Department of Computer Science

THE UNIVERSITY OF ARIZONA
TUCSON ARIZONA

Gould-Simpson Building
Tucson, Arizona 85721

RL-32

January - December 1993

Technical Report List

1993

Abstracts of technical reports available from our department are listed below. If you would like to receive a copy of any of these documents, return the form at the end of this list.

TR 93-01 **Efficient Timestamp Input and Output**
Curtis E. Dyreson and Richard T. Snodgrass
In this paper we provide efficient algorithms for converting between the internal form of a timestamp, and various external forms, principally character strings specifying Gregorian dates. We give several algorithms that explore a range of time and space tradeoffs. Unlike previous algorithms, those discussed here have a constant time cost over a greatly extended range of timestamp values. These algorithms are especially useful in operating systems and in database management systems. (25 pages)

TR 93-02 **On-Line, Alphabet-Independent, Gestural Recognition Using Probabilistic Properties**
Gary L. Newell
We present an approach to alphabet-independent gestural recognition which differs from existing techniques such as curve-matching in that it attempts to approximate the func- tionality of the feature-based approaches while allowing the advantages which result from alphabet independence. Our ap- proach uses a probabilistic model to analyze an alphabet based on user supplied samples and to effectively describe the symbols as a collection of properties or attributes which are either absent or present in the particular symbol with a known probability. These properties are then used to test an unknown symbol and a simple cost function is used to determine the recognition result. (10 pages)

TR 93-05 **Fbufs: A High-Bandwidth Cross-Domain Transfer Facility**
Peter Druschel and Larry L. Peterson
We have designed and implemented a novel operating system facility for I/O buffer management and data transfer across protection domain boundaries on shared memory machines. This facility, called *fast buffers* (fbufs), combines virtual page remapping with shared virtual memory, and exploits locality in I/O traffic to achieve high throughput without compromising protection, security, or modularity. Its goal is to help deliver the high-bandwidth afforded by emerging high-speed networks to user-level processes, both in monolithic and microkernel-based operating systems. (22 pages)

TR 93-06 **A Two-Level Approach to Information Retrieval**
Udi Manber and Sun Wu
A new indexing and query schemes for information retrieval of medium-size natural language text are presented in this paper. The novelty of the algorithms is that they use a very small index - in most cases 2-4% of the size of the text - and still allow very flexible full-text retrieval including the usual Boolean queries but also approximate matching. The ability to perform approximate queries - to search for misspelled keywords - is very powerful. Query times are typically slower than with inverted files, but they are still fast enough for many applications. We also describe a prototype system, called a Personal Information Retrieval System (PIRS) that we developed based on the new algorithms. Although the algorithms we present are general, PIRS was especially designed for personal information as opposed to typical IR systems that are designed for central collections used by many people. By personal information we mean information generated and collected by single users

for their purposes. It can include personal correspondence, articles of interest, e-mail messages, personal notes, bibliographic files, etc. The main characteristic of such information is that it is very non-uniform and includes many types of documents. An IR system for personal information should support low overhead, many types of queries, flexible interaction, and customization, all of which are important features of PIRS. (12 pages)

TR 93-07 **A Text Compression Scheme that Allows Fast Searching Directly in the Compressed File**

Udi Manber

A new text compression scheme is presented in this paper. The main purpose of this scheme is to speed up string matching by searching the compressed file directly. The scheme requires no modification of the string-matching algorithm, which is used as a black box, and any such program can be used. Instead, the pattern is modified; only the outcome of the matching of the modified pattern against the compressed file is decompressed. Since the compressed file is smaller than the original file, the search is faster both in terms of I/O time and processing time than a search in the original file. For typical text files, we achieve about 30% reduction of space and slightly less of search time. A 70% compression is not competitive with good text compression schemes, and thus should not be used where space is the predominant concern. The intended applications of this scheme are files that are searched often, such as catalogs, bibliographic files, and address books. Such files are typically not compressed, but with this scheme they can remain compressed indefinitely, saving space while allowing faster search at the same time. A particular application to an information retrieval system that we developed is also discussed. (12 pages)

TR 93-08 **A Functional and Attribute Based Computational Model for Fault-Tolerant Software**

Masato Suzuki, Takuya Katayama, and Richard D. Schlichting

Programs constructed using techniques that allow software or operational faults to be tolerated are typically written using an imperative computational model. Here, an alternative is described in which such programs are written using a functional and attribute based model called FTAG. This approach offers several advantages, including a declarative style, separation of semantic and syntactic definitions, and the simplicity of a functional foundation. While important for any type of programming, these advantages are especially pronounced for writing fault-tolerant programs that involve the use of state rollback, including the recovery block technique for software faults and checkpointing for operational faults. A pure value reference model is described in which redoing is introduced as a fundamental operation. A formal description of the model is also given, together with an outline of how this model can be implemented in a computer system containing multiple processors. Several rollback-oriented examples are used to illustrate the model. (18 pages)

TR 93-09 **X-Icon: An Icon Window Interface Version 8.10**

Clinton L. Jeffery and Gregg M. Townsend

This document describes the calling interface and usage conventions of X-Icon, a window system interface for the Icon programming language that supports high-level graphical user interface programming. It presently runs on UNIX and VMS systems under Version 11 of the X Window System, and on OS/2 2.0 under Presentation Manager. (52 pages)

TR 93-10 **An Approach to Constructing Modular Fault-Tolerant Protocols**

Matti A. Hiltunen and Richard D. Schlichting

Modularization is a well-known technique for simplifying complex software. Here, an approach to modularizing fault-tolerant protocols such as reliable multicast and membership is described. The approach is based on implementing a protocol's individual properties as separate micro-protocols, and then combining selected micro-protocols using an event-driven software framework; a system is constructed by composing these frameworks with traditional network protocols using standard hierarchical techniques. In addition to simplifying the software, this model helps clarify the dependencies among properties of fault-tolerant protocols, and makes it possible to construct systems that are customized to the specifics of the application or underlying architecture. An example involving reliable group multicast is given, together with a description of a prototype implementation using the SR concurrent programming language. An implementation based on the x-kernel and RT-Mach is also underway. (17 pages)

TR 93-11 **Memory Consistency Models**

David Mosberger

This paper discusses memory consistency models and their influence on software in the context of parallel machines. In the first part we review previous work on memory consistency models. The second part discusses the issues that arise due to weakening memory consistency. We are especially interested in the influence that weakened consistency models have on language, compiler, and run-time system design. We conclude that tighter interaction between those parts and the memory system might improve performance considerably. (10 pages)

TR 93-12 **Call Forwarding: A Simple Interprocedural Optimization Technique for Dynamically Typed Languages**

Koen De Bosschere, Saumya K. Debray, David Gudeman, and Sampath Kannan

This paper discusses *call forwarding*, a simple interprocedural optimization technique for dynamically typed languages. The basic idea behind the optimization is straightforward: find an ordering for the ''entry actions'' of a procedure, and generate multiple entry points for the procedure, such that the savings realized from different call sites bypassing different sets of entry actions, weighted by their estimated execution frequencies, is as large as possible. We show that the problem of computing optimal solutions to arbitrary call forwarding problems is NP-complete, and describe efficient heuristics for the problem. Experimental results indicate that (*i*) the heuristics are effective, in that the solutions produced are generally optimal or close to optimal; and (*ii*) the resulting optimization is effective, in that it leads to significant performance improvements for a number of benchmarks tested. (15 pages)

TR 93-13 **Efficient Support for Fine-Grain Parallelism**

Dawson R. Engler, Gregory R. Andrews, and David K. Lowenthal

It has long been thought that coarse-grain parallelism is much more efficient than fine-grain parallelism due to the overhead of process (thread) creation, context switching, and synchronization. On the other hand, there are several advantages to fine-grain parallelism: ease of programming for many applications, architecture independence, and load-balancing potential. This paper describes techniques that support efficient execution of fine-grain parallel programs on shared-memory multiprocessors. These are implemented in a package called Filaments, which supports three kinds of threads: run-to-completion, barrier (iterative), and fork/join. Efficiency results primarily from making threads stateless, i.e., they have no private stack; this also greatly reduces memory consumption. The gains in performance are such that a fine-grain implementation of Jacobi Iteration with a thread per point is within 11 percent of a coarse-grain program with only one task per processor on a Sequent Symmetry. Execution times for problems with more work per thread—such as matrix multiplication—are usually indistinguishable from coarse-grain programs and can be substantially faster when the amount of work per thread varies. (15 pages)

TR 93-14 **SRWin: A Graphics Library for SR**

Qiang Alex Zhao

This document describes the calling interface and usage conventions of SRWin, a graphics library for the SR concurrent programming language. SRWin provides a simple environment for building interactive graphics system. It currently runs on UNIX under Version 11 of the X Window System. (20 pages)

TR 93-15 **An Abelian Theorem for Completely Monotone Functions**

Peter J. Downey

When an average is taken over any completely monotone function using Poisson weights, the result is asymptotically equal to the completely monotone function evaluated at the Poisson average. Let N be a Poisson random variable with rate $\mathbf{E}N$. Then if g is any function completely monotone on $(0, \infty)$, we show that

$$\mathbf{E}[g(N)] = g(\mathbf{E}N) + O(\mathbf{E}Ng''(\mathbf{E}N)) \qquad \mathbf{E}N \to \infty .$$

For completely monotone functions that do not decay too rapidly (e.g., regularly varying functions), the error term will be of the order of $g(\mathbf{E}N)/\mathbf{E}N$. An application is given to the expectation of

random extremes. Let $Z_{(n)}$ denote the maximum of n independent random variables each with the distribution of Z. Since it can be shown that $\mathbf{E}Z_{(n)}/n$ is completely monotone for any random variable Z with finite expectation, it follows that

$$\mathbf{E}_N[\mathbf{E}_Z Z_{(N)}] = \mathbf{E}Z_{(\mathbf{E}N)} + o(1) \qquad \mathbf{E}N \to \infty ,$$

where $\mathbf{E}_N$ and $\mathbf{E}_Z$ represent expectations with respect to the d.f. of N and Z respectively. (7 pages)

TR93-16 **Interactive Displays for End-Users: A Pluto Tutorial**

Shamim P. Mohamed

This is a tutorial for Pluto, a system that allows end-users---technically sophisticated non-programmers---to design and implement graphical displays of data. Presenting data graphically can often increase its understandability---well-designed graphics can be more effective than a tabular display of numbers. Most visualization systems to date, however, have allowed users to only choose from a small number of pre-defined display methods. They also present a static display---users cannot interact with and explore the data. The more innovative displays and the systems that implement them tend to be extremely specialized, and closely associated with an underlying application. Pluto is a system that enables users to specify the displays to be drawn in a flexible manner. It provides facilities to integrate user-input devices into the display, encouraging an exploratory approach to data understanding. The specification takes the form of a visual language that also provides means for repeating elements of the display a variable number of times based on the amount of data to be displayed, and for conditional structures that can depend on either user input or the data itself. (14 pages)

TR 93-17 **Efficient Evaluation of the Valid-Time Natural Join**

Michael D. Soo, Richard T. Snodgrass, and Christian S. Jensen

Joins are arguably the most important relational operators. Poor implementations are tantamount to computing the Cartesian product of the input relations. In a temporal database, the problem is more acute for two reasons. First, conventional techniques are designed for the optimization of joins with equality predicates, rather than inequality predicates which are prevalent in valid-time queries. Second, the presence of temporally-varying data dramatically increases the size of the database. These factors require new techniques to efficiently evaluate valid-time joins.

We address this need for efficient join evaluation in databases supporting valid-time. A new temporal-join algorithm based on tuple partitioning is introduced. This algorithm avoids the quadratic cost of nested-loop evaluation methods; it also avoids sorting. The algorithm is then adapted to an incremental mode of operation, which is especially appropriate for temporal query evaluation. Performance comparisons between the recomputation algorithm and other evaluation methods are provided. While we focus on the important valid-time natural join, the techniques presented are also applicable to other valid-time joins. (26 pages)

TR 93-18 **Supporting Fault-Tolerant Parallel Programming in Linda**

David E. Bakken and Richard D. Schlichting

Linda is a language for programming parallel applications whose most notable feature is a distributed shared memory called tuple space. While suitable for a wide variety of programs, one shortcoming of the language as commonly defined and implemented is a lack of support for writing programs that can tolerate failures in the underlying computing platform. This paper describes FT-Linda, a version of Linda that addresses this problem by providing two major enhancements that facilitate the writing of fault-tolerant applications: stable tuple spaces and atomic execution of tuple space operations. The former is a type of stable storage in which tuple values are guaranteed to persist across failures, while the latter allows collections of tuple operations to be executed in an all-or-nothing fashion despite failures and concurrency. The design of these enhancements is presented in detail and illustrated by examples drawn from both the Linda and fault-tolerance domains. An implementation of FT-Linda for a network of workstations is also described. The design is based on replicating the contents of stable tuple spaces to provide failure resilience and then updating the copies using atomic multicast. This strategy allows an efficient implementation in which only a single multicast message is needed for each atomic collection of tuple space operations. (29 pages)

TR 93-19 **A Proof Methodology for Verification of Real-Time and Fault-Tolerance Properties of Distributed Programs**

Karen June Hay

From the early days of programming, the dependability of software has been a concern. The development of distributed systems that must respond in real-time and continue to function correctly in spite of hardware failure have increased the concern while making the task of ensuring dependability more complex. This dissertation presents a technique for improving confidence in software designed to execute on a distributed system of fail-stop processors.

The methodology presented is based on temporal logic augmented with time intervals and probability distributions. A temporal logic augmented with time intervals, Bounded Time Temporal Logic (BTTL), supports the specification and verification of real-time properties such as, "The program will poll the sensor every t to T time units." Analogously, a temporal logic augmented with probability distributions, Probabilistic Bounded Time Temporal Logic (PBTTL), supports reasoning about fault-tolerant properties such as, "The program will complete with probability less than or equal to p", and a combination of these properties such as, "The program will complete within t and T time units with probability less than or equal to p."

The syntax and semantics of the two logics, BTTL and PBTTL, are carefully developed. This includes development of a program state model, state transition model, message passing system model and failure system model. An axiomatic program model is then presented and used for the development of a set of inference rules. The inference rules are designed to simplify use of the logic for reasoning about typical programming language constructs and commonly occurring programming scenarios. In addition to offering a systematic approach for verifying typical behaviors, the inference rules are intended to support the derivation of formulas expressing timing and probabilistic relationships between the execution times and probabilities of individual statements, groups of statements, message passing and failure recovery. Use of the methodology is demonstrated in examples of varying complexity, including five real-time examples and four combined real-time and fault-tolerant examples. (268 pages)

TR 93-20 **An Interface for a Fragment Assembly Kernel**

Eugene W. Myers, Susan M. Larson, and Mudita Jain

This document includes a description of a C programming language interface for our Fragment Assembly Kernel. Inputs to the Fragment Assembly Kernel are DNA fragment sequences containing insertion and deletion errors, and optional constraints on fragment assembly such as known fragment overlaps or relative fragment orientation. Fragment sequence version control is also supported. The Fragment Assembly Kernel produces probable reconstructions of the original DNA sequence, subject to any specified constraints. Each fragment assembly includes multiple sequence alignment and consensus sequences. Multiple sequence alignment editing capabilities are provided to allow manual correction of sequence errors. (10 pages)

TR 93-21 **A Framework for Monitoring Program Execution**

Clinton L. Jeffery

Program execution monitors are used to improve human beings' understanding of program run-time behavior in a variety of important applications such as debugging, performance tuning, and the study of algorithms. Unfortunately, many program execution monitors fail to provide adequate understanding of program behavior, and progress in this area of systems software has been slow due to the difficulty of the task of writing execution monitors.

In high-level programming languages the task of writing execution monitors is made more complex by features such as non-traditional control flow and complex semantics. Additionally, in many languages, such as the Icon programming language, a significant part of the execution behavior that various monitors need to observe occurs in the language run-time system code rather than the source code of the monitored program.

This dissertation presents a framework for monitoring Icon programs that allows rapid development of execution monitors in the Icon language itself. Monitors have full source-level access to the target program with which to gather and process execution information, without intrusive modification to the target executable. In addition, the framework supports the monitoring of implicit

run-time system behavior crucial to program understanding.

In order to demonstrate its practicality, the framework has been used to implement a collection of program visualization tools. Program visualization provides graphical feedback about program execution that allows human beings to deal with volumes of data more effectively than textual techniques. Ideally, the user specifies program execution controls in such tools directly in the graphics used to visualize execution, employing the same visual language that is used to render the output. Some monitors that exhibit this characteristic are presented. (121 pages)

TR 93-22 **Testing Eigenvalue Software**

Lehman Edwin Henderson, Jr.

This dissertation describes a significant advance in automated testing of eigenvalue software. Several programs are described that assist the researcher in verifying that a new program is stable. Using backwards error techniques popularized by Wilkinson, a maximizer or "hill climber" systematically searches for instabilities in the program being tested. This work builds on software first reported by Miller and removes the restriction of not being able to work on iterative methods. Testing eigenvalue solver programs with sets of small random input data can often find instabilities, but the described hill climbing technique is more efficient. Using only ten sets of starting points, the maximizer will often find the instability, if it exists, in only a few tries. (297 pages)

TR 93-23 **FT-SR: A Programming Language for Constructing Fault-Tolerant Distributed Systems**

Vicraj Thomas

This dissertation focuses on the area of improving programming language support for constructing fault-tolerant systems. Specifically, the design and implementation of FT-SR, a programming language developed for building a wide variety of fault-tolerant systems, is described. FT-SR is based on the concurrent programming language SR and is designed as a set of extensions to SR.

A distinguishing feature of FT-SR is the flexibility it provides the programmer in structuring fault-tolerant software. It is flexible enough to be used for structuring systems according to any of the standard fault-tolerance structuring paradigms that have been developed for such systems, including the object/action model, the restartable action paradigm, and the state machine approach. This is especially important in systems building because different structuring paradigms are often appropriate for different parts of the system. This flexibility sets FT-SR apart from other fault-tolerant programming languages which provide language support for the one paradigm that is best suited for the class of applications they choose to support. FT-SR, on the other hand, is suitable for programming a variety of systems and applications. FT-SR derives its flexibility from a programming model based on fail-stop atomic objects. These objects execute operations as atomic actions except when a failure or series of failures cause underlying implementation assumptions to be violated; in this case, notification is provided. This dissertation argues that fail-stop atomic objects are the fundamental building blocks for all fault-tolerant programs. FT-SR provides the programmer with simple fail-stop atomic objects, and mechanisms that allow these fail-stop atomic objects to be composed to form higher-level fail-stop atomic objects that can tolerate a greater number of faults. The mechanisms for composing fail-stop atomic objects are based on standard redundancy techniques. This ability to combine the basic building blocks in a variety of ways allows programmers to structure their programs in a manner best suited to the application at hand. FT-SR has been implemented using version 3.1 of the x-kernel and runs standalone on Sun 3s. The implementation is interesting because of the novel algorithms and optimizations used within the language runtime system. (127 pages)

TR 93-24 **A LANGUAGE-BASED APPROACH TO PROTOCOL IMPLEMENTATION**

Mark Bert Abbott

This thesis explores two strategies for supporting the development of network communication software: imposing constraints on protocol design at the specification level, and using a special-purpose language for protocol implementation. It presents a protocol implementation language called Morpheus. Morpheus utilizes the new strategies to provide a higher level of abstraction, finer grain modularity, and greater software reusability than previous approaches.

Morpheus is able to provide a high level of abstraction because of built-in knowledge about its

problem domain. It has a narrow problem domain---network protocols---that is further narrowed by the application of specification-level constraints. One particular constraint---the {\m shapes} constraint, which partitions protocols into three basic kinds---is particularly effective in raising the level of abstraction.

Morpheus's support for modularity and, indirectly, software reuse hinges on reducing the performance penalty for layering. When protocol layering entails a high performance cost, developers are motivated to build complex monolithic implementations that are hard to design, implement, debug, modify, and maintain. Morpheus reduces the performance costs of layering by applying optimizations based on common patterns of protocol execution. If the degree of modularity is held fixed, then the optimizations simply improve performance. An optimization based on Integrated Layer Processing is particularly noteworthy for its dramatic contribution to network throughput while preserving modularity. (125 pages)

TR 93-25 **A Notation for the Visual Specification of Geometric Relations in Rule-Based User Interface Development Environments**

Andrey Yeatts and Scott Hudson

This paper describes a new visual notation for specifying geometric relationships (for example, in a user interface presentation application). This notation is designed to provide a centerpiece for the visual specification of predicates and rules in the BluePrint rule-based user interface development environment. The notation, while simple and using only a handful of operator symbols, is extremely powerful and expressive. It operates in an intuitive fashion using analogies to the physically based concepts of alignment and measurement to express a wide range of linear relationships between objects, and can be used to express dynamic as well as static properties. (11 pages)

TR 93-26 **Performance Experiments for the Filaments Package**

David K. Lowenthal and Dawson R. Engler

Ten representative benchmarks were run on two shared-memory multiprocessors using an efficient, fine-grain threads package called Filaments. This paper describes the implementation and performance of the applications and compares them to both coarse-grain and sequential counterparts. It also analyzes the results and explains why the fine-grain programs were faster or slower than the coarse-grain ones. (29 pages)

TR 93-27 **Representing Type Information in Dynamically Typed Languages**

David Gudeman

This report is a discussion of the various techniques for representing type information in dynamically typed languages, as implemented on general-purpose machines (and costs are discussed in terms of modern RISC machines). It is intended to make readily available a large body of knowledge that currently has to be absorbed piecemeal from the literature or re-invented by each language implementer. This discussion covers not only tagging schemes but other forms of representation as well, although the discussion is strictly limited to the representation of type information. It should also be noted that this report does not purport to contain a survey of the relevant literature. Information on dynamic type representation is widely scattered, and generally buried in large reports that concentrate on other aspects of language implementations. It would be a truly monumental task to try to uncover all of this information. Instead, this report gathers together a body of folklore, organizes it into a logical structure, makes some generalizations, and then discusses the results in terms of modern hardware. (40 pages)

TR 93-28 **Discrete Pattern Matching over Sequences and Interval Sets**

James Robert Knight

Finding matches, both exact and approximate, between a sequence of symbols *A* and a pattern *P* has long been an active area of research in algorithm design. Some of the more well-known byproducts from that research are the *diff* program and *grep* family of programs. These problems form a sub-domain of a larger area of problems called *discrete pattern matching* which has been developed recently to characterize the wide range of pattern matching problems. This dissertation presents new algorithms for discrete pattern matching over sequences and develops a new sub-domain of problems called discrete pattern

matching over interval sets. The problems and algorithms presented here are characterized by three common features: (1) a ''computable scoring function'' which defines the quality of matches; (2) a graph based, dynamic programming framework which captures the structure of the algorithmic solutions; and (3) an interdisciplinary aspect to the research, particularly between computer science and molecular biology, not found in other topics in computer science.

The first half of the dissertation considers discrete pattern matching over sequences. It develops the alignment-graph/dynamic-programming framework for the algorithms in the sub-domain and then presents several new algorithms for regular expression and extended regular expression pattern matching. The second half of the dissertation develops the sub-domain of discrete pattern matching over interval sets, also called *super-pattern matching.* In this sub-domain, the input consists of sets of typed intervals, defined over a finite range, and a pattern expression of the interval types. A match between the interval sets and the pattern consists of a sequence of consecutive intervals, taken from the interval sets, such that their corresponding sequence of types matches the pattern. The name super-pattern matching comes from those problems where the interval sets corresponds to the sets of sub-strings reported by various pattern matching problems over a common input sequence. The pattern for the super-pattern matching problem, then, represents a ''pattern of patterns,'' or super-pattern, and the sequences of intervals matching the super-pattern correspond to the substring of the original sequence which match that larger ''pattern.'' (91 pages)

TR 93-29 **Configuring Scientific Applications in a Heterogeneous Distributed System**
Patrick T. Homer and Richard D. Schlichting

Current scientific applications are often structured as a collection of individual software components that are manually executed on heterogeneous machines, with files being used to transfer data from one component to the next. Yet despite having the structure of a distributed application from the perspective of configuration management, the techniques and tools that have been used in this domain to address configuration have generally been minimal at best. Here, an approach to configuring scientific applications in a heterogeneous distributed system is described. The focus is on Schooner, an interconnection system that provides the programming model and base technology needed for realizing enhanced configurability. One key aspect of this technology is a machine- and language-independent interface specification that is used to generate interface code to bind components into the application and map them onto suitable host architectures. The other is a runtime system that implements support for both static and dynamic configuration. This paper describes the Schooner application model, outlines the method of creating component interfaces, and describes the runtime system and its various configuration options. (15 pages)

TR 93-30 9n**A Comparison of Implicit and Explicit Parallel Programming**
Vincent W. Freeh

This paper examines the impact of the parallel programming model on scientific computing. A comparison is made between sis, a functional language with implicit parallelism, and SR, an imperative language with explicit parallelism. Both languages are modern, high-level, concurrent programming languages. These two concurrent languages are also compared to programs written in C, using library calls for parallelism. The languages are evaluated by writing programs for six different scientific applications in each language. These programs are compared for programmability and performance on a shared memory multiprocessor. (24 pages)

TR 93-31 **Unifying Temporal Data Models via a Conceptual Model**
Christian S. Jensen, Michael D. Soo, and Richard T. Snodgrass

To add time support to the relational model, both first normal form (1NF) and non-1NF approaches have been proposed. Each has associated difficulties. Remaining within 1NF when time support is added may introduce data redundancy. The non-1NF models may be incapable of directly using existing relational storage structures or query evaluation technologies.

This paper describes a new, conceptual temporal data model that better captures the time-

dependent semantics of the data, while permitting multiple data models at the representation level. This conceptual model effectively moves the distinction between the various existing data models from a semantic basis to a physical, performance-relevant basis.

We define a conceptual notion of a bitemoral relation where tuples are stamped with sets of two-dimensional chronons in transaction-time/ valid-time space. Next, we describe five representation schemes that support both valid and transaction time; these representations include both 1NF and non-1NF models. We use snapshot equivalence to relate the representational data models with the bitemporal conceptual data model.

We then consider querying within the two-level framework. To do so, we define an algebra at the conceptual level. We then map this algebra to the representation level in such a way that new operators compute equivalent results for different representations of the same bitemporal conceptual relation. This demonstrates that all of these representations are faithful to the semantics of the conceptual data model, with many choices available that may be exploited to improve performance. (35 pages)

TR 93-32 **Type Inference in the Icon Programming Language**

Kenneth Walker and Ralph E. Griswold

The type system of the Icon programming language presents several problems for efficient implementation. Variables are untyped and can have values of any type. Structures are first-class values with pointer semantics. A naive implementation checks the types of all operations repeatedly during program execution.

A new optimizing compiler for Icon uses a type inference system to determine the possible types that expressions may have during program execution. This information is used to eliminate unneeded type-checking code. This paper describes this type inferencing system: its model of abstract interpretation, its implementation, its complexity, and its performance in practice. (21 pages)

TR 93-33 **Finding Similar Files in a Large File System**

Udi Manber

We present a tool, called sif, for finding all similar files in a large file system. Files are considered similar if they have significant number of common pieces, even if they are very different otherwise. For example, one file may be contained, possibly with some changes, in another file, or a file may be a reorganization of another file. The running time for finding all groups of similar files, even for as little as 25% similarity, is on the order of 500MB to 1GB an hour. The amount of similarity and several other customized parameters can be determined by the user at a post-processing stage, which is very fast. Sif can also be used to very quickly identify all similar files to a query file using a preprocessed index. Application of sif can be found in file management, information collecting (to remove duplicates), program reuse, file synchronization, data compression, and maybe even plagiarism detection. (10 pages)

TR 93-34 **GLIMPSE: A Tool to Search Through Entire File Systems**

Udi Manber, Sun Wu

GLIMPSE, which stands for GLobal IMPlicit SEarch, provides indexing and query schemes for file systems. The novelty of *glimpse* is that it uses a very small index - in most cases 2-4% of the size of the text - and still allows very flexible full-text retrieval including Boolean queries, approximate matching (i.e., allowing misspellings), and even searching for regular expressions. In a sense, *glimpse* extends agrep to entire file systems, while preserving most of its functionality and simplicity. Query times are typically slower than with inverted indexes, but they are still fast enough for many applications. For example, it took 5 seconds of CPU time to find all 19 occurrences of Usenix AND Winter in a file system containing 69MB of text spanning 4300 files. *Glimpse* is particularly designed for personal information, such as one's own file system. The main characteristic of personal information is that it is non-uniform and includes many types of documents. An information retrieval system for personal information should support many types of queries, flexible interaction, low overhead, and customization. All these are important features of *glimpse*. (10 pages)

TR 93-35 **Supporting Valid-time Indeterminacy**

Curtis E. Dyreson and Richard T. Snodgrass

In valid-time indeterminacy, it is known that an event stored in a temporal database did in fact occur, but it is not known exactly when the event occurred. We present an extension of a tuple-timestamped temporal data model to support valid-time indeterminacy. In this data model, each event is represented with a set of possible instants, delimiting when the event might have occurred, and a probability distribution over that set. We extend the TQuel query language with constructs that specify the user's credibility in the underlying valid-time data and the user's plausibility in the relationships among that data. We provide a formal tuple calculus semantics, and show that this semantics reduces to the determinate semantics on determinate data. We outline an efficient representation of valid-time indeterminacy and efficient query processing algorithms, demonstrating the practicality of our approach. (55 pages)

Technical Reports are available via anonymous FTP from cs.arizona.edu in the **reports** directory or via electronic mail by sending a message to ftpmail@cs.arizona.edu containing the word **help**. Questions may be directed to tr_libr@cs.arizona.edu.

Because of the costs of printing and mailing, only one free hard copy of each report will be sent to an address. Additional copies may be purchased for the amount indicated. Requests that require payment must be accompanied by a check or a prepaid purchase order in U.S. dollars for the proper amount payable to The University of Arizona. Free copies may also be ordered via electronic mail to tr_libr@cs.arizona.edu.

Please send me the reports checked below: (RL-33)

report	*additional copies*		
☐ TR 93-20	$ 1.50	☐ TR 93-28	$ 5.50
☐ TR 93-21	$ 6.00	☐ TR 93-29	$ 2.00
☐ TR 93-22	$11.48	☐ TR 93-30	$ 2.50
☐ TR 93-23	$ 6.00	☐ TR 93-31	$ 3.00
☐ TR 93-24	$ 6.00	☐ TR 93-32	$ 2.50
☐ TR 93-25	$ 2.00	☐ TR 93-33	$ 1.50
☐ TR 93-26	$ 2.50	☐ TR 93-34	$ 1.50
☐ TR 93-27	$ 3.00	☐ TR 93-35	$ 4.50

Icon Newsletter

This newsletter contains information about the Icon programming language. Typical topics include reports on language design, implementation, and notices of new publications. It is issued aperiodically, two or three times a year.

Software Distributions

The Department distributes a variety of software in machine-readable form. Examples are the Icon programming language, SB-Prolog, SR, *x*-kernel, and the Scorpion System. Information about the contents and availability of specific distributions can be obtained by checking the boxes listed below. Most distributions are available for a nominal fee, which includes media and the associated documentation, or via anonymous FTP from cs.arizona.edu.

☐ Please add my name to the distribution list for the Icon Newsletter

Please send me information on the following software distributions:

☐ The Icon programming language
☐ The SR programming language
☐ The SB-Prolog System
☐ The *x*-kernel
☐ The Scorpion System

Technical Librarian
Department of Computer Science
The University of Arizona
Gould-Simpson 721
Tucson, Arizona 85721
USA