Partial Order Programming

Extended Abstract

D. Stott Parker

Computer Science Department
University of California
Los Angeles, CA 90024-1596
stott@cs.ucla.edu

ABSTRACT

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

minimize	u
subject to	$u_1 \sqsupseteq v_1,\ u_2 \sqsupseteq v_2,\ \cdots$

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.

This work supported by a State of California MICRO – IBM Los Angeles Scientific Center grant, a State of California MICRO – Aerojet Electro Systems grant, and the Tangram project, DARPA contract F29601-87-C-0072.

1. Motivation

Consider the following three problems:

(1) *Laplace's Equation*

Let u_i be the solution of the discretized approximation to the equation

$$\nabla^2 u = \frac{\partial^2}{\partial x^2} u + \frac{\partial^2}{\partial y^2} u = 0.$$

The discretization is given by a rectilinear grid G, with boundary conditions defining u only on the boundary points of G. Numbering the points in G as $\{u_1, \ldots, u_n\}$ arbitrarily, let *left*(i), *right*(i), *up*(i), and *down*(i) give the neighbors of node i in the interior of the grid. A node i on the boundary of G takes on a boundary value $u_i = b_i$. Each node i in the interior of G satisfies the constraint

$$u_i = \frac{u_{left(i)} + u_{right(i)} + u_{up(i)} + u_{down(i)}}{4}.$$

The problem is to find values for $u_1, \ldots, u_n$.

(2) *Single-Source Shortest Path Problem*

Consider a graph $G = \langle V,E \rangle$, each of whose edges $\langle v_i,v_j \rangle$ has an associated *cost* a_{ij}. We assume here that costs are all nonnegative. One node of G, called v_0, is distinguished as a *source* node. A *path* from v_{i_1} to v_{i_m} in G is a sequence of edges

$$\langle v_{i_1},v_{i_2} \rangle, \langle v_{i_2},v_{i_3} \rangle, \cdots, \langle v_{i_{m-1}},v_{i_m} \rangle$$

in E; and the cost of this path is

$$a_{i_1 i_2} + a_{i_2 i_3} + \cdots + a_{i_{m-1} i_m}.$$

For every node v_i, the problem is to compute the *node cost* u_i, which is the least cost of all paths between the source v_0 and v_i. Hence, $u_0 = 0$.

(3) *Consistent Labeling Problem*

We are given a graph $G = \langle V,E \rangle$. Each node v_i is to be 'labeled' with one or more members of b_i, which are subsets of a finite label set Λ. These label sets are sometimes called *node constraints*, written

$$u_i \subseteq b_i$$

for each i, where u_i denotes the set of labels for v_i.

For each edge $\langle v_i,v_j \rangle$ in G, we also have an *arc constraint* a_{ij} that is a subset of $b_i \times b_j$. Specifically, the label sets u_i and u_j selected for nodes v_i and v_j must obey the constraint

$$u_i \subseteq a_{ij} \ltimes u_j,$$

where the *relational semijoin operation* $\ltimes$ is defined by

$$a \ltimes u = \{ x \mid \langle x,y \rangle \text{ is in } a, \text{ and } y \text{ is in } u \}.$$

The problem is to produce label sets that obey all of the node and arc constraints.

How are these three problems related? Each is commonly given as an example of a problem that can be solved by 'relaxation'. The notation used above suggests deeper similarities, however. This paper is concerned with showing that the problems are equivalent in a certain sense. Roughly, each problem seeks the solution of a linear *partial order program*. That is, each seeks a vector $\mathbf{u}$ solving the problem

minimize	$\mathbf{u}$		
subject to	$\mathbf{u}$	$\sqsupseteq$	$\mathbf{A}\,\mathbf{u} + \mathbf{b}$

for an appropriate partial order $\sqsupseteq$, matrix $\mathbf{A}$, vector $\mathbf{b}$, and multiplication and addition operator:

(1) With Laplace's equation, define the coefficient matrix $\mathbf{A} = (a_{ij})$ by

$$a_{ij} = \begin{cases} 1/4 & \text{if } i \text{ is an interior node and} \\ & \quad j \in \{left(i),right(i),up(i),down(i)\} \\ 0 & \text{otherwise} \end{cases}$$

and boundary condition vector $\mathbf{b} = (b_i)$ by

$$b_i = \begin{cases} u_i & \text{if } i \text{ is a boundary node} \\ 0 & \text{otherwise.} \end{cases}$$

Then the vector $\mathbf{u} = (u_i)$ satisfies the matrix equation $\mathbf{u} = \mathbf{Au} + \mathbf{b}$. Since the matrix $\mathbf{A}$ is nonnegative, the function $\mathbf{f(u)} = \mathbf{Au} + \mathbf{b}$ is monotone†. As it is also continuous, minimizing $\mathbf{u}$ such that $\mathbf{u} \geq \mathbf{Au} + \mathbf{b}$ is equivalent to finding $\mathbf{u}$ such that $\mathbf{u} = \mathbf{Au} + \mathbf{b}$ (the least fixed point of $\mathbf{f}$).

This problem may be solved iteratively [6]. Specifically, given any finite initial values for the interior nodes, the equation above may be used as an assignment, $\mathbf{u} := \mathbf{Au} + \mathbf{b}$. Since $\mathbf{A}$ is nonnegative and has spectral radius less than 1, the iteration always converges (though possibly in an infinite number of steps).

† Recall that a function f is *monotone* with respect to a partial order $\sqsubseteq$ if $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$.

(2) The single-source shortest path problem places constraints on the node costs u_i. If $i \neq 0$ and v_i is a node whose outgoing edges $\langle v_i,v_j \rangle$ have cost a_{ij}, then

$$u_i \geq \min\left\{ \min_{1 \leq j \leq n} a_{ij} + u_j,\ a_{i0} \right\}.$$

Since we seek the shortest path values, our problem becomes

minimize	$\langle u_0, \ldots, u_n \rangle$		
subject to	u_0	$\geq$	0
	u_i	$\geq$	$\min_{0 \leq j \leq n} a_{ij} + u_j \quad (i \neq 0).$

Again the inequalities may be used to find a solution iteratively. Given a suitable initial value (such as $+\infty$) for u_i, $1 \leq i \leq n$, we can use the equality above as an assignment to obtain successive values of u_i. The iteration converges in a finite number of steps when the costs a_{ij} are nonnegative, as we assumed above.

(3) With consistent labeling, we can combine the constraints for each node v_i into a single constraint

$$u_i \subseteq \left[\bigcap_{j=1}^{n} a_{ij} \ltimes u_j \right] \cap b_i.$$

This problem is naturally presented as a maximization problem with respect to the ordering $\subseteq$

maximize	$\langle u_1, \ldots, u_n \rangle$		
subject to	u_i	$\subseteq$	$\left[\bigcap_{j=1}^{n} a_{ij} \ltimes u_j \right] \cap b_i$

or equivalently, a minimization problem with respect to the reversal of $\subseteq$. (That is, we wish to minimize $\langle u_1, \ldots, u_n \rangle$ with respect to the ordering $\sqsubseteq$ defined by $x \sqsupseteq y$ iff $x \subseteq y$.) Once again, iterative solution is possible. Iterative approaches for consistent labeling have come under increasing investigation recently [8].

We see that the constraints of these three well-known problems can be expressed in the form

$$u_i \sqsupseteq f_i(u_1, \ldots, u_n)$$

for some ordering $\sqsubseteq$, where each f_i is a function that is monotone with respect to $\sqsubseteq$. We use $\sqsupseteq$ here instead of $\sqsubseteq$ in order to make direct connections with existing work on monotone functions. Furthermore each of the problems above is a *linear inequality system*. That is, for each problem there are binary operators $\boxplus$ and $\boxtimes$ with which the problem can be expressed as

minimize	$\mathbf{u}$		
subject to	$\mathbf{u}$	$\sqsupseteq$	$\mathbf{A} \boxplus.\boxtimes \mathbf{u} \boxplus \mathbf{b}$

where '$\boxplus.\boxtimes$' is a *generalized matrix product* using $\boxplus$ for (commutative, associative) addition and $\boxtimes$ for multiplication (as in the APL programming language). If $X = (x_{ij})$ and $Y = (y_{ij})$ are matrices of sizes $m \times n$ and $n \times p$, respectively,

the generalized matrix product $Z = X \boxplus\!\boxtimes Y$ is a matrix of size $m \times p$ defined by

$$z_{ij} = \boxplus_{k=1}^{n} x_{ik} \boxtimes y_{kj}.$$

For the examples above we have the following table:

Problem	$\boxplus$	$\boxtimes$	$\sqsupseteq$
Laplace's Equation	+	×	≥
Shortest Path	min	+	≥
Consistent Labeling	∩	⋉	⊆

Linearity is interesting not only because it provides a new point of view here. Linearity also immediately suggests both sequential and parallel *algorithms* for solving the problems. The arsenal of known algorithms for solving linear algebraic systems of this kind is vast and well understood [7]. By casting our problems in linear format we can immediately take advantage of this existing work.

Evidently these problems are not the only ones with this structure. How should we generalize upon these three examples?

2. Partial Order Programming

Partial order programming is a computational paradigm that expresses computation declaratively as statements of ordering. A *partial order program* P specifies a set of constraints C of the form

$$u \sqsupseteq v$$

where u and v are 'objects' and $\sqsupseteq$ is a given partial order. Each program thus specifies:

(1) a domain D of *values* with partial order $\sqsupseteq$;
(2) a set B of *objects*, which contains D;
(3) a set of ordering constraints C.

Typically the constraints $u \sqsupseteq v$ are such that u is never a value, while v can be a value. Programs with constraints of this form are called *reductive* programs, and we will discuss them shortly.

A *partial order programming problem* is then a statement of the form

minimize	g
subject to	P

where g is a specific object, which we call the *goal*, and $P = \langle B, C, D, \sqsupseteq \rangle$ is a partial order program. A *semantics* for the program P is an assignment of values in D to all of the objects in B in a way that satisfies all constraints in C. A *solution* of the programming problem is a semantics of P that simultaneously minimizes the value assigned to the goal object g. There may be no solutions, or more than one solution.

Consider the following example of a partial order program, where the domain of values D is the set of all subsets of $\{1,2,3\}$ and is partially ordered by set inclusion with least element $\emptyset$:

minimize	s		
subject to	s	$\supseteq$	t
	s	$\supseteq$	$\{3\}$
	t	$\supseteq$	$\{1,2\}$.

In this problem the objects B are $\{s,t\} \cup D$, and the inequalities listed here give the ordering constraints C. The unique solution of the problem is determined by the semantics

$$\begin{aligned} s &= \{1,2,3\} \\ t &= \{1,2\}, \end{aligned}$$

which gives the smallest possible value to s that is consistent with the constraints.

3. Solving Partial Order Programming Problems

With partial order programming problems defined, the issue of how to solve them arises immediately. After reviewing some relevant definitions and Kleene's fixed point theorem, we present three progressively restrictive classes of partial order programming problems, and show how they can be solved. These classes cover the three examples shown earlier.

3.1. Background

Definitions

A *partial order* is a pair $<D,\sqsupseteq>$ where $\sqsupseteq$ is a binary relationon D such that

(P1) For all x in D, $x \sqsupseteq x$.
(P2) For all x,y,z in D, if $x \sqsupseteq y$ and $y \sqsupseteq z$, then $x \sqsupseteq z$.
(P3) For all x,y in D, if $x \sqsupseteq y$ and $y \sqsupseteq x$, then $x = y$.

A *preorder* is a pair $<D,\sqsupseteq>$ satisfying (P1) and (P2).

The least element of D, if it exists, is written $\perp$.

In a partial order $<D,\sqsupseteq>$, an *upper bound* of a subset S of D is an element z in D such that for every x in S, $z \sqsupseteq x$. The *least upper bound* of a set S, written $\bigsqcup S$, is the least z such that z is an upper bound of S; by (P3) it is unique if it exists.

A *directed set* S in a partial order $<D,\sqsupseteq>$ is a nonempty subset of D with the property that if x,y are arbitrary elements in S, then there is another element z in S such that both $z \sqsupseteq x$ and $z \sqsupseteq y$.

A *complete partial order* (*cpo*) is a partial order $<D,\sqsupseteq>$ in which every directed set S of D has a least upper bound $\bigsqcup S$.

A function $f: D \to D$ on a cpo $<D,\sqsupseteq>$ is called *continuous monotone* if for every directed set S in D,

$$f(\bigsqcup S) = \bigsqcup f(S).$$

Readers familiar with domain theory will undoubtedly prefer the terminology 'continuous' to 'continuous monotone', finding 'monotone' redundant. We use the terminology 'continuous monotone' here to emphasize the monotonicity requirement on the function, because real-valued continuous, but non-monotone, functions can arise in problems like those in the first section of this paper.

Theorem (Kleene)

Let $f: D \to D$ be a continuous monotone map on the cpo $<D,\sqsupseteq>$ with least element $\perp$. Then f has a least fixed point equal to

$$\mathrm{fix} f = \bigsqcup \{ f^k(\perp) \mid k \in \omega \}$$

where ω is the set of natural numbers, and f^k is f iterated k times, i.e., $f^k = f \bullet f^{k-1}$, and f^0 is the identity.

3.2. Three Solvable Classes of Problems

We define three natural classes of partial order programming problems.

(1) *Reductive Partial Order Programming*

Reductive programs $P = <B,C,D,\sqsupseteq>$ require the value set $<D,\sqsupseteq>$ to be a complete partial order with least element $\perp$. Furthermore each object $u \in B$ is required to have a single constraint

$$u \sqsupseteq C(u).$$

That is, these programs view C as a function from B to B, rather than as a set of ordering constraints on pairs of objects drawn from B. Additionally, C is required to be the identity function on D.

Reductive partial order programs thus have constraints only of the form $u \sqsupseteq v$ where u is not a value, but v may be. Such programs can offer only lower bounds for an object u.

(2) *Continuous Monotone Partial Order Programming*

Continuous Monotone programs are Reductive programs that have constraints only of the form

$$u_i \sqsupseteq f_i(u_{i_1}, \ldots, u_{i_n})$$

where f_i is a symbol or expression denoting a specific continuous monotone function on D.

The set of objects B here includes not only 'atoms' (or 'variables') u_i, then, but also the 'expressions' constructible from these atoms and a specified set of continuous monotone functions. That is, we let B be a set of expressions using these function symbols over a set of *atoms* A and the values in D, and let C represent *reduction* or evaluation among expressions in B. Thus C defines a computation rule [9].

The partial order program definition given earlier is general enough to encompass this extension, although here the definition of a solution to a problem is made stricter, since the objects and constraints have more structure than before. Solutions must respect expressions. In other words, the value a solution assigns to the expression $f(u_1, \ldots, u_n)$ is required to be either $\perp$, or the value we get by applying the function denoted by f to the values the solution assigns to $u_1, \ldots, u_n$.

(3) *Semilinear Partial Order Programming*

Semilinear programs are Continuous Monotone programs that have constraints of the form

$$u_i \sqsupseteq \left(\boxplus_j a_{ij} \boxtimes u_{i_j} \right) \boxplus b_i$$

where $\boxplus$ and $\boxtimes$ are continuous monotone operators from a semiring that is also a complete partial order with least element 0. (In [10] we show that a more general semiring-like structure called a pointed continuous complete ordered semimodule can be used here instead.) Many Continuous Monotone programs can be 'linearized' into Semilinear programs. This is not surprising in retrospect, as monotonicity is a restriction very like linearity.

3.3. Procedural Semantics and Assignment Refinement

The three classes of problems are important because we have simple procedures for finding solution semantics for them.

First, it is not hard to show that solutions for all Reductive problems can be found by repeated *reduction*. We go about finding a value for any object u by constructing the sequence $C(u)$, $C^2(u)$, ..., $C^k(u)$ of objects. This sequence then satisfies the constraints

$$u \sqsupseteq C(u) \sqsupseteq C^2(u) \sqsupseteq \cdots \sqsupseteq C^k(u).$$

Either for some eventual value k, $C^k(u)$ is a value $v \in D$, in which case we can assign u the value v, or u can be assigned the value $\perp$. This assignment is always a solution to any Reductive problem, since it assigns the least possible value to each object. Furthermore the value for any goal object g can be found simply by constructing its sequence.

Second, we can obtain solutions for all three problem classes via *relaxation*. By relaxation we mean an iterative procedure which, given an initial value (namely: $\perp$) for the objects in B–D, repeatedly selects an unsatisfied constraint $u \sqsupseteq C(u)$ and enforces it by replacing the current assignment for u with the currently assigned value of the object given by $C(u)$. ('Out of kilter' constraints are thus brought 'into kilter'.) Repeated enforcement of the constraint $u \sqsupseteq C(u)$ gives an ascending sequence

$$\perp = v^{(0)} \sqsubseteq v^{(1)} \sqsubseteq \cdots \sqsubseteq v^{(N)}$$

of values for u. A consequence the Kleene fixed point theorem presented at the beginning of this section is that for the three classes of partial order programs above, enforcement of constraints in any 'fair' order (any order that eventually enforces all unsatisfied constraints) will ultimately produce a least solution.

'Single-assignment' semantics in programming languages have recently grown in importance. Relaxation semantics might be called *'assignment refinement'*: an assignment $u := v^{(k)}$ made in a partial order program can be superceded by $u := v^{(k+1)}$ provided that $v^{(k+1)} \sqsupseteq v^{(k)}$. In other words, we can replace the value of an object with better and better 'approximations', or refinements, for the value of the object. This idea appears to have applications in other programming contexts.

Linearity is interesting not only because it provides a new point of view, but also because it permits us to draw on known algorithms for solving linear algebraic systems. When **A** is a square matrix, the Semilinear programming program

$$\mathbf{u} \sqsupseteq \mathbf{A} \boxplus\!\boxtimes \mathbf{u} \boxplus \mathbf{b}$$

is solvable by *elimination* methods such as Gaussian or Gauss-Jordan elimination. Work remains in understanding how we can generalize on elimination methods. Zimmermann [14] gives an excellent survey of work in this area.

4. Programming Paradigms

The examples in the first section show that partial order programming has applications in relaxation computations. Also, it clearly has a great deal to do with mathematical programming. However, partial order programming can also be treated as a computer programming paradigm.

For example, logic programs can be expressed as partial order programs. All Horn rules are inequalities: the rule $H \leftarrow G$ expresses precisely the constraint "*truth*(H)" $\geq$ "*truth*(G)". Logical implication ($\leftarrow$) is just the ordering **true** $\geq$ **false** (i.e., **true** $\leftarrow$ **false**) on D = {**true**,**false**}.

Full logic programs can be expressed as systems of inequalities as follows. A collection of $m \geq 1$ Horn rules

$$p(t_{11}, \ldots, t_{1n}) \leftarrow G_1$$
$$\ldots$$
$$p(t_{m1}, \ldots, t_{mn}) \leftarrow G_m$$

is logically equivalent to the combined rule

$$p(X_1, \ldots, X_n) \leftarrow (X_1 = t_{11} \wedge \ldots \wedge X_n = t_{1n} \wedge G_1) \vee \ldots \vee (X_1 = t_{m1} \wedge \ldots \wedge X_n = t_{mn} \wedge G_m)$$

provided we include $X = X \leftarrow$ **true**. This combined rule can be viewed as an ordering among terms. With it, instances of the head of the rule can be rewritten to instances of its body, which is an expression involving equalities and other subgoals. The procedural semantics for logic programming can then be viewed accurately as a process of rewriting one goal to another that contains a reduced binding (reduced system of equalities) as a disjunct. The rewriting process combines the Prolog II reduction process for equations, presented in section 3 of [2], and goal elimination, the replacement of atomic subgoals by goals corresponding to the bodies of rules. It is similar to the "surface deduction" process developed by Cox and Pietrzykowski formally in [3], extending Colmerauer's work.

Rather than reproduce the reduction algorithm formally as a function C, we give an example that should illustrate the process clearly. With the convention above the standard 'append' predicate looks like

$$\text{append}(A,B,AB) \leftarrow (A = [\,] \wedge B = AB) \vee (A = [X|L] \wedge AB = [X|LB] \wedge \text{append}(L,B,LB)).$$

The goal append(Y,Z,[a]) is reducible to a binding with the following sequence:

append(Y,Z,[a])
$(Y = [\,] \wedge Z = [a]) \vee$ $(Y = [X_1|L_1] \wedge [a] = [X_1|LB_1] \wedge \text{append}(L_1,Z,LB_1))$
$(Y = [\,] \wedge Z = [a]) \vee$ $(Y = [X_1|L_1] \wedge X_1 = a \wedge LB_1 = [\,] \wedge \text{append}(L_1,Z,LB_1))$
$(Y = [\,] \wedge Z = [a]) \vee$ $(Y = [X_1|L_1] \wedge X_1 = a \wedge LB_1 = [\,] \wedge$ $((L_1 = [\,] \wedge Z = [\,]) \vee$ $(L_1 = [X_2|L_2] \wedge [\,] = [X_2|LB_2] \wedge \text{append}(L_2,Z,LB_2)))$

$$
\begin{array}{|l|}
\hline
(\ Y = [\,] \ \wedge\ Z = [a]\ ) \ \vee \\
(\ Y = [X_1 | L_1] \ \wedge\ X_1 = a \ \wedge\ LB_1 = [\,] \ \wedge \\
\qquad ((\ L_1 = [\,] \ \wedge\ Z = [\,]\ ) \ \vee \\
\qquad (\ L_1 = [X_2 | L_2] \ \wedge\ \textbf{false} \ \wedge\ \text{append}(L_2, Z, LB_2)\ ))) \\
\hline
(\ Y = [\,] \ \wedge\ Z = [a]\ ) \ \vee \\
(\ Y = [X_1 | L_1] \ \wedge\ X_1 = a \ \wedge\ LB_1 = [\,] \ \wedge \\
\qquad ((\ L_1 = [\,] \ \wedge\ Z = [\,]\ ) \ \vee \\
\qquad \textbf{false}\ )) \\
\hline
(\ Y = [\,] \ \wedge\ Z = [a]\ ) \ \vee \\
(\ Y = [X_1 | L_1] \ \wedge\ X_1 = a \ \wedge\ LB_1 = [\,] \ \wedge\ L_1 = [\,] \ \wedge\ Z = [\,]\ ) \\
\hline
\end{array}
$$

Partial order programming gives insights about different programming paradigms by expressing them in terms of ordering. For example, van Emden [5] proposes an extension of logic programming in which the semantics of a logic program are generalized from true-false assignments to assignments with attenuation factors, real values between 0 and 1, which can be viewed as representing some kind of certainty factor. The quantitative logic program

$$
\begin{array}{lll}
a & \leftarrow 0.50 - & b \,\&\, f \\
a & \leftarrow 0.50 - & c \,\&\, d \\
b & \leftarrow 0.20 - & \\
c & \leftarrow 0.45 - & \\
c & \leftarrow 0.30 - & \\
d & \leftarrow 1.00 - & \\
e & \leftarrow 0.50 - & \\
f & \leftarrow 0.90 - & e
\end{array}
$$

has semantics assigning the object b the value (greatest lower bound) 0.20, the object c the value max(0.45, 0.30) = 0.45, and the object a the value

$$
\begin{array}{l}
\max(\ 0.50 * \min(\ 0.20,\ 0.90*(0.50)\ ), \\
\qquad 0.50 * \min(\ 0.45,\ 1.00\ )\qquad\ \ ) \ =\ 0.225,
\end{array}
$$

because the '&' operator is defined to operate like 'min' and different clauses are combined with 'max', with the various attenuation factors multiplied in.

The corresponding partial order program makes the semantics of this program evident:

$$
\begin{array}{lll}
a & \geq & 0.50 * \min(\ b, f\ ) \\
a & \geq & 0.50 * \min(\ c, d\ ) \\
b & \geq & 0.20 \\
c & \geq & 0.45 \\
c & \geq & 0.30 \\
d & \geq & 1.00 \\
e & \geq & 0.50 \\
f & \geq & 0.90 * e.
\end{array}
$$

This translation from quantitative logic programs to inequalities is also sufficient to convert propositional logic programs (quantitative programs like the one above, but where all attenuation factors are 1.00) into systems of inequalities.

Other programming paradigms can be expressed as partial order programs as well. Functional programming, and more generally 'reductive' systems, can be expressed naturally in terms of a reduction ordering. Relationships such as

$$
\begin{array}{l}
(((\lambda x\ .\ \lambda y\ .\ x)\ a)\ b) \ \rightarrow\ ((\lambda y\ .\ a)\ b) \\
((\lambda y\ .\ a)\ b) \ \rightarrow\ a
\end{array}
$$

can be viewed as part of the definition of a partial order $\rightarrow$ (more precisely a preorder, since acyclicity of the ordering may not be guaranteed). In some situations, this inequality ordering can even be more natural than the 'one-way equality' relationship that is often associated with reduction rules.

A very rough tabular comparison can be made, then, to illustrate how other paradigms can be viewed as instances of partial order programming:

Partial Order Programming		*Logic Programming*		*Functional Programming*	
minimize	u	$\leftarrow$	u	evaluate	u
subject to	$u_1 \sqsupseteq v_1$	subject to	$u_1 \leftarrow v_1$	subject to	$u_1 \rightarrow v_1$
	$\cdots$		$\cdots$		$\cdots$
	$u_n \sqsupseteq v_n$		$u_n \leftarrow v_n$		$u_n \rightarrow v_n$

Casting problems in a partial order programming framework can give fresh perspective onto how programming paradigms can be structured, and onto how different paradigms can be combined. In addition, the relationship between basic fixed point results in denotational semantics and relaxation problem solving is made explicit. From the foregoing we can see that relaxation solves those problems that can be cast in the format of a system of inequalities and goal

minimize	$\mathbf{u}$
subject to	$\mathbf{u} \sqsupseteq \mathbf{f}(\mathbf{u})$

where $\mathbf{f}$ is a continuous monotone function, and the set D of values used is a complete partial order with a least element $\perp$. Here $\sqsupseteq$ is the order on D extended to a partial order of vector component-wise domination on the space of vectors over D.

Similarly, any recursive programming paradigm with least fixed point semantics can be presented as partial order programming. Least fixed point semantics assign to each program π a continuous monotone functional $T_\pi : D \rightarrow D$ on some space D of semantic "interpretations" of the program. These interpretations are typically subsets of the input-output relation defined by the program. The smallest interpretation I satisfying $I = T_\pi(I)$ gives the least fixed point semantics for π [9]. Given π then, the continuous monotone partial order programming problem

minimize	I
subject to	$I \sqsupseteq T_\pi(I)$

where D is the set of interpretations of π ordered by $\sqsupseteq$, which is inclusion among interpretations, B is the set of objects $\{\ T_\pi^k(x) \mid k \in \omega,\ x = I \text{ or } x \in D\ \}$ (which includes I), and C is defined by T_π. Any programming paradigm with least fixed point semantics is in this abstract sense a special case of partial order programming.

5. Prospects

This paper has informally introduced partial order programming, and has illustrated the nature of the paradigm by offering examples, by pointing out its basic semantic properties, and by relating it to existing paradigms. Clearly much more can be said about the issues the paradigm raises, and a more thorough analysis of many aspects discussed here can be found in [10].

Chandy and Misra point out in [1] that "the utility of any new approach is suspect, especially when the approach departs radically from the conventional." The remainder of this section highlights some characteristics of partial order programming that offer some further perspective on its potential.

5.1. Connection of Diverse Fields

The examples shown earlier demonstrate that problems in diverse fields can be expressed naturally with the partial order programming paradigm. The paradigm seems to have promise for use in operations research, modeling, some types of AI programming, and numerical problem solving. The blend of numerical and symbolic problems suitable for description with the paradigm is difficult to characterize at this point, but seems to be useful. Some partial order programs correspond to systems of linear inequalities, using addition and multiplication operators from some semiring or semimodule. Parallelism seems to be extractable from these problems. Furthermore partial order programming appears to embrace multiple programming paradigms and integration of environments, including for example fixed-point paradigms not discussed here [1,4,12].

5.2. Concurrency

Partial orders make natural models of concurrency in programs. Any sort of precedence or sequencing constraints gives rise to a partial order. Recently interest has grown in applying partial order models of concurrency. As Pratt remarks in [13], some concepts of concurrency are definable only for partial orders, the meaning of concurrency of two events in partial order models does not depend on the granularity of atomicity of the events, and partial order models are in certain cases easier to reason about than linear models. Recently we have shown [11] that partial order programming can naturally capture specifications of directedness and argument typing in logic programs. These results appear particularly advantageous for stream processing programs.

5.3. Modeling and Knowledge Representation

Although we have not discussed the issue in this brief overview, the partial order programming the paradigm fits many modeling (also known as knowledge representation) concepts naturally. These concepts include inference, type hierarchies, constraint satisfaction, hill-climbing, inexact reasoning, spreading activation, etc. Ordering also underlies important knowledge representation concepts such as composition, part-of relationships and aggregation, spatial relationships, temporal relationships, dependencies, causal relationships, possession, strength of conviction, preference, utility, planning, procedures, reductive problem-solving, chains of reasoning, and heuristics. *Humans are very good at reasoning about ordering.* In [10] more evidence is offered on why partial order programming may have advantages in modeling complex systems and human reasoning.

Acknowledgement

In 1974 Dave Kuck posed the question to the author of how to characterize the significance of the many incarnations of the generalized matrix product. In 1983 Paul Eggert and the author came up with the idea of 'relaxation programming'. This paper is a result of several years of subsequent ruminations. The author is indebted to many colleagues for suggestions that have corrected or improved the presentation here, but particularly to Paul Eggert for his careful readings, great ideas, and enthusiasm.

References

1. Chandy, K.M. and J. Misra, *Parallel Program Design: A Foundation,* Addison-Wesley, Reading, MA, 1988.
2. Colmerauer, A., "Equations and Inequations on Finite and Infinite Trees," *Proc. Intnl. Conf. on Fifth Generation Computer Systems (FGCS'84),* pp. 85-99, North-Holland, Tokyo, November 1984.
3. Cox, P.T. and T. Pietrzykowski, "Surface Deduction: a uniform mechanism for logic programming," *Proc. Symposium on Logic Programming,* pp. 220-227, IEEE Computer Society #636, Boston, 1985.
4. Dijkstra, E.W. and C.S. Scholten, "Termination Detection for Diffusing Computations," *Information Processing Letters,* vol. 11, no. 1, pp. 1-4, 29 August 1980.
5. van Emden, M.H., "Quantitative Deduction and Its Fixpoint Theory," *Journal of Logic Programming,* vol. 3, no. 1, pp. 37-53, April 1986.
6. Isaacson, E. and H.B. Keller, *Analysis of Numerical Methods,* J. Wiley & Sons, New York, 1966. (Chapter 9, Section 2: Solution of Laplace Difference Equations.)
7. Kuck, D.J., *The Structure of Computers and Computations,* J. Wiley & Sons, New York, 1978.
8. Mackworth, A.K. and E.C. Freuder, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence,* vol. 25, pp. 65-74, 1985.
9. Manna, Z., *Mathematical Theory of Computation,* McGraw-Hill, New York, 1974.
10. Parker, D.S., "Partial Order Programming," Technical Report CSD-870067, UCLA Computer Science Dept., Los Angeles, CA 90024-1596, 1987.
11. Parker, D.S. and R.R. Muntz, "A Theory of Directed Logic Programs and Streams," in *Logic Programming,* ed. R.A. Kowalski, K.A. Bowen, pp. 620-650, MIT Press, August 1988.
12. Parnas, D.L., "A Generalized Control Structure and Its Formal Definition," *Comm. ACM,* vol. 26, no. 8, pp. 572-581, August 1983.
13. Pratt, V., "Modelling Concurrency with Partial Orders," *International J. Parallel Programming,* vol. 15, no. 1, pp. 33-71, 1986. Also Stanford Tech. Report STAN-CS-86-1113, June 1986.
14. Zimmermann, U., *Linear and Combinatorial Optimization in Ordered Algebraic Structures,* North-Holland, New York, 1981. Annals of Discrete Mathematics, vol. 10.