuit with the smallest possible clock period. We show that the problem of determining an equivalent retimed circuit with minimum state (total number of registers) is polynomial-time solvable. This result yields a polynomial-time optimal solution to the problem of pipelining combinational circuitry with minimum register cost. We also give a characterization of optimal retiming based on an efficiently solvable mixed-integer linear programming problem.

Also in: *Algorithmica,* 1991, Volume 6, Number 1, pp 5-35, 1991.

- SRC Research Report 14

  An $O(n^2)$ Shortest Path Algorithm for a Non-Rotating Convex Body
  John Hershberger and Leonidas J. Guibas,
  November 27, 1986. 33 pages.

  We investigate the problem of moving a convex body in the plane from one location to another while avoiding a given collection of polygonal obstacles. The method we propose is applicable when the convex body is not allowed to rotate. If n denotes the total size of all polygonal obstacles, the method yields an $O(n^2)$ algorithm for finding a shortest path from the initial to the final location. In solving this problem, we develop some new tools in computational geometry that may be of independent interest.

  Also in: *Journal of Algorithms,* 1988, Number 9, pp 18-46.

- SRC Research Report 15

  A Simple Approach to Specifying Concurrent Systems
  Leslie Lamport
  December 25, 1986. 38 pages.

  In the transition axiom method, safety properties of a concurrent system can be specified by programs; liveness properties are specified by assertions in a simple temporal logic. The method is described with some simple examples, and its logical foundation is informally explored through a careful examination of what it means to implement a specification. Language issues and other practical details are largely ignored.

Also in: *Communications of the ACM,* January 1989, Volume 32, Number 1, pp 32-45.

- SRC Research Report 16

  A Generalization of Dijkstra's Calculus
  Greg Nelson
  April 2, 1987. 56 pages.

  This paper gives a self-contained account of a general calculus of program semantics, from first principles through the semantics of recursion. The calculus is like Dijkstra's guarded commands, but without the Law of the Excluded Miracle; like extended dynamic logic, but with a different approximation relation; like a relational calculus studied by deBakker, but with partial relations as well as total relations; like predicative programming, but with a more standard notion of total correctness. The treatment of recursion uses the fixpoint method from denotational semantics.

  Also in: *Transactions on Programming Languages and Systems,* October 1989, Volume 11, Number 4, pp 517-561.

- SRC Research Report 17

  win and sin: Predicate Transformers for Concurrency
  Leslie Lamport,
  May 1, 1987. 30 pages.

  Dijkstra's weakest liberal precondition and strongest postcondition predicate transformers are generalized to the weakest invariant and strongest invariant. These new predicate transformers are useful for reasoning about concurrent programs containing operations in which the grain of atomicity is unspecified. They can also be used to replace behavioral arguments with more rigorous assertional ones.

  Also in: *ACM Transactions of Programming Languages and Systems* July 1990, Volume 12, Number 3, pp 396-428.

- SRC Research Report 18 (Withdrawn)

  Synchronizing Time Servers,
  Leslie Lamport
  June 1, 1987. 34 pages.

  The paper has been withdrawn because, although we still believe its algorithm to be correct, the properties proved are not strong enough to demonstrate the algorithm's correctness.

- SRC Research Report 19

  Blossoming: A Connect-the-Dots Approach to Splines
  Lyle Ramshaw
  June 21, 1987. 172 pages.

  The standard explanations of the theory underlying the Bezier and B-spline curves and surfaces used in computer-aided geometric design are not as simple as they should be, because there is no easy way to tell, from the labels in the diagrams, what geometric relationships hold among the labeled points. This paper proposes a new labeling scheme, based on the work of P. de Casteljau. The key idea is a classical mathematical principle, which we christen the *Blossoming Principle*: a univariate polynomial of degree n is equivalent to a symmetric polynomial in n variables that is linear in each variable separately. Blossoming a Bezier curve or surface provides lucid labels both for its Bezier points and for all of the intermediate points that arise in the de Casteljau Algorithm. Blossoming a spline curve with parametric continuity provides lucid labels for its de Boor points and for the points that arise in the de Boor Algorithm. Spline curves with geometric continuity and spline surfaces with triangular patches present unsolved labeling challenges, however.

- SRC Research Report 20

  Synchronization Primitives for a Multiprocessor: A Formal Specification
  A. D. Birrell, J. V. Guttag, J. J. Horning, R. Levin
  August 20, 1987. 21 pages.

  Formal specifications of operating system interfaces can be a useful part of their documentation. We illustrate this by documenting the

Threads synchronization primitives of the Taos operating system. We start with an informal description, present a way to formally specify interfaces in concurrent systems, and then give a formal specification of the synchronization primitives. We briefly discuss both the implementation and what we have learned from using the specification for more than a year. Our main conclusion is that programmers untrained in reading formal specifications have found this one helpful in getting their work done.

- SRC Research Report 21

  Evolving the UNIX System Interface to Support Multithreaded Programs
  Paul R. McJones and Garret F. Swart
  September 28, 1987. 100 pages.

  Multiple threads (program counters executing in the same address space) make it easier to write programs that deal with related asynchronous activities and that execute faster on shared-memory multiprocessors. Supporting multiple threads places new constraints on the design of operating system interfaces. Part I of this report presents guidelines for designing (or redesigning) interfaces for multithreaded clients. We show how these guidelines were used to design an interface to UNIX-compatible file and process management facilities in the Topaz operating system. Two implementations of this interface are in everyday use: a native one for the Firefly multiprocessor, and a layered one running within a UNIX process. Part II is the actual programmer's manual for the interface discussed in Part I.

- SRC Research Report 22

  Building User Interfaces by Direct Manipulation
  Luca Cardelli
  October 2, 1987. 45 pages.

  User interfaces based on mice, bitmap displays, and windows are becoming commonplace, and there is a growing expectation that all programs, no matter how trivial or how complicated, should present a graphically elegant and sophisticated user interface. Unfortunately, such polished interfaces are normally difficult to build. Our goal is to make these tasks much simpler, so that application builders and even application users can confront them as routine and painless activities.

The approach described in this report achieves this goal by separating the user interface from the application program, as is done in many user interface management systems, and by using a user interface editor to build the interfaces. In a sense, we apply the direct manipulation style characteristic of user interfaces to the very process of building them, as opposed to building them by programming.

- SRC Research Report 23

  Firefly: A Multiprocessor Workstation
  C. P. Thacker, L. C. Stewart, and E. H. Satterthwaite, Jr.
  December 30, 1987. 17 pages.

  The Firefly is a shared-memory multiprocessor workstation that is used as the primary source of computing at the Digital Equipment Corporation Systems Research Center (SRC). Two versions of the Firefly have been built. The first version contains from one to seven MicroVAX 78032 processors, each with a floating point unit and a sixteen kilobyte cache. The caches are coherent, so that all processors see a consistent view of main memory. A system may contain from four to sixteen megabytes of storage. Input-output is done via a standard DEC QBus. Input-output devices are an Ethernet controller, fixed disks, and a monochrome 1024 x 768 display with keyboard and mouse. Optional hardware includes a high resolution color display and a controller for high capacity disks. The second version of the Firefly contains faster CVAX 78034 processors, sixty-four kilobyte caches, and a main memory of up to 128 megabytes.

  The Firefly runs a software system that emulates the Ultrix system call interface. It also supports medium and coarse-grained multiprocessing through multiple threads of control in a single address space. Communication is implemented uniformly through the use of remote procedure calls.

  This report describes the goals, architecture, implementation, and performance analysis of the Firefly. It then presents some measurements of hardware performance, and concludes with some brief remarks on the evolution of the software.

Also in: *IEEE Transactions on Computers,* August 1988, Volume 37, Number 8, pp 909-920.

- SRC Research Report 24

  A Simple and Efficient Implementation for Small Databases
  Andrew D. Birrell, Michael B. Jones, and Edward P. Wobber
  January 30, 1988. 13 pages.

  This paper describes a technique for implementing the sort of small databases that frequently occur in the design of operating systems and distributed systems. We take advantage of the existence of very large virtual memories, and quite large real memories, to make the technique feasible. We maintain the database as a strongly typed data structure in virtual memory, record updates incrementally on disk in a log, and occasionally make a checkpoint of the entire database. We recover from crashes by restoring the database from an old checkpoint, then replaying the log. We use existing packages to convert between strongly typed data objects and their disk representations, and to communicate strongly typed data across the network (using remote procedure calls). Our memory is managed entirely by a general purpose allocator and garbage collector. This scheme has been used to implement a name server for a distributed system. The resulting implementation has the desirable property of being simultaneously simple, efficient, and reliable.

- SRC Research Report 25

  Real-time Concurrent Collection on Stock Multiprocessors
  John R. Ellis, Kai Li, and Andrew W. Appel
  February 14, 1988. 24 pages.

  We've designed and implemented a copying garbage-collection algorithm that is efficient, real-time, concurrent, runs on commercial uniprocessors and shared-memory multiprocessors, and requires no change to compilers. The algorithm uses standard virtual-memory hardware to detect references to it "from space" objects and to synchronize the collector and mutator threads. We've implemented and measured a prototype running on SRC's 5-processor Firefly. It will be straightforward to merge our techniques with generational collection.

An incremental, non-concurrent version could be implemented easily on many versions of Unix.

- SRC Research Report 26

  Parallel Compilation on a Tightly Coupled Multiprocessor
  Mark Thierry Vandevoorde
  March 1, 1988. 87 pages.

  This thesis describes a C compiler that runs in parallel on a tightly coupled multiprocessor. The compiler, called PTCC, consists of a two-stage pipeline. The first stage performs extended lexical analysis for the second stage, which does the parsing and assembly code generation. The second stage processes units of the source program concurrently. Units as fine as a single statement are compiled in parallel.

  To avoid unproductive concurrency, a new scheduling abstraction, called WorkCrew, is used in PTCC. In the WorkCrew model of computation, the client creates tasks and specifies how they can be subdivided. WorkCrews favor serial execution when parallel execution is unproductive and coarser grains of parallelism over finer ones.

  Several experiments were done to measure the performance of PTCC. With 5 processors, PTCC performed up to 3.3 times better than a similar sequential compiler.

- SRC Research Report 27

  Concurrent Reading and Writing of Clocks
  Leslie Lamport
  April 1, 1988. 7 pages.

  As an exercise in synchronization without mutual exclusion, algorithms are developed to implement both a monotonic and a cyclic multiple-word clock that is updated by one process and read by one or more other processes.

  Also in: *ACM Transactions on Computer Systems,* November 1990, Volume 8, Number 4, pp 305-310.

- SRC Research Report 28

  A Theorem on Atomicity in Distributed Algorithms
  Leslie Lamport
  May 1, 1988. 21 pages.

  Reasoning about a distributed algorithm is simplified if we can ignore the time needed to send and deliver messages and can instead pretend that a process sends a collection of messages as a single atomic action, with the messages delivered instantaneously as part of the action. A theorem is derived that proves the validity of such reasoning for a large class of algorithms. It generalizes and corrects a well known folk theorem about when an operation in a multiprocess program can be considered atomic.

  Also in: *Distributed Computing,* 1990, Volume 4, pp 59-68.

- SRC Research Report 29

  The Existence of Refinement Mappings
  Martín Abadi and Leslie Lamport
  August 14, 1988. 42 pages.

  Refinement mappings are used to prove that a lower-level specification correctly implements a higher-level one. We consider specifications consisting of a state machine (which may be infinite-state) that specifies safety requirements, and an arbitrary supplementary property that specifies liveness requirements. A refinement mapping from a lower-level specification $\mathbf{S_1}$ to a higher-level one $\mathbf{S_2}$ is a mapping from $\mathbf{S_1}$'s state space to $\mathbf{S_2}$'s state space. It maps steps of $\mathbf{S_1}$'s state machine to steps of $\mathbf{S_2}$'s state machine and maps behaviors allowed by $\mathbf{S_1}$ to behaviors allowed by $\mathbf{S_2}$. We show that, under reasonable assumptions about the specifications, if $\mathbf{S_1}$ implements $\mathbf{S_2}$, then by adding auxiliary variables to $\mathbf{S_1}$ we can guarantee the existence of a refinement mapping. This provides a completeness result for a practical, hierarchical specification method.

  Also in: *Theoretical Computer Science,* May 1991, Volume 82, Number 2, pp 253-284.

- SRC Research Report 30

  The Power of Temporal Proofs
  Martín Abadi
  August 15, 1988. 57 pages.

  Some methods for reasoning about concurrent programs and hardware devices have been based on proof systems for temporal logic. Unfortunately, all effective proof systems for temporal logic are incomplete for the standard semantics, in the sense that some formulas hold in every intended model but cannot be proved. We evaluate and compare the power of several proof systems for temporal logic. Specifically, we relate temporal systems to classical systems with explicit time parameters.

  A typical temporal system turns out to be incomplete in a strong sense; we exhibit a short, valid formula it fails to prove. We suggest the addition of new rules to define auxiliary predicates. With these rules, we obtain nonstandard soundness and completeness results. In particular, one of the simple temporal systems we describe is as powerful as Peano Arithmetic.

  Also in: *Theoretical Computer Science,* Volume 65, Number 1, June 1989 and corrigendum in *Theoretical Computer Science,* Volume 70, Number 2, January 1990, page 275.

- SRC Research Report 31 (Superseded by report 52)

  Modula-3 Report
  Luca Cardelli, James Donahue, Lucille Glassman, Mick Jordan, Bill Kalsow, Greg Nelson
  August 24, 1988. 55 pages.

  See also: *Systems Programming with Modula-3,* edited by Greg Nelson. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1991.

- SRC Research Report 32

  Bounds on the Cover Time
  Andrei Broder and Anna Karlin
  October 15, 1988. 22 pages.

Consider a particle that moves on a connected, undirected graph G with n vertices. At each step the particle goes from the current vertex to one of its neighbors, chosen uniformly at random. The cover time is the first time when the particle has visited all the vertices in the graph starting from a given vertex.

In this paper, we present upper and lower bounds that relate the expected cover time for a graph to the eigenvalues of the Markov chain that describes the random walk above. An interesting consequence is that regular expander graphs have expected cover time $\Theta(n \log n)$.

Also in: *Journal of Theoretical Probability,* February 1989, pp 101-120.

- SRC Research Report 33

  A Two-view Document Editor with User-definable Document Structure
  Kenneth P. Brooks
  November 1, 1988. 193 pages.

  Lilac is an experimental document preparation system which combines the best features of batch-style document formatters and WYSIWYG editors. To do this it offers the user two views of the document: a WYSIWYG view and a formatter-like source view. Changes in either view are rapidly propagated to the other. This report describes both the user interface design and the implementation mechanisms used to build Lilac.

- SRC Research Report 34

  Blossoms Are Polar Forms
  Lyle Ramshaw
  January 2, 1989. 46 pages.

  Consider the functions $H(t) := t^2$ and $h(u,v) := uv$. The identity $H(t) = h(t,t)$ shows that H is the restriction of h to the diagonal $u = v$ in the uv-plane. Yet, in many ways, a bilinear function like h is simpler than a homogeneous quadratic function like H. More generally, if $F(t)$ is some n-ic polynomial function, it is often helpful to study the *polar form* of F, which is the unique symmetric, multiaffine function $f(u_1, \ldots, u_n)$ satisfying the identity $F(t) = f(t, \ldots, t)$. The mathematical theory underlying splines is one area where polar forms can be particularly helpful, because two pieces F and G of an n-ic spline meet at r with C^k parametric continuity if and only if their polar forms f and g satisfy

$$f(u_1, /ldots, u_k, \underbrace{r, \ldots, r}_{n-k}) = g(u_1, /ldots, u_k, \underbrace{r, \ldots, r}_{n-k})$$

for all u_1 through u_k.

This polar approach to the theory of splines emerged in rather different guises in three independent research efforts: Paul de Faget de Casteljau called it *shapes through poles*; Carl de Boor called it *B-splines without divided differences*; and Lyle Ramshaw called it *blossoming.* This paper reviews the work of de Casteljau, de Boor, and Ramshaw in an attempt to clarify the basic principles that underlie the polar approach. It also proposes a consistent system of nomenclature as a possible standard.

Also in: *Computer Aided Geometric Design,* November 1989, Volume 6, Number 4, pp 323–358.

- SRC Research Report 35

  An Introduction to Programming with Threads
  Andrew D. Birrell
  January 6, 1989. 35 pages.

  This paper provides an introduction to writing concurrent programs with threads. A threads facility allows you to write programs with multiple simultaneous points of execution, synchronizing through shared memory. The paper describes the basic thread and synchronization primitives, then for each primitive provides a tutorial on how to use it. The tutorial sections provide advice on the best ways to use the primitives, give warnings about what can go wrong and offer hints about how to avoid these pitfalls. The paper is aimed at experienced programmers who want to acquire practical expertise in writing concurrent programs.

- SRC Research Report 36

  Primitives for Computational Geometry
  Jorge Stolfi, January 27, 1989, 228 pages.

  Many geometric algorithms become simpler, more general, and more efficient when recast in the language of projective geometry. Some reasons for this are the uniform handling of points at infinity, the attendant reduction in the number of special cases, and the perfect duality between points and hyperplanes that are possible in the projective model. In fact, the homogeneous coordinates so widely used in computer graphics are essentially an analytical model of classical projective geometry. However, projective space is topologically quite different from Euclidean space. For example, in the projective plane lines have only one side, all triangles have the same handedness, and there are two distinct segments with any given pair of endpoints. These differences are a serious practical problem, since many geometric algorithms depend on orientation, ordering and separation tests that make sense only in the Euclidean model.

  This dissertation describes a slightly modified form of projective geometry which is free from this problem. Analytically, the change consists in making the signs of homogeneous coordinates more significant. Geometrically, the change consists in adopting *oriented* lines and planes as the elementary objects of the model, and redefining the basic geometric operation of meet and join so as to produce results with a definite orientation. Topologically, this is equivalent to working with a double covering projective space, which is equivalent to an n-dimensional sphere.

  The resulting framework, here called *oriented projective geometry*, combines the elegance of classical projective geometry with the ability to talk about oriented lines and planes, signed angles, line segments, convex figures, and many other concepts that cannot be conveniently defined within that model. The goals of this dissertation are: (1) to develop an intuitive understanding of oriented projective geometry in two and three dimensions; (2) to describe a formal geometric calculus for handling oriented lines, planes, and flat spaces of arbitrary dimension; and (3) to investigate the efficient representation of such objects in computers.

- SRC Research Report 37

  Ruler, Compass, and Computer: The Design and Analysis of Geometric Algorithms
  Leonidas J. Guibas and Jorge Stolfi
  February 14, 1989. 55 pages.

  In this paper the authors endeavor to convey the flavor of techniques, especially recent ones, that have been found useful in designing and analyzing efficient geometric algorithms. Each technique is presented by means of a worked out example. The paper presupposes some elementary knowledge of algorithmic geometric techniques and a more advanced knowledge of classical data structures. The aim is to share with the reader some of the excitement that permeates one of the most active areas in theoretical computer science today, namely the field of *Computational Geometry*. The paper is based on a series of lectures delivered at the 1987 NATO Symposium on Theoretical Foundations of Computer Graphics and CAD.

- SRC Research Report 38

  Can fair choice be added to Dijkstra's calculus?
  Manfred Broy and Greg Nelson
  February 16, 1989. 17 pages.

  The paper studies the incorporation of a fair nondeterministic choice operator into a generalization of Dijkstra's calculus of guarded commands. The new operator is not monotonic for the orderings that are generally used for proving the existence of least fixpoints for recursive definitions. To prove the existence of a fixpoint it is necessary to consider several orderings at once, and to restrict the class of recursive definitions.

- SRC Research Report 39

  A Logic of Authentication
  Michael Burrows, Martín Abadi, and Roger Needham
  February 28, 1989. Revised Febuary 22, 1990. 48 pages.

  Questions of belief are essential in analyzing protocols for authentication in distributed computing systems. In this paper we motivate, set out, and exemplify a logic specifically designed for this analysis; we show how various protocols differ subtly with respect to the required initial assumptions of the participants and their final beliefs. Our formalism has enabled us to isolate and express these differences with a precision that was not previously possible. It has drawn attention to features of protocols of which we and their authors were previously unaware, and allowed us to suggest improvements to the protocols. The reasoning about some protocols has been mechanically verified.

This paper starts with an informal account of the problem, goes on to explain the formalism to be used, and gives examples of its application to protocols from the literature, both with conventional shared-key cryptography and with public-key cryptography. Some of the examples are chosen because of their practical importance, while others serve to illustrate subtle points of the logic and to explain how we use it. We discuss extensions of the logic motivated by actual practice—for example, in order to account for the use of hash functions in signatures. The final sections contain a formal semantics of the logic and some conclusions.

In this revised version of the report, we have included a short note, "The Scope of a Logic of Authentication." The aim of the note is to clarify what the logic captures and what it does not capture, and where there is room for other formal or informal techniques. The note is self-contained.

Also in: *ACM Transactions on Computer Systems,* February 1990, Volume 8, Number 1, pp 18-36.
Proceedings of the Royal Society of London, December 1989, Series A, 426, 1871, pp 233-271.

- SRC Research Report 40

  Implementing Exceptions in C
  Eric S. Roberts
  March 21, 1989. 13 pages.

  Traditionally, C programmers have used specially designated return codes to indicate exception conditions arising during program execution. More modern languages offer alternative mechanisms that integrate exception handling into the control structure. This approach has several advantages over the use of return codes: it increases the likelihood that programming errors will be detected, makes it easier to structure the specification of an abstraction, and improves the readability of the implementation by providing better syntactic separation between handling of conventional and exceptional cases. This paper describes a set of language extensions to support exception handling in C, and a preprocessor-based implementation of those extensions that demonstrates both the feasibility and the portability of this approach.

- SRC Research Report 41

  Evaluating the Performance of Software Cache Coherence
  Susan Owicki and Anant Agarwal
  March 31, 1989. 29 pages.

  In a shared-memory multiprocessor with private caches, cached copies of a data item must be kept consistent. This is called *cache coherence*. Both hardware and software coherence schemes have been proposed. Software techniques are attractive because they avoid hardware complexity and can be used with any processor-memory interconnection. This paper presents an analytical model of the performance of two software coherence schemes and, for comparison, snoopy-cache hardware. The model is validated against address traces from a bus-based multiprocessor. The behavior of the coherence schemes under various workloads is compared, and their sensitivity to variations in workload parameters is assessed. The analysis shows that the performance of software schemes is critically determined by certain parameters of the workload: the proportion of data accesses, the fraction of shared references, and the number of times a shared block is accessed before it is purged from the cache. Snoopy caches are more resilient to variations in these parameters. Thus, when evaluating a software scheme as a design alternative, it is essential to consider the characteristics of the expected workload. The performance of the two software schemes with a multistage interconnection network is also evaluated, and it is determined that both scale well.

- SRC Research Report 42

  WorkCrews: An Abstraction for Controlling Parallelism
  Eric S. Roberts and Mark T. Vandevoorde
  April 2, 1989. 17 pages.

  When implementing parallel programs, it is important to find strategies for controlling parallelism that make the most effective use of available resources. In this paper, we introduce a dynamic strategy called WorkCrews for controlling the use of parallelism on small-scale, tightly-coupled multiprocessors. In the WorkCrew model, tasks are assigned to a finite set of workers. As in other mechanisms for specifying parallelism, each worker can enqueue subtasks for concurrent evaluation by other workers as they become idle. The WorkCrew paradigm has two advantages. First, much of the work associated with task division can be deferred until a new worker actually undertakes the subtask, and avoided altogether if the original worker ends up executing the subtask serially. Second, the ordering of queue requests under WorkCrews favors coarse-grained subtasks, which reduces further the overhead of task decomposition.

- SRC Research Report 43

  Performance of Firefly RPC
  Michael D. Schroeder and Michael Burrows
  April 15, 1989. 17 pages.

  In this paper, we report on the performance of the remote procedure call implementation for the Firefly multiprocessor and analyze the implementation to account precisely for all measured latency. From the analysis and measurements, we estimate how much faster RPC could be if certain improvements were made.

  The elapsed time for an inter-machine call to a remote procedure that accepts no arguments and produces no results is 2.66 milliseconds. The elapsed time for an RPC that has a single 1440-byte result (the maximum result that will fit in a single packet) is 6.35 milliseconds. Maximum inter-machine throughput using RPC is 4.65 megabits/second, achieved with 4 threads making parallel RPCs that return the maximum sized single packet result. CPU utilization at maximum throughput is about 1.2 on the calling machine and a little less on the server.

These measurements are for RPCs from user space on one machine to user space on another, using the installed system and a 10 megabit per second Ethernet. The RPC packet exchange protocol is built on IP/UDP, and the times include calculating and verifying UDP checksums. The Fireflies used in the tests had 5 MicroVAX II processors and a DEQNA Ethernet controller.

- SRC Research Report 44

  Pretending Atomicity
  Leslie Lamport and Fred B. Schneider
  May 1, 1989. 29 pages.

  We present a theorem for deriving properties of a concurrent program by reasoning about a simpler, coarser-grained version. The theorem generalizes a result that Lipton proved for partial correctness and deadlock-freedom. Our theorem applies to all safety properties.

- SRC Research Report 45

  Typeful Programming
  Luca Cardelli
  May 24, 1989. 63 pages.

  There exists an identifiable programming style based on the widespread use of type information handled through mechanical typechecking techniques.

  This typeful programming style is in a sense independent of the language it is embedded in; it adapts equally well to functional, imperative, object-oriented, and algebraic programming, and it is not incompatible with relational and concurrent programming.

  The main purpose of this paper is to show how typeful programming is best supported by sophisticated type systems, and how these systems can help in clarifying programming issues and in adding power and regularity to languages.

- SRC Research Report 46

  An Algorithm for Data Replication
  Timothy Mann, Andy Hisgen, and Garret Swart
  June 1, 1989. 55 pages.

  Replication is an important technique for increasing computer system availability. In this paper, we present an algorithm for replicating stored data on multiple server machines. The algorithm organizes the replicated servers in a master/slaves scheme, with one master election being performed at the beginning of each service period. The status of each replica is summarized by a set of monotonically increasing epoch variables. Examining the epoch variables of a majority of the replicas reveals which replicas have up-to-date data. The set of replicas can be changed dynamically. Replicas that have been off-line can be brought up to date in background, and witness replicas, which store the epoch variables but not the data, can participate in the majority voting. The algorithm does not require distributed atomic transactions. The algorithm also permits client machines to cache copies of data, with strict cache consistency being ensured by having the replicated servers keep track of which clients have cached what data. The work reported in this paper is part of an ongoing project to build a new replicated distributed file system with client caching, called Echo.

- SRC Research Report 47

  Dynamic Typing in a Statically Typed Language
  Martín Abadi, Luca Cardelli, Benjamin C. Pierce, Gordon D. Plotkin
  June 10, 1989. 35 pages.

  Statically typed programming languages allow earlier error checking, better enforcement of disciplined programming styles, and generation of more efficient object code than languages where all type-consistency checks are performed at run time. However, even in statically typed languages, there is often the need to deal with data whose type cannot be determined at compile time. To handle such situations safely, we propose to add a type `Dynamic` whose values are pairs of a value `v` and a type tag `T` where `v` has the type denoted by `T`. Instances of Dynamic are built with an explicit tagging construct and inspected with a type-safe `typecase` construct.

This paper explores the syntax, operational semantics, and denotational semantics of a simple language with the type `Dynamic`. We give examples of how dynamically typed values can be used in programming. Then, we discuss an operational semantics for our language and obtain a soundness theorem. We present two formulations of the denotational semantics of this language and relate them to the operational semantics. Finally, we consider the implications of polymorphism and some implementation issues.

Also in: *ACM Transactions on Programming Languages and Systems,* April 1991, Volume 13, Number 2, pp 237-268.

- SRC Research Report 48

  Operations on Records
  Luca Cardelli and John C. Mitchell
  August 25, 1989. 60 pages.

  We define a simple collection of operations for creating and manipulating record structures, where records are intended as finite associations of values to labels. A second-order type system over these operations supports both subtyping and polymorphism. We provide typechecking algorithms and limited semantic models.

  Our approach unifies and extends previous notions of records, bounded quantification, record extension, and parameterization by row-variables. The general aim is to provide foundations for concepts found in object-oriented languages, within a framework based on typed lambda-calculus.

  Also in: *Mathematical Structures in Computer Science,* 1991, Volume 1, pp 3-48.

- SRC Research Report 49

  The Part-Time Parliament
  Leslie Lamport
  September 1, 1989. 41 pages.

  Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems—an approach that has received limited attention because it leads to designs of insufficient complexity.

- SRC Research Reports 50a and 50b

  Report 50a
  An Efficient Algorithm for Finding the CSG Representation of a Simple Polygon
  David Dobkin, Leonidas Guibas, John Hershberger, and Jack Snoeyink
  September 10, 1989. 22 pages.

  Modeling two-dimensional and three-dimensional objects is an important theme in computer graphics. Two main types of models are used in both cases: boundary representations, which represent the surface of an object explicitly but represent its interior only implicitly, and constructive solid geometry representations, which model a complex object, surface and interior together, as a boolean combination of simpler objects. Because neither representation is good for all applications, conversion between the two is often necessary.

  We consider the problem of converting boundary representations of polyhedral objects into constructive solid geometry (CSG) representations. The CSG representations for a polyhedron P are based on the half-spaces supporting the faces of P. For certain kinds of polyhedra this problem is equivalent to the corresponding problem for simple polygons in the plane. We give a new proof that the interior of each simple polygon can be represented by a monotone boolean formula based on the half-planes supporting the sides of the polygon and using each such half-plane only once. Our main contribution is an efficient and practical $O(n \log n)$ algorithm for doing this boundary-to-CSG conversion for a simple polygon of n sides. We also prove that such nice formulæ do not always exist for general polyhedra in three dimensions.

  Videotape 50b
  Boolean Formulae for Simple Polygons
  John Hershberger and Marc H. Brown. Time 6:15.

This videotape shows the boundary-to-CSG conversion algorithm in action, featuring a visualization created with the Zeus algorithm animation system. Multiple color views, updated as the program runs, illustrate different aspects of the algorithm.

- SRC Research Report 51

  Experience with the Firefly Multiprocessor Workstation
  Susan Owicki
  September 15, 1989. 17 pages.

  Commercial multiprocessors are used successfully for a range of applications, including intensive numeric computations, time-sharing, and shared servers. The value of multiprocessing in a single-user workstation is not so obvious, especially in an environment where numeric problems do not dominate. The Digital Equipment Corporation Systems Research Center has had several years of experience using the five-processor Firefly workstation in such an environment. This report is an initial assessment of how much is gained from multiprocessing on the Firefly.

  Reported here are measurements of speedup and utilization for a variety of programs. They illustrate four sources of concurrency: between independent tasks, within a server, between client and server, and within an application. The nature of the parallelism in each example is explored, as well as the factors, if any, that constrain multiprocessing. The examples cover a wide range of multiprocessing, with speedups on a five-processor machine varying from slightly over 1 to nearly 6. Most uses derive most of their speedup from two or three processors, but there are important applications that can effectively use five or more.

- SRC Research Report 52

  Modula-3 Report (revised)
  Luca Cardelli, James Donahue, Lucille Glassman,
  Mick Jordan, Bill Kalsow, Greg Nelson
  November 1, 1989. 71 pages.

  The goal of Modula-3 is to be as simple and safe as it can be while meeting the needs of modern systems programmers. Instead of exploring new features, we studied the features from the Modula family of languages that have proven themselves in practice and tried to simplify them and fit them into a harmonious language. We found that most of the successful features were aimed at one of two main goals: greater robustness, and a simpler, more systematic type system.

Modula-3 descends from Mesa, Modula-2, Cedar, and Modula-2+. It also resembles its cousins Object Pascal, Oberon, and Euclid.

Modula-3 retains one of Modula-2's most successful features, the provision for explicit interfaces between modules. It adds objects and classes, exception handling, garbage collection, lightweight processes (or threads), and the isolation of unsafe features.

The Modula-3 report was published by Olivetti and Digital in August 1988. Implementation efforts followed shortly at both companies. In January 1989, the committee revised the language to reflect the experiences of these implementation teams. The main changes were the introduction of branded reference types, the requirement that opaque types be branded, the legalization of opaque supertypes, and the new flexibility in revealing information about an opaque type.

See also: *System Programming with Modula-3,* Edited by Greg Nelson, Prentice-Hall, Englewood Cliffs, New Jersey, 1991 and *Modula-3,* Samuel P. Harbison, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

- SRC Research Report 53

  IO Streams: Abstract Types, Real Programs
  Mark R. Brown and Greg Nelson
  November 15, 1989. 46 pages.

  The paper proposes standard Modula-3 interfaces for text input and output. It also describes an implementation of the interfaces, focusing on two novel features of Modula-3: the partially opaque type and the explicit isolation of unsafe code.

- SRC Research Report 54

  Explicit Substitutions
  Martín Abadi, Luca Cardelli, Pierre-Louis Curien, Jean-Jacques Levy
  February 6, 1990. 56 pages.

  The lambda, sigma-calculus is a refinement of the lambda-calculus where substitutions are manipulated explicitly. The lambda, sigma-calculus provides a setting for studying the theory of substitutions, with pleasant mathematical properties. It is also a useful bridge between the classical lambda-calculus and concrete implementations.

  Also in: *Journal of Functional Programming,* October 1991, Volume 1, Number 4, pp 375-416.

- SRC Research Report 55

  A Semantic Basis for Quest
  Luca Cardelli and Giuseppe Longo
  February 14, 1990. 51 pages.

  Quest is a programming language based on impredicative type quantifiers and subtyping within a three-level structure of kinds, types and type operators, and values.

  The semantics of Quest is rather challenging. In particular, difficulties arise when we try to model simultaneously features such as contravariant function spaces, record types, subtyping, recursive types, and fixpoints.

  In this paper we describe in detail the type inference rules for Quest, and we give them meaning using a partial equivalence relation model of types. Subtyping is interpreted as in previous work by Bruce and Longo, but the interpretation of some aspects, namely subsumption, power kinds, and record subtyping, is novel. The latter is based on a new encoding of record types.

  We concentrate on modeling quantifiers and subtyping; recursion is the subject of current work.

  Also in: *Journal of Functional Programming,* October 1991, Volume 1, Part 4, pp 417-458.

- SRC Research Report 56

  Abstract Types and the Dot Notation
  Luca Cardelli and Xavier Leroy
  March 10, 1990. 32 pages.

  We investigate the use of the dot notation in the context of abstract types. The dot notation—that is, $a.f$ referring to the operation f provided by the abstraction a—is used by programming languages such as Modula-2 and CLU. We compare this notation with the Mitchell-Plotkin approach, which draws a parallel between type abstraction and (weak) existential quantification in constructive logic. The basic operations on existentials coming from logic give new insights about the meaning of type abstraction, but differ completely from the more familiar dot notation. In this paper, we formalize simple calculi equipped with the dot notation, and relate them to a more classical calculus à la Mitchell and Plotkin. This work provides some theoretical foundations for the dot notation, and suggests some useful extensions.

- SRC Research Report 57 (Superseded by report 79)

  A Temporal Logic of Actions
  Leslie Lamport
  April 1, 1990, 24 pages.

- SRC Research Report 58

  Report on the Larch Shared Language: Version 2.3
  John V. Guttag, James J. Horning, Andrés Modet
  April 14, 1990. 43 pages.

  The Larch family of languages is used to specify program interfaces in a two-tiered definitional style. Each Larch specification has components written in two languages: one that is designed for a specific programming language and another that is independent of any programming language. The former are the *Larch interface languages*, and the latter is the *Larch Shared Language* (LSL). Version 2.3 of LSL is similar to previous versions, but contains a number of refinements based on experience writing specifications and developing tools to support the specification process. This report contains an informal introduction and a self-contained language definition.

This report supersedes Pieces II and III of *Larch in Five Easy Pieces,* SRC Report 5, by J. Guttag, J. Horning, and J. Wing.
Also in:
IEEE Software, September 1985, Volume 2, Number 5.
Science of Computer Programming, March 1986, Volume 6, Number 2, pp 103-134.
ACM Transations on Programming Languages and Systems, January 1987, Volume 9, Number 1.

- SRC Research Report 59

  Autonet: a High-speed, Self-configuring Local Area Network with Point-to-point Links
  Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, Charles P. Thacker
  April 30, 1990. 42 pages.

  Autonet is a self-configuring local area network composed of switches interconnected by 100 Mbit/second, full-duplex, point-to-point links. The switches contain 12 ports that are internally connected by a full crossbar. Switches use cut-through to achieve a packet forwarding latency as low as 2 microseconds per switch. Any switch port can be cabled to any other switch port or to a host network controller.

  A processor in each switch monitors the network's physical configuration. A distributed algorithm running on the switch processors computes the routes packets are to follow and fills in the packet forwarding table in each switch. This algorithm automatically recalculates the forwarding tables to incorporate repaired or new links and switches, and to bypass links and switches that have failed or been removed. Host network controllers have alternate ports to the network and fail over if the active port stops working.

  With Autonet, distinct paths through the set of network links can carry packets in parallel. Thus, in a suitable physical configuration, many pairs of hosts can communicate simultaneously at full link bandwidth. The aggregate bandwidth of an Autonet can be increased by adding more links and switches. Each switch can handle up to 2 million packets per second. Coaxial links can span 100 meters and fiber links can span 2 kilometers.

A 30-switch network with more than 100 hosts is the service network for Digital's Systems Research Center.

Also in: *IEEE Journal on Selected Areas in Communications,* October 1991.

- SRC Research Report 60

  Debugging Larch Shared Language Specifications
  Stephen J. Garland, John V. Guttag, James J. Horning
  July 4, 1990. 34 pages.

  The Larch family of specification languages supports a two-tiered definitional approach to specification. Each specification has components written in two languages: one designed for a specific programming language and another independent of any programming language. The former are called *Larch interface languages*, and the latter the *Larch Shared Language* (LSL).

  The Larch style of specification emphasizes brevity and clarity rather than executability. To make it possible to test specifications without executing or implementing them, Larch permits specifiers to make claims about logical properties of specifications and to check these claims at specification time. Since these claims are undecidable in the general case, it is impossible to build a tool that will automatically certify claims about arbitrary specifications. However, it is feasible to build tools that assist specifiers in checking claims as they debug specifications. This paper describes the checkability designed into LSL and discusses two tools that help perform the checking.

  This paper is a revised and expanded version of a paper presented at the April 1990 IFIP Working Conference on Programming Concepts and Methods.

  Also in: *IEEE Transactions Software Engineering,* September 1990, Volume 16, Number 9, pp 1044-57.

- SRC Research Report 61

  In Memoriam: J.C.R. Licklider 1915-1990
  August 7, 1990. 41 pages.

  In the 1960s, J.C.R. Licklider published his ideas about the future role of multiaccess interactive computing. He looked beyond the existing limitations of punched cards and paper tape to a time when computers would interact in real time with the human user. By performing numerous routine tasks on demand, computers could contribute to a person's ability to formulate new insights and decisions. He saw man-computer interaction as enhancing both the quality and efficiency of human problem solving.

  Articulating his vision was an important contribution in challenging people to examine the implications of an emerging technology. But through his work for the Advanced Research Projects Agency (ARPA), he was also able to give his vision reality. The projects sponsored by his program provided the research direction for computer science in this country for many subsequent years. Furthermore, his program was the first to provide the significant public funding necessary to guarantee the financial stability on which long-term research depended.

  Perhaps his most important influence, however, was in the area of computer science education. Prior to his work at ARPA, there were no departments in US universities offering a PhD in computer science. His program sponsored research at four of the first universities to offer graduate computer science degrees. These departments in turn provided role models for other departments that followed.

  J.C.R. Licklider thus played a central role in initiating and sustaining computer science research and education in this country. To commemorate his important contributions, we reprint here two of his papers, *Man-Computer Symbiosis* and *The Computer as a Communication Device.* In recognition of the debt owed to him by the whole computer science profession, and by every user of interactive computing, we dedicate this report in his memory.

- SRC Research Report 62

  Subtyping Recursive Types
  Roberto M. Amadio and Luca Cardelli
  August 14, 1990. 60 pages.

  We investigate the interactions of subtyping and recursive types in a simply typed lambda-calculus. The two fundamental questions here are whether two (recursive) types are in the subtype relation and whether a term has a type.

  To address the first question, we relate various definitions of type equivalence and subtyping that are induced by a model, an ordering on infinite trees, an algorithm, and a set of type rules. We show soundness and completeness between the rules, the algorithm, and the tree semantics. We also prove soundness and a restricted form of completeness for the model.

  To address the second question, we show that to every pair of types in the subtype relation we can associate a term whose denotation is the uniquely determined coercion map between the two types. Moreover, we derive an algorithm that, given a term with implicit coercions, can infer its least type whenever possible.

- SRC Research Report 63

  Heap Usage in the Topaz Environment
  John D. DeTreville
  August 20, 1990. 42 pages.

  Topaz, the experimental computing environment built and used at SRC, is implemented in the Modula-2+ programming language, which provides garbage collection. Garbage collection simplifies the construction of complex systems, and is tied to a number of other Topaz and Modula-2+ features, such as runtime polymorphism, language safety, information-hiding, object cleanup, persistent objects, and network objects.

  Although there are costs to using garbage collection, these are avoided or tolerated in Topaz. For example, because Topaz must avoid noticeable interruption of service due to garbage collection, it uses a concurrent garbage collector.

Measurements show that the use of the `REF` heap in Topaz is similar in many ways to the use of heaps in Lisp-like environments, but different in others. For example, in typical large programs, the `REF` heap contains millions of bytes, with tens of thousands of objects from among hundreds of statically-declared types; objects of only a few types predominate. Although most objects are small, most bytes are in relatively large objects. Cycles are rare; most cycles are of size 2. Most objects are short-lived, but not as short-lived as in Lisp-like environments that allocate large amounts of ephemeral data on the heap.

- SRC Research Report 64

  Experience with Concurrent Garbage Collectors for Modula-2+
  John DeTreville
  November 22, 1990. 54 pages.

  Garbage collection is an integral component of Modula-2+, the principal systems programming language at SRC. The initial Modula-2+ collector was a concurrent reference-counting collector; it did not reclaim cyclic structures, and the cost of assigning references was relatively high.

  I implemented three experimental collectors for Modula-2+ and tested them to explore alternatives to the initial collector: first a simple concurrent mark-and-sweep collector; then a modified concurrent mark-and-sweep collector that used VM synchronization between the mutator and the collector; and then a concurrent mostly-copying collector that also used VM synchronization.

  These collectors had advantages and disadvantages compared to the initial Modula-2+ collector. They reclaimed cyclic structures and tended to reduce the cost of assignments, but they provoked VM thrashing far more readily and sometimes produced noticeable interruptions of service. For this reason, we adopted a combined reference-counting and mark-and-sweep collector for Modula-2+ at SRC, in which the reference-counting collector reclaims most garbage and the mark-and-sweep collector executes infrequently to reclaim cyclic garbage.

- SRC Research Report 65

  An Axiomatization of Lamport's Temporal Logic of Actions
  Martín Abadi
  October 12, 1990. 18 pages.

  Lamport recently invented a temporal logic of actions suitable for expressing concurrent programs and for reasoning about their computations. In this logic, actions have syntactic representations, which can be combined and analyzed. The basic construct for relating actions and computations is [ ]; a computation satisfies the formula [A] if either the computation has halted or the first action in the computation is an A action. In addition, the language includes the temporal operators "always" and "eventually", and thus it is easy to write both safety and liveness formulas.

  However, the temporal logic of actions is not very expressive in some respects (just expressive enough). One cannot define the "next" and the "until" operators of many previous temporal logics. This is actually a feature, in that formulas with "until" are too often incomprehensible, and "next" violates the important principle of invariance under stuttering.

  A proof system for the logic of actions might be obtained by translating into previous, richer formalisms. In this translation we forfeit the logic and its advantages. A new suit of rules for temporal reasoning with actions is therefore wanted. A complete axiomatization can provide some guidance in choosing and understanding the rules used in practice, and in particular the laws for reasoning about programs.

  In this paper, we study a proof system for a propositional logic, PTLA. After an informal introduction, we define the syntax and semantics of PTLA precisely, and then present our proof system and prove its completeness.

- SRC Research Report 66

  Composing Specifications
  Martín Abadi and Leslie Lamport
  October 10, 1990. 90 pages.

  A rigorous modular specification method requires a proof rule asserting that if each component behaves correctly in isolation, then it behaves correctly in concert with other components. Such a rule is subtle because a component need behave correctly only when its environment does, and each component is part of the others' environments. We examine the precise distinction between a system and its environment, and provide the requisite proof rule when modules are specified with safety and liveness properties.

Also in: *ACM Transactions on Programming Languages and Systems,* January 1993, Volume 15, Number 2, pp 73-268.

- SRC Research Report 67

  Authentication and Delegation with Smart-cards
  M. Abadi, M. Burrows, C. Kaufman, B. Lampson
  October 22, 1990. 24 pages.

  The authentication of users in distributed systems poses special problems because users lack the ability to encrypt and decrypt. The same problems arise when users wish to delegate some of their authority to nodes, after mutual authentication.

  In most systems today, the user is forced to trust the node he wants to use. In a more satisfactory design, the user carries a smart-card with sufficient computing power to assist him; the card provides encryption and decryption capabilities for authentication and delegation.

  Authentication is relatively straightforward with a sufficiently powerful smart-card. However, for practical reasons, protocols that place few demands on smart-cards should be considered. These protocols are subtle, as they rely on fairly complex trust relations between the principals in the system (users, hosts, services). In this paper, we discuss a range of public-key smart-card protocols, and analyze their assumptions and the guarantees they offer.

- SRC Research Report 68

  Trestle Reference Manual
  Mark S. Manasse and Greg Nelson
  December, 1991. 154 pages.

  This is a reference manual for Trestle, a Modula-3 toolkit for the X window system. Trestle is a collection of interfaces structured around a central abstract type: a "virtual bitmap terminal" or VBT, which represents a share of the workstation's screen, keyboard, and mouse—a thing comparable to the viewers, windows, or widgets of other systems.

Trestle is included in SRC Modula-3 version 2.0, which is available via public ftp.

Trestle includes a fairly standard set of interactors, including menus, buttons, "container" classes that provide overlapping or tiled subwindows, and "leaf" windows that display text or other data. This reference manual also specifies the interfaces that allow you to create your own window classes. Knowledge of X is not required.

A Trestle window is an object whose behavior is determined by its methods. For example, a window's response to a mouse click is determined by calling its mouse method. This is fast becoming the standard architecture for toolkits, but Trestle carries it further than most. For example, you can change the way a Trestle window paints by overriding its paint method; this is useful for sophisticated effects like groupware.

Trestle provides a novel strategy for writing applications that are independent of the type of display screen they are running on. For example, it is easy to write a Trestle application that can be moved back and forth between a color display and a monochrome display where the application will look good on both.

- SRC Research Report 69

  Trestle Tutorial
  Mark S. Manasse and Greg Nelson
  May 1, 1992. 70 pages.

  This is a tutorial introduction to programming with Trestle, a Modula-3 window system toolkit currently implemented over the X window system. We assume that you have some experience as a user of window systems, but no previous experience programming with X or other window systems. To run Trestle, you need a copy of SRC Modula-3 and an X server.

  The tutorial begins with examples of programming using built-in Trestle interactors and continues by showing you how to build your own interactors: both leaf interactors and interactors that contain their own sub-windows and modify their behavior.

The source code presented in the tutorial is shipped as part of the Modula-3 release from SRC, in the package "trestletutorial." At SRC, you can fetch a copy of this by typing in your home directory:
`cp -r /proj/m3/pkg/trestletutorial .`
At other sites, you'll have to ask the people who installed SRC Modula-3 where they put the package sources. You will probably want to have a copy of the Trestle Reference Manual (SRC Report 68) nearby as you work through the tutorial.

The first few examples in the tutorial are programs; their source code is reproduced in subdirectories named after that program. The later examples are new classes of interactors. For these, the subdirectories are named after the interactor, and contain both *src* and *test* subdirectories. The *src* directories contain the source code for the interface and implementation of the new interactor, and the *test* directory contains a simple program to exercise the interactor.

- SRC Research Report 70

  A Calculus for Access Control in Distributed Systems
  M. Abadi, M. Burrows, B. Lampson, G. Plotkin
  March 4, 1991. 41 pages.

  We study some of the concepts, protocols, and algorithms for access control in distributed systems, from a logical perspective. We account for how a principal may come to believe that another principal is making a request, either on his own or on someone else's behalf. We also provide a logical language for access control lists, and theories for deciding whether requests should be granted.

- SRC Research Report 71

  Trading Space for Time in Undirected s-t Connectivity
  Andrei Z. Broder, Anna R. Karlin, Prabhakar Raghavan, Eli Upfal
  May 7, 1991. 19 pages.

  Aleliunas *et al.* posed the following question: "The reachability problem for undirected graphs can be solved in logspace and $O(mn)$ time [m is the number of edges and n is the number of vertices] by a probabilistic algorithm that simulates a random walk, or in linear time and space by a conventional deterministic graph traversal algorithm. Is there a spectrum of time-space trade-offs between these extremes?" We answer this question in the affirmative for graphs with a linear number of edges by presenting an algorithm that is faster than the random walk by a factor essentially proportional to the size of its workspace. For denser graphs, our algorithm is faster than the random walk but the speed-up factor is smaller.

- SRC Research Report 72

  LM3: A Larch Interface Language for Modula-3
  A Definition and Introduction, Version 1.0
  Kevin D. Jones
  June 13, 1991. 76 pages.

  This report describes a Larch interface language (LM3) for the Modula-3 programming language. LM3 is a complete example of a Larch interface language and addresses areas previously ignored in interface language definition, such as the specification of non-atomic procedures and object types.

  We give a complete definition of the syntax and illustrate it with some straightforward examples. We also give translation functions from LM3 specifications to Larch Shared Language traits and show their use for type checking. Finally, we present example specifications of standard Modula-3 interfaces.

  To remove the possibility of misunderstanding, this report presents LM3 using its base syntax and does not use any syntactic sugar. In practice, such sugar is convenient and the checker accepts a sugared form as well as the raw form presented here.

- SRC Research Report 73

  Decidability and Expressiveness for First-Order Logics of Probability
  Martín Abadi and Joseph Y. Halpern
  June 18, 1991. 39 pages.

  We consider decidability and expressiveness issues for two first-order logics of probability. In one, the probability is on possible worlds, while in the other, it is on the domain. It turns out that in both cases it takes very little to make reasoning about probability highly undecidable. We show that when the probability is on the domain, if the language contains only unary predicates then the validity problem is decidable. However, if the language contains even one binary predicate, the validity problem is Π_1^2 complete, as hard as elementary analysis with free predicate and function symbols. With equality in the language, even with no other symbol, the validity problem is at least as hard as that for elementary analysis, Π_∞^1 hard. Thus, the logic cannot be axiomatized in either case. When we put the probability on the set of possible worlds, the validity problem is Π_1^2 complete with as little as one unary predicate in the language, even without equality. With equality, we get Π_∞^1 hardness with only a constant symbol. We then turn our attention to an analysis of what causes this overwhelming complexity. For example, we show that if we require rational probabilities then we drop from Π_1^2 to Π_1^1. In many contexts it suffices to restrict attention to domains of bounded size; fortunately, the logics are decidable in this case. Finally, we show that, although the two logics capture quite different intuitions about probability, there is a precise sense in which they are equi-expressive.

- SRC Research Report 74

  Introduction to LCL, A Larch/C Interface Language
  J. V. Guttag and J. J. Horning
  July 24, 1991. 81 pages.

  This report is aimed primarily at the C programmer who wishes to begin to integrate formal specifications into the program development cycle. We present a specification language targeted specifically at C and discuss how it can be used to support a style of C programming in which abstraction plays a vital role.

  The report begins with a quick overview of the use of the Larch family of languages for program specification. It continues with an overview of LCL, a Larch interface language for (ANSI) standard C. It then describes LCL by means of an extended example. Parts of an implementation of the specified interfaces are provided in the body of the report. The remaining parts of the implementation are presented in an appendix. Another appendix contains a brief introduction to the Larch Shared Language.

- SRC Research Report 75

  Zeus: A System for Algorithm Animation and Multi-view Editing
  Marc H. Brown
  February 28, 1992. 23 pages.

  Algorithm animation is a form of program visualization that is concerned with dynamic and interactive graphical displays of a program's fundamental operations. This paper describes the Zeus algorithm animation system. Zeus is noteworthy for its use of objects, strong-typing, and parallelism. Also of interest is how the system can be used for building multi-view editors.

- SRC Research Reports 76a and 76b

  Report 76a
  Color and Sound in Algorithm Animation
  Marc H. Brown and John Hershberger
  August 30, 1991. 31 pages.

  Although systems for animating algorithms are becoming more powerful and easier for programmers to use, not enough attention has been given to the techniques that an algorithm animator needs to create effective visualizations. This paper reviews the techniques for algorithm animation reported in the literature thus far and introduces new techniques that we have developed for using color and, to a lesser extent, sound. The paper also presents six algorithm animations that illustrate the new techniques.

  Also in: *Computer* December 1992, Volume 25, Number 12, pp 52-63.

  Videotape 76b
  An Anthology of Algorithm Animations using Zeus
  Edited by Marc H. Brown
  Time: 59:00

  Contents:
  1. *An Introduction to Zeus*
     Marc H. Brown
  2. *Topologically Sweeping an Arrangement: A Parallel Implementation*
     Marc H. Brown and Harald Rosenberger

3. *Competitive Spinning Algorithms*
   Anna R. Karlin and Marc H. Brown
4. *Boolean Formulae for Simple Polygons*
   John Hershberger and Marc H. Brown
5. *Multilevel Adaptive Hashing*
   Andrei Broder and Marc H. Brown
6. *Compliant Motion in a Simple Polygon*
   John Hershberger

- SRC Research Report 77

  Automatic Reconfiguration in Autonet
  Thomas L. Rodeheffer and Michael D. Schroeder
  September 18, 1991. 47 pages.

  Autonet is a switch-based local area network using 100 Mbit/s full-duplex point-to-point links. Crossbar switches are interconnected to other switches and to host controllers in an arbitrary pattern. Switch hardware uses the destination address in each packet to determine the proper outgoing link for the next step in the path from source to destination. Autonet automatically recalculates these forwarding paths in response to failures and additions of network components. This automatic reconfiguration allows the network to continue normal operation without need of human intervention. Reconfiguration occurs quickly enough that higher-level protocols are not disrupted. This paper describes the fault monitoring and topology acquisition mechanisms that are central to automatic reconfiguration in Autonet.

- SRC Research Report 78

  Using Transformations and Verification in Circuit Design
  James B. Saxe, Stephen J. Garland, John V. Guttag, James J. Horning
  September 25, 1991. 27 pages.

  We show how machine-checked verification can support an approach to circuit design based on transformations. This approach starts with a conceptually simple (but inefficient) initial design and uses a combination of *ad hoc* and algorithmic transformations to produce a design that is more efficient (but more complex).

We present an example in which we start with a simplified CPU design and derive an efficient pipelined form, including circuitry for reverting the effects of partially executed instructions when a successful branch is detected late in the pipeline. The algorithmic stage of our derivation applies a transformation, retiming, that has been proven to preserve functional behavior in the general case. The *ad hoc* stage requires special justification, which we supply in the form of a machine-checked formal verification.

- SRC Research Report 79

  The Temporal Logic of Actions
  Leslie Lamport
  December 25, 1991. 73 pages.

  The temporal logic of actions (TLA) is a logic for specifying and reasoning about concurrent systems. Systems and their properties are represented in the same logic, so the assertion that a system meets its specification and the assertion that one system implements another are both expressed by logical implication. TLA is very simple; its syntax and complete formal semantics are summarized in a little over a page. Yet, TLA is not just a logician's toy; it is extremely powerful, both in principle and in practice. This report introduces TLA and describes how it is used to specify and verify concurrent algorithms. The use of TLA to specify and reason about open systems will be described elsewhere.

- SRC Research Report 80

  An Extension of System F with Subtyping
  Luca Cardelli, Simone Martini, John C. Mitchell, Andre Scedrov
  December 30, 1991. 42 pages.

  System F is a well-known typed lambda-calculus with polymorphic types, which provides a basis for polymorphic programming languages. We study an extension of F, called $F <:$ (pronounced ef-sub) that combines parametric polymorphism with subtyping.

  The main focus of the paper is the equational theory of $F <:$, which is related to PER models and the notion of parametricity. We study some categorical properties of the theory when restricted to closed terms, including interesting categorical isomorphisms. We also investigate proof-theoretical properties, such as the conservativity of typing judgments with respect to F.

We demonstrate by a set of examples how a range of constructs may be encoded in $F <:$. These include record operations and subtyping hierarchies that are related to features of object-oriented languages.

Also in: International Conference on Theoretical Aspects of Computer Software, *Lecture Notes in Computer Science* October 1991, Number 526, pp 750-770, Springer-Verlag, T. Ito and A. R. Meyers (Editors).

- SRC Research Report 81

  Extensible Records in a Pure Calculus of Subtyping
  Luca Cardelli
  January 3, 1992. 44 pages.

  Extensible records were introduced by Mitchell Wand while studying type inference in a polymorphic lambda-calculus with record types. This paper describes a calculus with extensible records, $F <: \rho$, that can be translated into a simpler calculus, $F <:$, lacking any record primitives. Given independent axiomatizations of $F <: \rho$ and $F <:$ (the former being an extension of the latter) we show that the translation preserves typing, subtyping, and equality.

  $F <: \rho$ can then be used as an expressive calculus of extensible records, either directly or to give meaning to yet other languages. We show that $F <: \rho$ can express many of the standard benchmark examples that appear in the literature.

  Like other record calculi that have been proposed, $F <: \rho$ has a rather complex set of rules but, unlike those other calculi, its rules are justified by a translation to a very simple calculus. We argue that thinking in terms of translations may help in simplifying and organizing the various record calculi that have been proposed, as well as in generating new ones.

- SRC Research Report 82

  A Guide to LP, The Larch Prover
  Stephen J. Garland and John V. Guttag
  December 31, 1991. 95 pages.

  This guide provides an introduction to LP (the Larch Prover), Release 2.2. It describes how LP can be used to axiomatize theories in a subset of multisorted first-order logic and to provide assistance in proving theorems. It also contains a tutorial overview of the equational term-rewriting technology that provides, along with induction rules and other user-supplied nonequational rules of inference, part of LP's inference engine.

- SRC Research Report 83

  Authentication in Distributed Systems: Theory and Practice
  Butler Lampson, Martín Abadi, Michael Burrows, Edward Wobber
  February 4, 1992. 45 pages.

  We describe a theory of authentication and a system that implements it. Our theory is based on the notion of principal and a "speaks for" relation between principals. A simple principal either has a name or is a communication channel; a compound principal can express an adopted role or delegation of authority. The theory shows how to reason about a principal's authority by deducing the other principals that it can speak for; authenticating a channel is one important application. We use the theory to explain many existing and proposed mechanisms for security. In particular, we describe the system we have built. It passes principals efficiently as arguments or results of remote procedure calls, and it handles public and shared key encryption, name lookup in a large name space, groups of principals, loading programs, delegation, access control, and revocation.

  Also in: *ACM Transactions on Computer Systems,* November 1992, Volume 13, Number 4, pp 265-310.

- SRC Research Reports 84 and 84b

  Report 84
  Graphical Fisheye Views of Graphs
  Manojit Sarkar and Marc H. Brown
  March 17, 1992. 24 pages.

  A fisheye camera lens is a very wide angle lens that magnifies nearby objects while shrinking distant objects. It is a valuable tool for seeing both local detail and global context simultaneously. This paper describes a system for viewing and browsing graphs using a software analog of a fisheye lens. We first show how to implement such a view using solely geometric transformations. We then describe a more general transformation that allows hierarchical or structured information about the graph to affect the view. Our general transformation is a fundamental extension to previous research in fisheye views.

  Videotape 84b
  Graphical Fisheye Views of Graphs
  Marc H. Brown, James R. Meehan, Manojit Sarkar
  July 1, 1992. Time: 3:51 minutes.

- SRC Research Report 85

  On-line Data Compression in a Log-structured File System
  Michael Burrows, Charles Jerian, Butler Lampson, Timothy Mann
  April 15, 1992. 20 pages.

  We have incorporated on-line data compression into the low levels of a log-structured file system (Rosenblum's Sprite LFS). Each block of data or meta-data is compressed as it is written to the disk and decompressed as it is read. The log-structuring overcomes the problems of allocation and fragmentation for variable-sized blocks. We observe compression factors ranging from 1.6 to 2.2, using algorithms running from 1.7 to 0.4 MBytes per second in software on a DECstation 5000/200. System performance is degraded by a few percent for normal activities (such as compiling or editing), and as much as a factor of 1.6 for file system intensive operations (such as copying multi-megabyte files). Hardware compression devices mesh well with this design.

Chips are already available that operate at speeds exceeding disk transfer rates, which indicates that hardware compression would not only remove the performance degradation we observed, but might well increase the effective disk transfer rate beyond that obtainable from a system without compression.

- SRC Research Report 86

  A Logical View of Composition
  Martín Abadi and Gordon D. Plotkin
  May 1, 1992. 35 pages.

  We define two logics of safety specifications for reactive systems. The logics provide a setting for the study of composition rules. The two logics arise naturally from extant specification approaches; one of the logics is intuitionistic, while the other one is linear.

- SRC Research Reports 87a and 87b

  Report 87a
  Animation of Geometric Algorithms: A Video Review
  Edited by Marc H. Brown and John Hershberger
  June 6, 1992. 23 pages.

  Geometric algorithms and data structures are often easiest to understand visually, in terms of the geometric objects they manipulate. Indeed, most papers in computational geometry rely on diagrams to communicate the intuition behind the results. Algorithm animation uses dynamic visual images to explain algorithms. Thus it is natural to present geometric algorithms, which are inherently dynamic, via algorithm animation.

  Videotape 87b
  Animation of Geometric Algorithms: A Video Review
  Edited by Marc H. Brown and John Hershberger
  June 6, 1992. Time: 70:00.

  This videotape presents a video review of geometric animations; the review was premiered at the 1992 ACM Symposium on Computational Geometry. The review includes single-algorithm animations and sample graphic displays from "workbench" systems for implementing multiple geometric algorithms. The accompanying report contains short descriptions of the algorithm, animation, and implementation techniques used in each video segment.

Contents:

1. *Real-Time Closest Pairs of Moving Points*
   Simon Kahan
2. *The XYZ GeoBench: Animation of Geometric Algorithms*
   Peter Schorn, Adrian Brüngger, Michele De Lorenzi
3. *Optimal Two-Dimensional Triangulations*
   Herbert Edelsbrunner, Roman Waupotitsch
4. *Boolean Formulae for Simple Polygons*
   John Hershberger, Marc H. Brown
5. *SHASTRA: A Distributed and Collaborative Design Environment*
   Chandrajit L. Bajaj
6. *Tetrahedral Break-Up*
   Leonidas Palios, Mark Phillips
7. *Compliant Motion in a Simple Polygon*
   Joseph Friedman
8. *Workbench for Computational Geometry*
   P. Epstein, J. Kavanagh, A. Knight, J. May, T. Nguyen, J.-R. Sack
9. *Topologically Sweeping and Arrangement:*
   A Parallel Implementation
   Marc H. Brown, Harald Rosenberger
10. *The New Jersey Line-Segment-Saw Massacre*
    Ayellet Tal, Bernard Chazelle, David Dobkin

- SRC Research Report 88

  Factors in the Performance of the AN1 Computer Network
  Susan S. Owicki and Anna R. Karlin
  June 15, 1992. 29 pages.

  AN1 (formerly known as Autonet) is a local area network composed of crossbar switches interconnected by 100Mbit/second, full-duplex links. In this paper, we evaluate the performance impact of certain choices in the AN1 design. These include the use of FIFO input buffering in the crossbar switch, the deadlock-avoidance mechanism, cut-through routing, back-pressure for flow control, and multi-path routing. AN1's performance goals were to provide low latency and high bandwidth in a lightly loaded network. In this it is successful. Under heavy load, the most serious impediment to good performance is the use of FIFO input buffers. The deadlock-avoidance technique has an adverse effect on the performance of some topologies, but it seems to be the best alternative, given the goals and constraints of the AN1 design. Cut-through switching performs well relative to store-and-forward switching, even under heavy load. Back-pressure deals adequately with congestion in a lightly-loaded network; under moderate load, performance is acceptable when coupled with end-to-end flow control for bursts. Multi-path routing successfully exploits redundant paths between hosts to improve performance in the face of congestion.

- SRC Research Report 89

  Compositional Refinement of Interactive Systems
  Manfred Broy
  July 15, 1992. 48 pages.

  We use functional specification techniques to describe systems and their components. We define the notions of property refinement and interaction refinement for interactive systems and their components. Interaction refinement allows changes to the syntactic interface (the number of channels and the sorts of messages on the channels) as well as the semantic interface (causality flow between messages and interaction granularity). We prove that these notions of refinement are compositional with respect to sequential and parallel composition, communication feedback, and recursive declarations of system components. These proofs demonstrate that refinements of networks can be accomplished in a modular way by refining their components. We generalize the notions of refinement to refining contexts. Finally, we define full abstraction for specifications and show compositionality with respect to this abstraction as well.

- SRC Research Report 90

  A High-speed DES Implementation for Network Applications
  Hans Eberle
  September 23, 1992. 24 pages.

This paper describes a high-speed data encryption chip implementing the Data Encryption Standard (DES). The DES implementation supports Electronic Code Book mode and Cipher Block Chaining mode. The chip is based on a gallium arsenide (GaAs) gate array containing 50K transistors. At a clock frequency of 250 MHz, data can be encrypted or decrypted at a rate of 1 GBit/second, making this the fastest single-chip implementation reported to date. High performance and high density have been achieved by using custom-designed circuits to implement the core of the DES algorithm. These circuits employ precharged logic, a methodology novel to the design of GaAs devices. A pipelined flow-through architecture and an efficient key exchange mechanism make this chip suitable for low-latency network controllers.

- SRC Research Report 91

  An Old-Fashioned Recipe for Real Time
  Martín Abadi and Leslie Lamport
  October 12, 1992. 67 pages.

  Traditional methods for specifying and reasoning about concurrent systems work for real-time systems. Using TLA (the temporal logic of actions), we illustrate how they work with the examples of a queue and of a mutual-exclusion protocol. In general, two problems must be addressed: avoiding the real-time programming version of Zeno's paradox, and coping with circularities when composing real-time assumption/guarantee specifications. Their solutions rest on properties of machine closure and realizability.

- SRC Research Reports 92a and 92b

  Report 92a
  Hector: Connecting Words with Definitions
  Lucille Glassman, Dennis Grinberg, Cynthia Hibbard James Meehan, Loretta Guarino Reid, Mary-Claire van Leunen
  October 20, 1992. 46 pages.

  Hector is a feasibility study on high-tech corpus lexicography. Oxford University Press provided the lexicographers and a corpus of 20 million words of running English text; Digital Equipment Corporation

Systems Research Center provided the high-tech tools to enable the lexicographers to do all of their work on-line.

The tools provide the ability to query the corpus in various ways and see the resulting matches, to write and edit dictionary entries, and to link each occurrence of a word in the corpus with its sense as displayed in the entry editor. Additional support tools give statistical information about words in the corpus, derivatives and related words, syntactic structures, collocates, and case-variants.

This report describes the tools and the status of the project as of July 1992.

Videotape 92b
Hector: Connecting Words with Definitions
Lucille Glassman, Dennis Grinberg, Cynthia Hibbard, James Meehan, Loretta Guarino Reid, Mary-Claire van Leunen
October 20, 1992. Time: 14:34

- SRC Research Report 93

  Experiences with Software Specification and Verification Using LP, the Larch Proof Assistant
  Manfred Broy
  November 12, 1992. 69 pages.

  We sketch a method for deduction-oriented software and system development. The method incorporates formal machine-supported specification and verification as activities in software and systems development. We describe experiences in applying this method. These experiences have been gained by using the LP, the Larch proof assistant, as a tool for a number of small and medium size case studies for the formal development of software and systems. LP is used for the verification of the development steps. These case studies include:

  - quicksort
  - the majority vote problem
  - code generation by a compiler and its correctness
  - an interactive queue and its refinement into a network

The developments range over levels of requirement specifications, designs and abstract implementations. The main issues are questions of a development method and how to make good use of a formal tool like LP in a goal-directed way within the development. We further discuss of the value of advanced specification techniques, most of which are deliberately not supported by LP and its notation, and their significance in development. Furthermore, we discuss issues of enhancement of a support system like LP and the value and the practicability of using formal techniques such as specification and verification in the development process in practice.

- SRC Research Report 94

  How to Write a Proof
  Leslie Lamport
  February 14, 1993. 12 pages.

  A method of writing proofs is proposed that makes it much harder to prove things that are not true. The method, based on hierarchical structuring, is simple and practical.

- SRC Research Report 95

  Baby Modula-3 and a Theory of Objects
  Martín Abadi
  February 2, 1993. 43 pages.

  Baby Modula-3 is a small, functional, object-oriented programming language. It is intended as a vehicle for explaining the core of Modula-3, from a biased perspective: Baby Modula-3 includes the main features of Modula-3 related to objects, but not much else. To the theoretician, Baby Modula-3 provides a tractable, concrete example of an object-oriented language, and we use it to study the formal semantics of objects.

  Baby Modula-3 is defined with a structured operational semantics and with a set of static type rules. A denotational semantics guarantees the soundness of this definition.

- SRC Research Report 96

  How to Make a Correct Multiprocess Program Execute Correctly on a Multiprocessor
  Leslie Lamport
  February 14, 1993. 10 pages.

  A multiprocess program executing on a modern multiprocessor must issue explicit commands to synchronize memory accesses. A method is proposed for deriving the necessary commands from a correctness proof of the algorithm.

- SRC Research Report 97

  An Implementation of $F <:$
  Luca Cardelli
  February 23, 1993. 49 pages.

  $F <:$ is a highly expressive typed lambda-calculus with subtyping. This paper describes an implementation of $F <:$ (extended with recursive types), and documents the algorithms used. Using this implementation, one can test $F <:$ programs and examine typing derivations.

  To facilitate the writing of complex encodings, we provide a flexible syntax-extension mechanism. New syntax can be defined from scratch, and the existing syntax can be extended on the fly. It is possible to introduce new binding constructs, while avoiding problems with variable capture.

  To reduce the syntactic clutter, we provide a practical type inference mechanism that is applicable to any explicitly typed polymorphic language. Syntax extension and type inference interact in useful ways.

- SRC Research Report 98

  The 1992 SRC Algorithm Animation Festival
  Marc H. Brown
  March 27, 1993. 12 pages.

  During the last two weeks of July 1992, twenty researchers at Digital Equipment Corporation's Systems Research Center participated in the 1st Annual SRC Algorithm Animation Festival. Only two of the researchers had previously animated an algorithm, and not too many more had ever written an application that involved graphics. In this paper, we report on the Animation Festival, describing why we did it and what we did, and commenting on what we learned.

- SRC Research Report 99

  High Speed Switch Scheduling for Local Area Networks
  Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker
  April 26, 1993. 37 pages.

  Current technology trends make it possible to build communication networks that can support high performance distributed computing. This paper describes issues in the design of a prototype switch for an arbitrary topology point-to-point network with link speeds of up to one gigabit per second. The switch deals in fixed-length ATM-style cells, which it can process at a rate of 37 million cells per second. It provides high bandwidth and low latency for datagram traffic. In addition, it supports real-time traffic by providing bandwidth reservations with guaranteed latency bounds. The key to the switch's operation is a technique called *parallel iterative matching*, which can quickly identify a set of conflict-free cells for transmission in a time slot. Bandwidth reservations are accommodated in the switch by building a fixed schedule for transporting cells from reserved flows across the switch; parallel iterative matching can fill unused slots with datagram traffic. Finally, we note that parallel iterative matching may not allocate bandwidth fairly among flows of datagram traffic. We describe a technique called *statistical matching*, which can be used to ensure fairness at the switch and to support applications with rapidly changing needs for guaranteed bandwidth.

2 Ordering Information

2.1 Reports

Many SRC Research Reports are available via anonymous ftp from Internet node:

`gatekeeper.dec.com` (16.1.0.2)

The ftp pathname to them is:

/pub/DEC/SRC/research-reports/

Please read the README file in this directory before retrieving reports.

For DEC sites without IP connectivity, the SRC Reports are also available via DECnet in the directory:

DECWRL::"/pub/DEC/SRC/research-reports"

For hardcopy orders of SRC research reports, please send electronic mail to one of the addresses below and include your full postal address and the number of the report you wish to receive.

`src-report@src.dec.com`
`decsrc::src-report`

Orders may also be placed by sending requests to:

Report Distribution
Digital Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301

2.2 Videotapes

There are currently videotapes available in the Research Report series. They are identified in the abstracts section. Their reference numbers are 50b, 76b, 84b, 87b, and 92b. The contents of tape 50b are now included in tape 76b. All are available in NTSC, PAL, and SECAM formats upon request.

2.3 Software

Software is available via anonymous ftp. The following describes what is available and gives the appropriate directory path on `gatekeeper.dec.com` (16.1.0.2).

- **Modula-3**

  `pub/DEC/Modula-3`:

  Modula-3 is a programming language developed jointly by DEC and Olivetti. The authors describe it as follows:

  The goal of Modula-3 is to be as simple and safe as it can be while meeting the needs of modern systems programmers. Instead of exploring new features, we studied the features of the Modula family of languages that have proven themselves in practice and tried to simplify them into a harmonious language. We found that most of the successful features were aimed at one of two main goals: greater robustness, and a simpler, more systematic type system.

  Modula-3 descends from Mesa, Modula-2, Cedar, and Modula-2+. It also resembles its cousins Object Pascal, Oberon, and Euclid.

  Modula-3 retains one of Modula-2's most successful features, the provision for explicit interfaces between modules. It adds objects and classes, exception handling, garbage collection, lightweight processes (or threads), and the isolation of unsafe features.

  The definition of Modula-3 is contained in the book *System Programming with Modula-3,* edited by Greg Nelson, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1991.

  The relevant Usenet newsgroup is `comp.lang.modula3`. The archives are available via anonymous ftp from `gatekeeper.dec.com` and all in pub/DEC/Modula-3/comp.lang.modula3. If you do not have access to Usenet you can use a relay mailing list; to be added to it, send a message to m3-request@src.dec.com.

There is only one implementation available today. It has been built by SRC and is available via anonymous ftp from gatekeeper.dec.com in pub/DEC/Modula-3/release. Contributed software can be found in pub/DEC/Modula-3/contrib.

- **Larch**

The Larch family of specification languages supports a two-tiered, definitional style of specification for program module interfaces. Each specification has components written in two languages: one language that is designed for a specific programming language and another language that is independent of any programming language. The former kind are called Larch interface languages, and the latter is the Larch Shared Language (LSL).

Readers interested in new developments of the Larch tools should subscribe to the electronic mailing list:

larch-interest@src.dec.com

This list is used for announcements and queries of general interest. Requests to be added to (or deleted from) this list, as well as more specialized queries, should be sent to:

larch-interest-request@src.dec.com.

Updated information on Larch tools is kept online on the Internet host gatekeeper.dec.com. It is available for anonymous ftp as:

/pub/DEC/Larch/Information.tex

A full bibliography on Larch is available by anonymous ftp from Internet host:

`larch.lcs.mit.edu` as `/pub/larch-bib/larch-bib.tex`.

Suggested additions for the online version should be sent to:

`ymtan@lcs.mit.edu`.

For documentation about Larch, and for descriptions of the Larch tools and their use, please refer to the Larch entries in the index of this report. The following book is also available:

Larch: Languages and Tools for Formal Specification, John V. Guttag and James J. Horning (editors), with Stephen J. Garland, Kevin D. Jones, Andrés Modet and Jeannette M. Wing, Springer-Verlag, Texts and Monographs in Computer Science, 1993.

- **Fsub_1.5.0**

  `/pub/DEC/Fsub_1.5.0`

  $F <:$ (pronounced ef-sub) is a typed lambda-calculus with polymorphism and sybtyping. It is a very minimal system, and as such it has been the focus of several theoretical studies. It can be considered as the kernel of the Quest language.

  The FSub system is an implementation of the $F <:$ calculus; it can be used to evaluate and typecheck $F <:$ expressions. It was implemented mostly to test typechecking algorithms for polymorphic languages with subtyping, in a clean setting.

  For details please refer to SRC Research Report 55.

- **Quest**

  `/pub/DEC/Quest`

  Quest is an experimental programming language. It was designed to integrate a number of advanced type-theoretical topics in a coherent language design. The main features are explicit polymorphism, subtyping, and type operators.

  Quest is also available on a UNIX CD-ROM from Prime Time Freeware, 370 Altair Way, Suite 150, Sunnyvale, CA 94086 (408 738 4832).

  For details please refer to Research Report 97.

3 List of SRC Research Reports 1-99

- 1. *A Kernel Language for Modules and Abstract Data Types*
  R. Burstall and B. Lampson
- 2. *Optimal Point Location in a Monotone Subdivision*
  Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi
- 3. *On Extending Modula-2 for Building Large, Integrated Systems*
  Paul Rovner, Roy Levin, John Wick
- 4. *Eliminating go to's while Preserving Program Structure.*
  Lyle Ramshaw
- 5. *Larch in Five Easy Pieces.*
  J. V. Guttag, J. J. Horning, and J. M. Wing
- 6. *A Caching File System for a Programmer's Workstation.*
  Michael D. Schroeder, David K. Gifford, and Roger M. Needham
- 7. *A Fast Mutual Exclusion Algorithm*
  Leslie Lamport
- 8. *On Interprocess Communication*
  Leslie Lamport
- 9. *Topologically Sweeping an Arrangement*
  Herbert Edelsbrunner and Leonidas J. Guibas
- 10. *A Polymorphic lambda-calculus with Type:Type*
  Luca Cardelli
- 11. *Control Predicates are Better Than Dummy Variables for Reasoning About Program Control*
  Leslie Lamport
- 12. *Fractional Cascading*
  Bernard Chazelle and Leonidas J. Guibas
- 13. *Retiming Synchronous Circuitry*
  Charles E. Leiserson and James B. Saxe

- 14. *An $O(n^2)$ Shortest Path Algorithm for a Non-Rotating Convex Body*
  John Hershberger and Leonidas J. Guibas

- 15. *A Simple Approach to Specifying Concurrent Systems*
  Leslie Lamport

- 16. *A Generalization of Dijkstra's Calculus*
  Greg Nelson

- 17. *win and sin: Predicate Transformers for Concurrency*
  Leslie Lamport

- 18. *Synchronizing Time Servers,*
  Leslie Lamport

- 19. *Blossoming: A Connect-the-Dots Approach to Splines*
  Lyle Ramshaw

- 20. *Synchronization Primitives for a Multiprocessor: A Formal Specification*
  A. D. Birrell, J. V. Guttag, J. J. Horning, R. Levin

- 21. *Evolving the UNIX System Interface to Support Multithreaded Programs*
  Paul R. McJones and Garret F. Swart

- 22. *Building User Interfaces by Direct Manipulation*
  Luca Cardelli

- 23. *Firefly: A Multiprocessor Workstation*
  C. P. Thacker, L. C. Stewart, and E. H. Satterthwaite, Jr.

- 24. *A Simple and Efficient Implementation for Small Databases*
  Andrew D. Birrell, Michael B. Jones, and Edward P. Wobber

- 25. *Real-time Concurrent Collection on Stock Multiprocessors*
  John R. Ellis, Kai Li, and Andrew W. Appel

- 26. *Parallel Compilation on a Tightly Coupled Multiprocessor*
  Mark Thierry Vandevoorde

- 27. *Concurrent Reading and Writing of Clocks*
  Leslie Lamport

- 28. *A Theorem on Atomicity in Distributed Algorithms*
  Leslie Lamport
- 29. *The Existence of Refinement Mappings*
  Martín Abadi and Leslie Lamport
- 30. *The Power of Temporal Proofs*
  Martín Abadi
- 31. *Modula-3 Report*
  Luca Cardelli, James Donahue, Lucille Glassman, Mick Jordan, Bill Kalsow, Greg Nelson
- 32. *Bounds on the Cover Time*
  Andrei Broder and Anna Karlin
- 33. *A Two-view Document Editor with User-definable Document Structure*
  Kenneth P. Brooks
- 34. *Blossoms are Polar Forms*
  Lyle Ramshaw
- 35. *An Introduction to Programming with Threads*
  Andrew D. Birrell
- 36. *Primitives for Computational Geometry*
  Jorge Stolfi
- 37. *Ruler, Compass, and Computer: The Design and Analysis of Geometric Algorithms*
  Leonidas J. Guibas and Jorge Stolfi
- 38. *Can fair choice be added to Dijkstra's calculus?*
  Manfred Broy and Greg Nelson
- 39. *A Logic of Authentication*
  Michael Burrows, Martín Abadi, and Roger Needham
- 40. *Implementing Exceptions in C*
  Eric S. Roberts
- 41. *Evaluating the Performance of Software Cache Coherence*
  Susan Owicki and Anant Agarwal

- 42. *WorkCrews: An Abstraction for Controlling Parallelism*
  Eric S. Roberts and Mark T. Vandevoorde
- 43. *Performance of Firefly RPC*
  Michael D. Schroeder and Michael Burrows
- 44. *Pretending Atomicity*
  Leslie Lamport and Fred B. Schneider
- 45. *Typeful Programming*
  Luca Cardelli
- 46. *An Algorithm for Data Replication*
  Timothy Mann, Andy Hisgen, and Garret Swart
- 47. *Dynamic Typing in a Statically Typed Language*
  Martín Abadi, Luca Cardelli, Benjamin C. Pierce, Gordon D. Plotkin
- 48. *Operations on Records*
  Luca Cardelli and John C. Mitchell
- 49. *The Part-Time Parliament*
  Leslie Lamport
- 50a *An Efficient Algorithm for Finding the CSG Representation of a Simple Polygon*
  David Dobkin, Leonidas Guibas, John Hershberger, Jack Snoeyink
- 50b (video)
  Boolean Formulae for Simple Polygons
  John Hershberger and Marc H. Brown
- 51. *Experience with the Firefly Multiprocessor Workstation*
  Susan Owicki
- 52. *Modula-3 Report (revised)*
  Luca Cardelli, James Donahue, Lucille Glassman,
  Mick Jordan, Bill Kalsow, Greg Nelson
- 53. *IO Streams: Abstract Types, Real Programs*
  Mark R. Brown and Greg Nelson

- 54. *Explicit Substitutions*
  Martín Abadi, Luca Cardelli, Pierre-Louis Curien, Jean-Jacques Levy

- 55. *A Semantic Basis for Quest*
  Luca Cardelli and Giuseppe Longo

- 56. *Abstract Types and the Dot Notation*
  Luca Cardelli and Xavier Leroy

- 57. *A Temporal Logic of Actions*
  Leslie Lamport

- 58. *Report on the Larch Shared Language: Version 2.3*
  John V. Guttag, James J. Horning, Andrés Modet

- 59. *Autonet: a High-speed, Self-configuring Local Area Network with Point-to-point Links*
  Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, Charles P. Thacker

- 60. *Debugging Larch Shared Language Specifications*
  Stephen J. Garland, John V. Guttag, James J. Horning

- 61. *In Memoriam: J.C.R. Licklider 1915-1990*

- 62. *Subtyping Recursive Types*
  Roberto M. Amadio and Luca Cardelli

- 63. *Heap Usage in the Topaz Environment*
  John D. DeTreville

- 64. *Experience with Concurrent Garbage Collectors for Modula-2+*
  John DeTreville

- 65. *An Axiomatization of Lamport's Temporal Logic of Actions*
  Martín Abadi

- 66. *Composing Specifications*
  Martín Abadi and Leslie Lamport

- 67. *Authentication and Delegation with Smart-cards*
  M. Abadi, M. Burrows, C. Kaufman, B. Lampson

- 68. *Trestle Reference Manual*
  Mark S. Manasse and Greg Nelson

- 69. *Trestle Tutorial*
  Mark S. Manasse and Greg Nelson

- 70. *A Calculus for Access Control in Distributed Systems*
  M. Abadi, M. Burrows, B. Lampson, G. Plotkin

- 71. *Trading Space for Time in Undirected s-t Connectivity*
  Andrei Z. Broder, Anna R. Karlin, Prabhakar Raghavan, Eli Upfal

- 72. *LM3: A Larch Interface Language for Modula-3*
  A Definition and Introduction, Version 1.0
  Kevin D. Jones

- 73. *Decidability and Expressiveness for First-Order Logics of Probability*
  Martín Abadi and Joseph Y. Halpern

- 74. *Introduction to LCL, A Larch/C Interface Language*
  J. V. Guttag and J. J. Horning

- 75. *Zeus: A System for Algorithm Animation and Multi-view Editing*
  Marc H. Brown

- 76a *Color and Sound in Algorithm Animation*
  Marc H. Brown and John Hershberger

- 76b (video)
  An Anthology of Algorithm Animations using Zeus
  Edited by Marc H. Brown

- 77. *Automatic Reconfiguration in Autonet*
  Thomas L. Rodeheffer and Michael D. Schroeder

- 78. *Using Transformations and Verification in Circuit Design*
  James B. Saxe, Stephen J. Garland, John V. Guttag, James J. Horning

- 79. *The Temporal Logic of Actions*
  Leslie Lamport

- 80. *An Extension of System F with Subtyping*
  Luca Cardelli, Simone Martini, John C. Mitchell, Andre Scedrov
- 81. *Extensible Records in a Pure Calculus of Subtyping*
  Luca Cardelli
- 82. *A Guide to LP, The Larch Prover*
  Stephen J. Garland and John V. Guttag
- 83. *Authentication in Distributed Systems: Theory and Practice*
  Butler Lampson, Martín Abadi, Michael Burrows, Edward Wobber
- 84. *Graphical Fisheye Views of Graphs*
  Manojit Sarkar and Marc H. Brown
- 84b (video)
  Graphical Fisheye Views of Graphs
  Marc H. Brown, James R. Meehan, Manojit Sarkar
- 85. *On-line Data Compression in a Log-structured File System*
  Michael Burrows, Charles Jerian, Butler Lampson, Timothy Mann
- 86. *A Logical View of Composition*
  Martín Abadi and Gordon D. Plotkin
- 87a *Animation of Geometric Algorithms: A Video Review*
  Edited by Marc H. Brown and John Hershberger
- 87b (video)
  Animation of Geometric Algorithms: A Video Review
  Edited by Marc H. Brown and John Hershberger
- 88. *Factors in the Performance of the AN1 Computer Network*
  Susan S. Owicki and Anna R. Karlin
- 89. *Compositional Refinement of Interactive Systems*
  Manfred Broy
- 90. *A High-speed DES Implementation for Network Applications*
  Hans Eberle
- 91. *An Old-Fashioned Recipe for Real Time*
  Martín Abadi and Leslie Lamport

- 92a *Hector: Connecting Words with Definitions*
  Lucille Glassman, Dennis Grinberg, Cynthia Hibbard, James Meehan, Loretta Guarino Reid, Mary-Claire van Leunen

- 92b (video)
  Hector: Connecting Words with Definitions
  Lucille Glassman, Dennis Grinberg, Cynthia Hibbard, James Meehan, Loretta Guarino Reid, Mary-Claire van Leunen

- 93. *Experiences with Software Specification and Verification Using LP, the Larch Proof Assistant*
  Manfred Broy

- 94. *How to Write a Proof*
  Leslie Lamport

- 95. *Baby Modula-3 and a Theory of Objects*
  Martín Abadi

- 96. *How to Make a Correct Multiprocess Program Execute Correctly on a Multiprocessor*
  Leslie Lamport

- 97. *An Implementation of $F <:$*
  Luca Cardelli

- 98. *The 1992 SRC Algorithm Animation Festival*
  Marc H. Brown

- 99. *High Speed Switch Scheduling for Local Area Networks*
  Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker.

4 Acknowledgements

Many people contributed to SRC Report 100.

Bob Taylor provided the environment in which the previous ninety-nine reports were produced.

Cynthia Hibbard worked on the index and provided editorial help, Jim Horning gave timely advice at all stages of production, Lyle Ramshaw assisted with technical graphics issues, and Lucille Glassman contributed to the index.

Martín Abadi, Marc H. Brown, Mike Burrows, Luca Cardelli, John Hershberger, Bill Kalsow, Leslie Lamport, and Paul McJones took time to review the index and make suggestions for improving the document. Allan Heydon provided help with formatting.

Index