Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

THE TANGRAM PROJECT: PUBLICATIONS 1987-88

Richard R. Muntz
D. Stott Parker
Gerald J. Popek

January 1989
CSD-890003

The Tangram Project: Publications 1987-88

Richard R. Muntz
D. Stott Parker
Gerald J. Popek

Computer Science Department
University of California
Los Angeles, CA 90024-1596

The Tangram Project at UCLA is aimed at the development of an environment for modeling of dynamic systems. It is an integration of DBMS and KBMS technology with distributed processing techniques. It is supported by DARPA, as contract F29601-87-C-0072. This is a summary of technical reports issued over the first year of the project,

Table of Contents

1. OVERVIEW

TANGRAM: PROJECT OVERVIEW
Richard R. Muntz, D. Stott Parker
CSD-880032 (39pp.)
April 1988

Today, most computers are used for the modeling of real-world systems. Demands on the extent and quality of the modeling are growing rapidly. There is an ever-increasing need for environments in which one can construct and evaluate complex models both quickly and accurately.

Successful modeling environments will require a cross-disciplinary combination of different technologies:

System modeling tools
Database management
Knowledge base management
Distributed computing

None of these technologies by itself provides all that is needed. A modeling environment must offer high-speed retrieval and exploration of knowledge about systems, as well as integration of diverse information sources with existing modeling tools.

Tangram is a distributed modeling environment being developed at UCLA. It is an innovative Prolog-based combination of DBMS and KBMS technology with access to a variety of modeling tools.

---

2. STREAM DATABASE PROCESSING

THE TANGRAM STREAM QUERY PROCESSING SYSTEM
D. Stott Parker, Richard R. Muntz, Lewis Chau
CSD-880025 (28pp.)
March 1988

Tangram is an environment for modeling. It supports development and management of models, simulation of models, analysis of simulation output and analysis of models in general. Its current focus is on computer system performance modeling.

Modeling applications routinely generate large quantities of simulation data, and analysis of this data requires a system that differs in significant ways from existing database systems. The data often takes the form of time series, and therefore query processing requires both stream processing techniques and heavy numerical computations (e.g., basic statistical and time series analysis) beyond ordinary aggregates.

One of the driving concepts behind Tangram has therefore been the combination of large-scale data access and data reduction with a powerful programming environment. The Tangram environment is based on Prolog, extending it with a number of features, including process management, distributed database access, and generalized stream processing.

This paper describes the *Tangram Stream Processor (TSP),* the part of the Tangram environment performing query processing on large streams of data. The paradigm of transducers on streams is used throughout this system, providing a *'database-flow' (database dataflow) computation capability.*

shorter version in *Proceedings of the Sixth International Conf. on Data Engineering,* Los Angeles, CA, February, 1989.

A THEORY OF DIRECTED LOGIC PROGRAMS AND STREAMS
D. Stott Parker, Richard R. Muntz
CSD-880031 (31pp.)
April 1988

For some time it has been recognized that logic programmers commonly write *directed predicates,* i.e., predicates supporting only certain input and output patterns among their arguments. In many logic programming implementations, programmers are encouraged to use 'mode declarations' to announce this directedness, both as a matter of style and as a directive for compiler optimization.

A common application of directed programming is stream or list processing. Programs that operate on streams or lists usually have specific input and output arguments. More generally, directed predicates can represent functions, with specific inputs and outputs.

We present a new declarative formalism for directedness in logic programming systems. The formalism is based on the use of partial ordering constraints rather than unification. Semantics of the resulting system are rigorously definable, and extend ordinary logic program semantics in a natural way.

The approach to directed logic programs presented here will probably provide higher performance than is possible with undirected programs. Furthermore, the approach provides perspective relating diverse concepts such as predicate 'modes', functional computation, constraint processing, and stream processing.

shorter version in R.A. Kowalski and K.A. Bowen (eds.), *Logic Programming,* MIT Press, 1988, pp. 620-650.

IMPLICIT REPRESENTATION FOR EXTENSIONAL ANSWERS
Chung-Dak Shum, Richard Muntz
CSD-880067 (17pp.)
August 1988

An exhaustive list of atomic objects is not always the best means of information exchange. This paper concerns the implicit representation of extensional answers. Expressions for answers are given in terms of concepts and individuals. Exceptions within individual concepts are allowed.

Two criteria are defined as measures of the *goodness* of such expressions: (i) minimizing the number of terms; (ii) positive terms preferred over negative terms. Expressions satisfying these two criteria are called optimal expressions. It is shown that under a strict taxonomy of concepts, any two optimal expressions for an extensional answer share the same set of terms. The inductive proof elicits an algorithm for obtaining such expressions.

Generalizing the strict taxonomy of concepts to a join-semilattice of concepts eliminates the term uniqueness property and also makes the problem of finding an optimal expression intractable. The problem under multiple taxonomies, although it involves a restricted type of join-semilattice, remains intractable.

in L. Kerschberg (ed.),*Expert Database Systems*, Benjamin Cummings, 1989, pp. 497-522.

AN INFORMATION-THEORETIC STUDY ON AGGREGATE RESPONSES
Chung-Dak Shum, Richard Muntz
CSD-880068 (12pp.)
August 1988

An enumeration of individual objects is not always the best means of information exchange. This paper concerns the problem of providing aggregate responses to database queries.

An aggregate response is an expression whose terms are quantified concepts. The tradeoff between the conciseness and preciseness of an aggregate response is studied. Conciseness is measured by the length (the number of terms) of an expression, and preciseness is measured by the entropy or the amount of uncertainty associated with the expression. For a given length, an expression with the minimum amount of entropy is called optimal.

Under a one-level taxonomy with the same cardinalities for all leaf concepts, the problem of finding an optimal expression can be solved inexpensively. An efficient heuristic is also proposed for the general one-level taxonomy. For a taxonomy of more than one level, an efficient heuristic is suggested which experiments indicate yields good solutions.

in *Proc. International Conf. on Very Large Databases,* Los Angeles, CA, August 29-September 1, pp. 479-490, 1989.

ASPEN: A STREAM PROCESSING ENVIRONMENT
Brian K. Livesey, Richard R. Muntz
CSD-880080 (26pp.)
October 1988

In this paper, we describe ASPEN, a concurrent stream processing system. ASPEN is novel in that it provides a programming model in which programmers use simple annotations to exploit varying degrees and types of concurrency. The degree of concurrency to be exploited is not fixed by the program specification or by the underlying system. Increasing or decreasing the degree of concurrency to be exploited during execution does not require rewriting the entire program, but rather, simply re-annotating it.

Examples are given to illustrate the varying types of concurrency inherent in programs written within the stream processing paradigm. We show how programs may be annotated to exploit these varying degrees of concurrency. We briefly describe our implementation of ASPEN.

ASPEN: A STREAM PROCESSING ENVIRONMENT
Brian K. Livesey
CSD-880098 (120pp.)
December 1988

Stream processing is an ideal paradigm for data-intensive applications. The solutions to a rich and varied set of problems that are, at best, awkward to express in other paradigms, can be expressed elegantly within the stream processing paradigm. Furthermore, stream processing presents an execution model in which such problems can be solved efficiently.

This thesis describes ASPEN, a stream processing environment. A programming language called Log(F) is extended to make it an appropriate language for expressing stream processing programs. The thesis focuses on those extensions that provide support for concurrent processing and access to distributed data.

The approach is novel in that the programming model allows the determination of the granularity of concurrency to be separated from the actual coding of the program. The degree of concurrency to be exploited is not fixed by the program specification or by the underlying system. Simple annotations allow the programmer to specify varying degrees of concurrency. Increasing or decreasing the degree of concurrency exploited during execution does not require rewriting the entire program, but rather, simply re-annotating it.

Several examples are given to illustrate the varying types of concurrency inherent in programs written within the stream processing paradigm. Examples are given which demonstrate how programs may be annotated to exploit these varying types and degrees of concurrency. The implementation of ASPEN is also described.

STREAM DATA ANALYSIS IN PROLOG
D. Stott Parker
CSD-890004 (54pp.)
January 1989

Today many applications routinely generate large quantities of data. The data often takes the form of a time series, or more generally just a *stream* – an ordered sequence of records. Analysis of this data requires stream processing techniques, which differ in significant ways from what current database query languages and statistical analysis tools support today. There is a real need for better stream data analysis systems.

Stream analysis, like most data analysis, is best done in a way that permits interactive exploration. It must support 'ad hoc' queries by a user, and these queries should be easy to formulate and run. It seems then that stream data analysis is best done in some kind of powerful programming environment.

A natural approach here is to analyze data with the stream processing paradigm of transducers (functional transformations) on streams. Data analyzers can be composed from collections of functional operators (transducers) that transform input data streams to output streams. A modular, extensible, easy-to-use library of transducers can be combined in arbitrary ways to answer stream data analysis queries of interest.

Prolog offers an excellent start for an interactive data analysis programming environment. However most Prolog systems have limitations that make development of real stream data analysis applications challenging.

We describe an approach for doing stream data analysis that has been taken in the Tangram project at UCLA. Transducers are implemented not directly in Prolog, but in a functional language called Log(F) that can be translated to Prolog. With Log(F), stream processing programs are straightforward to develop. A by-product of this approach is a practical way to interface Prolog and database systems.

STREAM PROCESSING: AN EFFECTIVE WAY TO INTEGRATE AI AND DBMS
D. Stott Parker
CSD-890005 (11pp.)
January 1989

We present a novel approach for integrating AI systems with DBMS. The 'impedance mismatch' that has made this integration a problem is, in essence, a difference in the two systems' models of data processing. Our approach is to avoid the mismatch by forcing both AI systems and DBMS into the common model of *stream processing.*

By a *stream* here we mean an ordered sequence of data items. Stream processing is a well-known AI programming paradigm in which functional operators (which we call 'transducers') are combined to obtain arbitrary mappings from streams to streams. The stream processing paradigm can be, and has been, applied equally well as an AI programming model and as a query processing model in databases.

We argue first that, in practice, the relational model of data is actually the stream model. The pure relational model cannot capture important aspects of relational databases such as column ordering, duplicate tuples, tuple ordering, and access paths, while the stream model does so naturally.

We then describe the approach taken in the *Tangram* project at UCLA, which integrates Prolog with relational DBMS. Prolog is extended to a functional language called Log(F) that facilitates development of stream processing programs. The integration of this system with DBMS is simultaneously elegant, easy to use, and relatively efficient.

shorter version in *Proceedings of the Sixth International Conf. on Data Engineering,* Los Angeles, CA, February, 1989.

STATISTICAL RULES: A NOTION OF DATABASE ABSTRACT AND ITS ROLE IN QUERY PROCESSING
Chung-Dak Shum, Richard Muntz
CSD-890007 (25pp.)
January 1989

A database instance is not an arbitrary collection of data, but rather many correlations exist among data items. The notion of *statistical rules* is introduced as a means of expressing such relationships. We demonstrate that statistical rules can be utilized in the query optimization process. In selectivity factor estimation, for example, statistical rules can actually be used to introduce relevant attributes the same manner as exact rules in semantic query optimization. Other uses of statistical rules include the enhancement of parallelism in database machines, and providing incomplete/quick answers as well as more informative responses.

We quantify the notion of how to measure the "inexactness" of a statistical rule using an entropy measure. The lower the entropy or uncertainty of a rule, the better the rule is. Based on such a measure, we show that constructing statistical rules using a "greedy" algorithm will result in a reasonable, although perhaps not optimal rule.

3-WAY HASH JOIN QUERY PROCESSING IN DISTRIBUTED RELATIONAL DATABASE SYSTEMS
Scott E. Spetka, Gerald J. Popek
CSD-890008 (17pp.)
January 1989

Initial distribution of relations as well as storage structures and organization have an important impact on performance and the appropriate choice of processing techniques for database operations. Consideration of data distribution for partitioned relations used in hash join processing lead us to experiment with a new algorithm for processing 3-way join queries in a distributed system.

Database cacheing is also important for performance of distributed database management systems. An important goal is to provide an algorithm that can complement existing algorithms to provide sufficient generality to operate in a network transparent environment where the location of available resources may be changing, and to use those resources effectively. We present a new algorithm for processing 3-way join queries that can take advantage of cacheing by providing improved performance when data is not ideally distributed for some other algorithms.

3. LANGUAGE SUPPORT

LOG(F): A NEW SCHEME FOR INTEGRATING REWRITE RULES, LOGIC PROGRAMMING AND LAZY EVALUATION
Sanjai Narain
CSD-870027 (18pp.)
July 1987

We present LOG(F), a new scheme for integrating rewrite rules, logic programming, and lazy evaluation. First, we develop a simple yet expressive rewrite rules system F* for representing functions. F* is non-Noetherian, i.e., an F* program *can* admit infinite reductions. For this system, we develop a reduction strategy called *select* and show that it possesses the property of reduction-completeness. Because of this property, *select* exhibits a weak form of lazy evaluation.

We then show how to implement F* in Prolog. Specifically, we compile rewrite rules of F* into Prolog clauses in such a way that when Prolog intereprets these clauses, it directly simulates the behavior of *select*. Since it is not necessary to change Prolog, it is possible to do lazy evaluation efficiently. Since Prolog is already a logic programming system, a combination of rewrite rules, logic programming and lazy evaluation is achieved.

IMPROVING CLAUSE ACCESS IN PROLOG
D. Stott Parker, Thomas W. Page, Richard Muntz
CSD-880024 (7pp.)
March 1988

One of the weakest aspects of Prolog is in its access to clauses. This weakness is lamentable as it makes one of Prolog's greatest strengths, its ability to treat programs as data and data as programs, difficult to exploit. This paper proposes modifications to Prolog and shows how they circumvent important problems in Prolog programming in a practical way. For example, the proposed modifications permit Prolog programs that perform efficient database query (join) processing, coroutining, and abstract machine interpretation. These modifications have been used successfully at UCLA, and should be easy to implement within any existing Prolog system.

LOG(F): AN OPTIMAL COMBINATION OF LOGIC PROGRAMMING, REWRITING, AND LAZY EVALUATION
Sanjai Narain
CSD-880040 (176pp.)
June 1988

A new approach for combining logic programming, rewriting, and lazy evaluation is described. It rests upon *subsuming* within logic programming, instead of upon *extending* it with, rewriting, and lazy evaluation.

A non-terminating, non-deterministic rewrite rule system, F* and a reduction strategy for it, select, are defined. F* is shown to be reduction-complete in that select simplifies terms whenever possible. A class of F* programs called Deterministic F* is defined and shown to satisfy confluence, directedness, and minimality. Confluence ensures that every term can be simplified in at most one way. Directedness eliminates searching in simplification of terms. Minimality ensures that select simplifies terms in a minimum number of steps. Completeness and minimality enable select to exhibit, respectively, weak and strong forms of laziness.

F* can be compiled into Horn clauses in such a way that when SLD-resolution interprets these, it directly simulates the behavior of select. Thus, SLD-resolution is made to exhibit laziness. LOG(F) is defined to be a logic programming system augmented with an F* compiler, and the equality axiom X=X. LOG(F) can be used to do lazy functional programming in logic, implement useful cases of the rule of substitution of equals for equals, and obtain a new proof of confluence for combinatory logic.

EXECUTABLE TEMPORAL SPECIFICATIONS WITH FUNCTIONAL GRAMMARS
H. Lewis Chau, D. Stott Parker
CSD-880046 (20pp.)
June 1988

The *Stream Pattern Analyzer (SPA)* is one part of the Tangram Stream Query Processing System being developed at UCLA. It uses *functional grammars* to specify pattern analysis for streams of data.

Parallel execution events in a distributed system may be captured in an event stream for analysis. Given a set of functional grammar rules, SPA can analyze arbitrarily complex behavior patterns in this stream. At the same time a SPA grammar can act as a declarative specification of valid event histories.

We define a simple but powerful scheme that coroutines recognition of multiple patterns in an event stream. Propositional temporal logic queries can be expressed in SPA in terms of predefined temporal operators such as *eventually, implies, not_until,* etc. Thus complex history-oriented specifications can be developed easily.

Functional grammar rules by themselves act as pattern generators or specifiers, and can be used to develop parsers by compilation to Log(F). Log(F) is a combination of Prolog and a functional language called F*. We describe a simple algorithm to compile functional grammars to Log(F), and prove its correctness.

PX REFERENCE MANUAL, VERSION 0.2
Ted Kim
CSD-880079 (47pp.)
October 1988

This manual describes an interface to the X Window System for Prolog. The X Window System is a network-based window system providing a desktop style of user interface and graphics. PX provides a low level interface to X for Prolog similar to that provided by "Xlib" for the C language. PX is designed for use with version 11 of the X Window System. Higher level interfaces (such as toolkits) are built on top of this one and are outside the scope of this document.

PX is implemented in the C language using the C language foreign function interface from Quintus Prolog. Almost any Prolog which supports the Quintus style interface can use this package with few restrictions. In particular, SICStus Prolog was used in the development of this system. This document is a reference manual. As such, it is not a tutorial or user's guide to X or Prolog.

FUNCTIONAL LOGIC GRAMMAR:
A NEW SCHEME FOR LANGUAGE ANALYSIS
H. Lewis Chau, D. Stott Parker
CSD-880097 (16pp.)
December 1988

We present a new kind of grammar. It combines concepts from logic programming, rewriting, lazy evaluation, and logic grammar formalisms such as Definite Clause Grammar (DCG). We call it *Functional Logic Grammar.*

A functional logic grammar is a finite set of rewrite rules. It is efficiently executable, like most logic grammars. In fact, fu^ctional logic grammar rules can be compiled to Prolog and executed by existing Prolog interpreters as generators or acceptors. Unlike most logic grammars, functional logic grammar also permits higher-order specification and modular composition.

This paper defines functional logic grammar and compares it with the successful and widely-used DCG formalism in logic programming. We show that pure DCG can be easily translated into functional logic grammar. Functional logic grammar enjoys the advantages of DCG, as well as its first-order logic foundation. At the same time, functional logic grammar ranks higher in aspects such as expressiveness and modularity, and permits lazy evaluation.

4. COMPUTER SYSTEM PERFORMANCE MODELING

A NOTE ON THE COMPUTATIONAL COST OF THE LINEARIZER ALGORITHM FOR QUEUEING NETWORKS
Edmundo de Souza e Silva and Richard R. Muntz
CSD-870025 (15pp.)
July 1987; revised February 1988

Linearizer is one of the best known approximation algorithms for obtaining numeric solutions for product form queueing networks. In the original exposition of Linearizer, the computational cost was stated to be O(MK^3) for a model with M queues and K job classes. We show in this note that with some straightforward algebraic manipulation Linearizer can be modified to require only O(MK^2) computational cost.

We also discuss the space requirements for Linearizer and show that the space can be reduced to 0(MK) but with some increased computational cost.

To appear, *IEEE Transactions on Computers,* 1989.

AN OBJECT ORIENTED METHODOLOGY FOR THE SPECIFICATION OF MARKOV MODELS
Steven Berson, Edmundo Silva, Richard Muntz
CSD-870030 (23pp.)
July 1987

Modelers wish to specify their models in a symbolic, high level language while analytic techniques require a low level, numerical representation. The translation between these description levels is a major problem.

We describe a simple, but surprisingly powerful approach to specifying system level models based on an object oriented paradigm. This basic approach will be shown to have significant advantages in that it provides the basis for modular, extensible modeling tools. With this methodology, modeling tools can be quickly and easily tailored to particular application domains. An implementation in Prolog, of a system based on this methodology and some example applications are given.

ANALYTIC MODELING METHODOLOGY FOR EVALUATING THE PERFORMANCE OF DISTRIBUTED, MULTIPLE-COMPUTER SYSTEMS
Alex Kapelnikov
CSD-870061 (201pp.)
November 1987

In this dissertation, we describe an analytic modeling methodology for evaluating the performance of distributed, multiple-computer systems. The concepts and techniques of this methodology are useful for the approximate analysis of a wide range of distributed computing environments and communication networks. The main strategy of our approach is to segregate, as much as possible, the model of the "logical" behavior of an application (a program or a process) from the model of its underlying execution environment. For representing program behavior, graph-based techniques are used, while extended queueing networks are utilized for modeling system architectures. The solutions of both types of models are combined to estimate the performance of a distributed system in executing some selected applications.

To illustrate the practical application of the methodology introduced in this dissertation and provide an indication of its expected accuracy level, we have included two case studies.

A MODELING METHODOLOGY FOR THE ANALYSIS OF CONCURRENT SYSTEMS AND COMPUTATIONS
Alex Kapelnikov, Richard R. Muntz, and Milos D. Ercegovac
in M.H. Barton, E.L. Dagless, G.L. Reijns (eds.), *Distributed Processing,* Elsevier Science Publishers, pp. 465-479, 1988.

In this paper, we describe a novel modeling methodology for evaluating the performance of distributed, multiple-computer systems. Our approach employs a set of analytic tools to obtain an estimate of the average execution time of a parallel implementation of a program (or transaction) in a distributed environment. These tools are based on an amalgamation of queueing network theory and graph models of program behavior. Hierarchical application of heuristic optimization techniques facilitates the analysis of large and complex programs. A realistic example is used to illustrate the practical application of our methodology.

A DISTRIBUTED ALGORITHM TO DETECT A GLOBAL STATE OF A DISTRIBUTED SIMULATION SYSTEM
Behrokh Samadi, Richard R. Muntz, D. Stott Parker
in M.H. Barton, E.L. Dagless, G.L. Reijns (eds.), *Distributed Processing,* Elsevier Science Publishers, pp. 19-34, 1988.

In this paper, we describe a novel modeling methodology for evaluating the performance of distributed, multiple-computer systems. Our approach employs a set of analytic tools to obtain an estimate of the average execution time of a parallel implementation of a program (or transaction) in a distributed environment. These tools are based on an amalgamation of queueing network theory and graph models of program behavior. Hierarchical application of heuristic optimization techniques facilitates the analysis of large and complex programs. A realistic example is used to illustrate the practical application of our methodology.

DISTRIBUTED SHARED MEMORY IN A LOOSELY COUPLED DISTRIBUTED SYSTEM (EXTENDED ABSTRACT)
Brett D. Fleisch
in *Proceedings COMPCON Spring 88,* San Francisco, CA, February-March 1988, pp.182-184.

In this work we describe new implementation experiences with a distributed shared memory system implemented in a loosely coupled distributed system. Our goal was to investigate the feasibility of distributed shared memory (dsm) in an operating system kernel. Li (1986) demonstrated the feasibility of such a system outside of the kernel with a number of numeric applications, but it remained a relatively open question as to how well dsm performs for a variety of non-numeric applications and what the effects of dsm are on other kernel services. The organization of dsm we describe resembles a cross-processor segmented paging system. Our talk relates implementation experiences and preliminary performance results. We plan to report results from experiments with symbolic computation, which emphasizes rearragement of data, where often the sequence of operations is highly data dependent and less amenable to compile time analysis than numerical computation. One general goal of this work is to describe a set of software primitives and to identify hardware features that can be used to support the conversion of applications from nondistributed shared memory to distributed shared memory. These features may include hints, user advice, control primitives, and architectural modifications that will improve functionality and performance.

BOUNDING AVAILABILITY OF REPAIRABLE COMPUTER SYSTEMS
Richard Muntz, Edmundo Silva, A. Goyal
CSD-880070 (26pp.)
September 1988

Markov models are widely used for the analysis of availability of computer/communication systems. Realistic models often involve state space cardinalities that are so large that it is impractical to generate the transition rate matrix let alone solve for availability measures. Various state space reduction methods have been developed, particularly for transient analysis. In this paper we present an approximation technique for determining steady state availability. Of particular interest is that the method also provides bounds on the error. Examples are given to illustrate the method.

5. CONSTRAINT-BASED MODELING

SET CONTAINMENT INFERENCE AND SYLLOGISMS
Paolo Atzeni, D. Stott Parker
CSD-870022 (34pp.)
March 1987

Type hierarchies and type inclusion (*isa*) inference are now standard in many knowledge representation schemes. In this paper, we show how to determine consistency and inference for collections of statements of the form

mammal isa vertebrate.

These *containment* statements relate the contents of two sets (or types). The work here is new in permitting statements with negative information: disjointness of sets, or non-inclusion of sets. For example, we permit the following statements also:

mammal isa **non**(*reptile*)
non(*vertebrate*) *isa* **non**(*mammal*)
not(*reptile isa amphibian*)

Binary containment inference is the problem of determining the consequences of positive constraints P and negative constraints **not**(P) on sets, where positive constraints have the form P: $X \subseteq Y$. Negations of these constraints therefore have the form **not**(P): $X \cap$ **non**(Y) $\neq \emptyset$, so positive constraints assert containment relations among sets, and negative constraints assert that two sets have a non-empty intersection.

We show binary containment inference is solved by rules essentially equivalent to Aristotle's *Syllogisms.* Necessary and sufficient conditions for consistency, as well as sound and complete sets of inference rules, are presented for binary containment. The sets of inference rules are compact, and lead to polynomial-time inference algorithms, so permitting negative constraints does not result in intractability for this problem.

To appear, *Theoretical Computer Science,* 1988.

PARTIAL ORDER PROGRAMMING
D. Stott Parker
CSD-870067 (80pp.)
December 1987

We introduce a programming paradigm in which statements are constraints over partial orders. A *partial order programming problem* has the form

$$\begin{array}{ll} \text{minimize} & u \\ \text{subject to} & u_1 \sqsupseteq v_1, u_2 \sqsupseteq v_2, \cdots \end{array}$$

where u is the *goal,* and $u_1 \sqsupseteq v_1, u_2 \sqsupseteq v_2, \cdots$ is a collection of constraints called the *program.* A solution of the problem is a minimal value for u determined by values for u_1, v_1, etc. satisfying the constraints. The domain of values here is a *partial order,* a domain D with ordering relation $\sqsupseteq$.

The partial order programming paradigm has interesting properties:

(1) It generalizes mathematical programming, dynamic programming, and computer programming paradigms (logic, functional, and others) cleanly, and offers a foundation both for studying and combining paradigms.

(2) It takes thorough advantage of known results for continuous functionals on complete partial orders, when the constraints involve expressions using only continuous and monotone operators. These programs have an elegant semantics coinciding with recent results on the relaxation solution method for constraint problems.

(3) It presents a framework that may be effective in modeling of complex systems, and in knowledge representation for cognitive computation problems.

ON CONSTRAINT-ORIENTED ENVIRONMENTS
FOR CONTINUOUS SYSTEMS SIMULATION
Richard A. Huntsinger
CSD-880018 (10pp.)
March 1988

Sets of simultaneous differential equations and sets of queries on those equations are naturally expressible as *constraint networks* in the *constraint satisfaction* modeling paradigm. Further, relaxation enhanced to exploit *typed valued* constraints provides a procedural semantics for such constraints which in the best case reduces to propagation, and in the worst case performs comparably to other paradigms. Accordingly, constraint satisfaction is advocated as the paradigm of choice on which to base continuous systems simulation environments.

Examples are presented illustrating constraint network characterizations of continuous systems models, and their corresponding procedural semantics.

REPRESENTATION TRANSFORMATION IN CONSTRAINT SATISFACTION SYSTEMS
Richard Huntsinger
CSD-880020 (11pp.)
March 1988

A practical class of constraint satisfaction systems operate on *relaxable representations* of the form $N = f(N)$, where N is a set of variables, and the declarative semantics is the set of instantiations of N which preserve the equality. In general, relaxation provides a complete procedural semantics for only a subset ρ of such representations. Of interest, then, is the set of *transformable* representations $\alpha \supset \rho$ in which for each representation $M_r \in \alpha$ there exists a determinable transformation $T: \alpha \rightarrow \rho$ such that the declarative semantics of M_r is identical to that of $T(M_r)$.

Relaxable representations for which $f(N)$ is a polynomial are transformable, each corresponding to a transform of the form $N = (f(N)N^n)^{1/(n+1)}$, where n is a function of the degree and coefficients of the polynomial. This observation provides some intuition about more general transformations, applicable to the implementation of powerful (complete over a superset of ρ) constraint satisfaction systems.

PARTIAL ORDER PROGRAMMING: EXTENDED ABSTRACT
D. Stott Parker
CSD-880086 (7pp.)
October 1988

We introduce a programming paradigm in which statements are constraints over partial orders. A *partial order programming problem* has the form

$$\boxed{\begin{array}{ll} \text{minimize} & u \\ \text{subject to} & u_1 \sqsupseteq v_1,\ u_2 \sqsupseteq v_2,\ \cdots \end{array}}$$

where u is the *goal,* and $u_1 \sqsupseteq v_1,\ u_2 \sqsupseteq v_2,\ \cdots$ is a collection of constraints called the *program.* A solution of the problem is a minimal value for u determined by values for u_1, v_1, etc. satisfying the constraints. The domain of values here is a *partial order,* a domain D with ordering relation $\sqsupseteq$.

The partial order programming paradigm has interesting properties:

(1) It generalizes mathematical programming and also computer programming paradigms (logic, functional, and others) cleanly, and offers a foundation both for studying and combining paradigms.

(2) It takes thorough advantage of known results for continuous functionals on complete partial orders, when the constraints involve expressions using only continuous and monotone operators. The semantics of these programs coincide with recent results on the relaxation solution method for constraint problems.

(3) It presents a framework that may be effective in modeling, or knowledge representation, of complex systems.

in *Proceedings of the Sixteenth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages,* Austin, Texas, January 11-13, 1989.

OPTIMIZATION BY NON-DETERMINISTIC, LAZY REWRITING
Sanjai Narain
CSD-880092 (19pp.)
November 1988

Given a set S and a condition C we address the problem of determining which members of S satisfy C. One useful approach is to set up the generation of S as a tree, where each node represents a subset of S. If from the information available at a node, we can determine that no members of the subset it represents satisfy C, then the subtree rooted at it can be pruned, or not generated. Thus, large subsets of S can be quickly eliminated from consideration. We show how such a tree can be simulated by interpretation of non-deterministic rewrite rules, and its pruning simulated by lazy evaluation.
