Preference Logic Programming: Optimization as Inference

Kannan Govindarajan
Bharat Jayaraman

Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
E-Mail: {bharat,govin-k}@cs.buffalo.edu

Surya Mantha

System Sciences Laboratory
Building 128
Xerox, Webster Research Center
Webster, NY 14580
E-Mail: mantha@wrc.xerox.com

Abstract

Preference Logic Programming (PLP) is an extension of Constraint Logic Programming (CLP) for declaratively specifying optimization problems. In the PLP framework, the *definite clauses* of a CLP program are augmented by two new kinds of clauses: *optimization* clauses and *arbiter* clauses. *Optimization* clauses specify which predicates are to be optimized and *arbiter* clauses specify the criteria to be used for optimization. Together, these three kinds of clauses form a *preferential theory*, for which a *possible worlds* semantics was first given by Mantha *et al.* This paper shows how modal concepts can be used to capture the notion of optimization: Essentially, each world in the possible-worlds semantics for a preference logic program is a model of the program, and an ordering over these worlds is enforced by the arbiter clauses in the program. We introduce the notion of *preferential consequence* as truth in the *optimal* worlds. We propose an operational semantics that is an extension of SLD derivation and prove its soundness. Finally, we provide a variety of examples to illustrate our paradigm: minimum and maximum predicates, partial-order programming, syntactic ambiguity resolution and its application in document formatting, and general optimization problems.

1 Introduction

This paper presents a principled approach to extending Constraint Logic Programming (CLP) languages [JL87, vH89, JM] for declaratively specifying optimization problems. The specification of an optimization problem in general has three components:

1. *specification of the constraints of the problem;*
2. *specification of what is to be optimized (the space of feasible solutions); and*
3. *specification of the criteria for determining the optimal solution.*

The CLP framework provides a declarative approach only to one of these three components, namely, (1), since optimization is a meta-level concept in constraint logic. While several approaches to optimization have been proposed in the LP community [GGZ91, Fag93, MS89, Par89, JOM93], it is not clear in these approaches that one can fully specify all three components of an optimization problem. The contribution of this paper lies in showing how the use of *preference logic* provides a natural, declarative, and general means of specifying optimization problems.

We first briefly illustrate our proposed approach of *preference logic programming* by showing how the above three components are specified for the shortest-distance example. The constraints are specified by definite clauses as follows (assuming a set of clauses for the predicate *edge*):

$$path(X, Y, C) \leftarrow edge(X, Y, C).$$

$$path(X, Y, C) \leftarrow edge(X, Z, C1),\ \ path(Z, Y, C2),\ \ C = C1 + C2.$$

To specify what is to optimized and the criteria for the optimal solution, we introduce two new kinds of clauses: *optimization* clauses and *arbiter* clauses. The optimization clause for this example is:

$$sh\_dist(X, Y, C) \rightarrow path(X, Y, C).$$

This clause states that the solution to the goal $sh\_dist(X, Y, C)$ is an *optimal* solution to the goal $path(X, Y, C)$. (We permit in general a conjuction of literals on the right-hand side of $\rightarrow$.) Lastly, the arbiter clause below specifies that, given two paths between two nodes of the graph, we prefer the path of lesser cost ($\mathcal{P}_f$ is a monadic modal operator of preference). The precise logical reading of this clause will be given later.

$$sh\_dist(X, Y, C1) \rightarrow \mathcal{P}_f(sh\_dist(X, Y, C2) \wedge C2 < C1).$$

We show in this paper that a variety of optimization problems can be formulated clearly and concisely with these three types of clauses. We provide semantics for preference logic programs using concepts from modal logic. The use of modal logic should not be surprising since optimization is a meta-level concept. Essentially, we give a possible-worlds semantics for a preference logic program in which each world is a model for the constraints of the program, and an ordering over these worlds is enforced by the arbiter clauses in the program. We introduce the concept of *preferential consequence* to refer to truth in the *optimal* worlds (in contrast with *logical consequence* which refers to truth in *all* worlds). An *optimal answer* to the query G is a substitution θ such that $G\theta$ is a *preferential consequence* of the preference logic program. We provide a derivation scheme called **PTSLD**-derivation, which stands for Pruned Tree SLD-derivation, for efficiently computing the *optimal answers* to queries. Because we are not carrying out general proving theorems in the logic of preference, we are able to devise an efficient derivation procedure.

Several approaches have been proposed to the problem of incorporating optimization in a CLP framework: [MS89] discusses how to incorporate optimization queries into a CLP system by mapping the solutions of a query to a partial order; [Fag93] describes a semantics for optimization predicates in CLP languages based on Kunen-Fitting's semantics for negation; [GGZ91] shows how under certain monotonicity conditions, optimization predicates such as minimum and maximum predicates can be efficiently computed; [Par89, JOM93] discuss partial-order programming over lattice domains in which the notion of maximizing (minimizing) is incorporated directly into the semantics by taking least upper bounds (greatest lower bounds); [BMMW89, WB93] discuss Hierarchical CLP (HCLP) which is an extension to CLP where (numeric) strengths are associated with constraints, and the desired behavior of the system is captured by trying to satisfy the constraints in the "best" possible way. Such an approach was shown to be natural for describing the behavior of interactive graphics applications declaratively.

Our proposed approach has many strengths: The key distinguishing feature of our approach is that it allows the programmer to program the preference criteria to suit the application. In contrast, [GGZ91, Fag93] only provide optimization predicates that can be expressed as maximizing or minimizing some objective function; similarly, in the partial-order programming framework of [Par89, JOM93], one computes the greatest lower bound or the least upper bound of elements in a partial order. Moreover, our semantics for optimization is expressed by explicit selection, rather than via negation, as in [GGZ91, Fag93]. In comparison with HCLP, we note that our focus is on the class of optimization problems where it is more natural to compare alternative solutions and specify which is "better" with respect to preference criteria. Another important feature of PLP is that it allows one to specify both the logic and the control components of an algorithm [Kow79] in a modular and declarative fashion. The constraints of a PLP program specify the logic, and the preference clauses specify the control. Essentially, the preference clauses offer control advice to the SLD inference engine about which paths in the proof tree are better.

The rest of this paper is organized as follows. Section 2 introduces the basic ideas of preference logic and general preferential theories. Section 3 defines preference logic programs as a restriction of general preferential theories, and also formalizes the concept of preferential consequence. Section 4 illustrates how various optimization problems can be expressed as preference logic programs. Section 5 describes the PTSLD-derivation scheme, and presents the soundness and completeness theorems and proofs for this derivation scheme. Section 6 discusses in some detail how a practical example, line-breaking [KP81], can be expressed in the PLP framework. Section 7 presents conclusions and areas for future work.

2 The Logic of Preference

The concept of preference has been studied extensively in decision theory, economics, ethics and philosophical logic. In a very broad sense preference is a binary relation over an appropriate domain. It has been suggested that the preference relation of primary importance is the one between *states of affairs* (or *possible worlds*) characterized by possibly infinite sets of propositions. Rather than characterize states of affairs themselves by single propositions, [Man91] suggests that the reason for the betterness of one state of affairs over another be captured by single propositions (*preference criteria*). Our discussion of preference logics below is based on [Man91, BMW94]. Familiarity with modal logic, as described in [Ram88] or [Che80], is sufficient.

2.1 The Modal Logic of Pure Preference $\mathcal{P}$

The syntax of $\mathcal{P}$ is obtained by extending the syntax of propositional logic $\mathcal{L}$ by a monadic modal operator $\mathcal{P}_f$, with a corresponding rule of formation: if F is a formula of $\mathcal{P}$, then so is $\mathcal{P}_f F$. In the tradition of modal logics, [BMW94] provides a *possible worlds* semantics for preference logics, as follows.

Definition 1 *A* **preference frame** *$\mathcal{F}$ is an ordered pair of the form $\langle \mathcal{W}, \preceq \rangle$, where $\mathcal{W}$ is a non-empty set of possible worlds and $\preceq$ is a binary relation over $\mathcal{W}$.*

Definition 2 *A* **preference model** *$\mathcal{M}$ is a preference frame $\mathcal{F}$ along with a valuation function $\mathcal{V}$ that determines the truth of atomic formulae at individual worlds. Boolean connectives such as $\wedge, \vee, \neg, \rightarrow$, etc. have the standard interpretation. The semantics of preference formulae of the form $\mathcal{P}_f F$ is given as follows:*

$$\models^w_{\mathcal{M}} \mathcal{P}_f F \text{ iff } (\forall v \in \mathcal{W}) \, [(\models^v_{\mathcal{M}} F) \rightarrow (w \preceq v)].$$

Informally, $\mathcal{P}_f F$ is true in a world w in a preference model iff every world v where F is true is related to w by the relation $w \preceq v$. If $\mathcal{P}_f F$ is true at a world w, then F is said to be a *preference criterion* at world w. In other words, any world v where F is true is at least as good as w.

Definition 3 *A preference model $\mathcal{M}$ is said to be* **supported** *if and only if, for any two worlds w and v, if $w \preceq v$ then, there is a formula $\mathcal{P}_f A$ such that $\models^w_{\mathcal{M}} \mathcal{P}_f A$ and $\models^v_{\mathcal{M}} A$.*

A supported preference model is also the preference model that minimizes the relation $\preceq$.

2.2 Properties of $\mathcal{P}$

The logic $\mathcal{P}$ is equipped with all the axioms schemes and rules of inference of standard propositional logic such as *modus ponens*. In addition we have the following rule of inference:

$$\frac{\vdash (A_1 \wedge \ldots \wedge A_n) \rightarrow A}{\vdash (\mathcal{P}_f(\neg A_1) \wedge \ldots \wedge \mathcal{P}_f(\neg A_n)) \rightarrow \mathcal{P}_f(\neg A)} \qquad (n \geq 0)$$

Definition 4 *Given a preference model $\mathcal{M}$, a formula F is said to be true in $\mathcal{M}$ (denoted $\models_{\mathcal{M}} F$) if it is true in every world in $\mathcal{M}$. F is said to be* **valid** *in a frame $\mathcal{F}$ (denoted $\models_{\mathcal{F}} F$) if it is true in every model built from $\mathcal{F}$ irrespective of the valuation function $\mathcal{V}$. F is said to be valid in a class of frames $\mathcal{F}'$ (denoted $\models_{\mathcal{F}'} F$) if it is valid in every frame in $\mathcal{F}'$.*

Theorem 1 [BMW94] *The Propositional Logic of Pure Preference $\mathcal{P}$ with the rule of inference mentioned above is both sound and complete in the class of all frames ($\mathcal{U}$). In other words, $\vdash F$ iff $\models_{\mathcal{U}} F$.*

We will be working with a first-order version of the logic of pure preference. The syntax of this logic is derived from the syntax of first-order logic (with quantifiers $\exists$ and $\forall$) by adding the modal operator $\mathcal{P}_f$ with the associated rule of formation. We also assume that the Barcan formulae [Che80, Ram88] are true, that each function symbol is a *rigid functor*, that each term is a *rigid designator*, and the domain over which terms are constructed is fixed in all worlds. Under these assumptions, the logic can be proved to be sound and complete. Moreover, these restrictions are natural for our applications, because each possible world in a preference model corresponds to a feasible solution. Since we wish to compare solutions across worlds, it is natural to assume that the domains are fixed and solutions denoted by the same term in different worlds are indeed the same solution. For our purposes, each world is derived from a pre-interpretation in which some of the function symbols are interpreted as in CLP, *e.g.*, $+$ in the shortest distance example, and the other function symbols are freely interpreted as in the Herbrand pre-interpretation.

2.3 General Preferential Theories

In this subsection, we briefly review the general preferential theories as introduced in [BMW94]. Informally a preferential theory is an abstract specification of an optimization problem. It has two components: a first order theory $\mathcal{T}$ and an arbiter $\mathcal{A}$. The preference model that we associate with a preferential theory is such that every world in the preference model is a model for the first-order theory and the ordering among the worlds is enforced by the arbiter.

Definition 5 *A* **preference clause** *is the universal closure of a formula of the form* $\wedge_k M_k \rightarrow \mathcal{P}_f(\wedge_i L_i)$ *where each of the* M_k*s and* L_i*s are literals. An* **arbiter** *is a finite collection of preference clauses. A* **preferential theory** *is a conjunction of a first-order theory* $\mathcal{T}$ *and an arbiter* $\mathcal{A}$.

Given a first-order language $\mathcal{L}$ consisting of constants, function symbols, *etc.*, a pre-interpretation consists of the following: a non-empty set $\mathcal{D}$ called the domain of the pre-interpretation; an assignment to each constant in $\mathcal{L}$ an element of $\mathcal{D}$; an assignment to every n-ary function symbol in $\mathcal{L}$, a mapping from $\mathcal{D}^n$ to $\mathcal{D}$. Given a pre-interpretation I, an I-based interpretation is one that assigns to every n-ary predicate a mapping from $\mathcal{D}^n$ to $\{true, false\}$.

Definition 6 *Given a pre-interpretation* I*, an* I**-based preference model** *is a 3-tuple* $\mathcal{M} = \langle \mathcal{W}, \preceq, \mathcal{V} \rangle$ *such that* $\mathcal{V}$ *(by slight abuse of notation) assigns* I*-based interpretations to the worlds in* $\mathcal{W}$.

Given a preferential theory $\mathcal{PT} = \mathcal{T} \bigwedge \mathcal{A}$ and a pre-interpretation I, there could be many I-based preference models in which the theory is true. These I-based preference models have the property that $\mathcal{V}$ assigns to worlds interpretations that are models of $\mathcal{T}$, and $\preceq$ is a relation that satisfies the arbiter. $\preceq$ is said to satisfy the arbiter if the following condition holds: if there exists a ground instance of a preference clause $B \rightarrow \mathcal{P}_f H$ such that $\models^w_{\mathcal{M}} B$ and $\models^{w'}_{\mathcal{M}} H$, then $w \preceq w'$. For our application, we are interested in preference models that take into account all possible feasible solutions and impose an ordering only when required. For any given preferential theory and a pre-interpretation, such a model exists and is unique upto an isomorphism [BMW94].

Definition 7 *Given a preferential theory* $\mathcal{PT}$ *and a pre-interpretation* I*, the* **intended I-based preference model** *is the supported model that maximizes the number of worlds in* $\mathcal{W}$ *and* $\mathcal{V}$ *does not assign the same interpretation to two distinct worlds* w*,* w' *in* $\mathcal{W}$.

Definition 8 *Given a preference model $\mathcal{M} = \langle \mathcal{W}, \preceq, \mathcal{V} \rangle$, a world $w \in \mathcal{W}$ is said to be* **strongly optimal** *if and only if there is no world w' different from w such that $w \preceq w'$. A world $w \in \mathcal{W}$ is said to be* **weakly optimal**, *if for every world $w' \in \mathcal{W}$, if it is the case that $w \preceq w'$ then $w' \preceq w$.*

Definition 9 *Given a preferential theory $\mathcal{PT}$ and a formula F,*

1. *F is said to be a* **strong skeptical preferential consequence** *of $\mathcal{PT}$, if, for all pre-interpretations, F is true in all the strongly optimal worlds in the intended preference model.*

2. *F is said to be a* **weak skeptical preferential consequence** *of $\mathcal{PT}$, if, for all pre-interpretations, F is true in all the weakly optimal worlds in the intended preference model.*

3. *F is said to be a* **strong credulous preferential consequence** *of $\mathcal{PT}$, if, for all pre-interpretations, F is true in some the strongly optimal world in the intended preference model.*

4. *F is said to be a* **weak credulous preferential consequence** *of $\mathcal{PT}$, if, for all pre-interpretations, F is true in some the weakly optimal world in the intended preference model.*

These notions of skeptical and credulous consequences are derived from similar notions in autoepistemic logics, [MT91]. In this paper we will be using the notion of weak credulous preferential consequence.

Constraint optimization problems are expressible as preferential theories as follows. The *constraints* form the first-order theory; each possible world corresponds to a feasible solution to the set of constraints; the *arbiter* orders the possible worlds; and the solutions of interest are those in weakly optimal worlds.

3 Preference Logic Programs

The general preferential theories in section 2, though clearly powerful enough to express optimization problems declaratively, are too general; it is not obvious what sort of procedural interpretation one should associate with these theories. In this section we introduce restrictions on the first-order theory and the arbiter that still provide adequate power to express a wide variety of optimization problems. We refer to this restricted class of theories as *Preference Logic Programs.* The relationship between preference logic programs and general preference theories is akin to that between definite clauses and first-order logic.

3.1 Restricted Preferential Theories

The preferential theories in the previous section had two parts: a first-order theory $\mathcal{T}$ and an arbiter $\mathcal{A}$. We now describe restrictions on $\mathcal{T}$ and $\mathcal{A}$ that give rise to preference logic programs.

Preference Logic Programs contain three kinds of predicates:

1. *C-predicates* are defined using other C-predicates through definite clauses.

2. *Opt-predicates* are defined using $\rightarrow$ clauses.

3. *O-predicates* are defined using $\leftarrow$ clauses and there is some goal in the body of a clause defining a O-predicate whose head is an Opt-predicate or an O-predicate.

The first-order part of a preference logic program is built up of the following kinds of clauses:

1. Core clauses of the form $H \leftarrow B_1, \ldots, B_n$, where each B_i is defined using other core clauses.
2. Optimization clauses of the following two forms
   - $H \rightarrow B_1, \ldots, B_m$. Where H is an Opt-predicate and each of the B_i could be a C-predicate, an Opt-predicate or an O-predicate. Note the direction of the implication in this formula. The intuition here is that the set of optimal solutions (*i.e.,* the set of binding for variables in the head of a $\rightarrow$ clause) is some subset of the feasible solutions (*i.e,* the set of bindings for the variables in the body).
   - Definite clauses of the form $H \leftarrow B_1, \ldots, B_n$, where at least one B_i is an Opt-predicate or an O-predicate.

The first order theory $\mathcal{T}$ can therefore be divided into two parts, the definitions of the C-predicates that make up the core program $\mathcal{T}_C$ and the definitions of the Opt-predicates and the O-predicates that make up the optimization program $\mathcal{T}_O$.

The arbiter part of a preference logic program has the following form:

$$p(\bar{t}) \;\rightarrow\; \mathcal{P}_f(p(\bar{u}) \wedge \bigwedge L_i) \qquad\qquad (i \geq 0)$$

where p is an Opt-predicate and each L_i is some atom. In all examples in this paper, each L_i is a constraint ($\geq, \leq, <, >$, etc.) over some domain. In essence this form of the arbiter states that the solution to the goal p in $\bar{u}$ is better than the solution in $\bar{t}$ because of $\wedge_i L_i$.

A preference logic program $\mathcal{P}$ can thus be characterized as a 3-tuple $\langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$, where $\mathcal{T}_C$ and $\mathcal{T}_O$ together form $\mathcal{T}$ and $\mathcal{A}$ is the arbiter.

For example, suppose we had the following preference logic program, which computes the list of edges along the shortest path from a source to a target node:

$$\begin{array}{rcl}
path(X,Y,C,[e(X,Y)]) & \leftarrow & edge(X,Y,C). \\
path(X,Y,C,[e(X,Z)|L1]) & \leftarrow & edge(X,Z,C1),\ path(Z,Y,C2,L1),\ C = C1 + C2. \\
\\
sh_dist(X,Y,C,L) & \rightarrow & path(X,Y,C,L). \\
sh_path(X,Y,L) & \leftarrow & sh_dist(X,Y,C,L). \\
\\
sh_dist(X,Y,C1,L) & \rightarrow & \mathcal{P}_f(sh_dist(X,Y,C2,L1) \wedge C2 < C1).
\end{array}$$

The first two definite clauses (and the clauses for the *edge* predicate) make up the *core program* $\mathcal{T}_C$. The third $\leftarrow$ clause and the $\rightarrow$ clause make up the *optimization program* $\mathcal{T}_O$. The only Opt-predicate is *sh_dist* and the only O-predicate is *sh_path*. The C-predicates are *path*, *edge* and $=$.

3.2 Preference Models for Preference Logic Programs

We now describe the intended preference model and the concept of *preferential consequence* of a preference logic program. The pre-interpretation I of interest to us interprets functions such as $+$ over the appropriate domain (as in CLP) and leaves all other function symbols uninterpreted (as in Herbrand interpretations). For the rest of the paper we fix on this pre-interpretation I. We are interested in a canonical model for the core program derived from the pre-interpretation I, as it specifies the constraints to be satisfied. For definite programs, this set is given by the least I-based model for the program. For programs with negation, we might use stable models, well-founded models, *etc.* The valuation for each world in the intended preference model is obtained by extending the canonical model of the core program with instances of the Opt-predicates and O-predicates so that it then becomes a model for the complete program.

We use the shortest distance example to illustrate the above ideas. Consider a directed graph with the following edges $\{edge(a,b,5), edge(b,c,10), edge(a,c,25)\}$ (see figure 1). The canonical model M for $\mathcal{T}_C$ of interest to us here is the set:

$$\{edge(a,b,5), edge(b,c,10), edge(a,c,25)\} \cup$$
$$\{path(a,b,5,[e(a,b)]), path(b,c,10,[e(b,c)]), path(a,c,25,[e(a,c)]), path(a,c,15,[e(b,c),e(a,b)])\}.$$

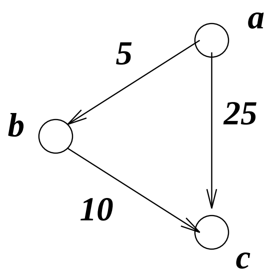

Figure 1: The Graph.

Definition 10 *Given a PLP* $= \langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$ *and a canonical model* M *for* $\mathcal{T}_C$*, a* **preference model** *for PLP is a 3-tuple* $\langle \mathcal{W}, \preceq, \mathcal{V} \rangle$*, where* $\mathcal{V}$ *assigns to each world* w *in* $\mathcal{W}$ *an extension of* M *to the Opt-predicates and O-predicates. Further the instances of C-predicates in each world are the same as in* M *and the O-predicates are supported (in the standard sense in Logic Programming, namely if a ground instance of O-predicate* A *is true at some world* w*, there must be some ground instance of a clause whose head unifies with* A *such that all the goals in the body are also true in* w*), such that* $\mathcal{T}_C$*,* $\mathcal{T}_O$ *and* $\mathcal{A}$ *are true at* w*.*

Definition 11 *Given a PLP, the* **intended preference model** $\mathcal{M}$ *is the preference model* $\langle \mathcal{W}, \preceq, \mathcal{V} \rangle$ *that maximizes the number of worlds in* $\mathcal{W}$ *and minimizes the relation* $\preceq$ *(i.e. is supported) and is such that* $\mathcal{V}$ *assigns different interpretations to different worlds.*

This definition of an intended preference model considers all possible solutions for the optimization predicates, and is crucial for the soundness and completeness of the execution model. Each world in the preference model is obtained by extending the least model for $\mathcal{T}_C$ so that it then becomes a model for $\mathcal{T}_C \wedge \mathcal{T}_O$. For the example program considered earlier, the intended preference model has three worlds. These are (restricted to C-predicates and O-predicates):

1. $M \cup \{sh\_dist(a,b,5,[e(a,b)]), sh\_dist(b,c,10,[e(b,c)]), sh\_dist(a,c,25,[e(a,c)]),$ $sh\_dist(a,c,15,[e(b,c),e(a,c)])\} \cup \{sh\_path(a,b,[e(a,b)]), sh\_path(b,c,[e(b,c)]), sh\_path(a,c,[e(a,c)]),$ $sh\_path(a,c,[e(b,c),e(a,b)])\}$

2. $M \cup \{sh\_dist(a,b,5,[e(a,b)]), sh\_dist(b,c,10,[e(b,c)]), sh\_dist(a,c,25,[e(a,c)])\}$ $\cup \{sh\_path(a,b,[e(a,b)]), sh\_path(b,c,[e(b,c)]), sh\_path(a,c,[e(a,c)])\}$

3. $M \cup \{sh\_dist(a,b,5,[e(a,b)]), sh\_dist(b,c,10,[e(b,c)]), sh\_dist(a,c,15,[e(b,c),e(a,c)])\}$ $\cup \{sh\_path(a,b,[e(a,b)]), sh\_path(b,c,[e(b,c)]), sh\_path(a,c,[e(b,c),e(a,b)])\}$

The arbiter is satisfied by the following binary relation $\preceq$ on the set of worlds:

$$\{1 \preceq 3, \quad 2 \preceq 3\}$$

We now define the notion of preferential consequence for formulae that do not involve any negation. Let us define **p-formulae** to be formulae that are constructed from atomic formulae by using the connective $\wedge$ and $\vee$ and the quantifiers $\exists$ and $\forall$, *i.e.,* formulae without negation.

Definition 12 *Given a PLP and a set of p-formulae X, X is said to be a* **positive preferential consequence** *of PLP, written PLP* $\mid\!\sim X$*, if X is true in some weakly optimal world in the intended preference model for PLP.*

We can extend this definition to formulae that involve negation as follows: Suppose F is $\neg G$ where G is a p-formula. F is a **preferential consequence** of a preference logic program PLP if G is not a positive preferential consequence of PLP.

Continuing the earlier example, the optimal world is the one in which only $sh\_dist(a,c,15,\_)$ is true and no other $sh\_dist(a,c,X,\_)$ for any X is true (model 3). Note that the $sh\_path(a,d,X)$ is true for only $X = [e(b,c),e(a,b)]$ in this world, confirming that sh_path computes actual shortest path. Note that adding the *edge* instance $edge(a,c,5)$ changes the shortest-path between a and c, *i.e.*, optimization is a non-monotonic notion [BMW94].

4 Examples of Preference Logic Programs

In this section we compare the paradigm with related approaches and we also give additional examples of preference logic programs

4.1 Minimum and Maximum Predicates

Ganguly *et al* [GGZ91] show how to express generalizations of graph-closure problems, such as the shortest path problem, using meta-level constructs such as *min.* They give a first-order semantics for these programs and describe an efficient *greedy fixpoint procedure* for computing the stable models of these programs under certain monotonicity restrictions. Fages [Fag93] asserts that to avoid dependence of the result on ordering of goals, the optimization process should be localized to the goal given as argument to the meta-level predicates such as *min.* He also suggests that the constraints inherited from other goals should not change the optimality condition. Under such conditions, Fages describes how to provide semantics for such optimization predicates using negation, and how a branch and bound algorithm can be derived as a refinement of the semantics. The important difference between [GGZ91, Fag93] and our approach is that optimization in our framework is captured by explicit selection, without making use of negation. The following is a formulation of the shortest-path problem in the framework of [GGZ91]:

$$\begin{aligned} sh\_path(X,Y,C) &\leftarrow min(C,(X,Y),path(X,Y,C)). \\ path(X,Y,C) &\leftarrow arc(X,Y,C). \\ path(X,Y,C) &\leftarrow path(X,Y,C1),\ arc(Z,Y,C2),\ C = C1 + C2. \end{aligned}$$

The semantics of the min predicate is captured by the first-order extension of the program which replaces the min clause with the following rule:

$$sh\_path(X,Y,C) \leftarrow path(X,Y,C),\quad \neg(path(X,Y,C1) \wedge C1 < C).$$

Clearly, the paradigm of preference logic programming is powerful enough to express problems expressible in the framework of [GGZ91], as illustrated by the shortest path example.

4.2 Partial Order Programming

Partial order assertions were introduced in [Par89] as a programming paradigm that generalizes mathematical and declarative programming (logic and functional programming). In the formulation discussed in [JOM93], partial order assertions have two basic forms:

$$\begin{aligned} f(\bar{t}) &\geq expr \\ f(\bar{t}) &\leq expr \end{aligned}$$

where each variable in $expr$ also occurs in $\bar{t}$. The declarative meaning of a partial order assertion is that for all its ground instances (i.e. replacing variables by ground terms), the function f applied to the argument $\bar{t}$ is $\geq$ (resp. $\leq$) the ground term denoted by the expression on the right-hand side. There could be many partial order assertions defining a single function. Any function is defined either using $\geq$ assertions only or $\leq$ assertions only. The meaning of a ground expression $f(\bar{t})$ is equal to the *least upper bound* (resp. *greatest lower bound*) of the resulting terms defined by the different partial order assertions for f. This semantics expresses a preference for the *smallest* element in the partial order which is $\geq$ all the applicable right hand sides for $f(\bar{t})$. The following is a simple program that computes the set of all permutations of a set, i.e., the set of all list arrangements of the set (see [Jay92] for more details on this example):

$$\begin{aligned} perms(\{\ \}) &\geq \{[\,]\}. \\ perms(\{X \backslash T\}) &\geq distr(X, perms(T)). \\ distr(H, \{L \backslash \_\}) &\geq \{[H|L]\}. \end{aligned}$$

The pattern $\{X\backslash T\}$ matches a set S such that $X \in S$ and $T = S - \{X\}$). For example, matching $\{X\backslash T\}$ with $\{1,2,3\}$ yields three matches: $\{X \leftarrow 1, T \leftarrow \{2,3\}\}$; $\{X \leftarrow 2, T \leftarrow \{1,3\}\}$; and $\{X \leftarrow 3, T \leftarrow \{1,2\}\}$. In general, the right-hand sides of clauses is instantiated with all such matches, and the *least-upper bound* of the results is computed. For example, $distr(1, \{[2,3],[3,2]\})$ yields the set $\{[1,2,3],[1,3,2]\}$.

These partial order assertions make up the *core* part of the first-order theory of the preference logic program. In addition, we have the following optimization clauses, which express our desire to equate $perms(X)$ and $distr(A,B)$ respectively to the sets S and T such that $perms(X) \geq S$ and $distr(A,B) \geq T$:

$$\begin{aligned} perms(X) = S &\rightarrow perms(X) \geq S. \\ distr(A,B) = T &\rightarrow distr(A,B) \geq T. \end{aligned}$$

Since different worlds might over-estimate these sets in different ways, the following arbiter clauses are needed to specify that we prefer solutions for $perms(X)$ and $distr(A,B)$ that do not contain any extraneous elements:

$$\begin{aligned} perms(X) = S &\rightarrow \mathcal{P}_f(perms(X) = S1 \ \wedge \ S \geq S1). \\ distr(A,B) = T &\rightarrow \mathcal{P}_f(distr(A,B) = T1 \ \wedge \ T \geq T1). \end{aligned}$$

4.3 Syntactic Ambiguity Resolution

The problem of resolving ambiguity in context free grammars is an interesting applications of preference logic programming. Suppose we had the following BNF grammar for a fragment of a programming language:

```
<stmtseq> ::= <stmt> | <stmt> ; <stmtseq>
<stmt>    ::= <assign> | <ifstmt>
<assign>  ::= <var> := <expr>
<ifstmt>  ::= if <cond> then <stmtseq> | if <cond> then <stmtseq> else <stmtseq>
```

This grammar exhibits the famous "dangling-else" ambiguity. Given a string of the form:

if cond1 then if cond2 then assign1 else assign2

there are two possible parses (we use parentheses to indicate the different parses):

if cond1 then (if cond2 then assign1 else assign2)
if cond1 then (if cond2 then assign1) else assign2

A natural way to resolve the ambiguity is to express our preference for one parse over the other. For example, suppose that we wanted to match each else with the closest previous unmatches then. The resulting solution, shown below, is far more succinct than rewriting the grammar to avoid ambiguity. Assuming definitions of the nonterminals *expr* and *cond* and *var* exist, the definite-clause parser for the ambiguous grammar is shown below. In a clause such as $stmt(In, Out, stmt(Stmt)) \leftarrow assign(In, Out, Stmt)$, In stands for the input list of tokens to be parsed and Out stands for the list of tokens remaining after an initial prefix of tokens from In have been consumed according to the rule <stmt> ::= <assign>.

$$\begin{aligned} stmtseq(In, Out, stmtseq(Stmt)) &\leftarrow stmt(In, Out, Stmt). \\ stmtseq(In, Out, stmtseq(Stmt, Stmtseq)) &\leftarrow stmt(In, [;|Out1], Stmt),\ stmtseq(Out1, Out, Stmtseq). \end{aligned}$$

$$
\begin{array}{rcl}
stmt(In,Out,stmt(Stmt)) & \leftarrow & assign(In,Out,Stmt). \\
stmt(In,Out,stmt(Stmt)) & \leftarrow & ifstmt(In,Out,Stmt). \\
assign(In,Out,assign(Var,Exp)) & \leftarrow & var(In,[:=|Out1],Var),\ expr(Out1,Out,Exp). \\
ifstmt([if|In],Out,if(Cond,Then)) & \leftarrow & cond(In,[then|Out1],Cond),\ stmtseq(Out1,Out,Then). \\
ifstmt([if|In],Out,if(Cond,Then,Else)) & \leftarrow & cond(In,[then|Out1],Cond), \\
 & & stmtseq(Out1,[else|Out2],Then),\ stmtseq(Out2,Out,Else).
\end{array}
$$

The ambiguity can be resolved by introducing an O-predicate, $unamb\_ifstmt$, and an arbiter, as follows:

$$
\begin{array}{l}
unamb\_ifstmt(In,Out,T) \rightarrow ifstmt(In,Out,T) \\
unamb\_ifstmt(In,Out,if(Cond,if(Cond1,Then),Else)) \rightarrow \\
\qquad \mathcal{P}_f(unamb\_ifstmt(In,Out,if(Cond,if(Cond1,Then,Else))))
\end{array}
$$

Lastly, we replace all occurrences of $ifstmt$ in the body of any clause in the parser with $unamb\_ifstmt$.

In the general case, if we have a sentential form in the grammar which has k parses, we would have $k-1$ arbiter clauses which specify which parse is the intended parse.

We should note that in some applications grammatical ambiguity is resolved by associating costs with different parses and preferring parses with lesser cost. Such problems may be termed *optimal parsing* problems, and they find application in a variety of areas, including document formatting [BMW92], *etc.* Indeed, this was one of the principal motivating problems for the theory developed in this paper. We illustrate this point in section 6.

5 Operational Semantics for Preference Logic Programs

We now present a top-down derivation scheme for computing the optimal answers, and prove the soundness of this derivation scheme. We do not incur the expense of general theorem proving in modal logic because we are only interested in computing preferential consequences. The derivation scheme is an extension of SLD-resolution where some of the derivation paths get pruned due to the arbiter: the arbiter can thus be thought of as offering control advice to the SLD engine about which paths are better. Unlike the negation based approach of [GGZ91], our derivation scheme performs optimization by explicit selection, *i.e.* it prunes paths that compute sub-optimal solutions.

Below we present an extension of SLD-resolution assuming that the core program consists of definite clauses without constraints; we subsequently describe how this scheme can be extended to the case where the core program is a constraint logic program.

Definition 13 *Let P be a definite clause program and G a goal. A* **partial SLD-tree** *for $P \cup \{G\}$, is a finite SLD-tree, not all of whose branches are successful or failed derivations of $P \cup \{G\}$.*

Because every edge in the SLD-tree is labeled by a substitution, we associate with each node in the SLD-tree a substitution which is the composition of the substitutions found on the path from the node to the root of the tree.

Definition 14 *Given two partial SLD-trees T_1 and T_2 for $P \cup \{G\}$, we define $T_1 \Rightarrow T_2$ to mean that T_2 is* **derived** *from T_1 by choosing a non-empty leaf $l = A_1, \ldots, A_m, \ldots, A_k$ of T_1, A_m being the selected goal, and creating children of l of the form:*

$$\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$$

for every clause $A \leftarrow B_1, \ldots, B_q$ in P such that θ is the most general unifier of A_m and A. The leaf l is said to be **expanded** *in T_1 to get T_2.*

Definition 15 *Let P be a definite clause program and G be a goal. A* **Tree** *SLD-derivation (***TSLD** *derivation) of $P \cup \{G\}$ is a finite or infinite sequence $T_0 = G, T_1, \ldots$ of partial SLD-trees for $P \cup \{G\}$ such that for all i, $T_i \Rightarrow T_{i+1}$.*

Definition 16 *Given a Preference Logic Program $PLP = \langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$, and a query G, a* **TSLD derivation for** *$PLP \cup \{G\}$ is an TSLD-derivation for $\mathcal{T}_C \wedge \mathcal{T}_O$. The $\rightarrow$ clauses are treated exactly as the $\leftarrow$ clauses.*

Definition 17 *Given a partial SLD-tree T for $P \cup \{G\}$, a node $n_1 = \leftarrow A_1, \ldots, A_j$ in T is said to be* **blocked** *if there exists another node $n_2 = \leftarrow B_1, \ldots, B_k$ and an internal node $n = \leftarrow D_1, \ldots, D_m, \ldots, D_n$, such that n_1 and n_2 are descendants of n, where D_m is $p(\bar{t})$ where p is an Opt-predicate, and is subject to an arbiter of the form:*

$$p(\bar{a}) \rightarrow \mathcal{P}_f(p(\overline{a_1}) \textstyle\bigwedge_i L_i)$$

In addition θ_1 and θ_2 are the substitutions associated with nodes n_1 and n_2 such that the following constraint is satisfiable:

$$\{p(\bar{t})\theta_1 = p(\bar{a}) \quad , \quad p(\bar{t})\theta_2 = p(\overline{a_1})\} \quad \cup \quad \textstyle\bigcup_i \{L_i\}$$

The substitution θ_2 is said to be **better** *than θ_1.*

A path in a partial SLD-tree that passes through a blocked node is said to be a pruned path. Given a preference logic program $PLP = \langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$, a Pruned TSLD-derivation is a TSLD-derivation in which, at each step, the leaf to be expanded is not a descendant of a blocked node. A tree occuring in a PTSLD derivation is said to be complete if all its paths are either successful, failed or pruned. A PTSLD-derivation $T_0, \ldots, T_s$ is *complete* if it ends in a complete tree. T_s is said to be the *result* of the complete PTSLD-derivation.

Definition 18 *Given a PLP and a goal G, θ is said to be a* **correct optimal answer** *to G with respect to PLP, if $PLP \mid\!\sim G\theta$. Given a PLP and a complete PTSLD-derivation for $PLP \cup \{G\}$ with result T_s, let $\Theta = \{\theta_i | \theta_i$ is the composition of the substitutions along a successful path in T_s restricted to the variables in $G\}$. Θ is said to be the set of* **computed optimal answers** *to the query G with respect to the program PLP.*

Theorem 2 (Soundness of PTSLD-derivation) *If θ belongs to the set of computed optimal answers to a query G with respect to a preference logic program PLP, then θ is a correct optimal answer to the query G with respect to PLP.*

Proof Sketch: The proof is based on the soundness of SLD resolution. In addition we need to show that a computed answer is optimal according to the arbiter. Suppose θ is a computed optimal answer then we can show that there is no answer θ_1 that is better than it. Suppose such an answer θ_1 existed, it would have caused the path leading to the answer θ to get pruned.

Incompleteness of PTSLD-derivation: The derivation scheme described above is sound but incomplete. Consider the following *PLP* program:

$$\begin{aligned} q \leftarrow & \quad p(X). \\ p(b) \leftarrow & \quad p(a). \\ p(a) \leftarrow & \quad p(b). \end{aligned}$$

$$p(a) \rightarrow \quad p(a).$$

$$p(a) \rightarrow \quad \mathcal{P}_f(p(b)).$$

In this program the $\mathcal{T}_C$ is empty. There are two optimal worlds in the intended preference model namely $W_1 = \phi$ and $W_2 = \{q, p(a), p(b)\}$. Given a goal $\leftarrow q$, the derivation is non-terminating and according to the semantics, q is true in some optimal world. Therefore the derivation scheme is incapable of computing all the preferential consequences of the program. We are investigating whether the derivation scheme is complete if we place syntactic restrictions on PLP programs such as:

1. A predicate appearing at the head of one optimization clause ($\rightarrow$) should not appear at the head of any other core or optimization clause.

2. $\mathcal{T}_C$ and $\mathcal{T}_O$ should be locally stratified disallowing looping programs.

3. Imposing restrictions on the arbiters so that the preference relation it imposes on the worlds is anti-symmetric, transitive *etc.*

4. Disallowing arbiters of the form $A \rightarrow \mathcal{P}_f A$.

We now describe the extension of the derivation scheme to the case when the core program is a constraint logic program. The key difference between definite clause programs and constraint logic programs is that unification is replaced by a more general mechanism, namely constraint solving. Each node in the SLD-tree for CLP programs is characterized by a pair, namely a set of goals and a set of constraints. We briefly describe below how the derivation would proceed in the CLP domain by defining how successive trees are derived and how nodes can get blocked.

Definition 19 *Given a CLP program P, a goal G and two partial SLD-trees T_1 and T_2 for $P \cup \{G\}$, we define $T_1 \Rightarrow T_2$ to mean that T_2 is* **derived** *from T_1 by choosing a non-empty leaf $l = \langle\{A_1, \ldots, A_m, \ldots, A_k\}, \{C_j\}\rangle$ of T_1, A_m being the selected goal, and creating children of l of the form:*

$$\langle\{A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k\}, \{C_j\} \cup \{C\} \cup \{C_i'\}\rangle$$

for every clause $A \leftarrow C'_1, \ldots, C'_k, B_1, \ldots, B_q$ *in* P *such that* $\{C\}$ *is the set of constraints generated by the equation* $A_m = A$*, and the* C'_i*s in the body of the clause are constraints subject to the condition that* $\{C_j\} \cup \{C\} \cup \{C'_i\}$ *is solvable. The leaf* l *is said to be* **expanded** *in* T_1 *to get* T_2*.*

Definition 20 *Given a partial SLD-tree* T *for* $P \cup \{G\}$*, a node* $n_1 = \langle\{A_1, \ldots, A_j\}, \{C_{n_1}\}\rangle$*, a node* $n_2 = \langle\{B_1, \ldots, B_k\}, \{C_{n_2}\}\rangle$ *and an internal node* $n = \langle\{D_1, \ldots, D_m, \ldots, D_{n'}\}, \{C_n\}\rangle$*, such that* n_1 *and* n_2 *are descendants of* n*, where* D_m *is* $p(\overline{t})$ *where* p *is an Opt-predicate, and is subject to an arbiter of the form:*

$$p(\overline{a}) \rightarrow \mathcal{P}_f(p(\overline{a_1}) \textstyle\bigwedge_i L_i)$$

In addition suppose the constraint

$$\{p(\overline{t}) = p(\overline{a})\} \cup \{p(\overline{t}) = p(\overline{a_1})\} \cup \textstyle\bigcup_i \{L_i\} \cup \{C_{n_1}\} \cup \{C_{n_2}\}$$

is satisfiable by the substitution η *such that the projection* (γ) *of* η *to the variables in* $\{C_{n_1}\}$ *satisfies the constraint* $\{p(\overline{t}) = p(\overline{a})\}$ *and the projection to the variables in* $\{C_{n_2}\}$ *satisfies the constraint* $\{p(\overline{t}) = p(\overline{a_1})\}$*. We then update the constraint* $\{C_{n_1}\}$ *of node* n_1 *to* $\{C_{n_1}\} \cup \{\neg\gamma\}$*, where* $\{\neg\gamma\}$ *is a constraint that states that* γ *is not a solution. The solution* γ *is said to have been* **blocked***, a node in the tree is said to be blocked if all the solutions to the constraints of the node get blocked.*

In contrast with the previous definition for simple definite programs, each node in a SLD-tree in the CLP framework has a constraint associated with it which may be satisfiable in more than one way. Therefore each node in the SLD-tree in the CLP framework abstracts a set of solutions. The addition of a constraint $\{\neg\gamma\}$ blocks the solution γ. Note further that the nodes n_1 and n_2 in the definition need not be different nodes, *i.e.* one solution to the set of constraints may block another solution to the set of constraints.

6 A Practical Application of Preference Logic Programming

This section describes how preference logic programs are powerful enough to express practical applications such as document formatting [BMW92]. In particular we describe how line-breaking as done by formatters such as TEXand LATEX [KP81], can be expressed as a preference logic program.

Logically, a paragraph is a sequence of lines where each line is a sequence of words. This view of a paragraph can be captured by the following context free grammar with regular right hand sides (allowing regular right hand sides does not add any expressive power to context free grammars):

<para> –> <line> +

<line> –> [unit]+

[KP81] describes how to lay out a sequence of words forming a paragraph by computing the badness of the paragraph which depends on the badness of the lines that make up the paragraph. The badness of a line is determined by the properties of the line such as the total width of the characters that make up the line, the number of white spaces in the line, the stretchability and shrinkability of the white spaces, the desired length of the line etc. [KP81] associates extra strechability with the last line in the paragraph and insists that each line in the paragraph be such that the ratio of the difference between actual length and the desired length and the stretchability or shrinkability (the adjustment ratio) be bounded.

These can be captured in the grammatical framework by associating attributes with the non-terminals <para> and <line> in the grammar and providing equations that define the attributes of each non-terminal. One of the attributes of <para> is a badness attribute which we want to minimize. Augmenting the grammar above with the attribute equations gives us the following attribute grammar with a MIN: (minimization) directive over the badness attribute of <para>:

```
<para> -> <line> +
        <para>.badness := SUM_i <line>_i.badness
        MIN: <para>.badness

<line> -> [unit]+
        <line>.adjustment := if <line>.difference > 0 then
                             <line>.difference / <line>.stretchability
                             else
                             <line>.difference / <line>.shrinkability
        <line>.badness := | <line>.adjustment|*3
```

The attributes such as stretchability, shrinkability *etc.* can be defined similarly. The grammar in the above example is extremely ambiguous. For instance given a sequence of n words, the number of parses from <para> is the number of binary trees with n leaves. The number of binary trees with n leaves is the Catalan number $\frac{1}{n+1}\binom{2n}{n}$, which is $O(4^n/(n+1))$ (using Stirling's approximation for $n!$). The naive method of constructing all possible parses and selecting the optimal one is too expensive. We want a mechanism using which we can prune parses which we know are not optimal. This is where preference logic is useful. If we assume that the functions that define badness are monotonic and separable, we can compute the optimal parse for the above grammar constructing only a polynomial number of partial parses. The translation from an attributed context free grammar to a definite clause program is a straight forward generalization of the translation from simple context free grammars to definite clause programs [War80]. Suppose $para(In, Out, Parse, Badness)$ is the predicate that is true if $Parse$ is the parse of the input tokens in the $In - Out$ with badness $Badness$, we add the following O-clause:

$$bestpara(In, Out, Parse, Badness) \rightarrow para(In, Out, Parse, Badness).$$

and the following arbiter:

$$bestpara(In, Out, Parse1, Badness1) \rightarrow \mathcal{P}_f(bestpara(In, Out, Parse2, Badness2) \wedge Badness2 < Badness1).$$

7 Conclusions and Future Research

We have provided a logical account of optimization by casting it as inference in a modal logic of preference. The use of modal logic for this purpose is not surprising because optimization is a non-monotonic notion and modal logics have been used in the past to characterize non-monotonic inference [MST93]. We have also given a declarative account of optimization in terms of preferential consequence, *i.e.*, truth in the optimal worlds. The resulting programming paradigm extends the CLP schemes of [JL87] by providing two new types of program clauses: optimization and arbiter. We showed the usefulness of the paradigm by expressing a variety of problems: shortest path, ambiguity resolution, partial order programming, and line breaking. In keeping with the spirit of Kowalski's famous equation "Algorithm = Logic + Control" [Kow79], preference logic programming allows both logic and control to be specified in a modular and declarative fashion: The arbiter clauses in the preference logic program provide advice to the SLD inference engine about which paths in the proof tree are better than others. Finally, we provided a sound derivation scheme which for preference logic programs. We are in the process of implementing the PLP framework described in this paper. Other areas of further research are:

1. Since the derivation scheme we provided is incomplete, we are studying alternate derivation schemes and reasonable syntactic restrictions on preference logic programs so that we have a complete derivation scheme.

2. The operational semantics constructs the or-parallel tree for definite clause programs, therefore the implementation strategies of interest are or-parallel implementations. Since we may need to prune branches before they succeed, we need to augment the traditional or-parallel implementation strategies with constructs for communication.

3. We are presently studying the use of the properties of the domain on the derivation strategy to improve efficiency, *e.g.* in the line breaking problem we need to construct only a polynomial and not an exponential number of parses.

4. Since relaxation is the dual of optimization, preference logic programs can be used to logically characterize relaxation [BMW93]. We are presently investigating this topic further.

5. Motivated by the line-breaking example, we are also interested in incrementally computing the optimal answers; extending [MS89, vH90] to preference logic programs.

Acknowledgements

This research was supported by a grant from the XEROX Research Foundation. Surya Mantha would like to acknowledge the encouragement of Prof. Howard Blair and Prof. Anil Nerode during the development of the theory of preference logic.

References

[BMMW89] A. Borning, M. J. Maher, A. Martindale, and M. Wilson. Constraint hierarchies and logic programming. In Giorgio Levi and Maurizio Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 149–164, Lisbon, 1989. The MIT Press.

[BMW92] A. Brown, S. Mantha, and T. Wakayama. The Declarative Semantics of Document Processing. In *Proc. of the First International Workshop on Principles of Document Processing*, 1992.

[BMW93] A. Brown, S. Mantha, and T. Wakayama. Logical Reconstruction of Constraint Relaxation Hierarchies in Logic Programming. In *Proc. Intl. Symp. on Methodologies for Intelligent Systems*, 1993.

[BMW94] A. Brown, S. Mantha, and T. Wakayama. Preference Logics : Towards a Unified Approach to Non-Monotonicity in Deductive Reasoning. *To Appear in the Annals of Mathematics and Artificial Intelligence*, 1994.

[Che80] B.F. Chellas. *Modal Logic, an introduction.* Cambridge University Press, 1980.

[Fag93] F. Fages. On the Semantics of Optimization Predicates in CLP Languages. In *Proc. of 13th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 1993.

[GGZ91] S. Ganguly, S. Greco, and C. Zaniolo. Minimum and Maximum Predicates in Logic Programming. In *Proc. Tenth ACM Symp. on Principles of Database Systems*, pages 154–163, 1991.

[Jay92] B. Jayaraman. Implementation of Subset-Equational Languages. *Journal of Logic Programming*, 12(4):299–324, 1992.

[JL87] J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proc. 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, January 1987.

[JM] J. Jaffar and M. J. Maher. Constraint Logic Programming : A Survey. *To appear in the Journal of Logic Programming.*

[JOM93] B. Jayaraman, M. Osorio, and K. Moon. Partial Order Logic Programming. Technical Report 93-040, Dept. of Computer Science, SUNY at Buffalo, 1993.

[Kow79] R. A. Kowalski. Algorithm = Logic + Control. *CACM*, 22:424–436, 1979.

[KP81] D. E. Knuth and M. F. Plass. Breaking Paragraphs into Lines. *Software Practice and Experience*, 11:1119–1184, 1981.

[Man91] S. Mantha. *First-Order Preference Theories and their Applications.* PhD thesis, University of Utah, November 1991.

[MS89] M. J. Maher and P. J. Stuckey. Expanding Query Power in Constraint Logic Programming Languages. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 20–36, Cleveland, Ohio, USA, 1989.

[MST93] V. W. Marek, G. F. Schwarz, and M. Truszczynski. Modal Nonmonotonic Logics : Ranges, Characterization, Computation. *JACM*, 40(4):963–990, September 1993.

[MT91] W. Marek and M. Truszczynski. Autoepistemic Logic. *JACM*, 38(3):588–619, July 1991.

[Par89] S. Parker. Partial Order Programming. In *Proc. ACM Conf. on Principles of Prog. Lang.*, pages 260–266, 1989.

[Ram88] A. Ramsay. *Formal Methods in Artificial Intelligence.* Cambridge University Press, 1988.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[vH90] P. van Hentenryck. Incremental constraint satisfaction in logic programming. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 189–202, Jerusalem, 1990. The MIT Press.

[War80] D. H. D. Warren. Logic Programming and Compiler Writing. *Software Practice and Experience*, 10(II), 1980.

[WB93] M. Wilson and A. Borning. Hierarchical Constraint Logic Programming. *Journal of Logic Programming*, 16:277–318, 1993.