Generalized Local Propagation: A Framework for Solving Constraint Hierarchies

Hiroshi Hosobe,[1] Satoshi Matsuoka,[2] and Akinori Yonezawa[1]

[1] Department of Information Science
[2] Department of Mathematical Engineering
University of Tokyo
3-1, Hongo 7-chome, Bunkyo-ku, Tokyo 113, JAPAN
{detail@is.s, matsu@ipl.t, yonezawa@is.s}.u-tokyo.ac.jp

Abstract. 'Constraint hierarchy' is a nonmonotonic system that allows programmers to describe over-constrained real-world problems by specifying constraints with hierarchical preferences, and has been applied to various areas. An important aspect of constraint hierarchies is the existence of efficient satisfaction algorithms based on local propagation. However, past local-propagation algorithms have been limited to multi-way equality constraints. We overcome this by reformulating constraint hierarchies with a more strict definition, and proposing *generalized local propagation* as a theoretical framework for studying constraint hierarchies and local propagation. Then, we show that *global semi-monotonicity* in satisfying hierarchies turns out to be a practically useful property in generalized local propagation. Finally, we discuss the relevance of generalized local propagation with our previous *DETAIL* algorithm for solving hierarchies of multi-way equality constraints.

Keywords: constraint hierarchies, nonmonotonicity, local propagation, multi-way constraints.

1 Introduction

Constraint hierarchies allow programmers to describe over-constrained real-world problems by specifying constraints with hierarchical *strengths* or preferences [1, 2], and have been applied to various research areas such as constraint logic programming [11, 13], constraint imperative programming [3], and graphical user interfaces [8, 9]. Intuitively, in a constraint hierarchy, the stronger a constraint is, the more it influences the solutions of the hierarchy. For example, the hierarchy of the constraints strong $x = 0$ and weak $x = 1$ yields the solution $x \leftarrow 0$. This property enables programmers to specify preferential or default constraints that may be used in case the set of required or non-default constraints are under-constrained. Moreover, constraint hierarchies are general enough to handle powerful constraint systems such as arithmetic equations and inequalities over reals. Additionally, they allow 'relaxing' of constraints with the same strength by applying, e.g., the least-squares method.

Theoretically, the key property of constraint hierarchies is *nonmonotonicity*. That is, addition of a new constraint to an existing hierarchy may change the set of solutions completely,[3] while in ordinary monotonic constraint systems, it would either preserve or reduce the solution set. For instance, if we add the constraint strong $x = 0$ to the hierarchy of weak $x = 1$, the solution will change from $x \leftarrow 1$ into $x \leftarrow 0$. Clearly, this nonmonotonic property gives us the power to specify default constraints.

An important aspect of constraint hierarchies as a nonmonotonic system is that there are efficient satisfaction algorithms proposed. We can categorize them into the following two approaches:

The refining method first satisfies the strongest level, and then, weaker levels successively. It is employed in the DeltaStar algorithm [11] and a hierarchical constraint logic programming language CHAL [10].

Local propagation gradually solves hierarchies by repeatedly selecting uniquely satisfiable constraints. It is mainly used in constraint solvers for graphical user interfaces such as DeltaBlue [4], SkyBlue [9], and *DETAIL* [6].

First, to illustrate the refining method, suppose we have a hierarchy consisting of required $x = y$, strong $y = z + 1$, medium $z = 0$, and weak $z = 1$. This is solved as follows: first, by satisfying the strongest constraint required $x = y$, the method reduces the set $\boldsymbol{\Theta}$ of all variable assignments (mappings from variables to their values) to $\{\theta \in \boldsymbol{\Theta} \mid \theta(x) = \theta(y)\}$; second, by fulfilling the next strongest one strong $y = z + 1$, we obtain $\{\theta \in \boldsymbol{\Theta} \mid \theta(x) = \theta(y) \wedge \theta(y) = \theta(z) + 1\}$; third, evaluating medium $z = 0$ yields $\{\theta \in \boldsymbol{\Theta} \mid \theta(x) = 1 \wedge \theta(y) = 1 \wedge \theta(z) = 0\}$; now, the weakest constraint weak $z = 1$ conflicts with the assignments that have been generated from the stronger constraints, and therefore, remains unsatisfied. As shown in this example, the refining method is a 'straightforward' algorithm for solving constraint hierarchies.

Next, to demonstrate local propagation, reconsider the hierarchy in the last example. Local propagation handles it as follows: first, since medium $z = 0$ can be uniquely solved, it acquires $\{\theta \in \boldsymbol{\Theta} \mid \theta(z) = 0\}$; next, since the instantiation of z makes strong $y = z + 1$ uniquely satisfiable, it produces $\{\theta \in \boldsymbol{\Theta} \mid \theta(y) = 1 \wedge \theta(z) = 0\}$; finally, computing required $x = y$, it outputs $\{\theta \in \boldsymbol{\Theta} \mid \theta(x) = 1 \wedge \theta(y) = 1 \wedge \theta(z) = 0\}$. Note it must reject the weakest constraint weak $z = 1$ at the beginning; otherwise, it would yield an incorrect or empty solution. As suggested with this example, local-propagation algorithms must *plan* in what order they will choose and solve constraints, discarding the ones that lead to incorrect solutions.

Local propagation takes advantage of the potential locality of typical (possibly, non-hierarchical) constraint networks in graphical user interfaces. Basically, it is efficient because it uniquely solves a single constraint in each step. In addition, when a variable value is repeatedly updated by an operation such

[3] Wilson and Borning refer to the property, in a less familiar word, as 'disorderly' [12]. Instead, they use nonmonotonicity for another concept in hierarchical constraint logic programming.

as dragging in interactive interfaces, it can easily re-evaluate only the necessary constraints. However, local propagation has been restricted to *multi-way equality constraints* which can be uniquely solved for each variable, e.g. linear equations over reals. Also, it cannot find multiple solutions for a given constraint hierarchy due to the uniqueness.

Naturally, a question arises whether we can 'generalize' local propagation to solve hierarchies of more powerful constraints without losing its efficiency. In this research, we first reformulate the constraint hierarchy theory, and then introduce a property of constraint systems called *global semi-monotonicity*, which is weaker than monotonicity but not disordered nonmonotonicity. Next, we propose *generalized local propagation*, a theoretical framework for investigating local propagation on constraint hierarchies, and show that global semi-monotonicity exhibits a practically useful property in generalized local propagation. Finally, to illustrate the utilization of GLP, we relate it with our previous *DETAIL* algorithm for multi-way equality constraints that can be simultaneously solved or properly relaxed [6].

2 Related Work

This section briefly overviews previous researches on nonmonotonic constraint systems from the viewpoint of local propagation.

Borning et al., the originators of constraint hierarchies [1], studied properties of hierarchies [2, 12], and also developed local-propagation algorithms called DeltaBlue [4] and SkyBlue [9]. However, their research on theoretical properties did not cover local propagation on constraint hierarchies, but rather mainly focused on hierarchical constraint logic programming (HCLP) [11, 12, 13].

Jampel constructed a certain HCLP instance that separates the HCLP scheme into compositional and non-compositional parts [7]. The method is expected to improve the efficiency of interpreters and compilers since the compositional part is efficiently implementable. However, it is unclear whether such a method is applicable to local propagation.

Freuder and Wallace proposed partial constraint satisfaction for handling constraint satisfaction problems that are impossible or impractical to solve [5]. Theoretically, it is general enough to simulate constraint hierarchies. However, the presented algorithms were ones searching for approximate solutions by 'weakening' problems over finite domains.

3 Generalized Local Propagation

In this section, we first reformulate constraint hierarchies, and then introduce global semi-monotonicity of constraint hierarchies. Next, we generalize local propagation on constraint hierarchies, and study its properties for obtaining correct solutions.

3.1 A Reformulation of Constraint Hierarchies

Before generalizing local propagation, we modify the original formulation of constraint hierarchies in [2] so that it will allow us to better investigate local propagation. Intuitively, the main changes are to explicitly parameterize target hierarchies, and to replace concrete embedded functions/relations with abstract ones satisfying reasonable conditions. First, we define basic terms and symbols. Let $\boldsymbol{X}$ be the set of all variables, $\boldsymbol{D}$ the domain of the variables, and $\boldsymbol{C}$ the set of all constraints.[4] Given a constraint c, $\boldsymbol{X}(c)$ denotes the set of all the variables constrained by c, and given a set C of constraints, we define $\boldsymbol{X}(C) = \{x \in \boldsymbol{X} \mid \exists c \in C.\, x \in \boldsymbol{X}(c)\}$. A strength of a constraint is an integer l such that $0 \leq l \leq w$, where w is some positive integer. Intuitively, the larger the integer is, the weaker the strength is. Let $\boldsymbol{L}$ be the set of all the strengths. A constraint c with a strength l is represented by c/l. A constraint hierarchy is a finite set H of constraints with strengths, and $\boldsymbol{H}$ expresses the set of all constraint hierarchies. For brevity, we write a variable as x, a constraint as c, a strength as l, and a constraint hierarchy as H, possibly with primes or subscripts.

To represent solutions to constraint hierarchies, we use variable assignments. A variable assignment, denoted as θ, is a mapping from $\boldsymbol{X}$ to $\boldsymbol{D}$, and $\boldsymbol{\Theta}$ indicates the set of all variable assignments. Given a set X of variables, we define $\theta(X) = \theta'(X)$ as $\forall x \in X.\, \theta(x) = \theta'(x)$.

To assign semantics to constraints, we first introduce *error functions* in the same manner as the original formalization of constraint hierarchies [2]:

Definition 1 (error function). An error function for l is a mapping $e_l : \boldsymbol{C} \times \boldsymbol{\Theta} \to \{0\} \cup \boldsymbol{R}^+$ such that for any c, θ, and θ', $\theta(\boldsymbol{X}(c)) = \theta'(\boldsymbol{X}(c)) \Rightarrow e_l(c, \theta) = e_l(c, \theta')$.

Intuitively, $e_l(c, \theta)$ indicates the error of c/l under θ, which is zero if c/l is exactly satisfied, and positive otherwise. The condition requires that errors of a constraint under two variable assignments are equal if the assignments have equal values for each constrained variable.

Next, we introduce *level comparators*:

Definition 2 (level comparator). A level comparator for l is a ternary relation $\overset{\cdot/l}{\leq} : \boldsymbol{H} \times \boldsymbol{\Theta} \times \boldsymbol{\Theta}$ such that for any H, H', θ, θ', and θ'',[5]

$$\forall c \in \boldsymbol{C}.\, (c/l \in H \Leftrightarrow c/l \in H') \Rightarrow (\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \theta \overset{H'/l}{\leq} \theta') \tag{1}$$

$$\forall c/l \in H.\, e_l(c, \theta) = e_l(c, \theta'') \Rightarrow (\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \theta'' \overset{H/l}{\leq} \theta') \tag{2}$$

$$\forall c/l \in H.\, e_l(c, \theta') = e_l(c, \theta'') \Rightarrow (\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \theta \overset{H/l}{\leq} \theta'') \tag{3}$$

$$\forall c/l \in H.\, e_l(c, \theta) \leq e_l(c, \theta') \Rightarrow \theta \overset{H/l}{\leq} \theta' \tag{4}$$

[4] We simply define variables and constraints as elements in the corresponding sets, and separately provide their semantics using certain functions and relations.

[5] When we write $\forall c/l \in H$, we mean that the universal quantifier $\forall$ is associated only with c. In other words, l is either free or quantified by another preceding one.

$$\theta \overset{H/l}{\leq} \theta' \wedge \theta' \overset{H/l}{\leq} \theta'' \Rightarrow \theta \overset{H/l}{\leq} \theta'' \tag{5}$$

$$\theta \overset{H/l}{\leq} \theta' \wedge \theta \overset{H'/l}{\leq} \theta' \Rightarrow \theta \overset{H \cup H'/l}{\leq} \theta' \ . \tag{6}$$

Intuitively, $\theta \overset{H/l}{\leq} \theta'$ means "θ is better than or similar to θ' according to l of H." Conditions (1)–(3) say that the scope of a level comparator is restricted to be inside a designated level. Condition (4) indicates that if errors of all constraints at a level under an assignment are smaller than or equal to those under another assignment, then the former assignment is better than or similar to the latter according to the level. Condition (5) is 'transitivity' of a level comparator. Condition (6) means that if, in two hierarchies, an assignment is better than or similar to another according to the level, then the relation holds in the combination of the hierarchies.

For convenience, we define $\overset{\cdot/l}{\geq}$ (worse than or similar to), $\overset{\cdot/l}{\sim}$ (similar to), $\overset{\cdot/l}{<}$ (better than), $\overset{\cdot/l}{>}$ (worse than), and $\overset{\cdot/l}{\not\sim}$ (incomparable with) as follows: $\theta \overset{H/l}{\geq} \theta' \Leftrightarrow \theta' \overset{H/l}{\leq} \theta$; $\theta \overset{H/l}{\sim} \theta' \Leftrightarrow \theta \overset{H/l}{\leq} \theta' \wedge \theta \overset{H/l}{\geq} \theta'$; $\theta \overset{H/l}{<} \theta' \Leftrightarrow \theta \overset{H/l}{\leq} \theta' \wedge \neg\, \theta \overset{H/l}{\sim} \theta'$; $\theta \overset{H/l}{>} \theta' \Leftrightarrow \theta \overset{H/l}{\geq} \theta' \wedge \neg\, \theta \overset{H/l}{\sim} \theta'$; $\theta \overset{H/l}{\not\sim} \theta' \Leftrightarrow \neg\, \theta \overset{H/l}{\leq} \theta' \wedge \neg\, \theta \overset{H/l}{\geq} \theta'$.

The original definition of level comparators is quite different from Definition 2 in the following ways: it separates $\overset{\cdot/l}{\leq}$ into $\overset{\cdot/l}{<}$ and $\overset{\cdot/l}{\sim}$, and defines them constructively; it includes (1)–(3) operationally; it seems to implicitly assume (4); it does not require the transitivity of $\overset{\cdot/l}{\sim}$ unlike (5); it presents no condition like (6). Theoretically, the greatest difference is the lack of the transitivity of $\overset{\cdot/l}{\sim}$, which we will discuss later in Subsect. 3.4.

A useful example of a level comparator is the *least-squares level comparator*, defined as $\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \sum_{c/l \in H} e_l(c,\theta)^2 \leq \sum_{c/l \in H} e_l(c,\theta')^2$. Here, two variable assignments are compared by summing squares of errors of constraints at the level. It is easy to prove that the definition fulfills the conditions for level comparators. Used in satisfaction of constraint hierarchies, it works as the least-squares method within level l.

Next, we define *constraint-hierarcy comparators* that totally compare hierarchies by combining level comparators:

Definition 3 (constraint-hierarcy comparator). A constraint-hierarcy comparator is a ternary relation $\dot{<}$: $\boldsymbol{H} \times \boldsymbol{\Theta} \times \boldsymbol{\Theta}$ such that for any H, θ, and θ', $\theta \overset{H}{<} \theta' \Leftrightarrow \exists l \in \boldsymbol{L}.(\forall l' \in \boldsymbol{L}.\, l' < l \Rightarrow \theta \overset{H/l'}{\sim} \theta') \wedge \theta \overset{H/l}{<} \theta'$.

Intuitively, $\theta \overset{H}{<} \theta'$ means "θ is better than θ' according to H." It is defined as lexicographic ordering with level comparators as its components. Consequently, the result of a level comparator has absolute priority over those of weaker ones.

For convenience, we define $\dot{>}$ (worse than), $\dot{\sim}$ (similar to), $\dot{\leq}$ (better than or similar to), $\dot{\geq}$ (worse than or similar to), and $\dot{\not\sim}$ (incomparable with) as follows:

$\theta \overset{H}{>} \theta' \Leftrightarrow \theta' \overset{H}{<} \theta$; $\theta \overset{H}{\sim} \theta' \Leftrightarrow \forall l \in \boldsymbol{L}.\theta \overset{H/l}{\sim} \theta'$; $\theta \overset{H}{\leq} \theta' \Leftrightarrow \theta \overset{H}{<} \theta' \vee \theta \overset{H}{\sim} \theta'$; $\theta \overset{H}{\geq} \theta' \Leftrightarrow \theta \overset{H}{>} \theta' \vee \theta \overset{H}{\sim} \theta'$; $\theta \overset{H}{\not\sim} \theta' \Leftrightarrow \neg\theta \overset{H}{\leq} \theta' \wedge \neg\theta \overset{H}{\geq} \theta'$.

The following definition describes the satisfaction of constraint hierarchies using a constraint-hierarcy comparator:

Definition 4 (constraint-hierarcy satisfier). A constraint-hierarcy satisfier is a mapping $S : 2^{\boldsymbol{\Theta}} \times \boldsymbol{H} \to 2^{\boldsymbol{\Theta}}$ defined as $S(\Theta, H) = \{\theta \in \Theta \mid \neg\exists\theta' \in \Theta.\, \theta' \overset{H}{<} \theta\}$.

As a shorthand, we write $S(H)$ instead of $S(\boldsymbol{\Theta}, H)$. Intuitively, $S(\Theta, H)$ is the set of assignments obtained by nonmonotonically satisfying H in Θ. By definition, an assignment in $S(\Theta, H)$ is an element in Θ such that there is no better assignment in Θ when compared according to H.

Finally, we define solutions of constraint hierarchies:

Definition 5 (solution). A solution to H is a variable assignment in $S(H)$.

In other words, a solution to H is an assignment found by nonmonotonically satisfying H in the set of all assignments.

One difference between the original and our formulations in defining constraint-hierarcy comparators is that the original restricts top-level constraints to be required, whereas ours allows conflicting constraints at the top level. This is because our definition of hierarchy satisfiers excludes the special treatment of the top level. However, the resulting solutions are the same so far as the top level is not over-constrained. Also, even if we add the condition for the top level to be required, we can easily accommodate it in our following proofs.

3.2 Global Semi-Monotonicity

We define a useful property called *global semi-monotonicity* (GSM) in satisfying constraint hierarchies as follows:

Definition 6 (global semi-monotonicity). S is globally semi-monotonic iff for any H and H', $S(H) \cap S(H') \subseteq S(H \cup H')$.

GSM requires that any common solution to two constraint hierarchies is also a solution to their combination. It is not only natural but also weak (or general) in a sense that the condition is true for any two hierarchies sharing no solutions.

GSM, by definition, is not limited to constraint hierarchies. In a similar style, we can express basic properties of constraint systems. For example, we can represent ordinary monotonicity as $S(H) \cap S(H') = S(H \cup H')$, where the difference from GSM is that it has $S(H) \cap S(H') \supseteq S(H \cup H')$. Thus, we can see that GSM lacks the familiar style of the monotonic property, $S(H) \supseteq S(H \cup H')$. (Such a universal style of formal properties is helpful in comparing different nonmonotonic systems.)

We present a useful class of GSM constraint-hierarcy satisfiers called *global constraint-hierarcy satisfiers*, using *global level comparators* and *global constraint-hierarcy comparators*:

Definition 7 (global level comparator). A level comparator $\overset{\cdot/l}{\leq}$ is global iff for any H, H', θ, and θ',

$$\theta \overset{H/l}{<} \theta' \wedge \theta \overset{H'/l}{\sim} \theta' \Rightarrow \theta \overset{H\cup H'/l}{<} \theta' \tag{7}$$

$$\theta \overset{H/l}{\not\sim} \theta' \Rightarrow \neg\, \theta \overset{H\cup H'/l}{\sim} \theta' \tag{8}$$

$$\neg\, \theta \overset{H/l}{<} \theta' \wedge \neg\, \theta \overset{H'/l}{<} \theta' \Rightarrow \neg\, \theta \overset{H\cup H'/l}{<} \theta' \ . \tag{9}$$

Definition 8 (global constraint-hierarcy comparator). A constraint-hierarcy comparator is global iff each level comparator is global.

Definition 9 (global constraint-hierarcy satisfier). A constraint-hierarcy satisfier is global iff its constraint-hierarcy comparator is global.

An example of global level comparators is the least-squares level comparator. Most level comparators presented in the original formulation are also global. The following theorem proves that global satisfiers are GSM:

Theorem 10. *S is GSM if S is global.*

Proof. By contradiction: Assume that there exists a θ that is in $S(H)$ and $S(H')$, but not in $S(H \cup H')$. Then, for some θ', $\theta' \overset{H\cup H'}{<} \theta$ holds, that is, for some l, $(\forall l' \in \boldsymbol{L}.\, l' < l \Rightarrow \theta' \overset{H\cup H'/l'}{\sim} \theta) \wedge \theta' \overset{H\cup H'/l}{<} \theta$. By (7) and (8), $\theta' \overset{H\cup H'/l'}{\sim} \theta$ implies $(\theta' \overset{H/l'}{<} \theta \wedge \theta' \overset{H'/l'}{>} \theta) \vee (\theta' \overset{H/l'}{\sim} \theta \wedge \theta' \overset{H'/l'}{\sim} \theta) \vee (\theta' \overset{H/l'}{>} \theta \wedge \theta' \overset{H'/l'}{<} \theta)$, and by (9), $\theta' \overset{H\cup H'/l}{<} \theta$ implies $\theta' \overset{H/l}{<} \theta \vee \theta' \overset{H'/l}{<} \theta$. Hence, it must be either of the following two cases:

Case $\exists l' \in \boldsymbol{L}.\, l' \leq l \wedge (\forall l'' \in \boldsymbol{L}.\, l'' < l' \Rightarrow \theta' \overset{H/l''}{\sim} \theta \wedge \theta' \overset{H'/l''}{\sim} \theta) \wedge \theta' \overset{H/l'}{<} \theta$. Then, $\theta' \overset{H}{<} \theta$ holds, which is a contradiction to $\theta \in S(H)$.

Case $\exists l' \in \boldsymbol{L}.\, l' \leq l \wedge (\forall l'' \in \boldsymbol{L}.\, l'' < l' \Rightarrow \theta' \overset{H/l''}{\sim} \theta \wedge \theta' \overset{H'/l''}{\sim} \theta) \wedge \theta' \overset{H'/l'}{<} \theta$. Then, $\theta' \overset{H'}{<} \theta$ holds, which is a contradiction to $\theta \in S(H')$. □

The converse, that GSM satisfiers are global, is not true; in fact, we have not found weaker conditions for level comparators that yield a set equivalent to GSM. However, we believe that most useful GSM satisfiers are global.[6]

[6] Actually, we could make the converse true if we strengthened the formulation of constraint hierarchies by allowing only 'modular' hierarchy comparators as follows: let level comparators be in a certain set including the least-squares level comparator, and also let hierarchy comparators need to be arbitrarily composed of level comparators in the set. For modular hierarchy comparators, the truth of the converse is easily provable since we can create a non-GSM satisfier by combining any non-global and the least-squares level comparators. Another set of level comparators without the least-squares level comparator may exist, but is unlikely to be more useful.

Global hierarchy comparators might seem strongly related to globally-better comparators in the original formulation, but in fact, they are different. A globally-better comparator is a hierarchy comparator composed of level comparators that compare reals generated by combining errors of constraints. One instance, least-squares-better, is composed of the least-squares level comparators, and therefore, is global. However, worst-case-better, composed of the worst-case level comparators defined as $\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \max_{c/l \in H} e_l(c,\theta) \leq \max_{c/l \in H} e_l(c,\theta')$, is not global because (7) does not hold. Generally, for level comparators of globally-better comparators, (8) is true since they compare reals, i.e. $\neg\, \theta \overset{H/l}{\not\sim} \theta'$. However, it depends on actual instances of level comparators whether both (7) and (9) hold.

3.3 Generalized Local Propagation

Classical local propagation satisfies a constraint network by successively solving individual constraints in an order closely associated with the network topology. Here we generalize local propagation so that it can solve a set of constraints in one step and can also introduce an arbitrary order among such constraint sets. For this purpose, we introduce *ordered partitions* as follows: a partition of a constraint hierarchy is a set generated by decomposing the hierarchy into disjoint subsets called *blocks*; given a partition P, an ordered partition of P is a pair $\langle P, \leq_P \rangle$, where $\leq_P$ is an arbitrary partial order among blocks in P. For brevity, we write $B <_P B'$ instead of $B \leq_P B' \wedge B \neq B'$.

Using ordered partitions into blocks, we define *generalized local propagation* (GLP) in the following way:

Definition 11 (generalized local propagation). Generalized local propagation with S is a mapping $\pi_S(\langle P, \leq_P \rangle)$ defined as follows:

$$\pi_S(\langle P, \leq_P \rangle) = \begin{cases} \boldsymbol{\Theta} & \text{if } |P| = 0 \\ \bigcap\limits_{B \in terminals(\langle P, \leq_P \rangle)} S(\pi_S(before(\langle P, \leq_P \rangle, B)), B) & \text{otherwise} , \end{cases}$$

where *terminals* and *before* are as follows:

$$terminals(\langle P, \leq_P \rangle) = \{B' \in P \mid \neg\, \exists\, B'' \in P.\, B' <_P B''\}$$

$$before(\langle P, \leq_P \rangle, B) = \langle P', \leq_{P'} \rangle \begin{cases} P' = \{B' \in P \mid B' <_P B\} \\ \leq_{P'} = \{\langle B', B'' \rangle \in P' \times P' \mid B' \leq_P B''\} \end{cases} .$$

Intuitively, $terminals(\langle P, \leq_P \rangle)$ is the set of all blocks at terminal positions, and $before(\langle P, \leq_P \rangle, B)$ is the 'ordered sub-partition' of $\langle P, \leq_P \rangle$, where all blocks are before B. For example, consider the ordered partition $\langle P, \leq_P \rangle$ of the blocks B_1, B_2, ..., B_9, as illustrated in Fig. 1. The partial order $\leq_P$ is defined as the reflexive transitive closure of all the arrows in Fig. 1. Then, $terminals(\langle P, \leq_P \rangle)$ is the set $\{B_8, B_9\}$. Also, $before(\langle P, \leq_P \rangle, B_9)$ is the pair consisting of the set

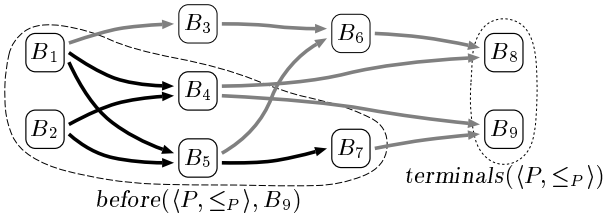

Fig. 1. An ordered partition

$\{B_1, B_2, B_4, B_5, B_7\}$ and the partial order defined as the reflexive transitive closure of the black arrows. Thus, B_9 is satisfied in the set of assignments obtained by applying GLP to blocks before B_9. Accordingly, we can view GLP as a process that successively solves each blocks in some order respecting $\leq_P$. This is always possible because $\leq_P$ is a partial order.

The next lemma shows that by using a global satisfier, GLP respects the similarity of variable assignments for ordered partitions that satisfy the conditions below:

Lemma 12. *Let S be global. Given an arbitrary H, $\langle P, \leq_P \rangle$ of H, and θ in $\pi_S(\langle P, \leq_P \rangle)$, then any θ' is in $\pi_S(\langle P, \leq_P \rangle)$ if $\theta' \overset{H}{\sim} \theta$ and*

$$\forall B \in P. \forall c/l \in B. e_l(c, \theta) > 0 \Rightarrow \forall B' \in P. B' <_P B \Rightarrow \forall c'/l' \in B'. l' < l \ . \quad (10)$$

Proof. By contradiction: Assume that there exists some θ' which is not in $\pi_S(\langle P, \leq_P \rangle)$. Then, it is necessary that for some B_1 in P, θ' is in $\pi_S(before(\langle P, \leq_P \rangle, B_1))$, but not in $S(\pi_S(before(\langle P, \leq_P \rangle, B_1)), B_1)$. Because $\theta' \overset{H}{\sim} \theta$ holds and S is global, $\theta \overset{B_1}{\not\sim} \theta'$ does not hold. Therefore, $\theta \overset{B_1}{<} \theta'$ must hold, that is, there exists some l_1 such that $(\forall l \in \boldsymbol{L}. l < l_1 \Rightarrow \theta \overset{B_1/l}{\sim} \theta') \wedge \theta \overset{B_1/l_1}{<} \theta'$. This implies that for some B in P, $\theta \overset{B/l_1}{>} \theta'$ holds. Since θ must be in $S(\pi_S(before(\langle P, \leq_P \rangle, B)), B)$, it must be either of the following two cases:

Case $\theta' \in \pi_S(before(\langle P, \leq_P \rangle, B)) \wedge \theta' \notin S(\pi_S(before(\langle P, \leq_P \rangle, B)), B)$. Since $\theta \overset{B/l_1}{>} \theta'$ holds, there must exist some l_2 such that $l_2 < l_1$ and $\theta \overset{B/l_2}{<} \theta'$.

Case $\theta' \notin \pi_S(before(\langle P, \leq_P \rangle, B))$. Then, for some B' in P such that $B' <_P B$, θ' is in $\pi_S(before(\langle P, \leq_P \rangle, B'))$, but not in $S(\pi_S(before(\langle P, \leq_P \rangle, B')), B')$. Since $\theta \overset{B/l_1}{>} \theta'$ implies $\exists c/l_1 \in B. e_{l_1}(c, \theta) > 0$, and also since (10) holds, B' contains only stronger constraints than l_1. Therefore, there exists some l_2 such that $l_2 < l_1$ and $\theta \overset{B'/l_2}{<} \theta'$.

Beginning with $\theta \overset{B_1/l_1}{<} \theta'$, both of the two cases resulted in that there exist some l_2 and B_2 such that $l_2 < l_1$ and $\theta \overset{B_2/l_2}{<} \theta'$. Clearly, it causes an infinite

sequence $l_1, l_2, \ldots$ such that $l_i > l_{i+1}$. However, since each l_i is a non-negative integer, it is a contradiction. □

Intuitively, Lemma 12 says that if GLP using a global satisfier generates a variable assignment under which constraints with errors have only stronger constraints before them, then it yields all similar (i.e. $\overset{H}{\sim}$) assignments. Note that the sufficient condition (10) allows constraints without errors to be placed after weaker ones.

In the following theorem, we prove that such variable assignments are solutions to the constraint hierarchy:

Theorem 13. *Let S be global. Given an arbitrary H, $\langle P, \leq_P \rangle$ of H, and θ in $\pi_S(\langle P, \leq_P \rangle)$, then θ is a solution to H if* (10) *holds.*

Proof. By induction on the size of P:

Induction base. If $|P| = 0$, the proposition holds.

Induction step. Assume that if $|P| < n$, the proposition holds. Now, let $|P| = n$. For any B in $\mathit{terminals}(\langle P, \leq_P \rangle)$, θ must be in $S(\pi_S(\mathit{before}(\langle P, \leq_P \rangle, B)), B)$. Therefore, by the induction hypothesis, θ is in $S(H_B)$, where H_B is the union of blocks of $\mathit{before}(\langle P, \leq_P \rangle, B)$. Now, we assume (for contradiction) that there exists some θ' such that $\theta' \overset{H_B \cup B}{<} \theta$, that is, for some l, $(\forall l' \in \boldsymbol{L}. l' < l \Rightarrow \theta' \overset{H_B \cup B/l'}{\sim} \theta) \wedge \theta' \overset{H_B \cup B/l}{<} \theta$. It must be either of the following two cases:

Case $\exists l' \in \boldsymbol{L}. l' \leq l \wedge (\forall l'' \in \boldsymbol{L}. l'' < l' \Rightarrow \theta' \overset{H_B/l''}{\sim} \theta \wedge \theta' \overset{B/l''}{\sim} \theta) \wedge \theta' \overset{H_B/l'}{<} \theta$. Then, $\theta' \overset{H_B}{<} \theta$ holds. Therefore, $\theta \notin S(H_B)$, which is a contradiction.

Case $\exists l' \in \boldsymbol{L}. l' \leq l \wedge (\forall l'' \in \boldsymbol{L}. l'' < l' \Rightarrow \theta' \overset{H_B/l''}{\sim} \theta \wedge \theta' \overset{B/l''}{\sim} \theta) \wedge \theta' \overset{B/l'}{<} \theta$. Then, for some c/l' in B, $e_{l'}(c, \theta) > 0$ must hold. By (10), H_B contains only stronger constraints than l'. Therefore, $\theta' \overset{H_B}{\sim} \theta$ holds. By Lemma 12, θ' is also in $\pi_S(\mathit{before}(\langle P, \leq_P \rangle, B))$. However, since $\theta' \overset{B}{<} \theta$ holds, it implies $\theta \notin S(\pi_S(\mathit{before}(\langle P, \leq_P \rangle, B)), B)$, which is a contradiction.

Both cases caused contradiction. Therefore, there never exists such θ', i.e. θ is in $S(H_B \cup B)$. Since S is global, θ is also in $S(H)$ by Theorem 10. □

The theorem presents a strategy to design algorithms for solving constraint hierarchies. As noted, the sufficient condition permits constraints without errors to be located after weaker ones. In other words, we can delay the satisfaction of a strong constraint with no error until some appropriate time, for example, "when the constraint becomes uniquely satisfiable." Actually, Theorem 13 gracefully explains why the *DETAIL* algorithm obtains solutions by using local propagation, which we will describe in Sect. 4.

An important instance of such GLP is the refining method. Since constraints have no weaker constraints before them in the method, it can be easily understood by Theorem 13 that it generates only correct solutions, i.e. is sound. In addition, using a certain kind of global satisfiers, the refining method yields all solutions, i.e. is complete:

Proposition 14. *Let S be global such that for any H, θ, and θ', $\neg\, \theta \overset{H}{\not\sim} \theta'$. For any H and $\langle P, \leq_P \rangle$ of H, $\pi_S(\langle P, \leq_P \rangle) = S(H)$ if $P = \{B_l \mid l \in \boldsymbol{L}\}$ and $B_l \leq_P B_{l'} \Leftrightarrow l \leq l'$, where B_l is the level l of H.*

By this proposition, a refining-method algorithm using a global hierarchy and globally-better comparator, e.g. least-squares-better, is sound and complete, because any globally-better comparator satisfies $\neg\, \theta \overset{H}{\not\sim} \theta'$.[7]

Next, we define *local level comparators*, *local constraint-hierarcy comparators*, and *local constraint-hierarcy satisfiers*:

Definition 15 (local level comparator). A level comparator $\overset{\cdot/l}{\leq}$ is local iff for any H, θ, and θ', (11) $\theta \overset{H/l}{\leq} \theta' \Rightarrow \forall c/l \in H.\, e_l(c, \theta) \leq e_l(c, \theta')$.

Definition 16 (local constraint-hierarcy comparator). A constraint-hierarcy comparator is local iff each level comparator is local.

Definition 17 (local constraint-hierarcy satisfier). A constraint-hierarcy satisfier is local iff its constraint-hierarcy comparator is local.

By (4) and (11), a local level comparator results in $\theta \overset{H/l}{\leq} \theta' \Leftrightarrow \forall c/l \in H.\, e_l(c, \theta) \leq e_l(c, \theta')$, which is equivalent to level comparators of locally-better comparators in the original formalization. With additional restrictions on multi-way equality constraints, we can regard our formulation as a theoretical basis of efficient constraint-hierarcy satisfaction algorithms such as DeltaBlue.

The following proposition indicates a critical difference between global hierarchy and globally-better comparators:

Proposition 18. *Any local constraint-hierarcy comparator is global.*

The original formulation presented locally-better and globally-better as separate concepts. However, we successfully integrated locally-better and an important class of globally-better into global hierarchy comparators via GSM.

Using a local satisfier, we can obtain a theorem with a weaker sufficient condition than that of Theorem 13:

Theorem 19. *Let S be local. Given an arbitrary H, $\langle P, \leq_P \rangle$ of H, and θ in $\pi_S(\langle P, \leq_P \rangle)$, then θ is a solution to H if*

$$\forall B \in P. \forall c/l \in B.\, e_l(c, \theta) > 0 \Rightarrow \forall B' \in P.\, B' <_P B \Rightarrow \forall c'/l' \in B'.\, l' \leq l \ . \quad (12)$$

The difference of (12) from (10) is the existence of equality in $l' \leq l$, which indicates that (12) is weaker than (10). Since it will provide more freedom to organize ordered partitions, we can expect to develop more efficient constraint solving algorithms using local satisfiers. For example, we can regard the blocked constraint lemma presented in the DeltaBlue paper [4] as a specialization of Theorem 19.

[7] It is probably possible to weaken the sufficient conditions for level comparators since the condition for ordered partitions is too strong in the refining method.

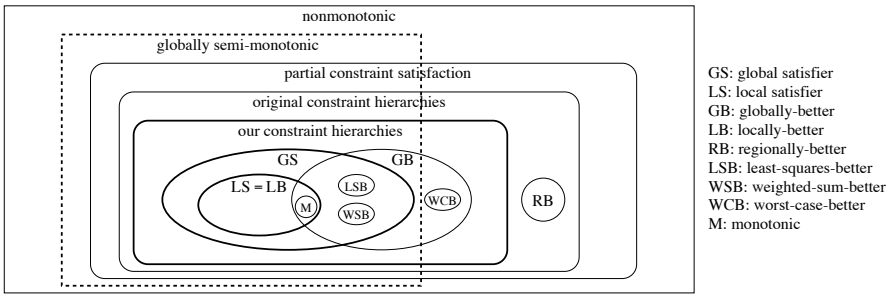

Fig. 2. Relationship of nonmonotonic constraint systems

3.4 Discussion

In this subsection, we review the relationship among nonmonotonic constraint systems, which is roughly illustrated in Fig. 2.

Partial constraint satisfaction [5] is a considerably general theory. Therefore, it will include various nonmonotonic systems, which are not necessarily efficiently solvable.

Our reformulation of constraint hierarchies has become narrower than the original one,[8] because we necessitated $\overset{\cdot/l}{\sim}$ to be transitive by (5). For example, we exclude regionally-better in [11] since its level comparator is defined as $\theta \overset{H/l}{<} \theta' \Leftrightarrow \forall c/l \in H.\, e_l(c,\theta) \leq e_l(c,\theta') \wedge \exists c/l \in H.\, e_l(c,\theta) < e_l(c,\theta')$ and $\theta \overset{H/l}{\sim} \theta' \Leftrightarrow \neg\theta \overset{H/l}{<} \theta' \wedge \neg\theta \overset{H/l}{>} \theta'$, where $\overset{H/l}{\sim}$ is not transitive. However, excluding such level comparators contributed to theoretical cleanness and development of generalized local propagation.

It is important to find an expressive and efficiently solvable class of nonmonotonic constraint systems. Except regionally-better and worst-case-better, all the hierarchy comparators presented in the original formulation are global by our formulation. We believe that this fact supports the expressiveness of our global satisfiers with respect to constraint hierarchies. Also, we claim that Theorem 13 for global satisfiers and Theorem 19 for local satisfiers are useful in designing efficient constraint satisfaction algorithms.

4 The *DETAIL* Algorithm

To show how to employ the results in the last section, we relate them with the *DETAIL* algorithm, which we proposed in [6]. *DETAIL* is an incremental algorithm for solving constraint hierarchies based on local propagation. It always

[8] Strictly speaking, as noted earlier, our theory allows conflicting constraints at the top level, while the original theory restricts top-level constraints to be required.

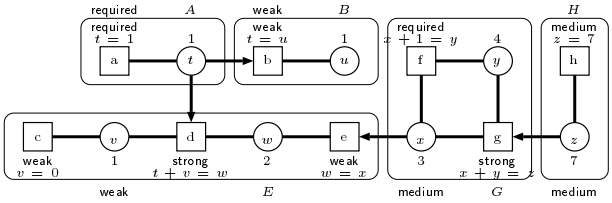

Fig. 3. A configuration of constraint cells

stores planning data instead of an appropriate ordered partition of the current hierarchy, and modifies the plan if a constraint is added to or removed from the hierarchy.

DETAIL handles multi-way equality constraints extended so that it can simultaneously satisfy or properly relax them, in addition to solving them individually as is with classical local propagation. To process such constraints, *DETAIL* maintains a set of *constraint cells* instead of an ordered partition into blocks. A constraint cell can be regarded as a block including output variables, where the constraints in the block are uniquely solved for the output variables. Also, it never shares variables with any other cells. For example, to solve the constraint strong $x + y = 3$ for variable x, *DETAIL* yields a cell of strong $x + y = 3$ and x. By contrast, to simultaneously solve strong $x + y = 3$ and weak $x - y = 1$, it generates a cell of the two constraints and the variables x and y. Similarly, to relax strong $x = 0$ and strong $x = 1$, it produces a cell consisting of the two constraints and x.[9] *DETAIL* solves such constraint cells with pluggable numerical modules called *subsolvers* using e.g. Gaussian elimination.

By the definition of constraint cells, we can determine dependency among cells. Additionally, if we prohibit cyclic dependency, we can naturally identify the overall dependency among cells with a partial order among blocks. Then, we can perform GLP in a 'unique' manner as is with conventional local propagation. For example, consider the hierarchy with the constraints [a], [b], ..., [h] in Fig. 3, where the squares and circles represent constraints and variables respectively, and the boxes with round corners indicate cells. Clearly, in the order respecting the cell dependencies, such as A, B, H, G, and E, we can uniquely solve constraints in each cell.

The other issue is how to determine configurations of cells that obtain correct solutions. To guarantee the sufficient condition (10) for Theorem 13, we employed *walkabout strengths*, which had been first introduced in DeltaBlue [4]. In *DETAIL*, walkabout strengths, associated with constraint cells, are defined to propagate strengths of the weakest constraints. For example, in Fig. 3, the walkabout strength medium of cell G is inherited from the weakest constraint [h]

[9] SkyBlue also realizes simultaneous satisfaction by calling 'cycle solvers,' but provides no features for relaxing constraints [9].

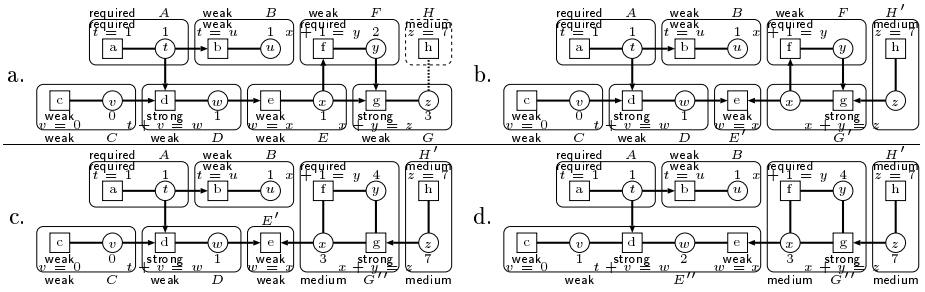

Fig. 4. Adding a constraint to a hierarchy of equalities

among all the constraints that the variables in G depend on, i.e. [f], [g], and [h]. Therefore, the walkabout strength of a cell indicates that there are only constraints with equal or stronger strengths before/in the cell. Thus, it can be easily verified whether a configuration of cells satisfies the sufficient condition for Theorem 13. For example, in Fig. 3, although the weak constraints [c] and [e] in E have positive errors, the walkabout strengths required and medium of the preceding cells A and G indicate that all the forward constraints are stronger than weak.

Now, we demonstrate the *DETAIL* algorithm by example. Figure 4a illustrates the initial configuration of cells, and suppose that we add a new constraint [h] medium $z = 7$ to it. The current solution $z = 3$ conflicts with [h], and the walkabout strength weak of G shows that there is one or more weak constraints in or before G. Therefore, we must change the configuration in the following steps:

1. First, move along the path from the new cell to the nearest source of the walkabout strength, i.e. from H to E, reversing the dependency between them, as shown in Fig. 4b. Note the multi-way equality property of constraints always enables us to perform the reversing operation [6].
2. Next, merge cyclic dependencies generated from the previous step if any. In the example, we collapse the cycle of G' and F as illustrated in Fig. 4c.
3. Third, check whether the victimized cell E' has any preceding cells with the same walkabout strength weak. Figure 4c shows that D is such a cell. Since it violates the sufficient condition for generating solutions, merge all the transitively adjacent cells with the same walkabout strength, i.e. E', D, and C (but not B). Then, we obtain the final configuration in Fig. 4d.

In step 3, we merged all the transitively adjacent cells with the same walkabout strength to ensure the sufficient condition (10) for global satisfiers. However, if we use a local satisfier, we only need to guarantee the weaker condition (12), and therefore, we can omit step 3 (the final configuration would have been Fig. 4c). *DETAIL* also provides the support for local satisfiers, which usually results in smaller constraint cells that can be solved more efficiently.

5 Conclusions and Status

We reformulated the definition of constraint hierarchies, and proposed generalized local propagation to theoretically study local propagation therein. We showed that globally semi-monotonic satisfaction of hierarchies exhibits a practically useful property for generalized local propagation.

By applying the results, we are extending the *DETAIL* algorithm to handle 'multi-way inequality constraints.' We already established its basis, and actually implemented a prototype constraint solver. Due to the existence of inequalities, the new algorithm is exponential in time complexity unlike the original *DETAIL*, which is polynomial. Therefore, we are mainly exploring performance techniques such as efficient scheduling and pruning of constraints.

References

1. Borning, A., R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf, "Constraint Hierarchies," in *OOPSLA'87*, ACM, Oct. 1987, pp. 48–60.
2. Borning, A., B. Freeman-Benson, and M. Wilson, "Constraint Hierarchies," *Lisp and Symbolic Computation*, vol. 5, 1992, pp. 221–268.
3. Freeman-Benson, B. N. and A. Borning, "Integrating Constraints with an Object-Oriented Language," in *ECOOP'92*, no. 615 in LNCS, Springer-Verlag, June/July 1992, pp. 268–286.
4. Freeman-Benson, B. N., J. Maloney, and A. Borning, "An Incremental Constraint Solver," *Comm. ACM*, vol. 33, no. 1, Jan. 1990, pp. 54–63.
5. Freuder, E. C. and R. J. Wallace, "Partial Constraint Satisfaction," *Artificial Intelligence*, vol. 58, 1992, pp. 21–70.
6. Hosobe, H., K. Miyashita, S. Takahashi, S. Matsuoka, and A. Yonezawa, "Locally Simultaneous Constraint Satisfaction," in *PPCP'94*, no. 874 in LNCS, Springer-Verlag, Oct. 1994, pp. 51–62.
7. Jampel, M., "A Compositional Theory of Constraint Hierarchies (Operational Semantics)," in *Proc. Workshop on Over-Constrained Systems at CP'95*, Sept. 1995.
8. Maloney, J. H., A. Borning, and B. N. Freeman-Benson, "Constraint Technology for User-Interface Construction in ThingLab II," in *OOPSLA'89*, ACM, Oct. 1989, pp. 381–388.
9. Sannella, M., "SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction," in *UIST'94*, ACM, Nov. 1994, pp. 137–146.
10. Satoh, K. and A. Aiba, "Computing Soft Constraints by Hierarchical Constraint Logic Programming," Tech. Rep. TR-610, ICOT, Japan, Jan. 1991.
11. Wilson, M., "Hierarchical Constraint Logic Programming (Ph.D. Dissertation)," Tech. Rep. 93-05-01, Dept. of Computer Science and Engineering, University of Washington, May 1993.
12. Wilson, M. and A. Borning, "Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison," in *Proc. North American Conference on Logic Programming*, 1989.
13. Wilson, M. and A. Borning, "Hierarchical Constraint Logic Programming," *J. Logic Programming*, vol. 16, no. 3/4, July/Aug. 1993, pp. 277–319.