Extending HCLP with Partially Ordered Hierarchies and Composite Constraints

(Partially Ordered HCLP and Composite Constraints)

C.K. Chiu and J.H.M. Lee
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Phone: +852 2609 8426
Fax: +852 2603 5024
Email: {ckchiu,jlee}@cse.cuhk.edu.hk

Abstract

Hierarchical Constraint Logic Programming (HCLP) extends the expressive power of Constraint Logic Programming (CLP) by allowing both required and non-required constraints, making the framework suitable for resolving and relaxing over-constrained problems. Each non-required constraint in HCLP is associated with a strength and constraint strengths are totally ordered. This artificial restriction is incompatible with a sizable class of real-life problems, in which some constraints cannot be classified as stronger or weaker than one another. Formulating this class of problems in HCLP in an *ad hoc* fashion would result in some valid solutions being discarded. To cope with this problem, we enhance the HCLP framework with a theory of partially ordered constraint hierarchy, resulting in PO-HCLP. Another anomaly of HCLP is due to the often need to model a complex constraint using a group of primitive constraints, which is a standard practice in the constraint programming community. We say that the primitive constraints are *related*. The lack of language facilities in HCLP to enforce the simultaneous satisfaction or relaxation of a group of related primitive constraints would result in, besides valid solution, nonsensical solutions. To resolve this problem, we introduce composite constraints into (PO-)HCLP. We present the formal syntax and semantics of PO-HCLP, and establish its soundness and completeness results. A prototype of PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) is constructed using CLP($\mathcal{R}$).

1 Introduction

Hierarchical Constraint Logic Programming (HCLP) [15, 16] extends the CLP scheme [8] by including non-required constraints (or soft constraints) so that overconstrained problems can be modeled more declaratively in this extended framework. HCLP, however, suffers from two limitations. First, constraints are classified into different levels according to their strength. The levels of constraints are then put under a totally ordered constraint hierarchy. In many overconstrained problems, however, it is unnecessary, and sometimes undesirable, to have an ordering on every two constraints with different strength. We thus argue that this totally ordered hierarchy is incompatible with the nature of some overconstrained problems, and the expressive power of HCLP is inherently limited. Second, in constraint programming, a complex constraint is usually declared as a conjunction of several primitive constraints. The primitive constraints should be regarded as an entity. This relation is unimportant in traditional constraint solving since all constraints must be satisfied simultaneously. In HCLP, non-required constraints can be relaxed. The relations between those non-required constraints defining a complex constraint must be recognized. Otherwise, undesirable answers may be computed. HCLP does not provide any mechanism to specify such relations. For example, users cannot declare which set of constraints must be satisfied or relaxed simultaneously.

In this paper, we propose two extensions to HCLP. First, we replace the totally ordered constraint hierarchy in HCLP by a partially ordered constraint hierarchy. A theory of partially ordered constraint hierarchy is developed from first principle. We show that the extended theory results in more intuitive and reasonable answers to certain types of overconstrained problems. Second, the notion of composite constraints are introduced for declaring a complex constraint using primitive constraints. Composite constraints allow programmers to enforce the satisfaction or relaxation of a group of primitive constraints simultaneously.

By extending HCLP with partially ordered constraint hierarchies and composite constraints, we obtain *Partially Ordered HCLP* (PO-HCLP), which subsumes the HCLP framework and can specify some overconstrained problems which cannot be specified declaratively in HCLP. We give the formal syntax and semantics of PO-HCLP. Soundness and completeness results are also presented. To demonstrate the feasibility of our proposal, we have built a prototype implementation of PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) in CLP($\mathcal{R}$). We also discuss the difficulties of constructing an efficient implementation of PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$).

The rest of the paper is organized as follows. Section 2 reviews related work, both inside and outside the logic programming community. Section 3 gives an overview of constraint hierarchies and the HCLP framework. Section 4 exemplifies the limitations of HCLP using two examples. Section 5 presents a theory of partially ordered constraint hierarchy and compares the theory with a similar proposal by Borning *et al* [1]. Section 6 defines composite constraints and describes how the error function in (partially ordered) constraint hierarchies should be modified to accommodate the new type of constraints. The PO-HCLP framework is detailed in section 7. We outline an instance of PO-HCLP and its prototype implementation in section 8. Section 9 concludes the paper by summarizing our contributions and sheds light on future direction of research.

2 Related Work

Borning *et al* [1] propose a constraint hierarchy formalism by using the notation of combining functions and lexicographic ordering. The possibility of introducing partially ordered hierarchies is also briefly discussed. They suggest the set of solutions of a partially ordered constraint hierarchy to be the union of solutions from all totally ordered hierarchies that are consistent with the partially ordered one. Wilson and Borning [16] adopt the constraint hierarchy theory to extend the CLP scheme [8] for non-required constraint processing. Jampel and Hunt [10] propose a weakening of the HCLP semantics, in which the incrementality and compositionality properties are preserved.

Govindarajan *et al* [6] propose preference logic programming (PLP), another extension of CLP, for declaratively specifying problems requiring optimization or selection among alternative solutions to a query. Constraint relaxations can be emulated in PLP using meta-programming techniques.

Outside the logic programming community, Fox [3] introduces the concepts of constraint "relaxation," "preferences," and "importance" to cope with conflicting constraints in job-shop scheduling. Shapiro and Haralick [13] treat inexact matching of structural descriptions using an extension of constraint satisfaction, which searches a solution within a given error bound. Snow and Freuder [14] suggest to assign weights to individual combinations of values. Freuder [4] and Freuder and Wallace [5] propose a general model, partial constraint satisfaction problem (PCSP), for modeling overconstrained problems. Several such solving techniques as branch and bound are proposed. Parker [12] realizes that partial ordering can be used as a programming paradigm.

In this paper, we pursue a general theory of partially ordered constraint hierarchy *from the first principle*. As far as we know, there is no previous work on the notion of composite constraints. We show how the partially ordered constraint hierarchy theory and composite constraints extend HCLP in such a way that its elegant semantic properties are retained.

3 Preliminaries

This section provides the theoretical background to this paper. We first outline such important concepts as combining functions and comparators in the constraint hierarchy theory. The hierarchical constraint logic programming (HCLP) framework is then presented. Interested readers may refer to [1] and [15] for details.

3.1 Constraint Hierarchies

A *constraint hierarchy* is a collection of constraints, each of which is associated with a *label* for stating its *strength*. The strength labels are *totally ordered*. According to the associated labels, constraints are classified into different disjoint sets, i.e. different *levels*. We use H_0 to denote the set of constraints which must be satisfied, i.e. *required constraints* (or *hard constraints*). The symbols $H_1, H_2, \ldots, H_n$ denote different sets of *non-required constraints* (or *soft constraints*), ranging from the strongest strength to the weakest one.

A *valuation* for a set of constraints is a function that maps the free variables in the constraints to elements in the domain over which the constraints are defined. We define an *error function* $e(c\theta)$ to measure how well a constraint c is satisfied by valuation θ. The error function returns non-negative real number and must have the property: $e(c\theta) = 0 \Leftrightarrow c\theta$ holds. To map e over a vector of constraints $C = [c_1, \ldots, c_k]$, we define a function $\mathbf{E}(C\theta)$ as

$$\mathbf{E}(C\theta) = [e(c_1\theta), \ldots, e(c_k\theta)].$$

An *error sequence* is a vector $[\mathbf{E}(H_1\theta), \mathbf{E}(H_2\theta), \ldots, \mathbf{E}(H_n\theta)]$.

A number of comparators can be defined to compare the "quality" of different valuations on the constraints of a given level. Since some comparators need to combine the errors at every level in a constraint hierarchy before doing comparisons, we introduce the notion of combining function. A *combining function* g is a function which applies to real-valued vector and returns some values that can be compared using the associated relations $<>_g$ and $<_g$. The $<>_g$ relation indicates that two valuations cannot be compared using $<_g$. We require the former relation to be reflexive and symmetric, and the latter to be irreflexive, antisymmetric, and transitive. To combine error sequences, we further generalize g to $\mathbf{G}$ such that if $R = [\mathbf{E}(H_1\theta), \mathbf{E}(H_2\theta) \ldots, \mathbf{E}(H_n\theta)]$, then

$$\mathbf{G}(R) = [g(\mathbf{E}(H_1\theta)), g(\mathbf{E}(H_2\theta)), \ldots, g(\mathbf{E}(H_n\theta))].$$

A *comparator* is defined by stating its associated combining function, relations $<>_g$ and $<_g$. All comparators can be classified as either *local*, *global*, or *regional*. For a local comparator, each constraint is considered individually. A valuation θ is "better" than a valuation σ if and only if θ satisfies all non-required constraints at least as well as σ does, and strictly better for at least one. They are incomparable if they achieve exactly the same degree of satisfiability on each constraint. Let $\mathbf{v} = [v_1, \ldots, v_n]$ and $\mathbf{u} = [u_1, \ldots, u_n]$. An example of local comparator is defined as follows:

$$g(\mathbf{v}) = \mathbf{v}$$

$$\mathbf{v} <_g \mathbf{u} \equiv \forall i \;\; v_i \leq u_i \wedge \exists j \text{ such that } v_j < u_j$$

$$\mathbf{v} <>_g \mathbf{u} \equiv \forall i \;\; v_i = u_i.$$

For a global comparator, the errors for all constraints at a given level are aggregated using g. For a regional comparator, each constraint at a given level is considered individually. Unlike local comparator, two valuations are incomparable if neither of them is better than the other one. An example of regional comparator is

$$g(\mathbf{v}) = \mathbf{v}$$

$$\mathbf{v} <_g \mathbf{u} \equiv \forall i \;\; v_i \leq u_i \wedge \exists j \text{ such that } v_j < u_j$$

$$\mathbf{v} <>_g \mathbf{u} \equiv \neg((\mathbf{v} <_g \mathbf{u}) \vee (\mathbf{u} <_g \mathbf{v})).$$

The solution set S to a constraint hierarchy is defined as:

$$\begin{aligned} S_0 &= \{\theta | \forall c \in H_0 \;\; e(c\theta) = 0\} \\ S &= \{\theta | \theta \in S_0 \wedge \forall \sigma \in S_0 \\ &\quad \neg(\mathbf{G}([\mathbf{E}(H_1\sigma), \ldots, \mathbf{E}(H_n\sigma)]) <_{\mathbf{g}} \mathbf{G}([\mathbf{E}(H_1\theta), \ldots, \mathbf{E}(H_n\theta)]))\} \end{aligned}$$

where $<_{\mathbf{g}}$ is a *lexicographic ordering* operator defined on error sequences $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$.

The $<_{\mathbf{g}}$ operator is defined as:

$$
\begin{aligned}
u_1, \ldots, u_n \quad <_{\mathbf{g}} \quad & v_1, \ldots, v_n \text{ if} \\
& \exists k \in \{1, \ldots, n\} \text{ such that} \\
& \forall i \in \{1, \ldots, k-1\}\ u_i <>_g v_i \ \wedge\ u_k <_g v_k.
\end{aligned}
$$

3.2 Hierarchical Constraint Logic Programming

HCLP adopts constraint hierarchies to extend the traditional CLP framework [8] for non-required constraint processing. An HCLP system is parameterized by a constraint domain and a comparator.

An HCLP program is a collection of clauses of the form

$$p \leftarrow q_1, \ldots, q_m, l_1 c_1, \ldots, l_n c_n \quad \text{where} \quad m, n \geq 0.$$

The symbols p, $q_1, \ldots, q_m$ denote atoms, $c_1, \ldots, c_n$ denote constraints, and $l_1, \ldots, l_n$ denote the associated labels of $c_1, \ldots, c_n$ respectively. For required constraints, the associated labels can be omitted.

The declarative semantics of HCLP is defined using the extended model, extended interpretation, and preferred model [15], which is an extension of the CLP model theory [8] with constraint hierarchy and comparator. Operationally, given a goal and an HCLP program, the goal is reduced as in CLP. Required constraints are passed to the solver immediately, while non-required constraints are pushed onto a stack. If the goal succeeds with non-unique valuations, the set of valuations are refined based on the non-required constraints. Non-required constraint solving algorithm may vary for different domains and comparators. For example, HCLP($\mathcal{R}$,$\star$) [15] adapts a variant of the Simplex algorithm [11] in its constraint solver.

4 Limitations of HCLP

"Why should apples be better or worse than oranges?"

D. Stott Parker [12]

HCLP provides a framework to model constraint relaxations by declaring and ranking required and non-required constraints under a constraint hierarchy. This framework, however, suffers from two limitations. First, all levels of constraints are totally ordered. Total ordering is unnecessary and even incompatible with the nature of some non-required constraints since the strength of two constraint levels is not necessarily comparable. Second, HCLP provides no mechanism to specify the relations between non-required constraints. In particular, there is no way to specify that some constraints (when used to specify a complex constraints) must either be satisfied or relaxed simultaneously. We observe that this mechanism is essential for eliminating undesirable valuations. Due to these two limitations, certain classes of overconstrained problems cannot be expressed declaratively. In the following, we support our claims using two simple but realistic examples.

The strength of two non-required constraints may be incomparable. Therefore, we see no reasons why there must be an ordering on any two levels of constraints. Imposing unnecessary ordering usually eliminates desirable valuations. Consider the following scenario. A president plans to meet representative(s) of his staff for half an hour after 1:00pm. He has no preferences on which department the representatives come from. However, in each department, senior staff should have higher priority than junior staff to attend the meeting but it is not necessarily true that a senior staff member in one department has higher priority than a junior staff member in another. In addition, if a staff member is available at more than one time slot, the president prefers a time slot in which as many other staff members are available as possible, provided that the above priority criteria is respected. There are two managers and two assistant managers in Department A, and one manager only in Department B. A secretary is shared by both departments and she reports directly only to the managers of the departments. The assistant managers are not more senior than the secretary and vice versa. The available time slots of the staff are summarized in the following table.

<table>
<tr><th></th><th>Manager 1</th><th>Manager 2</th><th>Assistant manager 1</th><th>Assistant manager 2</th><th>Secretary</th></tr>
<tr><td>Dept. A</td><td>after 3:00pm</td><td>after 4:00pm</td><td>after 4:00pm</td><td>at 5:00pm</td><td rowspan="2">at 6:00pm</td></tr>
<tr><td>Dept. B</td><td>at 2:00pm</td><td>—</td><td>—</td><td>—</td></tr>
</table>

Based on the above requirements, reasonable time to schedule the meeting is either at 2:00pm (feasible for manager 1 of Department B), 5:00pm (feasible for all staff members of Department A), or 6:00pm (feasible for the secretary, both managers and assistant manager 1 of Department A).

One may try to model the problem in HCLP by constructing a four-level constraint hierarchy, which consists of required (for the president), strong (for the managers), medium (for the assistant managers), and weak (for the secretary) constraints. Let T denote the meeting time. The HCLP program is shown as follows.

```
meeting(T) :-
      T > 1, deptA(T), deptB(T), common(T).
deptA(T) :-
      strong T>3, strong T>4, medium T>4, medium T=5.
deptB(T) :-
      strong T=2.
common(T) :-
      weak T=6.
```

This program, however, has two problems. First, the modeling is semantically incorrect since it implies: (1) the manager in Department B has higher priority than the assistant managers in Department A, and (2) the secretary has lower priority than the assistant managers. Second, given the query `?- meeting(X)`, we obtain only the answer $\{X = 5\}$ using the weighted-sum-predicate-better comparator [15]. Two other possible answers $\{X = 2\}$ and $\{X = 6\}$ are pruned.

This class of overconstrained problems cannot be modeled in the HCLP framework easily unless meta-programming techniques are employed.

A more natural way to model this problem is to classify the constraints into a partially ordered constraint hierarchy, as shown in figure 1. H_r denotes the required constraint level, while H_a, H_b, H_c, and H_d are four non-required constraint levels. Note that there is no specific ordering between H_c and H_a, between H_c and H_b, and also between H_b and H_d.

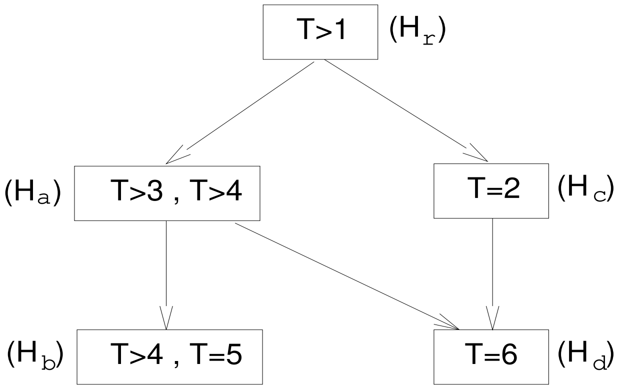

Figure 1: A Partially Ordered Constraint Hierarchy

The second problem of HCLP is that it provides no mechanism to specify the relations between non-required constraints. In constraint programming, it is common to use several primitive constraints to specify a more complex constraint. In classical constraint satisfaction where the main concern is the satisfaction of all constraints, this way of modeling complex constraint works well. In hierarchical constraint solving where constraint relaxation is allowed, we have to ensure that the primitive constraints modeling one complex constraint must either be satisfied or relaxed simultaneously. If such relations between non-required constraints cannot be recognized in constraint relaxations, undesirable valuations may be given as answers. We illustrate our point using a simple example: two managers need to meet their president, who is free from 2:00pm to 5:00pm on Wednesday and Thursday. They try to make only one appointment if a time can be compromised. Otherwise, two separate appointments are also acceptable. Assume that manager A is free at or before 3:00pm on the days before Thursday, and manager B is free at or after 4:00pm on the days after Wednesday. Let D and T denote the date and time of the meeting respectively. We model the problem in HCLP by four required constraints $\{D \geq 3, D \leq 4, T \geq 2, T \leq 5\}$ and four equally strong non-required constraints $\{D \leq 3, T \leq 3, D \geq 4, T \geq 4\}$. Using the locally-predicate-better comparator [15], we obtain the following four set of valuations: $\{D = 3, T \geq 2, T \leq 3\}$, $\{D = 4, T \geq 4, T \leq 5\}$, $\{D = 4, T \geq 2, T \leq 3\}$, and $\{D = 3, T \geq 4, T \leq 5\}$. Clearly, the latter two sets of valuations are undesirable since we can never make an appointment at 4:00pm on Wednesday. The undesirable answers cannot be eliminated due to the lack of insights into the relations of non-required constraints: namely that $\{D \leq 3, T \leq 3\}$ and $\{D \geq 4, T \geq 4\}$ must be regarded as two indecomposable constraints.

We tackle the first problem by replacing totally ordered hierarchies by partially ordered hierarchies, and resolve the second problem by introducing composite constraints.

5 Partially Ordered Constraint Hierarchies

The concept of partial ordering is not new to HCLP. All valuations are partially ordered according to their associated error sequences, which indicate the degree of satisfiability for each level of constraints. However, the levels of constraints are totally ordered. Thus the error sequences for the constraints at levels $H_1, H_2, \ldots, H_n$ are compared using a lexicographical ordering.

We generalize HCLP by replacing the totally ordered constraint hierarchy by a partially ordered hierarchy. The generalized framework, besides being more expressive, can also be explained coherently using partial ordering. We give the definition of partially ordered hierarchy and define its associated solution set in this section.

Similar to HCLP, constraints are classified into different sets, each of which consists of constraints with the same strength. The sets are, however, imposed only a partial ordering $\sqsupseteq$, instead of a total ordering. A *partial ordering on a set* is given by a reflexive, antisymmetric, and transitive relation defined for certain ordered pairs of elements of the set.

A partially ordered set of constraint sets, $\{H_r, H_{l_1}, \ldots, H_{l_n}\}$, forms a partially ordered constraint hierarchy H. The set H_r contains all the required constraints. By definition, $H_r \sqsupseteq H_{l_i}$ for $i = 1, \ldots, n$. We define the solution set S to H as follows:

$$\begin{aligned} S_r &= \{\theta \mid \forall c \in H_r \; e(c\theta) = 0\} \\ S &= \{\theta \mid \theta \in S_r \wedge \forall \sigma \in S_r \\ &\quad \sim (\mathbf{G}([\mathbf{E}(H_{l_1}\sigma), \ldots, \mathbf{E}(H_{l_n}\sigma)]) \;\; <_{\sqsupseteq_g} \;\; \mathbf{G}([\mathbf{E}(H_{l_1}\theta), \ldots, \mathbf{E}(H_{l_n}\theta)]))\} \end{aligned}$$

where $<_{\sqsupseteq_g}$ is a partial ordering operator.

The above definition is the same as the one defined for the totally ordered hierarchy in HCLP except that the lexicographical ordering operator $<_{\mathrm{g}}$ is replaced by a partial ordering operator $<_{\sqsupseteq_g}$. Informally speaking, given two error sequences $\mathbf{u} = u_{l_1}, u_{l_2}, \ldots, u_{l_n}$ and $\mathbf{w} = w_{l_1}, w_{l_2}, \ldots, w_{l_n}$, we say that $\mathbf{u} <_{\sqsupseteq_g} \mathbf{w}$ if

1. at least one error in $\mathbf{u}$ is smaller than the corresponding one in $\mathbf{w}$, and
2. if the errors in the two error sequences cannot be compared using either $<_g$ or $<>_g$ at one level, there must exist another stronger level, at which the error of $\mathbf{u}$ is smaller than the error of $\mathbf{w}$.

The above definition can be stated formally as follows:

$$\begin{aligned} u_{l_1}, u_{l_2}, \ldots, u_{l_n} \quad <_{\sqsupseteq_g} \;\; & w_{l_1}, w_{l_2}, \ldots, w_{l_n} \text{ if} \\ & (\exists u_{l_i} u_{l_i} <_g w_{l_i}) \; \wedge \\ & (\forall w_{l_j} \; w_{l_j} <_g u_{l_j}, \; \exists u_{l_k} \; u_{l_k} <_g w_{l_k} \wedge H_{l_k} \sqsupseteq H_{l_j}) \wedge \\ & (\forall u_{l_m} \; \sim ((u_{l_m} <_g w_{l_m}) \vee (u_{l_m} <>_g w_{l_m})), \; \exists u_{l_n} \; u_{l_n} <_g w_{l_n} \wedge H_{l_n} \sqsupseteq H_{l_m}). \end{aligned}$$

Again, $<_g$ and $<>_g$ can be either local, regional, or global comparators defined in the HCLP framework.

We illustrate our definition by comparing different valuations regarding the partially ordered hierarchy in figure 1. Given four valuations $X = 2, X = 4, X = 5$, and $X = 6$ with the weighted-sum-predicate-better comparator, the four corresponding partially ordered error sequences are shown in figure 2. The symbols u_i, v_i, w_i, x_i denote the aggregated errors for constraints at level i in the

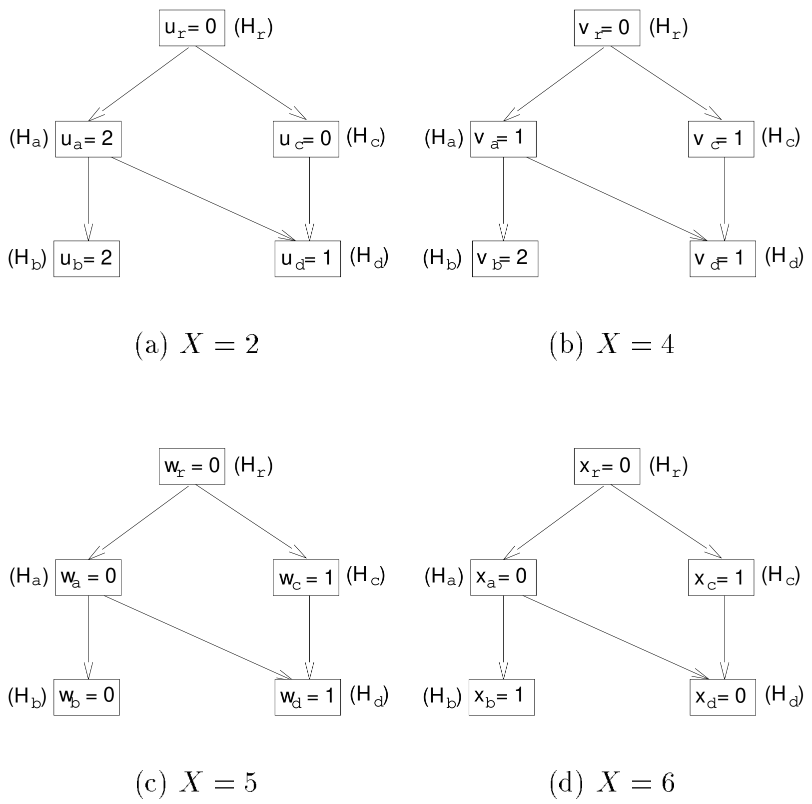

Figure 2: Four Partially Ordered Error Sequences

four hierarchies respectively.

We conclude that $X = 5$ and $X = 6$ are better than $X = 4$ since the first two valuations satisfy more constraints at level H_a, while the error values of the three valuations at level H_c are the same. The valuation $X = 4$ is then pruned.

The valuations $X = 2$, $X = 5$, and $X = 6$ are, however, incomparable since $u_a >_g w_a$ (or $u_a >_g x_a$) and $u_c <_g w_c$ (or $u_c <_g x_c$) for comparing $X = 2$ and $X = 5$ (or $X = 6$), while $w_b <_g x_b$ and $w_d >_g x_d$ for comparing $X = 5$ and $X = 6$. Thus, none of them can remove the other from the solution set of the partially ordered hierarchy.

Borning *et al* [1] propose another formulation of the set of solutions to a partially ordered constraint hierarchy. The set of all totally ordered hierarchies which are consistent with the original constraint hierarchy are first formed by adding permissible ordering, i.e. stronger, weaker, or equal, between the partially ordered strengths. The set of solutions to the original constraint hierarchy is then the union of the sets of solutions to the totally ordered hierarchies. The formal definitions can be stated as follows [1].

Definition 1 *If P is a partially ordered hierarchy, a totally ordered hierarchy H is* consistent *with P if (1) both hierarchies contain the same constraints, and (2) there is a mapping m from the strengths of P to the strengths of H such that if $s_1 < s_2$ in P, then $m(s_1) < m(s_2)$ in H, and (3) $\forall i, s_i c_i \in P$ iff $m(s_i) c_i \in H$.*

Definition 2 *Let P be a partially ordered hierarchy. Then*

$$solutions(P) = \bigcup_{H \in \mathcal{H}} solutions(H)$$

where $\mathcal{H}$ is the set of all totally ordered hierarchies consistent with P.

Their formulation, however, may give non-intuitive solutions to some partially constraint hierarchies. We discuss this issue in details in the rest of this section.

Consider the partially ordered hierarchy in figure 3. According to Borning *et al*'s definition, the solutions to the partially ordered hierarchy is equivalent to the union of the solutions of the three totally ordered hierarchies as shown in figure 4.

Using the locally-predicate-better comparator, the solution set is $\{\{X = 3\}, \{X = 4\}\}$, which

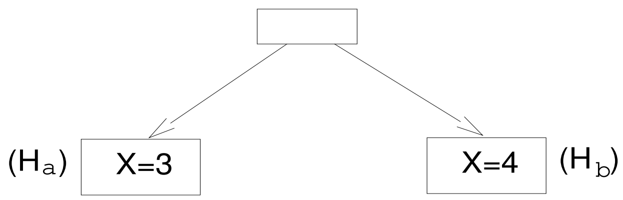

Figure 3: A Trivial Partially Ordered Hierarchy

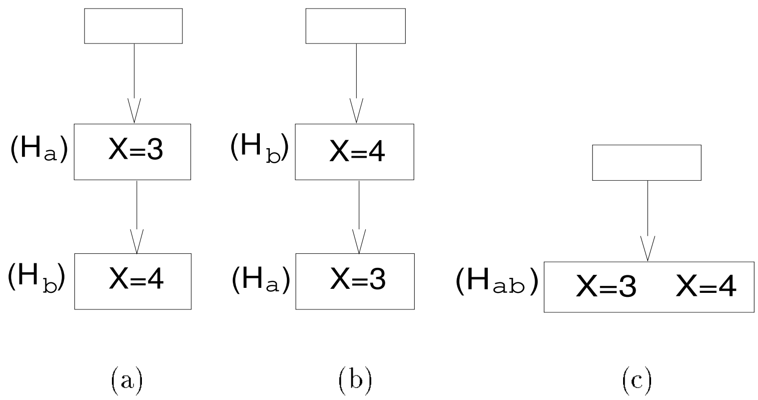

Figure 4: Three Possible Totally Ordered Hierarchies

is consistent with our formulation. For such global comparators as least-square-better [15], the solutions to the three totally ordered hierarchies are $X = 3$, $X = 4$, and $X = 3.5$ respectively. The solution set of the original hierarchy is thus concluded to be $\{\{X = 3\}, \{X = 3.5\}, \{X = 4\}\}$. This solution set is counter-intuitive since the error sequences of any two valuations $\{X = a\}$ and $\{X = b\}$, where a and b are between 3 and 4, should be equally good. They serve equally well as solutions to the example hierarchy. Obviously, a more intuitive solution set to this hierarchy should be $\{\{X = a\} \mid a \in [3, 4]\}$. This phenomenon is discussed in [1], which also gives a variant definition for the solutions to partially ordered hierarchies. Instead of just combining constraints from two incomparable strengths into a single strength, all possible weightings between the constraints would also be used. The solution set for the hierarchy in figure 3 is thus equivalent to the union of the solution set of the following *infinite* set of hierarchies:

$$\begin{array}{llllll} strong & X = 3 & strong & X = 4 & medium[w_1] & X = 3 \\ weak & X = 4 & weak & X = 3 & medium[w_2] & X = 4 \\ & & & & \multicolumn{2}{l}{\text{for all positive } w_1 \text{ and } w_2} \end{array}$$

The set of solutions in this case is $\{\{X = a\} \mid a \in [3, 4]\}$, which is compliant with our partially ordered hierarchy framework.

Borning *et al* [1] are brief on the exposition of a theory of partially ordered hierarchies. Based on the limited description provided in their classic paper, it is difficult for us to perform further comparisons on the two frameworks. While Borning *et al*'s definition is simple and elegant, our definition is derived from first principle and easy to verify. We conjecture that the two definitions are equivalent but it is a worthwhile exercise to establish the proof of the conjecture.

6 Composite Constraints

As stated in section 4, it is sometimes desirable to regard several constraints specifying a complex relation as an indecomposable entity. This can be achieved by defining them as a composite constraint.

Semantically, a *composite constraint* is a set of primitive constraints which can only be satisfied or relaxed simultaneously. Syntactically, a composite constraint is a set of constraints embraced by curly brackets. Since each primitive constraint can be seen as a composite constraint, each level in a constraint hierarchy can be generalized as a set of composite constraints. There are two points to note in defining a composite constraint.

First, we require that all primitive constraints in a composite constraint must be grouped together and embraced by curly brackets. This simple syntactic restriction should not decrease the expressive power significantly since it is natural for us to put the constraints specifying a relation together. On the contrary, this restriction simplifies the implementation since a composite constraint can be collected in one step.

Second, it is obvious to find that the traditional error function $e(c\theta)$, which indicates how nearly constraint c is satisfied for a valuation θ, does not apply to composite constraint since a composite constraint is a set of primitive constraints. To derive a new error function, we should keep in mind that the set of primitive constraints is regarded as an individual entity so that error values of the function should not be defined in terms of the accumulation of error values of all constraints in the constraint set. Otherwise, the degree of satisfiability on a composite constraint will be related to the cardinality of its associated constraint set. This is surely undesirable since the violation of a composite constraint will then have a more significant effect than the violation of a primitive constraint. Given a composite constraint c_x constructed from a set of primitive constraints $\{c_1, c_2, \ldots, c_n\}$, one possible definition of the error function is

$$e_x(c_x\theta) = \max(e(c_1), e(c_2), \ldots, e(c_n)).$$

We conclude this section by reformulating the two-manager-meeting problem in section 4. Let D and T denote the date and time of the meeting respectively. The free time of the two managers are specified using two equally strong composite constraints: $\{\{D \leq 3, T \leq 3\}, \{D \geq 4, T \geq 4\}\}$. We compare the four sets of valuations $\{D = 3, T \geq 2, T \leq 3\}$, $\{D = 4, T \geq 4, T \leq 5\}$, $\{D = 4, T \geq 2, T \leq 3\}$, and $\{D = 3, T \geq 4, T \leq 5\}$ using the locally-predicate-better comparator g:

Valuations	$e_x(\{D \leq 3, T \leq 3\}\theta)$	$e_x(\{D \geq 4, T \geq 4\}\theta)$	Combined Error Sequence
$\theta \in \{D = 3, T \geq 2, T \leq 3\}$	0	1	[[0,1]]
$\theta \in \{D = 4, T \geq 4, T \leq 5\}$	1	0	[[1,0]]
$\theta \in \{D = 4, T \geq 2, T \leq 3\}$	1	1	[[1,1]]
$\theta \in \{D = 3, T \geq 4, T \leq 5\}$	1	1	[[1,1]]

The two sets of valuations, $\{D = 4, T \geq 2, T \leq 3\}$ and $\{D = 3, T \geq 4, T \leq 5\}$, are pruned by the two other better sets of valuations since $[[0,1]] <_{\sqsupseteq_g} [[1,1]]$ and $[[1,0]] <_{\sqsupseteq_g} [[1,1]]$. We cannot compare $\{D = 3, T \geq 2, T \leq 3\}$ and $\{D = 4, T \geq 4, T \leq 5\}$ since we cannot conclude either that $[[0,1]] <_{\sqsupseteq_g} [[1,0]]$ or that $[[1,0]] <_{\sqsupseteq_g} [[0,1]]$. Therefore, the two sets of valuations become two different answers to the problem.

7 Partially Ordered HCLP

We propose to extend HCLP with partially ordered hierarchies and composite constraints, resulting in the Partially Ordered HCLP (PO-HCLP) framework. We present the syntax and semantics of PO-HCLP in this section.

7.1 Syntax

A PO-HCLP program consists of two different parts: constraint hierarchy declaration and generalized HCLP clauses.

Constraint Hierarchy Declaration The *constraint hierarchy declaration* is a collection of statements of the form

$$l > l' \quad \text{where } l \text{ and } l' \text{ are constraint labels,}$$

which declares that label l is stronger than label l'. We assume that an implicit label l_r is attached to the set of required constraints. By definition, we have $l_r > l$ for every strength label l. Such trivial statements can be omitted from the constraint hierarchy declaration.

The constraint hierarchy declaration forms a directed graph with strength labels as nodes. We require the graph to be acyclic.

Generalized HCLP Clauses A *generalized HCLP clause* is of the form

$$p \leftarrow q_1, \ldots, q_k, r_1, \ldots, r_m, l_1C_1, \ldots, l_nC_n \quad \text{where } k, m, n \geq 0.$$

The symbols p, $q_1, \ldots, q_k$ are atoms and $r_1, \ldots, r_m$ are required constraints as in [8]. $C_1, \ldots, C_n$ are non-required constraints, each labeled with a strength l_i defined in the constraint hierarchy declaration. A non-required constraint C_i can either be a primitive constraint or a composite constraint.

A composite constraint is constructed using a non-empty set of primitive constraints and has the form

$$\{c_1, c_2, \ldots, c_n\} \quad \text{where } n \geq 1 \text{ and } c_i \text{ is a primitive constraint for } i = 1, \ldots, n.$$

Note that a composite constraint must be declared as a non-required constraint. Thus, it must be assigned a strength label.

7.2 Semantics

PO-HCLP and HCLP share similar semantic properties. Operationally, PO-HCLP inherits from HCLP the same computation rules except that a more powerful constraint solver is required for a PO-HCLP system to generate solutions for partially ordered constraint hierarchies. Declaratively, PO-HCLP replaces the lexicographical ordering $<_{\mathbf{g}}$ by a partial ordering $<_{\sqsupseteq_g}$ in defining the preference relations over the extended interpretations [15]. We present both the operational and declarative semantics of PO-HCLP in this section. Since composite constraints can be regarded as indecomposable entities with different error functions, we simply refer both primitive constraints and composite constraints as constraints in the following discussion.

7.2.1 Operational Semantics

Let P be a PO-HCLP program and G_0 be a goal in the form ?– $\vec{\Theta}, \vec{\Delta}$, where $\vec{\Theta}$ is a set of constraints and $\vec{\Delta}$ is a set of atoms. A derivation step that reduces a goal G_p to another G_{p+1} follows:

> $\gamma \in \vec{\Delta}$ and the program P contains a rule $R, H :- \vec{\Theta'}, \vec{\Delta'}$, that the head atom H can be unified with γ, i.e. $H\theta = \gamma\theta'$. G_{p+1} is
>
> $$?\!-\ (\vec{\Theta'}\theta \bigcup \vec{\Theta}\theta'), (\vec{\Delta} \setminus \gamma)\theta' \bigcup \vec{\Delta'}\theta.$$

The set of constraints $(\vec{\Theta'}\theta \bigcup \vec{\Theta}\theta')$ forms the constraint hierarchy for G_{p+1} and we required that $S_r(\vec{\Theta'}\theta \bigcup \vec{\Theta}\theta') \neq \phi$, i.e. the set of required constraints are solvable. A derivation is a sequence of goals, possibly infinite. A derivation is *successful* if it is finite, the last goal is empty, and the corresponding constraint hierarchy, i.e. the *final constraint hierarchy*, has solutions. A valuation s is a *computed solution* for the query Q if and only if Q has a successful derivation and s is a solution for the final constraint hierarchy. A derivation is *finitely-failed* if it is finite but either the last goal has one or more atoms, the set of required constraints are unsolvable at some steps, or the final constraint hierarchy has no solutions. A query is finitely failed if every derivation for the query is finitely failed.

7.2.2 Declarative Semantics

PO-HCLP can be regarded as an extension of HCLP in which the lexicographical ordering $<_{\mathrm{g}}$ is replaced by a partial ordering $<_{\sqsupseteq_g}$. The declarative semantics of PO-HCLP can thus be defined using the extended model, extended interpretation, and preferred model in the HCLP model theory [15].

A *constraint domain* $\mathcal{D}$ is a structure defining the meaning of the primitive constraints. A *primitive constraint* is of the form $r(t_1, \cdots, t_n)$ where r is from the set $\prod_{\mathcal{D}}$ of predicate symbols of $\mathcal{D}$, and $t_1, \cdots, t_n$ are *terms* formed from function symbols and constants of $\mathcal{D}$. A *valuation* θ for a set V of variables is an assignment of values from the domain $\mathcal{D}$ to the variables in V. Suppose $V = \{x_1, \cdots, x_n\}$. We write θ as $\{x_1 \mapsto a_1, \cdots, x_n \mapsto a_n\}$ to indicate that each x_i is assigned the value a_i of $\mathcal{D}$. The *extended base* of a PO-HCLP program is defined as:

$$
\begin{aligned}
P_{ext-base} = \{ \quad & \langle p(x_1, \ldots, x_n)\theta, R\rangle \mid \\
& p \text{ is a predicate in } \textstyle\prod_{\mathcal{D}} \text{ and} \\
& \theta \text{ is a valuation on the variables } x_1, \ldots, x_n \text{ and} \\
& R \text{ is an error sequence } \}.
\end{aligned}
$$

Assume that every rule in a PO-HCLP program has a unique number. Let $\oplus$ be an operator which interleaves error sequences $R_1(= [r_{11}, \ldots, r_{1n}])$, $R_2(= [r_{21}, \ldots, r_{2n}])$, $\ldots$, $R_m(= [r_{m1}, \ldots, r_{mn}])$ so that

$$R_1 \oplus R_2 \oplus \cdots \oplus R_m = [r_{11}, r_{21}, \ldots, r_{m1}, \ldots, r_{1n}, r_{2n}, \ldots, r_{mn}].$$

Let $\mathcal{P}(B)$ denote the power set of B. An *extended model* for a program P is a subset I of $\mathcal{P}(P_{ext-base})$ such that every rule in P

$$(i) \quad A \leftarrow B_1, B_2, \ldots, B_m, H$$

and for every valuation $\theta \in S_r$,

$$\langle B_1\theta, R_1\rangle \in I_1, \langle B_2\theta, R_2\rangle \in I_2, \ldots, \langle B_m\theta, R_m\rangle \in I_m, \text{ for } I_1, I_2, \ldots, I_m \in I$$

implies

$$\langle A\theta, R_1 \oplus R_2 \oplus \cdots \oplus R_m \oplus [\mathbf{E}(H_{l_1}\theta), \mathbf{E}(H_{l_2}\theta), \ldots, \mathbf{E}(H_{l_n}\theta)]\rangle \in I_{i,1,2,\ldots,m} \in I.$$

MM_P is the *minimal extended model* for P if there is no other extended model M'_P for P such that $M'_P \subset MM_P$.

An *extended interpretation* is a subset of $\mathcal{P}(P_{ext-base})$. Let I and I' be two extended interpretations for a program, J and J' are their members respectively which have identical subscripts. Given a comparator g, we say that $I' \sqsubset_g I$ if

1. $J' \subset J$ and
2. if $\exists \langle p(x_1, \ldots, x_n)\sigma, R_\sigma \rangle \in J$ and $\langle p(x_1, \ldots, x_n)\sigma, R_\sigma \rangle \notin J'$, then $\exists \langle p(x_1, \ldots, x_n)\theta, R_\theta \rangle \in J'$ and $\mathbf{G}(R_\theta) <_{\sqsupset_g} \mathbf{G}(R_\sigma)$

A *preferred model* for a program P using a comparator g is defined as

$$\{p(x_1, \ldots, x_n)\theta \mid \exists J \in M \wedge \langle p(x_1, \ldots, x_n)\theta, R_\theta \rangle \in J\}$$

if

1. MM_P is an extended minimal model for P using g and $M \sqsubset_g MM_P$, and
2. there is no other extended interpretation M' such that $M' \sqsubset_g M$.

PO-HCLP generalizes HCLP by replacing the lexicographical ordering $<_{\mathbf{g}}$ by a partial ordering $<_{\sqsupset_g}$. Since the ordering operator is not related to the definitions of minimal extended model and successful derivation, the following lemma from the HCLP theory also holds true in PO-HCLP. We adopt the lemma to show that, in the PO-HCLP framework, the operational semantics agrees with the declarative semantics and vice versa.

Lemma 1 *[15] A query $Q = \{q(\vec{x})\}$ has a successful derivation of length i with final constraint hierarchy H iff there is a set $I_s \in MM_P$, with $|s| = i$, such that*

$$I_s = \{\langle q(\vec{x})\theta, R_q \rangle | \theta \in S_r \text{ and } R_q = [\mathbf{E}(H_{l_1}\theta), \ldots, \mathbf{E}(H_{l_n}\theta)]\},$$

where $\vec{x}$ is a vector of variables.

Theorem 1 (Soundness and Completeness of PO-HCLP) *A valuation σ is a computed solution for a query Q and program P if and only if $Q\sigma$ is in the preferred model for P.*

Proof: Consider the case in which σ is a computed solution for the query $Q = \{q(\vec{x})\}$ and program P. The query Q must have a successful derivation with final constraint hierarchy H and $\sigma \in S$. By lemma 1, we have a set I_s in MM_P such that

$$I_s = \{\langle q(\vec{x})\theta, R_q\rangle | \theta \in S_r \text{ and } R_q = [\mathbf{E}(H_{l_1}\theta), \ldots, \mathbf{E}(H_{l_n}\theta)]\}.$$

Since I_s contains all tuples for the valuations $\theta \in S_r$ and $\sigma \in S$, $q(\vec{x})\sigma$ must be in the preferred model for P.

If $q(\vec{x})\sigma$ is in the preferred model for P, by lemma 1, there must be a successful derivation with final constraint hierarchy H and $\langle q(\vec{x})\sigma, R_q\rangle \in MM_P$. It follows that $\sigma \in S_r$. Since $q(\vec{x})\sigma$ is in the preferred model, by definition, there are no other valuations better than σ in the partial ordering $<_{\sqsupseteq_g}$ defined by the comparator g. We conclude that $\sigma \in S$.

8 An Instance of PO-HCLP and its Implementation

PO-HCLP defines a class of languages sharing the same semantic properties. A PO-HCLP system is parameterized by its domain and comparator. In the following, we present the PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) system and discuss some implementation considerations.

PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) is a PO-HCLP instance working on the domain of real numbers, using the weighted-sum-predicate-better comparator [15]. The *weighted-sum-predicate-better* ($\mathcal{WSPB}$) comparator is a global comparator, in which the constraint errors at a given level are calculated using the trivial error function (returns 0 if the constraint is satisfied; 1 if it is not) and combined by taking the sum.

The two examples in section 4 can be formulated in PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) as follows.

```
%%%%%%%%%%%%%%%%%%
% Program 1
%%%%%%%%%%%%%%%%%%
```

```
% Generalized HCLP Clauses
meeting1(T) :-
      T > 1, deptA(T), deptB(T), common(T).
deptA(T) :-
      strong1 T>3, strong1 T>4, weak1 T>4, weak1 T=5.
deptB(T) :-
      strong2 T=2.
common(T) :-
      weak2 T=6.
% Constraint Hierarchy Declaration
strong1 > weak1.
strong1 > weak2.
strong2 > weak2.

%%%%%%%%%%%%%%%%
% Program 2
%%%%%%%%%%%%%%%%
meeting2(D,T) :-
      D>=3, D<=4, T>=2, T<=5, managerA(D,T), managerB(D,T).
managerA(D,T) :-
      strong {D<=3, T<=3}.
managerB(D,T) :-
      strong {D>=4, T>=4}.
```

There is no need for a constraint hierarchy declaration in program 2 since all the non-required constraints are of the same strength.

We implement a simple PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) meta-interpreter in CLP($\mathcal{R}$) [9]. Goals in our interpreter are reduced as in CLP($\mathcal{R}$). We pass all required constraints to the CLP($\mathcal{R}$) solver immediately, while accumulating non-required constraints to form a partially ordered constraint hierarchy. After all goals are reduced, if the computed solutions are not unique, we use the final constraint hierarchy to refine the solutions.

To solve the final constraint hierarchy, we enumerate every possible subset of non-required constraints in the hierarchy. A list is maintained to store valuations and their corresponding error sequences computed during the enumeration process. Once a valuation is calculated, it is added to the list immediately. A pruning procedure is then invoked to remove inferior valuations from the list. Valuations remaining in the list after the power set of non-required constraints is exhausted constitutes the computed solutions to the query.

Given the queries `?- meeting1(T)` and `?- meeting2(D,T)` for the above two programs respectively, the interpreter generates the solutions as follows.

```
PO-HCLP(R) Version 1.0 (Beta)
(c) Copyright The Chinese University of Hong Kong
1996 All Rights Reserved

1 ?- load(prog1).
2 ?- load(prog2).
3 ?- meeting1(T).
[T=2]
[T=6]
[T=5]
4 ?- meeting2(D,T).
[D=3,T<=3,2<=T]
[D=4,4<=T,T<=5]
5 ?-
```

Next, we consider a larger and a more complex problem instance, both in terms of the constraint hierarchy and programming. The task at hand is to allocate five jobs (A, B, C, D, and E) to two workers. Both workers have the same capability and can handle all jobs equally well. Each job must be handled by one worker and cannot be preempted during handling. Except job E which takes 10 time units to complete, all jobs have a duration of 5 time units. We require that all jobs to be finished within 15 time units. Job A is important and urgent. We prefer to complete it before starting the others (B, C, D, and E) if possible. However, there is no measure to tell whether it is more important for job A to precede B, to precede C, to precede D, or to precede E. Besides, we have two weaker preferences: job B precedes job D and job C precedes job E. While these two preferences are incomparable with each other, the former is less important than the precedence relationship between A and B & D, and the latter is less important than the precedence relationship between A and C & E. The corresponding partially ordered constraint hierarchy is shown in figure 5 and this problem can be formulated in PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) as follows.

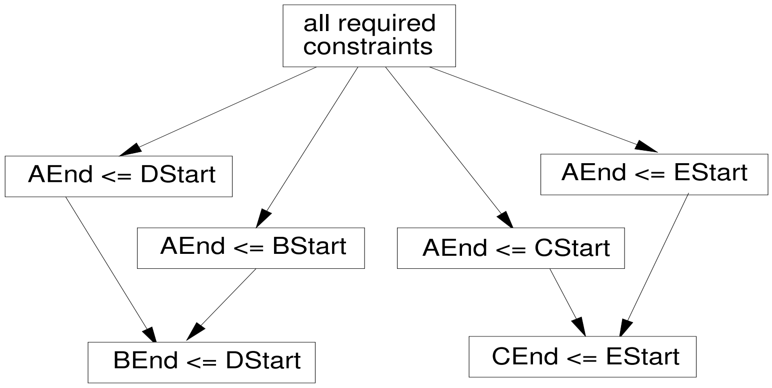

Figure 5: The Partially Ordered Hierarchy for the Job Allocation Problem

```
% Constraint Hierarchy Declaration
strong1 > weak1.
strong2 > weak1.
strong3 > weak2.
strong4 > weak2.

% Top-level Predicate
schedule([job(AWorker,AStart),job(BWorker,BStart), job(CWorker,CStart),
          job(DWorker,DStart), job(EWorker,EStart)]) :-
      Jobs = [[AWorker,AStart,AEnd,5], [BWorker,BStart,BEnd,5],
              [CWorker,CStart,CEnd,5], [DWorker,DStart,DEnd,5],
              [EWorker,EStart,EEnd,10]],
      % Required constraints
      duration(Jobs), workers(Jobs), nonOverlap(Jobs),
      % Non-required constraints
      strong1 AEnd<=BStart, strong2 AEnd<=DStart,
      strong3 AEnd<=CStart, strong4 AEnd<=EStart,
      weak1 BEnd<=DStart, weak2 CEnd<=EStart.

duration([]).
duration([[_,Start,End,Duration]|Rest]) :-
      Start+Duration=End, Start>=0, End<=15,
      duration(Rest).

workers([]).
workers([[Worker,_,_,_]|T]) :-
      % Disjunction constraint
      (Worker = 1) | (Worker = 2),
      workers(T).
```

```
nonOverlap([_]).
nonOverlap([H|T]) :-
      nonOverlapJobs(H,T),
      nonOverlap(T).

nonOverlapJobs(_,[]).
nonOverlapJobs([Worker1,Start1,End1,Duration1],[[Worker2,Start2,End2,_]|T]) :-
      % Implication constraint
      (Worker1 = Worker2) --> ((Start2 >= End1) | (Start1 >= End2)),
      nonOverlapJobs([Worker1,Start1,End1,Duration1],T).
```

We associate each job with its worker assignment, start time, end time, and duration. The required constraints enforce that all jobs must start after time 0 and end before time 15. They must last for the duration requirement as specified. We use disjunctive constraints to help specify the requirement that each job can only be assigned to either worker 1 or 2. Last but not least, we use an implication constraint to ensure that handling of jobs assigned to the same worker must not overlap. The disjunction constraint and the implication constraint used in the definition of predicates `workers/1` and `nonOverlapJobs/2` respectively are similar to those from [7].

Posing the query `?- schedule(Jobs)` to our system generates twenty-four different valuations. Many of these valuations are symmetrical variants of one another since worker assignment is taken into account. If we are concerned only with the start time of each job, we can replace the head of the top level predicate (`schedule/1`) definition to `schedule(AStart,BStart,CStart,DStart,EStart)`. The new definition yields six different valuations as follows.

```
1 ?- load(job).
2 ?- schedule(A,B,C,D,E).
[A=0, B=0, C=10, D=5, E=5]
[A=0, B=0, C=5, D=10, E=5]
[A=0, B=10, C=5, D=0, E=5]
[A=0, B=5, C=10, D=0, E=5]
[A=0, B=5, C=0, D=10, E=5]
[A=0, B=5, C=10, D=10, E=0]
3 ?-
```

Each valuation corresponds to four answer valuations of the old formulation.

Suppose now that we want the handling of jobs A and B and that of jobs B and D to be non-overlapping respectively. We can add the following required constraints to the body of the definition of the `schedule/1` predicate.

```
% A and B cannot be handled simultaneously
(AStart >= BEnd) | (BStart >= AEnd)
% C and D cannot be handled simultaneously
(CStart >= DEnd) | (DStart >= CEnd)
```

The solutions to the modified problem are:

```
1 ?- load(newjob).
2 ?- schedule(A,B,C,D,E).
[A=0, B=10, C=5, D=0, E=5]
[A=0, B=5, C=10, D=0, E=5]
[A=0, B=5, C=0, D=10, E=5]
[A=0, B=10, C=5, D=10, E=0]
[A=0, B=10, C=10, D=5, E=0]
3 ?-
```

Note that the first, the second, and the last answer valuations of the original problem are eliminated since they are inconsistent with the two new required constraints. It is, however, interesting to find that the two valuations, `[A=0, B=10, C=5, D=10, E=0]` and `[A=0, B=10, C=10, D=5, E=0]`, which are pruned by the sixth answer valuation in the original problem, become new answer valuations since (1) the sixth valuation to the original problem is no longer an answer valuation to the modified problem and (2) the new answer valuations are incomparable to the other three answer valuations in the restricted problem. This demonstrates the non-monotonic property of the (PO-)HCLP framework.

An important issue regarding PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$), which is work in progress, is an efficient implementation. We outline the main difficulties in the following.

The $\mathcal{WSPB}$ comparator is a global comparator. It implies that given a set of valuations (represented by several constraints), we cannot determine whether this is a solution without taking other valuations into account. In addition, unlike constraints on finite domain, valuations cannot be enumerated systematically.

The HCLP($\mathcal{R}$,$\star$) interpreter [15] employs the Simplex algorithm to support such global comparators as the weighted-sum-metric-better, the worse-case-better, and the locally-metric-better comparators. It works by first transforming the totally-ordered constraint hierarchy into a series of linear programming problems. The constraints at the strongest level are then solved by minimizing the maximum of the weighted errors. If a unique solution is obtained, the algorithm terminates. Otherwise, a constraint is added to the linear programming problem to attain the minimum value as computed in the previous step. Another linear programming problem, in which the goal is to minimize the weighted sum of the errors of the next strongest level of constraints, is set up. The algorithm proceeds in this fashion until all levels of constraints are considered or a unique answer is obtained.

The HCLP approach cannot be adopted directly in the implementation of PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$) since the constraint hierarchy is partially ordered. The order to process the levels of non-required constraints cannot be determined in advance. It implies that all different topological ordering must be exhausted, thus incurring a high complexity. Furthermore, this time-consuming algorithm is still not guaranteed to locate all possible solutions. Consider the constraint hierarchy with only non-required constraints in figure 6. Minimizing the errors of constraints at level H_a gives the valuation $\{X = 5, Y = 5\}$, while minimizing the errors of constraints at level H_b gives another valuation $\{X = 3, Y = 2\}$. In fact, two more valuations, $\{X = 5.5, Y = 4.5\}$ and $\{X = 2.5, Y = 2.5\}$, can be obtained by satisfying one constraint in each level.

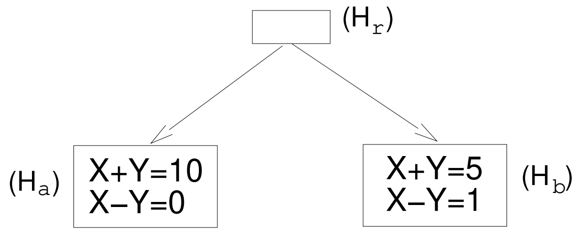

Figure 6: A Simple Partially Ordered Constraint Hierarchy

9 Concluding Remarks

Overconstrained problems constitute an important class of real-life applications. The HCLP scheme provides a new approach in including preference information in constraint logic programming, which offers an expressive and flexible language for problem specification and heuristics programming, allowing rapid program development for complex problems and enabling programs to be easily modified and extended. We exemplify two limitations of the HCLP framework. To remedy the situation, we propose a theory of partially ordered constraint hierarchies and the notion of composite constraints. We show how the two extensions can be incorporated into HCLP without sacrificing its elegant semantic properties. The resulting framework PO-HCLP allows the specification of a larger class of overconstrained problems naturally and declaratively. We identify a useful PO-HCLP instance, PO-HCLP($\mathcal{R}$,$\mathcal{WSPB}$), and implement its interpreter in CLP($\mathcal{R}$). We also raise and discuss issues related to efficient implementation of the instance.

Another important and interesting instance worth investigating is PO-HCLP(FD,$\mathcal{WSPB}$) for tackling finite domain constraints (*a la* CHIP [2]), which find applications in scheduling, binpacking, resource allocation, among others. Partially ordered hierarchical constraint solving on finite domains is closely related to the partial constraint satisfaction problems (PCSP) [4]. Some constraint solving techniques developed for PCSP can be adapted for the PO-HCLP(FD,$\mathcal{WSPB}$) constraint solver. For example, in PCSP constraint solving, the branch and bound method [4] works by keeping track of the best solution found so far and maintaining the number of constraint violations accumulated during searching. When it becomes clear that some branches in the search tree cannot lead to better solutions, the branches will simply be pruned. To adapt this method for solving partially ordered hierarchical constraints, we can simply maintain the combined error sequences of non-required constraints at each node of the search tree, instead of the number of constraint violations. The ordering of two error sequences can be determined in linear time.

References

[1] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[2] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, December 1988.

[3] M. Fox. *Constraint-Directed Search : A Case Study of Job-Shop Scheduling.* Morgan Kaufmann Publishers, 1987.

[4] E.C. Freuder. Partial constraint satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 278–283, 1989.

[5] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[6] K. Govindarajan, B. Jayaraman, and S. Mantha. Optimization and relaxation in constraint logic languages. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages (to appear)*, Florida, 1996.

[7] ILOG. *ILOG: Solver Reference Manual Version 3.0*, 1995.

[8] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[9] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.

[10] M. Jampel and S. Hunt. Composition in hierarchical CLP. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 640–645, 1995.

[11] K.G. Murty. *Linear Programming.* Wiley, 1983.

[12] D.S. Parker. Partial order programming. Technical report, Computer Science Department, University of California, Los Angeles, CA 90024-1596, 1987.

[13] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[14] P. Snow and E. Freuder. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. In *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 227–230, 1990.

[15] M.A. Wilson. *Hierarchical Constraint Logic Programming.* PhD thesis, Department of Computer Science and Engineering, University of Washington, 1993.

[16] M.A. Wilson and A. Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16(3):277–318, 1993.