Optimization and Relaxation in Constraint Logic Languages

Kannan Govindarajan
Dept. of Computer Science
SUNY at Buffalo
Buffalo, NY 14260
govin-k@cs.buffalo.edu

Bharat Jayaraman
Dept. of Computer Science
SUNY at Buffalo
Buffalo, NY 14260
bharat@cs.buffalo.edu

Surya Mantha
Corporate Research & Technology
Xerox Corporation
Webster, NY 14580
mantha@wrc.xerox.com

Abstract

Optimization and relaxation are two important operations that naturally arise in many applications involving constraints, e.g., engineering design, scheduling, decision support, etc. In optimization, we are interested in finding the optimal (i.e., best) solutions to a set of constraints with respect to an objective function. In many applications, optimal solutions may be difficult or impossible to obtain, and hence we are interested in finding suboptimal solutions, by either relaxing the constraints or relaxing the objective function. The contribution of this paper lies in providing a logical framework for performing optimization and relaxation in a constraint logic programming language. Our proposed framework is called *preference logic programming* (PLP), and its use for optimization was discussed in [8]. Essentially, in PLP we can designate certain predicates as optimization predicates, and we can specify the objective function by stating preference criteria for determining the optimal solutions to these predicates. This paper extends the PLP paradigm with facilities to formulate relaxation problems in a natural manner. We introduce the concept of a relaxation goal, and discuss its use for preference relaxation. Our model-theoretic semantics of relaxation is based on simple concepts from modal logic: Essentially, each world in the possible-worlds semantics for a preference logic program is a model for the constraints of the program, and an ordering over these worlds is determined by the objective function. Optimization can then be expressed as truth in strongly optimal worlds, while relaxation becomes truth in suitably-defined suboptimal worlds. We also present an operational semantics for relaxation as well as correctness results. Our conclusion is that the concept of preference provides a unifying framework for formulating optimization as well as relaxation problems.

To appear in the Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages. January 1996. Also available as Department of Computer Science SUNY at Buffalo, Technical Report TR 95-37.

1 Motivation and Approach

Constraint optimization and relaxation are practical techniques for solving problems arising in various applications, such as engineering design, document layout, interactive graphics, scheduling, and decision support. In these settings one is interested in finding the *optimal* solutions to constraints with respect to some objective function; and, if the optimal solutions are impossible to obtain, one is interested in finding *suboptimal* solutions either by relaxing the objective function or by relaxing the constraints themselves. While optimization and relaxation are important in practice, they are meta-level operations—as they require comparing alternative solutions and choosing the best one—and therefore they fall outside the standard constraint logic programming (CLP) framework [10, 11]. While several approaches (discussed in section 2) have been proposed in the literature to address these problems, they do not provide a *unified*, logical treatment of both optimization and relaxation, nor do they provide the flexibility of our proposed approach. The contribution of this paper is two-fold:

1. We present a principled extension of the CLP framework for declaratively specifying optimization and relaxation problems. We provide logical constructs to specify in a modular way which predicate is to be optimized and the criterion for optimization, as well as which (optimization) predicate is to be relaxed and the criterion for relaxation. These criteria are understood using the concept of *preference*, hence the resulting programming paradigm is called *preference logic programming* (PLP).

2. We formalize the declarative semantics of optimization and relaxation using concepts from modal logic. We give a *possible-worlds semantics* for a PLP program in which each world stands for a certain subset of feasible solutions. The ordering among worlds—determined from the preferences—explicitly conveys the ordering among solutions. Optimization is then expressed as truth in *strongly optimal* worlds, while relaxation becomes truth in suitably-defined suboptimal worlds.

We briefly illustrate our proposed programming constructs. The following CLP clauses for the predicate `path(X,Y,C,P)` computes `P` as a path (list of edges) with cost `C` from node `X` to `Y` in a directed graph:

```
path(X,Y,C,[e(X,Y)]) ← edge(X,Y,C).
path(X,Y,C1+C2,[e(X,Z)|L1]) ←
                    edge(X,Z,C1), path(Z,Y,C2,L1).
```

To illustrate the constructs for optimization in PLP, a logical specification of the shortest-path problem can be given as follows (we present another solution to this problem in section 2):

```
sh_path(X,Y,C,P) → path(X,Y,C,P).
sh_path(X,Y,C1,P1) ⪯ sh_path(X,Y,C2,P2) ←
                                    C2 < C1.
```

The first clause is called an optimization clause, and sh_path is called an optimization predicate. Its space of feasible solutions is some subset of the solutions for path (hence the use of a → clause). The second clause is called an arbiter clause and it states the criterion for optimization: given two solutions for sh_path, the one with lesser cost is preferred. The symbol $\preceq$ is to be read as 'is less preferred than'.

To illustrate the constructs for relaxation in PLP, suppose that we want to use the above definition of sh_path to compute the shortest path between a and b that does not go through c. Note that the query ?- sh_path(a,b,C,P), notinpath(c,P) will not fully accomplish our objective: If all the shortest paths between a and b pass through c, the above query fails and we obtain no answer. Hence we introduce a *relaxation goal* to solve our stated problem, as follows:

```
?- RELAX sh_path(a,b,C,P) WRT notinpath(c,P)
```

The goal that follows the keyword RELAX must be some optimization goal, and the criterion for relaxing this goal is stated after the WRT keyword (read as 'with respect to'). The intended meaning of the relaxation goal is as follows: If the best solutions to sh_path(a,b,C,P) satisfy notinpath(c,P), then those are the solutions to the relaxation goal. Otherwise, the intended solutions are got by restricting the feasible solution space of sh_path(a,b,C,P) by treating the predicate notinpath(c,P) as an additional constraint that sh_path(a,b,C,P) has to satisfy, and then finding the optimal solutions in this restricted space. This example illustrates what we call *preference relaxation.* We also show that the PLP paradigm subsumes the notion of *constraint relaxation* as defined in HCLP [24].

The *model-theoretic semantics* for PLP is given in terms of *possible worlds* where each world is a model for the definite clauses of the program, and the ordering among the worlds is determined by the preferences. We are interested in the *preferential consequences* of a program, or truth in *strongly optimal worlds.* This is in constrast with *logical consequence* which refers to truth in all worlds. To provide semantics for relaxation goals, we consider only the worlds satisfying the relaxation criterion to determine the best solution; thus we effectively relax our preferences for those worlds that do not satisfy this criterion. We introduce the notion of *relaxed preferential consequence* to refer to truth in the corresponding strongly optimal worlds. We also provide *operational semantics* for optimization and relaxation, and show that appropriate correctness results are obtained. The operational semantics of optimization is given in terms of *Pruned-Tree SLD (PTSLD)* derivations. To compute the relaxed preferential consequences of a program, we first perform a program transformation on the definitions of the optimization predicates that must be relaxed, and then use a variation of the operational semantics for PLP to compute the relaxed preferential consequences.

The remainder of this paper is organized as follows: Section 2 compares our work with related approaches to optimization and relaxation. Section 3 presents the syntax of programs and several paradigmatic examples to show the power and flexibility of the paradigm. Section 4 gives the declarative semantics of PLP in terms of the preferential consequences of a program, i.e., truth in strongly optimal worlds. Section 5 provides the operational semantics of PLP programs in terms of Pruned-Tree SLD (PTSLD) derivations, states the main correctness results, and outlines their proofs.

2 Related Work

As noted earlier, related approaches have dealt either with optimization or relaxation but have not provided a unified framework for both. Our approach also offers advantages over these approaches both from a programming as well as semantic viewpoint.

In the area of optimization, Maher and Stuckey [16] discuss how to incorporate optimization queries into a CLP system by mapping the solutions of a query to a partial order; Fages [3] describes a semantics for optimization predicates in CLP languages based on Kunen-Fitting's semantics for negation; and Marriott and Stuckey [18] discuss the semantics of constraint logic programs with optimization by translating the optimization predicates into predicates with negation. The key distinguishing feature of our approach is that it allows the programmer to program the preference criteria to suit the application at hand. In contrast, [18, 3] only provide optimization predicates that can be expressed as maximizing or minimizing some objective function. Fages *et al* [4] allow the programmer to program the ordering among solutions but the semantics of optimization is provided *via* negation. Compared with the negation-based approaches [18, 3, 4], the advantage of our approach is that it gives an explicit treatment of the ordering among solutions, which results in a simple and direct way of formalizing both optimization and relaxation.

Our notion of optimization is closely related to the notion of first-order aggregate operations (such as *min* and *max*) in the field of deductive databases, where there has been a considerable recent interest in providing the semantics for aggregation. A program with such aggregate operations has an equivalent first-order formulation using negation. Ganguly *et al* [5] considered fisrt-order aggregates and showed that under certain monotonicity conditions, the first-order equivalent program has a total well-founded model [23] that can be computed using a greedy fixed-point procedure. There has also been interest in more general aggregate operations, such *count*, *sum*, etc. Kemp and Stuckey [13] examined programs with recursion through aggregation. To give semantics for programs with aggregation, they extended two well-known semantics for programs with negation, namely, well-founded models and stable models [6]. Ross and Sagiv [20] provide semantics for aggregation where the domain over which the aggregation is performed is a complete lattice and the program is monotonic. By Tarski's theorem, we are guaranteed the existence of a least fixed-point for the aggregate operation. Sudarshan *et al* [21, 22] provide semantics for a class of aggregate operators that includes operations such as the k^{th}-best solution, using *valid models* for normal programs. Our formulation of preference relaxation enables the user to program such operations in a ver flexible manner.

In the area of relaxation, Mantha *et al* [1] introduced Relaxable Horn Clauses, where a relaxable clause is a definite clause with a partial order over the goals in the body; the partial order dictates the order in which the goals are to be relaxed if all the goals in the body are not satisfiable. However, stating the relaxation criteria in this way, i.e., in terms of goals local to a clause, provides only limited expressiveness for our intended applications. Wilson and Borning [24] propose Hierarchical CLP (HCLP), a paradigm in which a constraint may be optionally tagged with a weight (such as `required`, `strong`, `weak`, etc.) that indicates the importance of a constraint relative to other constraints and serves to organize all constraints them into linear hierarchy. The notion of a *comparator* is introduced in order to compare and order alternative solutions to the required constraints by determining how well they satisfy the remaining (i.e., relaxable) constraints. Given a constraint hierarchy, the solutions of interest are those that satisfy the required constraints and are optimal according to the comparator. We show that the notion of optimization in PLP is powerful enough to capture the notion of relaxation in HCLP. PLP is more powerful than HCLP because the latter does not provide a general support for optimization or preference relaxation.

Our formulation of optimization and relaxation offers the benefits of having a simple declarative semantics, providing modularity, and being amenable to an efficient implementation. We illustrate the first two points in detail in this paper, but demonstration of the last point (efficient implementation) is beyond the scope of this paper. Finally, we note that this paper extends our earlier work [8] in that the latter only treats optimization (not relaxation). We gave an initial formulation of relaxation in [9] which did not permit relaxation goals in bodies of clauses; they were permitted only as top level goals. This paper extends our earlier work in providing a more comprehensive account of optimization and relaxation.

3 Programming Paradigms in PLP

3.1 The PLP Framework

A preference logic program may be thought of as containing two parts: a **first-order theory** and an **arbiter**. The first-order theory consists of clauses each of which can have one of two forms:

1. $H \leftarrow B_1, \ldots, B_n$, $(n \geq 0)$, i.e., *definite clauses.* Each B_i is of the form $p(\bar{t})$ where p is a predicate and $\bar{t}$ is a sequence of terms. In general, some of the B_is could be constraints as in CLP [10, 11].
2. $H \rightarrow C_1, \ldots, C_l \mid B_1, \ldots, B_m$, $(l, m \geq 0)$, i.e., *optimization clauses.* $C_1, \ldots, C_l$ are constraints as in CLP [10, 11] that must be satisfied for this clause to be applicable to a goal[1]; they are to be read as antecedents of the implication.

Moreover, the predicate symbols can be partitioned into three disjoint sets depending on the kinds of clauses used to define them:

1. *C-predicates* appear only in the heads of definite clauses and the bodies of these clauses contain only other *C-predicates* (*C* stands for *core*).
2. *O-predicates* appear in the heads of only optimization clauses (*O* stands for *optimization*). For each ground instance of an optimization clause, the instance of the *O-predicate* at the head is a candidate for the optimal solution provided the corresponding instance of the body of the clause is true. The constraints that appear before the | in the body of an optimzation clause are referred to as the *guard* and must be satisfied in order for the head H to be reduced.
3. *D-predicates* appear in the heads of only *definite* clauses and at least one goal in the body of at least one such clause is either an *O-predicate* or a *D-predicate.* (*D* stands for *derived from O-predicates.*)

The **arbiter** part of a preference logic program, which specifies the **optimization criterion** for the *O-predicates*, has clauses of form:

$$p(\bar{t}) \preceq p(\bar{u}) \leftarrow L_1, \ldots, L_n \qquad (n \geq 0)$$

where p is an *O-predicate* and each L_i is an atom whose head is a *C-predicate* or a constraint as in CLP. In essence this form of the arbiter states that $p(\bar{t})$ *is less preferred than* $p(\bar{u})$ if $L_1, \ldots, L_n$.

A **relaxation goal** has the form

`RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$,

where p is an *O-predicate* and c is a *C-predicate* or a constraint as in CLP. The predicate p is said to be a **relaxable predicate** and c is said to be the **relaxation criterion** of the relaxation goal. The semantics provided in this paper also captures the meaning of relaxation goals where c is any *O-predicate* that is not defined in terms of p. However, in this paper, we will only deal with relaxation goals where c is a *C-predicate.* A relaxation goal may appear in the body of an *O-predicate*, a *D-predicate* or in a toplevel query. A preference logic program is a triple of the form $\langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$ where $\mathcal{T}_C$ is made up of the definition of the *C-predicates*, $\mathcal{T}_O$ consists of the definitions of the *O-predicates* and *D-predicates*, and $\mathcal{A}$ consists of the arbiter clauses in the program.

3.2 Programming Paradigms

We now present representative examples of optimization and relaxation in PLP. We first present an example of *dynamic programming*, followed by an example of a *greedy heuristic.* We then consider a *preference grammar*, which illustrates the use of arbiter clauses for ambiguity resolution. We also show how the use of weighted constraints and comparators for constraint relaxation in HCLP can be expressed in PLP, and finally illustrate preference relaxation (or relaxation goals) with an example from scheduling and the n^{th}-shortest-path problem.

[1] Note that the variables that appear only on the RHS of the $\rightarrow$ clause are existentially quantified, not universally quantified. The intended meaning of this clause is that the set of solutions to the head is some subset of the set of solutions to the body.

Dynamic Programming: The program below is a dynamic-programming formulation of the shortest-path problem.

```
sh_dist(X,X,N,0).
sh_dist(X,Y,1,C) → X <> Y | edge(X,Y,C).
sh_dist(X,Y,N+1,C1+C2) → N > 1, X <> Y |
            sh_dist(X,Z,1,C1), sh_dist(Z,Y,N,C2).
sh_dist(X,Y,N,C1) ⪯ sh_dist(X,Y,N,C2) ←
                                          C2 < C1.
```

(We show only the computation of the shortest distance; the the associated path can be computed with the aid of an extra argument). This program explicitly expresses the *optimal sub-problem property* of a dynamic-programming algorithm: each call to `sh_dist` uses only the *optimal* solutions to subsequent recursive calls on `sh_dist`. Thus pruning of sub-optimal solutions occurs at each recursive call. In the previous formulation of this problem (in section 1), domain knowledge, such as the monotonicity of `+`, would be necesary to achieve a similar effect. This example also shows the need for the guard: the conditions `X <> Y` and `N > 1, X <> Y` should be read as antecedents of the implication.

Heuristics: Preferences can also be used to formulate *greedy* algorithms, or *heuristics*, which are useful in obtaining acceptable solutions to combinatorially hard problems such as the traveling salesman problem (TSP). Heuristics are used in these problems to make the search space polynomial in size, without completely compromising the quality of the solution. In many cases, it may be possible to specify such heuristics by comparing partial solutions and preferring one partial solution over the other, as illustrated below for the nearest neighbor heuristic for TSP [14].

```
tsp(Tour,Cost) ←
        get_vertices(V), tsp(V,[],Tour,0,Cost).

tsp([],T,T,C,C).
tsp([H|T],[],Tour,Cin,Cout) ←
                      tsp(T,[H],Tour,Cin,Cout).
tsp(List,[City|Tour],T,Cin,Cout) ←
            closest(City,List,Next,Rest,Cost),
     tsp(Rest,[Next,City|Tour],T,Cin+Cost,Cout).

closest(City,List,Next,Rest,Cost) →
   select(List,Next,Rest), edge(City,Next,Cost).

closest(A,_,_,_,C1) ⪯ closest(A,_,_,_,C2) ←
                                     C1 > C2.
```

The top-level predicate, `tsp`, computes the best tour and its cost, by making use of the optimization predicate, `closest`, which incorporates the nearest-neighbor heuristic. (The definitions of the predicates `get_vertices`, `select`, and `edge` are straightforward and hence omitted.)

Preference Grammars: The concept of preference also provides a general means for specifying the selection criteria for choosing among alternative solutions to a goal. A good use of this capability is ambiguity resolution—which may be viewed as a form of optimization—in programming language and natural language grammars. We illustrate with a well-known example:

ifstmt `::=` `if` *cond* `then` *stmtseq* |
`if` *cond* `then` *stmtseq* `else` *stmtseq*

This grammar exhibits the famous "dangling else" ambiguity (the nonterminals *cond* and *stmtseq* have their usual definitions). A natural way to resolve the ambiguity is to express our preference that each `else` pairs up with the closest previous unmatched `then`. The resulting solution, shown below using *definite clause grammars* (DCG) [19], is far more succinct than rewriting the grammar to avoid ambiguity:

```
ifstmt(if(C,T)) --> [if], cond(C), [then],
                                    stmtseq(T).
ifstmt(if(C,T,E)) --> [if], cond(C), [then],
                 stmtseq(T), [else], stmtseq(E).
ifstmt(if(C1,if(C2,T),E)) ⪯

                        ifstmt(if(C1,if(C2,T,E))).
```

The first two productions are the usual grammatical rules of a definite clause grammar. The argument terms `if(C,T)` and `if(C,T,E)` represent the constructed parse trees for the corresponding grammar rules. The third clause is a *grammatical arbiter clause*, and it specifies in a modular and declarative manner the criterion that each `else` pairs up with the closest previous unpaired `then`. We refer to this extension of definite clause grammars as *preference logic grammars* (PLGs) [7]. There is a straightforward translation from PLGs into PLPs; the scheme is analogous to that from DCGs into definite clause programs. Since the two DCG rules have a common prefix, namely, `[if], cond(C), [then], stmtseq(T)`, it should be possible to obtain an efficient parsing scheme (e.g., using memoization) for efficiently performing the disambiguation.

Constraint Relaxation: We show how constraint relaxation in HCLP [24] can be simulated in PLP. A constraint c is a relation over an appropriate domain, and a labeled constraint $l\ c$ is a constraint c with strength l, where the strengths of constraints are taken from a totally-ordered domain. A constraint hierarchy H is a finite collection of labeled constraints. The constraints in H can be partitioned according to their strengths. If H_i is the collections of constraints with strength i, we write $H = \langle H_0, H_1, \ldots, H_n \rangle$, where H_0 is the set of required constraints in the constraint hierarchy. A HCLP scheme is parametrized both by the domain of the constraints and the *comparator* to be used in determining the optimal answer. We can systematically translate a HCLP program into a PLP program. Essentially, the arbiter clauses of the PLP program enforce the same ordering among the solutions that the comparator in the HCLP program enforces. Therefore, in PLP the comparator can be programmed to suit the application. The translation scheme is similar to the translation from definite clause grammars to definite clause programs. For example, consider the following HCLP program adapted from [24]:

```
b(X) ← a(X), weak X > 6.
a(X) ← strong X = 1.
a(X) ← required X > 0, required X < 10,
        X < 4.
```

In accordance with the operational semantics of HCLP [24], given a top-level query q, the translated PLP program collects all relaxable constraints arising from q into a list, and processes them after all required constraints arising from q have been satisfied. For example, the above HCLP program is translated as follows:

```
b(X,[weak X > 6 | I],O) ← a(X,I,O).
a(X,[strong X = 1 | O],O).
a(X,[required X > 0, required X < 10,
      weak X < 4 | O],O).
```

Given a HCLP query $q(\bar{t})$, the translated query would be:

$q(\bar{t}, L, [\,]),\ hclp(L, ErrorSeq).$

The definition of the predicate *hclp* is independent of the HCLP program to be translated, and depends only on the parameters of the scheme, in particular, the comparator being used.

$$hclp(L, ErrorSeq) \rightarrow compute\text{-}error(L, ErrorSeq).$$
$$hclp(L, ErrorSeq_1) \preceq hclp(L, ErrorSeq_2) \leftarrow ErrorSeq_2 < ErrorSeq_1.$$

The *hclp* predicate computes an error-sequence for the relaxable constraints. The specific comparator used in the HCLP scheme is incorporated in the definition of the predicate *compute-error*. The i^{th} entry in the sequence is the aggregate error for the relaxable constraints at the i^{th} level of the hierarchy. The error measures how well the relaxable constraints satisfy a particular solution to the required constraints. The comparison of two error-sequences is done lexicographically.

Preference Relaxation: In many scheduling problems, we associate costs with schedules and we are interested in schedules with the least cost. For instance, suppose we wanted to schedule m jobs to n processors and we are interested in minimizing the time taken to finish all the jobs. Such problems can be expressed in the PLP framework as follows. Below, N0, N1, and N2 are nodes that contain schedules; P is a processor; S, S1, and S2 are schedules; and T is a task.

```
opt_schedule(S) → schedule(S).
opt_schedule(S2) ⪯ opt_schedule(S1) ←
              samejobs(S1,S2), lesscost(S1,S2).

schedule(S) ← initial(N), schedule(N,S).
schedule(N,N) ← alltasksdone(N).
schedule(N0,N2) ← step(N0,N1), schedule(N1,N2).
```

The toplevel predicate is opt_schedule(S). The step predicate takes a schedule in node N0 as input and produces a new schedule in node N1 after selecting and scheduling a particular task to a processor. The definitions of step, initial, alltasksdone, samejobs, and lesscost are omitted as their intended meanings should be clear in this example. In such a scenario, it is natural to subject the optimal schedule to additional requirements *without explicitly changing the definition of* opt_schedule, i.e., we seek a modular solution to the problem. For instance, if some processor (say p1) fails, we would then want the best schedule that does not involve this processor. This requirement can be expressed by a relaxation goal such as

```
RELAX opt_schedule(S) WRT free(p1,S),
```

where we define free(P,S) to be true if in schedule S, no job has been assigned to processor P.

Our second example illustrates use of a relaxation goal in the body of a clause to compute the path with the n^{th}-lowest cost between any two nodes in a graph:

```
n_sh_path(1,X,Y,C,P) ← sh_path(X,Y,C,P).
n_sh_path(N+1,X,Y,C,P) ← n_sh_path(N,X,Y,D,_),
                  RELAX sh_path(X,Y,C,P) WRT C > D.
```

Given a query, ?- n_sh_path(2,a,b,C,P), the computed answer for C will be the second-lowest cost between a and b. The only relaxation goal that is called is:

```
RELAX sh_path(1,a,b,C,P) WRT C > C0,
```

where C0 is already the cost of the shortest path between a and b. The above program gives a modular and declarative specification of the problem. It might appear at first that the above program incurs a lot of recomputation because of repeated calls to the sh_path predicate. However, by making use of memoization or an equivalent method, we can avoid this potential problem.

4 Declarative Semantics

We first review the model theory of simple preference logic programs (without relaxation goals in bodies of clauses) [8], and then extend it to preference logic programs with relaxation goals in the bodies of clauses.

4.1 Model Theory of Optimization

Preference logic programs are viewed as theories in the modal logic of preference [17], and hence the model theory for preference logic programs uses ideas from modal logic. We provide a possible world semantics for preference logic programs where each world is a model for the program and an ordering among the worlds is enforced by the arbiter clauses.

We the review of the model theory starting with a brief introduction to the modal logic of preference [17]. The syntax of this logic extends the syntax of first-order logic by adding a new modal operator $\mathcal{P}_f$ with the associated rule of formation: If F is a formula then so is $\mathcal{P}_f F$. We treat each *definite preference clause* $p(\bar{t}) \preceq p(\bar{u}) \leftarrow L_1, \ldots, L_n$ in a preference logic program as a formula

$$p(\bar{t}) \rightarrow \mathcal{P}_f(p(\bar{u})\ \wedge\ L_1\ \wedge\ \ldots\ \wedge\ L_n)$$

In the tradition of modal logic, [17] provides a *possible worlds* semantics for this logic. 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 $\mathcal{W}$. 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. The semantics of preference formulae of the form $\mathcal{P}_f F$ is given as follows:[2]

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

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. A preference model $\mathcal{M}$ is said to be **supported** if, for any two

[2]We write $\models^w_{\mathcal{M}} G$ to indicate that the formula G is assigned the truth value *true* at the world w in the preference model $\mathcal{M}$.

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 $\preceq$. 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 there is no world w' different from w such that $w \preceq w'$.

4.2 Models for Preference Logic Programs

We build models for preference logic programs in stages. We assign ordinal levels to the *O-predicates* in the program such that if an *O-predicate* O_2 appears in the body of a clause defining another *O-predicate* O_1, the ordinal assigned to O_2 is less than the ordinal assigned to O_1[3]. Suppose the number of levels of *O-predicates* in the program is n, and they are numbered $1, \ldots, n$. The model is constructed in n stages as follows:

1. The *O-predicates* at level 1 are defined only in terms of *C-predicates* (or other *O-predicates* at level 1 in a mutually recursive manner). Since the *C-predicates* are defined using only definite clauses, they have a unique minimal model. Each world in the preference model at level 1 extends the minimal model of the *C-predicates* with instances of *O-predicates* at level 1 so that it becomes a model for the clauses defining the *O-predicates* at level 1 in the program. The ordering among these worlds is enforced by the arbiter clauses over *O-predicates* at level 1. The **intended preference model** at level 1 contains all the possible worlds that can model the clauses defining the *O-predicates* at level 1. The set of **preferential consequences** of the program at level 1 is the set of atoms that are true in some strongly optimal world in the preference model at level 1.

2. The worlds in the preference model at level $k+1$ extend the set of preferential consequences of the preference model at level k with instances of *O-predicates* at level $k+1$ so that each world becomes a model for the clauses defining the *O-predicates* at level $k+1$. The **intended preference model** at level $k+1$ contains all those worlds that model the clauses defining the *O-predicates* at level $k+1$. The ordering among the worlds is enforced by the arbiter clauses for the *O-predicates* at level $k+1$.

The reader is referred to [8] for examples and a more detailed description of the model theory. Given a preference logic program P with n levels of *O-predicates*, the **intended preference model** of P is the intended preference model at level n. Given a preference logic program P with n levels of *O-predicates* whose Herbrand Base[4] is B_P and an atom $A \in B_P$, we say that A is a preferential consequence of the program (written $P \approx\!\!\!| A$) if A is a preferential consequence of the preference model at level n. The **declarative semantics**, D_P, is defined to be the set $\{A \in B_P \mid P \approx\!\!\!| A\}$. Given a preference logic program P and a goal G, a valuation θ is said to be a **correct optimal valuation** for P and G if $G\theta$ is a preferential consequence of P.

[3] We can construct the predicate call graph among the *O-predicates* and topologically sort it. All the predicates in a cycle get assigned the same ordinal.

[4] In the presence of constraints, this refers to the generalized Herbrand Base [12].

In the above approach, the worlds constructed at different levels are different. This enables us to obtain the optimal solutions at one level without regard to the orderings enforced by arbiter clauses at higher levels. If we maintained one set of worlds for all levels as in [2], the orderings enforced by arbiter clauses at one level might conflict with those at a lower level, thereby disallowing optimal solutions to *O-predicates* at the lower level. Furthermore, the stagewise model construction captures the notion of hierarchical optimization because only the optimal solutions at level k contribute to the solutions at level $k+1$.

4.3 Model Theory of Relaxation

We first develop the model theory for the relaxation queries to preference logic programs that do not have any relaxation goals in bodies of clauses. We then extend the model theory to programs that have relaxation goals in the bodies of clauses too.

4.3.1 Relaxation Queries

We begin with the following plausible definition for a correct valuation of a relaxation query.

Definition 1 *Given a preference logic program P and a relaxation query G such that the set of correct valuations to G is empty, θ is said to be a* **naive relaxed correct valuation** *if there is a world w in the intended preference model for P such that $w \models G\theta$ and $\neg\exists w' \exists\theta'((w' \models G\theta') \wedge (w \preceq w') \wedge (\theta \neq \theta'))$.*

Intuitively, θ is a *naive relaxed correct valuation* to a relaxable query G, if there is a world w where $G\theta$ occurs and there is no better world w' than w where there is an occurrence of $G\theta'$ for some substitution θ' different from θ. This definition of relaxation, though intuitively appealing, suffers from a pathological problem as illustrated by the following proposition.

Proposition 4.1 *Given a preference logic program P and a relaxation query $G =$ `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$ such that the set of correct optimal valuations to $p(\bar{t}) \wedge c(\bar{u})$ with respect to P is empty, and there are at least two solutions for $p(\bar{t})$ that satisfy $c(\bar{u})$. Then the set of naive relaxed correct valuations to G with respect to P is also empty.*

Proof Sketch: Suppose θ_1 is the optimal solution of $p(\bar{t})$; clearly, θ_1 does not satisfy $c(\bar{u})$. Suppose further that θ_2 and θ_3 are two non-optimal solutions for $p(\bar{t})$ that satisfy $c(\bar{u})$. Assume without loss of generality that θ_1 is the solution that pruned both of them. Since p is defined using an optimization ($\rightarrow$) clause, there are worlds in the intended preference model that contain the following instances of $p(\bar{t})$:

1. $w_1 \supseteq \{p(\bar{t})\theta_1\ ,\ p(\bar{t})\theta_2\}$.
2. $w_2 \supseteq \{p(\bar{t})\theta_1\ ,\ p(\bar{t})\theta_3\}$.

The solution θ_2 is not a relaxed correct valuation because in world w_2—which is better than any world with the θ_2 instance—the only solution to the relaxable query is θ_3. By similar reasoning, θ_3 is also not a relaxed correct valuation. Furthermore, notice that this argument would hold irrespective of the number of substitutions θ_i such that both $p(\bar{t})\theta_i$ and $c(\bar{u})\theta_i$ are satisfiable. ■

In order to obtain a satisfactory definition, we need to remove the worlds that have instances of the *O-predicate* that are not solutions of the *C-predicate*.

Definition 2 *A preference frame F_1 is said to be a* **sub-frame** *of a preference frame F_2 if the set of worlds in F_1 is a subset of the set of worlds in F_2 and the relation among the worlds among the worlds in F_1 is a restriction of the relation among the worlds in F_2 to the worlds in F_1.*

Definition 3 *Given a preference logic program P and a relaxable query $G =$ `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$, the* **relaxed preference model** *for P and G is a sub-frame M_r of the intended preference model M for P such that M_r contains all the worlds in M such that the only instances of $p(\bar{t})$ that appear in each world correspond to valuations that satisfy $c(\bar{u})$.*

Theorem 1 *Given a preference logic program P and a relaxation query G, the relaxed intended preference model exists and is unique.*

Proof Sketch: For any preference logic program P, its intended preference model is unique [8]. The relaxed intended preference model is constructed from the intended preference model. Since the *C-predicate* in the relaxation goal is interpreted uniformly across the worlds, the set of worlds in the relaxed intended preference model is well defined. ■

For example, consider the shortest path program from section 1 and the relaxation query `?- RELAX sh_path(a,b,C,P) WRT notinpath(c,P)`. The worlds in the relaxed intended model will contain only those instances of `sh_path(a,b,C,P)` such that `notinpath(c,P)` is also true. The instances of `sh_path(a,b,C,P)` that are true in strongly optimal worlds in the relaxed preference model will correspond to shortest paths between `a` and `b` that do not pass through `c`.

4.3.2 Programs with Relaxation Goals

We now extend the model theory presented above to programs where relaxation goals occur in the bodies of clauses. The difficulty in providing a model theory for preference logic programs with relaxation goals in the bodies is that *each instance* of the clause with a relaxation goal in the body might need to consult a *different* relaxed preference model to ascertain the truth of the relaxation goal.

As before, we assign ordinal levels to the *O-predicates* and *D-predicates* in the program and construct the model in stages. Let us assume that any *O-predicate* O_1 that appears in a relaxation goal in the body of a clause defining an *O-predicate* O_2 is such that the level of O_1 is less than the level than that of O_2. The semantics presented here can be easily extended to the case when the *O-predicate* that appears in a relaxation goal is at the same level as the *O-predicate* at the head. However, if a *D-predicate* (or an *O-predicate*) is defined in terms of a relaxation goal containing an *O-predicate* of level n, it is assigned a level that is at least $n+1$. The *O-predicates* at level 1 do not have any relaxation goals in the bodies of clauses defining them. Therefore the semantics at level 1 is the same as before, i.e., each world is constructed by extending the least model for the core predicates with instances of the *O-predicates* and *D-predicates* so that it becomes a model for the definitions of the predicates at level 1 in the program.

Given a preference logic program P without relaxation goals in bodies of clauses, but with only one level of *O-predicates*, and a relaxation query $G =$ `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$, the **relaxed preference model** for P and G is a **sub-frame** M_r of the intended preference model M for P such that M_r contains all the worlds in M such that the only instances of $p(\bar{t})$ that appear in each world are those for which the corresponding instances of $c(\bar{u})$ are also present (if c is a constraint, the instances of $p(\bar{t})$ should correspond to valuations such that $c(\bar{u})$ is satisfiable). The set of atoms that are true in some strongly optimal world in the relaxed preference model of a program P and a relaxation query G are the **relaxed consequences** of P and G.

The *O-predicates* and *D-predicates* at higher levels may be defined in terms of relaxation goals. In general, the relaxed consequences of a program (with relaxation goals in the bodies of clauses) and a relaxation query with an *O-predicate* at any level can be defined once the intended preference model at that level has been defined. We define the intended preference model at a level by defining the worlds and showing how the arbiter orders the worlds. Once the intended preference model at level k has been defined, we can determine the various relaxed preference models with respect to relaxation goals whose *O-predicates* are of level k. When constructing the intended model at level $k+1$, the relaxed preference models of relaxation queries up to level k are used to define the worlds at level $k+1$. We now illustrate the above process by describing the semantics at level 2 in greater detail. For defining the preference model at level 2, we need to know the relaxed consequences for *O-predicates* whose level is at most 1. Since the clauses defining predicates at level 1 cannot have any relaxation goals in the body, we can determine the relaxed preference models as outlined in the previous paragraph.

In order to keep track of the relaxation goals that may be used in the definitions of *D-* and *O-predicates* at level 2, the structure of each world in the preference model at level 2 is a pair $\langle S, R\rangle$ where S is a subset of the base of the program[5] and R is an indexed set of sets, where each member of R is a set, denoted as $R_{\langle p(\bar{t}), c(\bar{u})\rangle}$, where $\langle p(\bar{t}), c(\bar{u})\rangle$ is the index, p is an *O-predicate* at level 1, and c is a *C-predicate* (or a constraint as in CLP). The set S is an extension of the set of preferential consequences at level 1 of the program P. A set $R_{\langle p(\bar{t}), c(\bar{u})\rangle}$ belongs to R if $p(\bar{t})$ is an instance (not necessarily ground) of an *O-predicate* (at level 1) and $c(\bar{u})$ is an instance of a *C-predicate* (or a constraint as in CLP)[6] and $R_{\langle p(\bar{t}), c(\bar{u})\rangle}$ is the set of instances of $p(\bar{t})$ that are *relaxed consequences* of P and the relaxation query `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$.

Definition 4 *Suppose q is an O-predicate in the program (at level 2) defined by a clause (with m ordinary and n relaxation goals) of the form:*

$$\begin{aligned} q(\bar{x}) \rightarrow\ & d_1(\bar{w}_1), \ldots, d_l(\bar{w}_l) \mid \\ & p_1(\bar{x}_1), \texttt{RELAX}\ r_1(\bar{v}_1)\ \texttt{WRT}\ c_1(\bar{u}_1), \ldots, \\ & p_m(\bar{x}_m), \texttt{RELAX}\ r_n(\bar{v}_n)\ \texttt{WRT}\ c_n(\bar{u}_n). \end{aligned}$$

A world $w = \langle S, R\rangle$ is said to **satisfy** *such a clause if the following holds: For every ground instance $q(\bar{x})\theta$ (where θ is a valuation that satisfies $d_1(\bar{w}_1), \ldots, d_l(\bar{w}_l)$) of the head to belong to S, if $q(\bar{x})\theta$ belongs to S, then there exist η_i, $i = 1 \ldots m$ and σ_j, $j = 1 \ldots n$ such that the conjunction $\bigwedge_i p_i(\bar{x}_i)\theta\eta_i \ \wedge\ \bigwedge_j r_j(\bar{v}_j)\theta\sigma_j$ is satisfiable, i.e., for all j, σ_j is a valuation such that $r_j(\bar{v}_j)\theta\sigma_j$ belongs to the set in R*

[5] Based on the pre-interpretation that interprets the constraint symbols, rather than the Herbrand pre-interpretation.

[6] It should be noted that the semantics presented here is powerful enough to capture the meaning of programs where the relaxation criterion is an *O-predicate* that is at a lower level than the relaxable predicate. We have, however, not discussed such programs in this paper.

indexed by $\langle r_j(\bar{v}_j)\theta, c_j(\bar{u}_j)\theta\rangle$, and, for all i, η_i is a valuation such that $p_i(\bar{x}_i)\theta\eta_i$ is a member of S.

Definition 5 *Suppose q is an D-predicate in the program (at level 2) defined by a clause (with m ordinary and n relaxation goals) of the form:*

$$q(\bar{x}) \leftarrow p_1(\bar{x}_1), \texttt{RELAX}\ r_1(\bar{v}_1)\ \texttt{WRT}\ c_1(\bar{u}_1), \ldots,$$
$$p_m(\bar{x}_m), \texttt{RELAX}\ r_n(\bar{v}_n)\ \texttt{WRT}\ c_n(\bar{u}_n).$$

A world $w = \langle S, R\rangle$ is said to **satisfy** *such a clause if the following holds: For every ground instance $q(\bar{x})\theta$, $q(\bar{x})\theta$ belongs to S if there exist valuations η_i, $i = 1 \ldots m$ and σ_j, $j = 1 \ldots n$ such that the conjunction $\bigwedge_i p_i(\bar{x}_i)\theta\eta_i \ \wedge \ \bigwedge_j r_j(\bar{v}_j)\theta\sigma_j$ is satisfiable, where, for all j, σ_j is a valuation such that $r_j(\bar{v}_j)\theta\sigma_j$ belongs to the set in R indexed by $\langle r_j(\bar{v}_j)\theta, c_j(\bar{u}_j)\theta\rangle$, and, for all i, η_i is a valuation such that $p_i(\bar{x}_i)\theta\eta_i$ is a member of S.*

The valuations must be consistent in that if a variable occurs in two different goals in the body of the clause, the corresponding valuations have to assign the same value to the variable. In other words, to determine whether a world $\langle S, R\rangle$ satisfies a clause, the truth of the ordinary goals in the body is determined by consulting the set S and the truth of the relaxation goals is determined by consulting the set in R that is indexed by the relevant instance of the relaxation goal. The example later in the section illustrates that the instance $r_j(\bar{v}_j)\theta$ (or $c_j(\bar{u}_j)\theta$) is not necessarily ground.

Definition 6 *The* **preference model** *at level 2 consists of all possible worlds $\langle S, R\rangle$ that satisfy the clauses defining predicates at level 2 such that the relation among the worlds is supported by the arbiter. The ordering among the worlds is determined by the instances of O-predicates in the set S.*

Definition 7 *The set of* **preferential consequences** *at level 2 is the set of atoms that are members of the set S in some strongly optimal world $\langle S, R\rangle$ in the preference model at level 2.*

Once the preference model at level 2 has been determined, the various relaxed preference models of the program with respect to relaxation queries with *O-predicates* of level 2 can also be determined. Essentially the relaxed models for P and a relaxation goal G (whose *O-predicate* is of level 2) are sub-frames of the preference model at level 2 such that the set S in each world contains only those instances of the relaxable predicate of G such that corresponding instances of the relaxation criterion of G are also present (satisfiable) in S.

Now at level k, each world has access to the relaxed consequences of *O-predicates* of level at most $k-1$. Each world $\langle S, R\rangle$ in the preference model at level k is such that it *satisfies* the clauses defining predicates at level k. If a preference logic program has n levels of predicate symbols, the **relaxed intended preference model** is the preference model at level n as described above.

For example, consider the `n_sh_path` program with the edges {`edge(a,b,5), edge(b,c,10), edge(a,c,25)`}. The set of preferential consequences at level 1 is:

{`edge(a,b,5), edge(b,c,10), edge(a,c,25)`} ∪

{`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)])`} ∪

{`sh_path(a,b,5,[e(a,b)]), sh_path(b,c,10,[e(b,c)]), sh_path(a,c,15,[e(a,b),e(b,c)])`}

Since there are no *O-predicates* at level 2, there is only one world $\langle S, R\rangle$ in the intended preference model at level 2. The set of instances of `n_sh_path` that are present in S are:

```
{ n_sh_path(1,a,b,5,[e(a,b)]),
n_sh_path(1,b,c,10,[e(b,c)]),
n_sh_path(1,a,c,15,[e(a,b),e(b,c)]),
n_sh_path(2,a,c,25,[e(a,c)])}
```

Notice that for the instance `n_sh_path(2,a,c,25,[e(a,c)])` to be present, we need to determine the truth of the the relaxable goal `RELAX sh_path(a,c,C,P) WRT C > 15`. In the corresponding relaxed intended preference model, the only optimal valuation for `sh_path(a,c,C,P)` is `C = 25`, `P = [e(a,c)]`. The set in R indexed by ⟨ `sh_path(a,c,C,P), C > 15` ⟩ is {`sh_path(a,c,25,[e(a,c)])`}.

Definition 8 *Given a preference logic program P, a ground atom A is said to be a* **relaxed preferential consequence** *of P (written $P \models_{\approx} A$) if A belongs to the set S in some strongly optimal world $\langle S, R\rangle$ in the intended preference model of P.*

Theorem 2 *For any preference logic program P with relaxable goals in the body, the relaxed intended preference model exists and is unique.*

The proof is similar to the case for preference logic programs without relaxation goals [17, 8].

5 Operational Semantics

After briefly reviewing the operational semantics of preference logic programs without relaxation goals [8], we present an extension of the operational semantics for programs with relaxation goals in the bodies of clauses.

5.1 Operational Semantics of Optimization

We now briefly review a derivation scheme called **PTSLD**-derivation, which stands for Pruned Tree SLD-derivation, for efficiently computing the *optimal valuations* to queries presented in [8]. Below we assume that the program consists of definite and optimization clauses without constraints in the sense of [10, 11]; we subsequently describe how this scheme can be extended to the case where the program contains constraints as goals in bodies of clauses.

5.1.1 PTSLD-Derivations

Let P be a definite clause program and G a positive 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 9 *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 $l = \leftarrow A_1, \ldots, A_m, \ldots, A_k$ 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. 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, $T_i \Rightarrow T_{i+1}$.*

Given a Preference Logic Program $P = \langle \mathcal{T}_C, \mathcal{T}_O, \mathcal{A} \rangle$, and a query G, a **TSLD derivation** for $P \cup \{G\}$ is an TSLD-derivation for $\mathcal{T}_C \wedge \mathcal{T}_O$. When the head of the selected goal in the node to be *expanded* is an *O-predicate* p, we assume that every ground instance of an *O-predicate* is supported by at most one $\rightarrow$ clause. This restriction only simplifies the description of the operational semantics and the proofs; otherwise, if instance of an *O-predicate* unifies with the heads of multiple $\rightarrow$ clauses, then it is a candidate for the optimal answer only if the appropriate instances of the bodies of each of the $\rightarrow$ clauses succeed.

Since the variables that appear only on the right hand sides of $\rightarrow$ clauses are existentially quantified, and in view of the above restriction, we can treat the $\rightarrow$ clause exactly as we would treat a $\leftarrow$ clause for the purpose of creating the children of a node. Furthermore, in order to achieve soundness, an *O-predicate* p must be invoked with unbound variables at those argument positions where the pair of instances of p in any arbiter clause differ. This requirement is needed because the values at these positions are computed by the body of the optimization clause and made use of by the arbiter to prefer one solution over another. If this requirement is not met at run-time, we simply replace the argument to the goal instance of p being considered by an unbound variable for the purpose of solving for p, and we enforce the original binding at this argument position by "back unification."

Definition 10 *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* **pruned** *if there exists a 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 O-predicate, and is subject to an arbiter of the form:*

$$p(\overline{a}) \preceq p(\overline{a_1}) \leftarrow L_1, \ldots, L_n$$

In addition θ_1 and θ_2 are the substitutions associated with nodes n_1 and n_2 such that $p(\bar{t})\theta_1$ is an instance of $p(\bar{a})$, $p(\bar{t})\theta_2$ is an instance of $p(\bar{a}_1)$ and the following constraint is satisfiable:

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

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

Definition 11 *Given a preference logic program P, a* **Pruned TSLD-derivation** *(***PTSLD***-derivation) is a TSLD-derivation in which, at each step, the leaf to be expanded is not a descendant of a pruned 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 12 *Given a program P and a goal G, θ is said to be a* **correct optimal answer** *to G with respect to P, if $P \models\!\!\approx G\theta$. Given P and a complete PTSLD-derivation for $P \cup \{G\}$ with result T_s, let $\Theta = \{\theta | \theta$ 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 P. We write $P \mid\!\sim G\theta$ if $\theta \in \Theta$.*

Theorem 3 (Soundness) *Given a preference logic program P and a goal G, suppose θ is a computed optimal answer for G with respect to P, then θ is a correct optimal answer to G with respect to P.*

Proof Sketch: By induction on the levels of the PTLSD search tree and the soundness and completeness of SLD-derivations for definite clauses [15]. All the solutions to *O-predicates* are considered as the $\rightarrow$ clauses are interpreted exactly as the $\leftarrow$ clauses by the operational semantics. Since *O-predicates* have to be sufficiently uninstantiated when invoked, we can show that the pruning is sound. ∎

Given a preference logic program P, a goal G, and a correct optimal answer θ of G with respect to P, we say that the operational semantics is complete if there exists a computed optimal answer η and a substitution γ such that $\theta = \eta\gamma$. However, PTSLD derivations are *not* complete for arbitrary preference logic programs. Incompleteness can arise because the PTSLD derivation for some goal cannot be completed, and this can happen when the PTSLD search tree emanating from an optimization goal is infinite but there is a well-defined correct optimal answer to the goal.

5.1.2 Stratified Preference Logic Programs

It turns out that even when the PTSLD search tree is finite, incompleteness can arise because the optimization predicates are not stratified. A preference logic program P is said to be **stratified** if the following holds: There is a mapping f from the set of *O-predicates* to the set $\{1, \ldots, n\}$, for some least n, such that if an instance of an *O-predicate* P_1 appears in the body of a $\rightarrow$ clause defining an *O-predicate* P_2, then $f(P_1) < f(P_2)$. For any *O-predicate* P, its **rank** is defined to be $f(P)$.

Stratified preference logic programs presented above are restrictive in that they do not allow recursive definitions of *O-predicates*. However, there are many programs, such as the dynamic programming formulation of shortest path, that need recursive definitions of *O-predicates*. Hence we are interested in locally stratified programs, described below.

Definition 13 *A preference logic program P is said to be* **locally stratified** *if the following two conditions hold:*

1. *There is a mapping f from the set of O-predicates to $\{1, \ldots, n\}$ for the least n such that if an instance of an O-predicate P_1 appears in the body of a $\rightarrow$ clause defining an O-predicate P_2, then $f(P_1) \leq f(P_2)$.*

2. *There is a well-founded ordering $\prec_k$ over the set of ground instances of all O-predicates of rank k, defined*

as follows: (i) ground instances of base facts of O-predicates of rank k all map to the $\perp$ of the ordering $\prec_k$; and (ii) ground instances of optimization clauses have the property that each instance of an O-predicate of rank k that appears in the body is $\prec_k$ the instance of the O-predicate of rank k that appears in the head.

Note that in defining $\prec_k$ we consider only those argument positions of a ground instance that are not used by the arbiter clauses to prefer one solution over another.

Theorem 4 *PTSLD derivations are complete for locally stratified preference logic programs with finite search trees.*

Proof Sketch: By induction on the rank of *O-predicates*, the ordering $\prec_k$ defined on the ground instances of *O-predicates* of each rank and soundness and completeness of SLD-resolution. ■

We now briefly describe the operational semantics when the first-order theory is a constraint logic program. The main difference from the definite clause case is that each node in the tree is not labeled by a single substitution but by a set of constraints. Pruning is accomplished by adding constraints to the node that rule out the pruned solution. Since the set of constraints associated with a node can have multiple solutions, it is possible that one solution to the set of constraints prunes another solution.

Definition 14 *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, choosing a goal A_m (whose head is a C-predicate or a D-predicate), a clause $A \leftarrow C'_1, \ldots, C'_k, B_1, \ldots, B_q$ in P, and creating children of l of the form:
$\langle \{A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k\}, \{C_j\} \cup \{C\} \cup \{C'_i\} \rangle$
if $\{C_j\} \cup \{C\} \cup \{C'_i\}$ is solvable[7], where $\{C\}$ is the set of constraints generated by the equation $A_m = A$, and the C'_is in the body of the clause are constraints. The leaf l is said to be **expanded** *in T_1 to get T_2.*

Definition 15 *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 $p(\bar{t})$ where p is an O-predicate, and is subject to an arbiter of the form: $p(\overline{a}) \preceq p(\overline{a_1}) \leftarrow L_1, \ldots, L_n$. In addition suppose the constraint*

$$\{p(\bar{t}) = p(\overline{a})\} \cup \{p(\overline{t'}^{8}) = p(\overline{a_1})\} \cup \bigcup_i \{L_i\} \cup \{C_{n_1}\} \cup \{C_{n_2}\}$$

is satisfiable by the substitution η such that the projection (γ) of η to the variables in $\{C_{n_1}\}$ is such that $p(\bar{t})\gamma$ is an instance of $p(\overline{a})$ and the projection to the variables in $\{C_{n_2}\}$, σ, is such that $p(\overline{t'})\sigma$ is an instance of $p(\overline{a_1})$.

We then update the constraint set $\{C_{n_1}\}$ 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 be **pruned**, *and a node in the tree is said to be pruned if all the solutions to the constraints of the node get pruned.*

[7] We use $\{C\}$, $\{C_j\}$, etc., to stand for a set of constraints.

[8] $\overline{t'}$ is a variant of $\bar{t}$ obtained by renaming those argument positions of $\bar{t}$ where $\overline{a}$ and $\overline{a_1}$ differ with new variables .

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.

Using the two definitions above, we can define PTSLD-derivations for preference logic programs with constraints in the bodies in a manner similar to PTSLD derivations for preference logic programs without constraints in bodies of clauses.

Definition 16 *Given a preference logic program P and a goal G, a* **computed optimal valuation** *to G with respect to P is is a valuation θ that satisfies the constraints at a successful node in a tree T at the end of a PTSLD derivation for P and G.*

We can also formulate soundness and completeness theorems in a manner similar to the definite clause case.

Theorem 5 *Given a preference logic program P and a goal G, suppose θ is a computed optimal valuation for G with respect to P, then θ is a correct optimal answer to G with respect to P.*

The proof is similar to the definite clause case.

Theorem 6 *Given a stratified preference logic program P and a goal G such that the PTSLD derivation emanating from P and G is finite, then if θ is a correct optimal answer to G with respect to P, then there is a successful node in the successful PTSLD tree that is satisfied by an valuation σ such that there exists a substitution η such that $\theta = \sigma\eta$.*

The proof is similar to the definite clause case.

5.2 Operational Semantics of Relaxation

We now present an extension of PTSLD-derivations to compute the relaxed correct optimal valuations to queries.

Definition 17 *Given a preference logic program P and a relaxation query G =* `RELAX` *$p(\bar{t})$* `WRT` *$c(\bar{u})$, the function $relax(p(\bar{t}), c(\bar{u}))$ returns the set of* **relaxed clauses** *for p by including, for every clause $p(\bar{x}) \rightarrow p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n)$ for p, the following pair of clauses:*

1. $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{x}) \rightarrow \bar{x} = \bar{t} \;\mid\; p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n), c(\bar{u})$
2. $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{x}) \rightarrow \bar{x} \neq \bar{t} \;\mid\; p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n)$

Furthermore, for every arbiter clause of p of the form $p(\bar{t}_1) \preceq p(\bar{t}_2) \leftarrow L_1, \ldots, L_n$, $relax(p(\bar{t}), c(\bar{u}))$ includes the following arbiter clauses:

1. $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t}\theta_1) \preceq relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t}\theta_2) \leftarrow L_1\sigma_1\sigma_2, \ldots, L_n\sigma_1\sigma_2, c(\bar{u}\theta_1), c(\bar{u}\theta_2)$, *where θ_1 and θ_2 are variable renaming substitutions such that $\bar{t}\theta_1$ and $\bar{t}\theta_2$ do not share any common variables. The substitutions σ_1 is the most general unifier of $\bar{t}_1$ and $\bar{t}\theta_1$, and σ_2 is the most general unifier of $\bar{t}_2$ and $\bar{t}\theta_2$.*
2. $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t}_1) \preceq relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t}_2) \leftarrow L_1, \ldots, L_n, (\bar{t} \neq \bar{t}_1 \vee \bar{t} \neq \bar{t}_2)$.

The **relaxed query** *corresponding to* G *is* $relax\_p(\bar{t})$.

For example, given the query

```
?- RELAX sh_dist(a,b,C,P) WRT notinpath(c,P),
```

$relax(\mathtt{sh\_dist(a,b,C,P)}, \mathtt{notinpath(c,P)})$ returns the following set of clauses:

1. $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(X,Y,C,P)} \rightarrow$ $\mathtt{(X,\ Y)\ =\ (a,\ b)} \mid \mathtt{path(X,Y,,C,P),\ notinpath(c,P)}$.

2. $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(X,Y,C,P)} \rightarrow$ $\mathtt{(X,\ Y)} \neq \mathtt{(a,\ b)} \mid \mathtt{path(X,Y,C,P)}$.

3. $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(a,b,C1,P1)} \preceq$ $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(a,b,C2,P2)} \leftarrow$ $\mathtt{C2\ <\ C1,\ notinpath(c,P1),\ notinpath(c,P2)}$.

4. $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(X,Y,C1,P1)} \preceq$ $\mathtt{relax\_sh\_dist}_{(a,b,C,P)}\mathtt{\_notinpath}_{(c,C)}\mathtt{(X,Y,C2,P2)} \leftarrow$ $\mathtt{C2\ <\ C1,\ (X,\ Y)} \neq \mathtt{(a,\ b)}$.

The set $relax(p(\bar{t}), c(\bar{u}))$ introduces a new *O-predicate* $relax\_p_{\bar{t}}\_c_{\bar{u}}$ into the program, the clauses for which are obtained from the clauses for p. Every clause for p that is applicable to $p(\bar{t})$ in the relaxation goal is modified by adding $c(\bar{u})$ to the body; clauses for p that are not applicable to $p(\bar{t})$ are not modified. If some instance of a clause for p is applicable to $p(\bar{t})$ then we obtain two clauses in the relaxed version, one that is applicable to $\bar{t}$ with the body containing $c(\bar{u})$, and the other that is not applicable to $\bar{t}$ with the body as before.

The key difference between our presentation here and the presentation in [9] is the following. Since we now allow relaxation goals to occur in the bodies of clauses, the relaxed version of the *O-predicate* gets a very specific name depending on the exact arguments with which it is invoked. [9] did not allow relaxation goals to occur in the bodies of clauses and therefore the operational semantics described there did not require the name of the relaxed version of an *O-predicate* to be so specialized.

We extend PTSLD-derivations by enabling them to augment the program when a relaxation goal is encountered. This is captured by presenting the derivation as a derivation of pairs of program and partial SLD-trees.

Definition 18 *A* **program-tree pair** *is a pair of the form* $\langle C, T\rangle$ *where* C *is a collection of clauses and* T *is a partial PTSLD-tree. The pair* $\langle C_{i+1}, T_{i+1}\rangle$ *is* **derived** *from the pair* $\langle C_i, T_i\rangle$, *if the following holds:*

1. *If the selected goal in the leaf to be expanded in* T_i *is not a relaxable goal,* $C_{i+1} = C_i$ *and* $T_i \Rightarrow T_{i+1}$ *with respect to clauses in* C_i.

2. *If the selected goal in the leaf to be expanded in* T_i *is a relaxable goal* RELAX $p(\bar{t})$ WRT $c(\bar{u})$, $C_{i+1} = C_i \cup relax(p(\bar{t}), c(\bar{u}))$, *the relaxable goal is replaced by* $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t})$, *and* $T_i \Rightarrow T_{i+1}$ *with respect to the clauses in* C_{i+1}.

Definition 19 *Given a preference logic program* P *and a goal* G, *a* **relaxable PTSLD-derivation** *is a sequence of pairs* $\langle P, G\rangle, \langle C_1, T_1\rangle, \ldots, \langle C_i, T_i\rangle, \ldots$ *where* C_i *is a set of clauses and* T_i *is a partial PTSLD-tree and each* $\langle C_{i+1}, T_{i+1}\rangle$ *is derived from* $\langle C_i, T_i\rangle$.

A relaxed PTSLD-derivation is said to be **successful** if there is a pair $\langle C, T\rangle$ at the derivation, where T is a successful PTSLD tree.

Definition 20 *Given a preference logic program* P *and a goal* G, *a* **relaxed computed optimal valuation** *to* G *with respect to* P *is is a valuation* θ *that satisfies the constraints at a successful node in* T *where* $\langle C, T\rangle$ *is a pair at the end of a relaxed PTSLD derivation* }.

Lemma 1 *Given a preference logic program* P *and a relaxation goal* $G =$ RELAX $p(\bar{t})$ WRT $c(\bar{u})$, $p(\bar{t})\theta$ *is a relaxed preferential consequence of* P *and* G, *if and only if* $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t})\theta$ *is a relaxed preferential consequence of* $P \cup relax(p(\bar{t}), c(\bar{u}))$.

Proof Sketch: By induction on the levels of the *O-predicates* in the program P. For the base case the number of levels of *O-predicates* in the program is 1. Suppose there is an instance $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t})\theta$ in a world in the intended preference model for $P \cup relax(p(\bar{t}), c(\bar{u}))$. Clearly, the instance $p(\bar{t})\theta$ belongs to a world in the relaxed intended preference model of P and G as $c(\bar{u})\theta$ is true. Similarly, we can show that if an instance $p(\bar{t})\sigma$ occurs in any world in the relaxed intended preference model of P and G, then the instance $relax\_p_{\bar{t}}\_c_{\bar{u}}(\bar{t})\sigma$ occurs in some world in the intended preference model of $P \cup relax(p(\bar{t}), c(\bar{u}))$. The arbiter clauses for $relax\_p_{\bar{t}}\_c_{\bar{u}}$ enforce the same ordering when the solutions to p satisfy c. Essentially the same argument works for the inductive case too. ∎

Theorem 7 (Soundness) *If* P *is a preference logic program and* G *is a goal such that* θ *is a relaxed computed optimal valuation then* $P \models_{\approx} G\theta$.

Proof Sketch: The proof is by induction on the levels of the *O-predicates* in the program. It makes use of the lemma above for soundness of answers computed for relaxation goals that appear in bodies of clauses. ∎

Lemma 2 *Given a stratified preference logic program* P *and a relaxation query* G *such that the relaxed PTSLD-derivation emanating from* P *and* G *is finite, suppose* $G\theta$ *is a relaxed preferential consequence of* P, *then there is a successful node in the successful PTSLD tree at the end of the relaxed PTSLD derivation such that* θ' *is a valuation that satisfies the constraints at the node and furthermore,* $\theta = \theta'\eta$ *for some substitution* η.

Proof Sketch: By induction on the level of the *O-predicates* in the program. For the base case: Suppose the relaxation query G was RELAX $p(\bar{t})$ WRT $c(\bar{u})$. Furthermore, by our assumptions and PTSLD-derivations are complete for the program P and the query $p(\bar{t})$ as there are no relaxation goals in bodies of clauses defining *O-predicates*. Therefore an answer at least as general as any correct optimal answer for the query $p(\bar{t})$ is computed by the operational semantics. Consider the relaxed program $P \cup relax(p(\bar{t}), c(\bar{u}))$, the search tree emanating from $c(\bar{u})$ is finite (the original program was such that the operational semantics was complete). Therefore, the operational semantics is capable of computing all the potential answers to $relax\_p_{\bar{t}}\_c\bar{u}(\bar{t})$ and the arbiter prunes the sub-optimal answers. Therefore if the operational semantics was complete for the original program, it is complete for the transformed program. Since the declarative semantics of the transformed program coincides with the relaxed intended preference model as far as answers to the relaxable query are concerned, we have our desired result. The inductive case is similar. ∎

Theorem 8 (Completeness) *If P is a stratified preference logic program with relaxation goals in bodies of clauses and G is a goal such that the relaxed PTSLD-derivation emanating from it is finite, then if θ is a valuation such that $G\theta$ is a relaxed preferential consequence of P, then there is a successful node in the PTSLD tree at the end of the PTSLD derivation emanating from P and G such that θ' is a valuation that satisfies the constraints at the node and there is a valuation η such that $\theta = \theta'\eta$.*

Proof Sketch: The proof is by induction on the levels of the *O-predicates* in the program. It makes use of the lemma above. ∎

6 Conclusions

The concept of preference provides a unifying approach to formulating problems requiring optimization and relaxation. In earlier work [8], we showed how optimization problems can be specified declaratively in the paradigm of preference logic programming. This paper extends our previous work in showing how the paradigm can also capture the notion of relaxation.

Motivated by practical considerations, we introduced the notion of preference relaxation in this paper. Essentially, given a query `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$, where p is an *O-predicate* and c is a *C-predicate*, we want the best answer for p that satisfies c. However, the best answer for p on its own may not satisfy c and we must consider sub-optimal answers for p to answer the query. We provided model theoretic and operational semantics for such relaxations and we introduced the notion of *relaxed preferential consequence*. The operational semantics consisted of transforming the original program by changing the definition of p. The changes to the $\rightarrow$ clauses defining p ensure that PTSLD-derivations with respect to the resulting program are sound for computing *relaxed preferential consequences*. The changes to the arbiter clauses for p ensure that if PTSLD-derivations were complete for the program to start with, they are also complete for the transformed program for computing *relaxed preferential consequences*.

We are investigating conditions under which the operational semantics for preference relaxation can be made more efficient. Given a relaxable goal `RELAX` $p(\bar{t})$ `WRT` $c(\bar{u})$, the operational semantics "pushes" $c(\bar{u})$ into the definition of p so that the solutions to $p(\bar{t})$ that do not satisfy $c(\bar{u})$ do not succeed. However, if p is defined recursively, we can try to gain better efficiency by pushing $c(\bar{u})$ into the recursive call, Intuitively, this is possible when the *C-predicate* in the relaxable query "distributes" over the structure that is passed to it. For example, the predicate `notinpath` distributes over paths, because a node `c` is not in some path P in a graph if and only if it is not in any sub-path of P. We cannot enforce `evenlength` for every sub-path as there are even length paths that are composed of paths of odd length. Therefore, a constraint such as `evenlength` does not distribute over paths.

The programs with relaxation goals in bodies of clauses considered in this paper can be termed stratified relaxation programs as the level of the *O-predicate* present in a relaxation goal in the body of a $\rightarrow$ clause had to be strictly lower than the level of the *O-predicate* at the head. We are investigating the semantics when we allow the level of the *O-predicates* in relaxation goals in bodies of the $\rightarrow$ clauses to be the same as the level of the *O-predicate* at the head.

References

[1] A. Brown, S. Mantha, and T. Wakayama. Logical Reconstruction of Constraint Relaxation Hierarchies in Logic Programming. In *Proc. of 7th Intl. Symp. on Methodologies for Intelligent Systems*, LNAI 689, Trondheim Norway, 1993.

[2] A. Brown, S. Mantha, and T. Wakayama. Preference Logics: Towards a Unified Approach to Non-Monotonicity in Deductive Reasoning. *Annals of Mathematics and Artificial Intelligence*, 10:233–280, 1994.

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

[4] F. Fages, J. Fowler, and T. Sola. Handling Preferences in Constraint Logic Programming with Relational Optimization. In *Proc. 6th Intl. Symposium PLILP '94*, LNCS 844, 1994.

[5] S. Ganguly, S. Greco, and C. Zaniolo. Minimum and Maximum Predicates in Logic Programming. In *Proc. 10th ACM Symp. on Principles of Database Systems*, pages 154–163, 1991.

[6] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proc. 5th Joint International Conference and Symposium on Logic Programming*, pages 1081–1086, 1988.

[7] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Logic Grammars. Technical Report 94-27, Dept. of Computer Science, SUNY at Buffalo, 1994.

[8] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Logic Programming. In *Proc. 12th Intl. Conf. on Logic Programming*, pages 731–745, 1995.

[9] K. Govindarajan, B. Jayaraman, and S. Mantha. Relaxation in Constraint Logic Languages. Technical Report 95-22, Dept. of Computer Science, SUNY at Buffalo, 1995.

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

[11] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 1994.

[12] P. C. Kannelakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. In *Proc. ACM Symp. on Principles of Database Systems*, pages 299–313, 1990.

[13] D. B. Kemp and P. J. Stuckey. Semantics of Logic Programs with Aggregates. In *Proc. International Logic Programming Symposium*, 1991.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization.* Wiley and Sons, Chichester, United Kingdom, 1985.

[15] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[16] M. J. Maher and P. J. Stuckey. Expanding Query Power in Constraint Logic Programming Languages. In E. L. Lusk and R. A. Overbeek, editors, *Proc. North American Conference on Logic Programming*, pages 20–36, 1989.

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

[18] K. Marriott and P. J. Stuckey. Semantics of Constraint Logic Programs with Optimization. *Letters on Programming Languages and Systems*, 2(1-4):181–196, 1993.

[19] F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.

[20] K. A. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Databases. In *Proc. ACM Symp. on Principles of Database Systems*, pages 114–126, 1992.

[21] S. Sudarshan and R. Ramakrishnan. Aggregation and Relevance in Deductive Databases. In *Proceedings of the International Conference on Very Large Databases*, 1991.

[22] S. Sudarshan, D. Srivastava, R. Ramakrishnan, and C. Beeri. Extending the Well-Founded and Valid Semantics for Aggregation. In *Proc. International Logic Programming Symposium*, pages 590–608, 1993.

[23] A. van Gelder, K. Ross, and J.S. Schlipf. Unfounded Sets and Well-Founded Semantics for General Logic Programs. *JACM*, 38(3):620–650, 1991.

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