Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 238 (2009) 189–205
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Which Soft Constraints do you Prefer?

Matthias Hölzl, Max Meier, Martin Wirsing [1,2]

Institut für Informatik
Ludwig-Maximilians-Universität
München, Germany

Abstract

Soft constraints are gaining popularity in diverse areas such as orchestration of Web services or optimization of scheduling decisions. However, current approaches to soft constraints preclude them from modelling certain decision problems with multiple preference criteria. We propose a new approach to soft constraints which allows a natural expression of these problems, describe an implementation in the rewriting logic system Maude, and prove its correctness.

Keywords: Soft Constraints, Rewriting Logic

1 Introduction

Soft constraints are gaining popularity in diverse areas such as orchestration of Web services (see e.g. [13]) or optimization of scheduling decisions (see e.g. [18]). However, current approaches to soft constraints preclude them from modelling certain decision problems with multiple preference criteria.

As an example for this phenomenon consider the meeting planning component of a university management system. There are both mandatory and optional participants, which are able to attend the meeting only on certain dates and places. In a real system, this information could be collected automatically from the persons' calendars. To successfully schedule a meeting, all mandatory persons and a meeting room of the appropriate size have to be available. Among all the dates satisfying these conditions, the scheduling service has to choose one that optimizes certain other criteria, e.g. the number of optional participants. If possible, the meeting should not be scheduled on a weekend. Many more criteria are imaginable, for example each participant could give a personal rating on dates and places.

[1] This work has been partially sponsored by the project SENSORIA, IST-2005-016004.
[2] Email: firstname.lastname@ifi.lmu.de

doi:10.1016/j.entcs.2009.05.020

We will argue that current formalisms for expressing soft-constraint problems, e.g., semiring-based soft constraint satisfaction problems (SCSPs) are not adequate to model the above situation under certain reasonable assumptions about the importance of the preferences. We propose a generalized soft-constraint formalism, Monoidal Soft Constraints, that can express these kinds of problems and describe an implementation (in the Maude rewriting logic) of a branch-and-bound algorithm to solve these generalized constraint problems. We have applied the formalism and solver to several application domains, e.g., to orchestration of service-oriented architectures [20] and to the optimization of software-defined radio networks (for an earlier approach see [21]). Our current implementation of the solver is based on a predecessor which was written in Maude and integrated into the Pagoda framework for modelling SDRs. Building on this foundation reduced our implementation effort and allows us to easily integrate the current solver into the Pagoda system.

The structure of the paper is as follows: In Section 2 we introduce the theory of Monoidal Soft Constraints. We describe the transformations that we perform on constraint systems before submitting them to the solver in Section 3. In Section 4 we describe the branch-and-bound algorithm and prove its correctness. The final two sections present related work and our conclusions.

2 Soft Constraints over Partially-Ordered Monoids

2.1 C-Systems of Monoids

A *monoid* $\mathcal{M}$ is an algebra $(M, \otimes, 1)$ with carrier set M, a constant $1 \in M$ and an associative binary operation $\otimes$ with identity element 1. If $\otimes$ is commutative $\mathcal{M}$ is called a commutative monoid.

An *ordered monoid* $\mathcal{M}$ is an algebra $(M, \otimes, \leq, 1)$ with carrier set M, a constant $1 \in M$, a binary operation $\otimes$ and a binary relation $\leq$ such that $(M, \otimes, 1)$ is a commutative monoid and $\leq$ is a partial order which is compatible with $\otimes$, i.e., $\forall x, y, z \in M. x \leq y \implies x \otimes z \leq y \otimes z$.

Examples for ordered monoids are the natural numbers with addition or multiplication and the usual comparison operation:

$$Nat_{add} = (\mathbb{N}, +, \leq, 0)$$
$$Nat_{mult} = (\mathbb{N}, \cdot, \leq, 1).$$

Another frequently used ordered monoid is the monoid of natural numbers with reverse order $Nat\,^{\geq}_{add}$. Its carrier is the set of non-negative integers extended with an "infinite" element ∞, the multiplication operation is addition and the order relation is the $\geq_{\mathbb{N}}$ relation on the natural numbers extended with $\infty \geq_{\mathbb{N}} n$ for all $n \in \mathbb{N}$:

$$Nat\,^{\geq}_{add} = \big(\mathbb{N} \cup \{\infty\}, +, \geq, 0\big).$$

To express crisp constraints the ordered monoid of Boolean values is often useful: its carrier is the set consisting of the two values **true** and **false**, the multiplication operation corresponds to conjunction; the order relation is generated by the relation

false $<$ **true** and coincides with implication:

$$Bool = (\{\mathbf{true}, \mathbf{false}\}, \wedge, \rightarrow, \mathbf{true}).$$

Let $\mathcal{M} = (M, \otimes, 1)$ be a monoid and E a set. We write $\mathbf{Id}_E$ for the identity mapping on E: $\mathbf{Id}_E : E \rightarrow E, x \mapsto x$. A map $p : M \rightarrow (E \rightarrow E)$ is called a *(left) operation of M on E* if $p_1 = \mathbf{Id}_E$ and $p_{\alpha\otimes\beta} = p_\alpha \circ p_\beta$. We often write αx for $p_\alpha(x)$; the previous conditions can then be written as

$$1x = x \tag{1}$$

$$(\alpha \otimes \beta)x = \alpha(\beta x). \tag{2}$$

Let E be an ordered set. Then we call p *left-compatible* (with the orders of $\mathcal{M}$ and E) if $\forall \alpha, \beta \in M, x \in E.\alpha \leq \beta \implies \alpha x \leq \beta x$. We call the operation p *intensive* if $\forall \alpha \in M, x \in E.\alpha x \leq x$. It follows immediately from these two definitions and (1) that for an intensive left-compatible operation $1x$ is a maximal element of $\{\beta x \mid \beta \in M\}$ for every $x \in E$.

We call p a *c-operation* of an ordered monoid $\mathcal{M}$ on an ordered set E if p is an left-compatible operation of $\mathcal{M}$ on E. In this case we also call $\mathcal{M}$ a *c-monoid* over E. If p is an intensive c-operation we call it an *ic-operation* and $\mathcal{M}$ an *ic-monoid* over E.

Note 1 *The use of intensive c-operations enables many additional techniques for solving constraint problems but modelling with ic-monoids is often cumbersome for practical applications. In Section 4 we describe a branch-and-bound algorithm for intensive monoids. This algorithm can be modified to work for non-intensive problems by taking into account the possible improvements of partial solutions by constraints defined over non-intensive monoids.*

Let $\mathcal{E} = (E, \leq_E)$ be an ordered set, I a set, $(\mathcal{M}_i)_{i\in I} = \big((M_i, \otimes_i, \leq_i, 1_i)\big)_{i\in I}$ a family of ordered monoids, and $(p^i)_{i\in I}$ a family of c-operations, $p^i : M_i \rightarrow (E \rightarrow E)$. If $p^i_{\alpha_i}$ and $p^j_{\alpha_j}$ commute (with respect to function composition) for all $\alpha_i \in M_i$, $\alpha_j \in M_j$, i.e., if

$$\forall i, j \in I, x \in E.\forall \alpha_i \in M_i, \alpha_j \in M_j.\alpha_i(\alpha_j x) = \alpha_j(\alpha_i x)$$

we call $\big\langle(\mathcal{M}_i); (p^i); E\big\rangle$ a *c-system* of monoids (over E). If all (p^i) are ic-operations then we call $\big\langle(\mathcal{M}_i); (p^i); E\big\rangle$ an *ic-system* of monoids (over E).

We write inj_i for the injection into the i-th component and π_i for the projection of the i-th component of a cartesian product. Let $(M_i)_{i\in I}$ be a family of monoids and $E = \prod_i M_i$. We then define the *canonical injective multiplication $imult_i$* as

$$\begin{aligned} imult_i : M_i &\rightarrow (E \rightarrow E) \\ \alpha &\mapsto (x \mapsto inj_i(\alpha \otimes_i \pi_i(x))) \end{aligned}$$

For example, let M_1, M_2 be ordered monoids, $E = M_1 \times M_2$, let $\leq_\times$ be the pointwise ordering on E and $p^i = imult_i$. Then $\big\langle(M_1, M_2); (p^1, p^2); (M_1 \times M_2, \leq_\times)\big\rangle$

is a c-system of monoids. The same holds if we equip E with the lexicographic order $\leq_{\text{lex}}$.

To summarize, a c-system of monoids is a triple $\big((\mathcal{M}_i)_{i\in I}, (p^i)_{i\in I}; E\big)$ where $(\mathcal{M}_i)$ is a family of ordered monoids and each p^i is an intensive left-compatible operation of $(\mathcal{M}_i)$ on E, i.e., the following properties hold:

$$\begin{aligned} 1^i x &= x \\ (\alpha^i \otimes \beta^i)x &= \alpha^i(\beta^i x) \\ \alpha^i \leq \beta^i &\implies \alpha^i x \leq \beta^i x \qquad (3) \\ \alpha_i(\alpha_j x) &= \alpha_j(\alpha_i x) \qquad (4) \end{aligned}$$

An ic-system of monoids satisfies the additional property

$$\alpha^i x \leq x \qquad (5)$$

2.2 Soft Constraints

A soft constraint assigns grades to different valuations of a set of problem variables. In a soft-constraint problem the grades of all soft constraints are combined by a preference relation into the rank, which measures the quality of the result.

Let R (the set of ranks) be an ordered set, $\mathcal{G}$ (the set of *grades*) a c-monoid with carrier set G over R, D a finite problem domain, and Var the set of all problem variables. A *valuation* $v : \mathit{Var} \to D$ is a partial map from Var to D which has a finite support. If a valuation is defined for all elements of a set $V \subseteq Var$ we call it a valuation over V and also regard it as a function $V \to D$. Given a finite set $V \subseteq \mathit{Var}$ of variables, a soft constraint assigns a grade in G to each possible valuation over V; more formally, a *soft constraint* over $\mathcal{G}$ is a triple $\langle V; \mathit{cst}; \mathcal{G}\rangle$ where the function $\mathit{cst} : (V \to D) \to G$ maps valuations over V to elements of G. As V is finite, any soft constraint has a finite domain of definition and can therefore be represented either in an *explicit* way by a finite set of pairs $(v \mapsto g)$ with $v \in (V \to D)$ and $g \in G$, or in a more *implicit* way by particular constructors or function declarations in a programming language. In our framework we support both possibilities; here we present only the explicit form. For fixed V of size k, the type $(V \to D) \to G$ is isomorphic to $D^k \to G$ and thus any constraint definition can be written as a map from D^k to G. For example, for a constraint with $V = V_1, V_2$ we write cst in the form $(v_1, v_2) \mapsto v_1 + v_2$ instead of $v \mapsto v(V_1) + v(V_2)$. For explicit constraints we sometimes write the mapping in the form of (domain values $\mapsto$ grade) pairs, e.g., $(\mathit{Monday} \mapsto 0)(\mathit{Tuesday} \mapsto 1)$ denotes the constraint mapping the domain value *Monday* to the grade 0 and the domain value *Tuesday* to the grade 1. If c is a soft constraint of the form $\langle V; \mathit{cst}; \mathcal{G}\rangle$ we sometimes write $c(x_1, \ldots, x_n)$ instead of $\mathit{cst}(x_1, \ldots, x_n)$.

In practice a soft constraint is often defined as a combination of several simpler soft constraints: given two soft constraints $c_1 = \langle V_1; \mathit{cst}_1; \mathcal{G}\rangle$ and $c_2 = \langle V_2; \mathit{cst}_2; \mathcal{G}\rangle$ over the same monoid of grades $\mathcal{G}$, their combination $c_1 \otimes c_2$ is the constraint $\langle V; \mathit{cst}; \mathcal{G}\rangle$ defined by $V = V_1 \cup V_2$ and $\mathit{cst}(t) = \mathit{cst}_1(t \downarrow^{V}_{V_1}) \otimes \mathit{cst}_2(t \downarrow^{V}_{V_2})$, where $t \downarrow^{X}_{Y}$

denotes the tuple of values over the variables in Y, obtained by projecting tuple t from X to Y.

In the example of the meeting scheduling service, the monoid of grades $\mathcal{G}_1$ is used to express the preferences of each key person for meeting dates and places. We limit the values for each person to an interval $[0, \mathit{max\text{-}pref}\,]$ so that preferences of different persons can be compared. By choosing the multiplicative monoid of natural numbers $\mathit{Nat}_{\mathit{mult}}$ to evaluate the results we can ensure that the inability of a single key person to attend the meeting leads to an "unacceptably bad" grade 0 for the meeting. The combination of the constraints for all key persons measures the desirability of dates and places for scheduling the meeting.

$$\begin{aligned}\mathit{key\text{-}person}_i &= \langle \mathit{Date}, \mathit{Place}; \mathit{cst}^k_i; \mathit{Nat}_{\mathit{mult}}\rangle\\ \mathit{key\text{-}persons} &= \bigotimes_i \mathit{key\text{-}person}_i\end{aligned}$$

Similarly, we define a constraint expressing preferences for certain days and places for each other person, again with 0 meaning the person is unavailable and higher number meaning preferred dates and places. However, here we do not want the unavailability of a non-key person to reduce the overall evaluation of a meeting opportunity to 0. Therefore we combine these constraints in the additive monoid of natural numbers:

$$\begin{aligned}\mathit{opt\text{-}person}_i &= \langle \mathit{Date}, \mathit{Place}; \mathit{cst}^o_i; \mathit{Nat}_{\mathit{add}}\rangle\\ \mathit{opt\text{-}persons} &= \bigotimes_i \mathit{opt\text{-}person}_i\end{aligned}$$

Another constraint "*not-on-weekend*" expresses that the meeting should not be scheduled on a weekend: $\mathit{not\text{-}on\text{-}weekend} = \langle \mathit{Date}; \mathit{cst}^w; \mathit{Bool}\rangle$.

2.3 Soft-Constraint Problems

A monoidal soft-constraint problem combines the grades of several soft constraints into a single *rank* which measures the overall quality of a solution. We use a family of c-operations, called the preference relation to specify this combination process. While the grades represent the quality of individual constraints the preference relation expresses the importance we assign to the individual constraints.

Formally, a soft-constraint problem $\mathcal{C}$ is defined as a quintuple $\langle(\mathcal{G}_i)_{i\in I}, (c_i)_{i\in I}, (p^i)_{i\in I}, R, \mathcal{I}\rangle$, where $(\mathcal{G}_i)$ is a family of monoids, each c_i is a soft constraint over $\mathcal{G}_i$, $\mathcal{I} \in R$ is the initial value, and $\big((\mathcal{G}_i), (p^i), R\big)$ is a c-system of monoids. We call the family (p^i) the *preference relation.*

For each valuation $v \in Var \to D$ the *rank of the constraint problem $\mathcal{C}$ for v* is defined as the function composition of all p^i applied to the initial value:

$$S(v) = \big(\bigcirc_{i\in I} p^i_{c_i(v)}\big)(\mathcal{I}), \tag{6}$$

i.e., if $I = \{1, \ldots, n\}$ we have $S(v) = p^1_{c_1(v)}(p^2_{c_2(v)}(\cdots(p^n_{c_n(v)}(\mathcal{I}))\cdots))$. Properties

(2) and (4) together with the commutativity of the operations on $\mathcal{G}_i$ ensure that S does not depend on the order in which the individual constraints are applied.

We define the set of *maximal solutions* of a constraint problem $\mathcal{C}$ in the following way:

$$maxSol(\mathcal{C}) = \big\{(v \mapsto S(v)) \mid S(v) \text{ maximal in } S[Var \to D]\big\}.$$

The scheduling example demonstrates the usefulness of the preference relation in the computation of the rank: a meeting may only be scheduled at times when all mandatory persons are available, or more generally, only times and places in which the availability of mandatory persons is maximal should be considered. The other constraints can only be used to influence the quality of these maximal solutions but even very high grades for the other constraints cannot offset the negative impact of a missing key person. This situation can be modelled, e.g., by defining R as a lexicographically ordered product, where the preference relation injects the grade for availability of key-persons into the first component and all other grades into the second component. Suppose that $R = \mathbb{N}_{\leq} \times_{lex} (\mathbb{N}_{\leq} \times Bool_{\leq})$ with the grade of *key-persons* in the first component, the grade of *opt-persons* in the second component and the grade of *not-on-weekend* in the third component. Then, if we have ranks $(8, (1, \textbf{false}))$, $(7, (100, \textbf{false}))$ and $(7, (5, \textbf{true}))$ the lexicographic order ensures that the first rank, $(8, (1, \textbf{false}))$, is the only maximum since the value of its first component, 8, dominates the value 7 of the other ranks. The second and third rank are not comparable: they both have the same value 7 for the first component, so the pairs of second and third component are used to decide the order. Since these pairs are compared by the pointwise order and $100 > 5$ but $\textbf{false} < \textbf{true}$ the pairs are not comparable.

We can therefore express the scheduling service with the soft-constraint problem $\mathcal{C} = \langle (\mathcal{G}_i)_{i \in I}, (c_i)_{i \in I}, (p^i)_{i \in I}, R, \mathcal{I} \rangle$ with

$$\begin{aligned}
(\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3) &= (Nat_{mult}, Nat_{add}, Bool) \\
(c_1, c_2, c_3) &= (\textit{key-persons}, \textit{opt-persons}, \textit{not-on-weekend}\ ) \\
p^i &= imult_i \\
R &= \mathbb{N}_{\leq} \times_{lex} (\mathbb{N}_{\leq} \times Bool_{\leq}) \\
I &= (1, 0, true)
\end{aligned}$$

$\mathbb{N}_{\leq}$ denotes the set of natural numbers together with their usual order; $Bool_{\leq}$ denotes the set of Boolean values $\{\textbf{true}, \textbf{false}\}$ together with the order $\to$, i.e., the order generated by $\textbf{false} < \textbf{true}$. This model satisfies the property that whenever there is a date and place where all key-persons can attend the meeting, no meeting dates and places where key people are missing will be considered as solution.

Suppose that we have three possible dates, d_1, d_2, d_3, only one possible location l, two key persons with the mappings

$$\begin{aligned}
cst_1^k &= (d_1, l \mapsto 2)(d_2, l \mapsto 0)(d_3, l \mapsto 4) \\
cst_2^k &= (d_1, l \mapsto 1)(d_2, l \mapsto 5)(d_3, l \mapsto 0)
\end{aligned}$$

and 5 optional members, all with the same function

$$cst_i^o = (d_1, l \mapsto 0)(d_2, l \mapsto 5)(d_3, l \mapsto 3)$$

and a function $cst^w = (d_1, \mathit{false})(d_2, \mathit{true})(d_3, \mathit{true})$. Then the ranks of the constraint problem for the different valuations are

$$\begin{aligned} S((\mathit{Date} \mapsto d_1)) &= (2, (0, \mathit{false})) \\ S((\mathit{Date} \mapsto d_2)) &= (0, (25, \mathit{true})) \\ S((\mathit{Date} \mapsto d_3)) &= (0, (15, \mathit{true})) \end{aligned}$$

(we ignore the constant assignment $(\mathit{Place} \mapsto l)$ in the notation for the valuation). Hence, $\mathit{maxSol}(\mathcal{C}) = \{(\mathit{Date} \mapsto d_1) \mapsto (2, (0, \mathit{false}))\}$. Here the lexicographic order ensures that the date where both key persons can attend is chosen, even though this means that no optional member can attend the meeting, and the meeting takes place on a weekend.

The preceding model of the scheduling problem makes use of non-intensive operations. We show that the problem can be modelled using only ic-monoids as well. To achieve this we have to modify the *key-person*$_i$ and *opt-person*$_i$ constraints by defining them as penalties instead of preferences and modify R:

$$\begin{aligned} \mathit{opt\text{-}person}_i' &= \langle \mathit{Date}, \mathit{Place}; \mathit{pcst}_i^o; \mathit{Nat}\ {}^{\geq}_{\mathit{add}} \rangle \\ R &= \mathbb{N}^{\infty}_{\geq} \times_{\mathit{lex}} (\mathbb{N}^{\infty}_{\geq} \times \mathit{Bool}_{\leq}) \end{aligned}$$

Here the function pcst_i^o denotes the penalty for each date and location, not the preference. It can be defined by $\mathit{pcst}_i^o(d, l) = \mathit{max\text{-}pref} - \mathit{cst}_i^o(d, l)$.

We can model the constraints for key persons in the same way, by assigning penalties in the monoid $\mathit{Nat}\ {}^{\geq}_{\mathit{add}}$ instead of preferences in $\mathit{Nat}_{\mathit{mult}}$. Dates and locations where a key person is not available receive the value ∞. In this model, as in all models based on ic-monoids, the initial value $\mathcal{I}$ is the greatest possible value, i.e., $\mathcal{I} = (0, 0, \mathit{true})$.

3 Towards an Implementation

The soft-constraint theory described in the last section is very expressive and allows the modelling of many soft-constraint problems in a natural manner. However it is parametric in a variable number of monoids and makes extensive use of higher-order functions. Both factors complicate the implementation of the theory in a first-order rewriting logic framework. In many cases it is possible to transform soft-constraint problems into a form that is more amenable to an implementation in Maude: When the grades are totally ordered and the set of ranks R can be equipped with the structure of a commutative monoid that is compatible with the preference relation, we can precompute the composition of the constraints and the preference relation and thereby simplify the computation the constraint solver has to perform.

Let $\mathcal{C} = \langle (\mathcal{G}_i)_{i\in I}, (c_i)_{i\in I}, (p^i)_{i\in I}, R, I\rangle$ be a soft-constraint problem with totally ordered grades $\mathcal{G}_i$, and let R be equipped with the structure of a commutative monoid $\mathcal{R} = (R, \otimes_{\mathcal{R}}, 1)$. If $\mathcal{R}$ satisfies the relation

$$p^i_\alpha(x) = p^i_\alpha(1) \otimes_{\mathcal{R}} x \tag{7}$$

we say that $\otimes_{\mathcal{R}}$ is *compatible* with the preference relation $(p^i)_{i\in I}$. In this case, we can define for every soft constraint $c_i = \langle V; cst; \mathcal{G}\rangle$ the flattened constraint

$$c'_i = \langle V; (\alpha \mapsto p^i_\alpha(1)) \circ cst\rangle. \tag{8}$$

The *flattened constraint problem* corresponding to $\mathcal{C}$ can then be defined as $\langle (c'_i)_{i\in I}; \mathcal{R}; I\rangle$. In order to be able to distinguish between $\mathcal{G}$ and $\mathcal{R}$ we call $\mathcal{R}$ the rank of c'_i and $\mathcal{G}$ the the original monoid or grade of c'_i.

Note that in general the algebra $\langle R; \otimes_{\mathcal{R}}; \leq; 1\rangle$ is not an ordered monoid since multiplication is not monotone. For example, if $R = \mathbb{Z} \times \mathbb{Z}$ with the lexicographic order $\leq_{\text{lex}}$ and $\otimes_{\mathcal{R}}$ the elementwise minimization, then we have

$$\begin{gathered}(2,3) \leq_{\text{lex}} (3,1)\\ (2,2) = (2,3) \otimes_{\mathcal{R}} (2,2) \geq_{\text{lex}} (3,1) \otimes_{\mathcal{R}} (2,2) = (2,1)\end{gathered}$$

But for compatible relations there is a weaker monotonicity condition that is sufficient to establish the correctness of branch-and-bound search: Let

$$B_i = (\alpha \mapsto p^i_\alpha(1))[G_i]$$

the image of p^i under all possible valuations for the rank value 1. Then the following property, which we call *semi-monotonicity*, holds for all $i \in I$:

$$\forall x, y \in B_i . \forall z \in R . x \leq y \implies x \otimes_{\mathcal{R}} z \leq y \otimes_{\mathcal{R}} z. \tag{9}$$

Proof. Let $x, y \in B_i$ with $x < y$. Then by the definition of B_i there exist $\alpha, \beta \in \mathcal{G}_i$ with $x = \alpha 1$ and $y = \beta 1$. We show that $\alpha < \beta$. Suppose that $\alpha \geq \beta$. Then by (3) $\alpha 1 \geq \beta 1$; by definition of α and β we have $x \geq y$, a contradiction.

Let $z \in R$. From $\alpha < \beta$ and (3) we obtain $\alpha z \leq \beta z$, from (7) we then get $\alpha 1 \otimes_{\mathcal{R}} z \leq \beta 1 \otimes_{\mathcal{R}} z$, hence $x \otimes_{\mathcal{R}} z \leq y \otimes_{\mathcal{R}} z$. □

The solution of the flattened problem for a valuation $v \in Var \to D$ is defined by

$$S'(v) = \Big(\bigotimes_i c'_i(v)\Big) \otimes_{\mathcal{R}} \mathcal{I}$$

We define the set of *maximal solutions* of a flattened constraint problem $\mathcal{C}$ in the same way as for monoidal constraint problems:

$$maxSol(\mathcal{C}') = \big\{(v \mapsto S'(v)) \mid S'(v) \text{ maximal in } S'[Var \to D]\big\}.$$

Lemma 3.1 *Let $\mathcal{C}$ be a monoidal constraint problem and let $\mathcal{C}'$ be the corresponding flattened constraint problem. Then*

(i) *for any valuation* v*:* $S(v) = S'(v)$

(ii) $\mathit{maxSol}(\mathcal{C}) = \mathit{maxSol}(\mathcal{C}')$.

Proof. The first statement follows from the definitions of S and S' and equations (8) and (7). The second statement is an immediate consequence of the first and the definition of *maxSol*. □

It is easy to see that for c-monoids $\mathcal{G}_1 = (G_1, \otimes_1, \le_1, 1)$ and $\mathcal{G}_2 = (G_2, \otimes_2, \le_2, 1)$ both

$$\mathcal{G}_1 \times_{\text{cart}} \mathcal{G}_2 := \big(G_1 \times G_2, \otimes_1 \times \otimes_2, \le_\times, (1,1)\big)$$
$$\mathcal{G}_1 \times_{\text{lex}} \mathcal{G}_2 := \big(G_1 \times G_2, \otimes_1 \times \otimes_2, \le_{\text{lex}}, (1,1)\big)$$

satisfy (7) for the functions

$$p^i = \mathit{imult}_i.$$

Both $\mathcal{G}_1 \times_{\text{cart}} \mathcal{G}_2$ and $\mathcal{G}_1 \times_{\text{lex}} \mathcal{G}_2$ are therefore semi-monotone for sets $B_i = \mathit{imult}_i[G_i](1) = \mathit{inj}_i[G_i]$ even though the order of $\mathcal{G}_1 \times_{\text{lex}} \mathcal{G}_2$ is not compatible with its multiplication as the example above shows. More generally, products of arbitrarily many factors are semi monotone relative to sets created by multi-injections $B_{\{i,\dots,k\}} = \mathit{inj}_{\{i,\dots,k\}}[G]$ where

$$\pi_j(\mathit{inj}_{\{i,\dots,k\}}(g)) = \begin{cases} g & \text{if } j \in \{i,\dots,k\} \\ 1 & \text{otherwise} \end{cases}$$

if $G_j = G$ for $j \in \{i \dots k\}$ and the product is ordered by $\times$ and $\times_{\text{lex}}$.

To return to the scheduling example: the important parts of the flattened problem of the last (intensive) model are:

$$\begin{aligned}
R &= \mathit{Nat}\,{}^{\ge}_{\mathit{add}} \times_{\mathit{lex}} (\mathit{Nat}\,{}^{\ge}_{\mathit{add}} \times \mathit{Bool}) \\
\otimes:\quad & (r_1,(r_2,r_3)) \otimes (r_1',(r_2',r_3')) \mapsto (r_1 + r_1', (r_2 + r_2', r_3 \wedge r_3')) \\
\mathit{key\text{-}person}_i' &= \langle \mathit{Date}, \mathit{Place}; (d,l) \mapsto (\mathit{pcst}_i^k(d,l), (0, \mathit{true}))\rangle \\
\mathit{opt\text{-}person}_i' &= \langle \mathit{Date}, \mathit{Place}; (d,l) \mapsto (0, (\mathit{pcst}_i^o(d,l), \mathit{true}))\rangle \\
\mathit{not\text{-}on\text{-}weekend} &= \langle \mathit{Date}; d \mapsto (0, (0, \mathit{cst}^w(d)))\rangle
\end{aligned}$$

4 Solving Soft Constraint Problems with Preferences

The Maude implementation of Monoidal Soft Constraints consists of a package of Maude theories and parameterized functional modules which can be integrated easily in any other Maude application by instantiating the parameter theories with the particular settings of the application. The axiomatic theory of ordered monoids is modelled as Maude functional theory; special ordered monoids such as $\mathit{Nat}\,{}^{\ge}_{\mathit{add}}$ are modelled as functional Maude modules; all other modules of the framework (such as implicit and explicit soft constraints as well as the constraint solving algorithms) are parameterized by the choice of the constraint domain and monoid. In order to improve the efficiency of constraint solving, all recursive specifications are written in tail-recursive form. In the following we present shortly the implementation of the constraints for the meeting scheduling service and the branch-and-bound algorithm for solving soft constraints. The implementation is based on our earlier work for solving soft constraints over constraint semirings [21]. The main changes are: solving soft constraints over partially ordered monoids instead of totally ordered constraint semirings, and a flexible approach for specifying preferences between constraints by dynamically constructing cartesian and lexicographic products of monoids instead of using a (possibly composite) fixed constraint semiring.

4.1 Implementing soft constraints with preferences

We formalize ordered monoids and the monoid structures with ordering relation of a flattened constraint problem as Maude theories in a straightforward way. E.g. the latter has the following form where the monoid structure with ordering $\langle R; \otimes_{\mathcal{R}}; \leq; 1\rangle$ is represented by the sort `Rank`, the operation `*`, the boolean operation `<=`, and the unit element `one`.

```
fth MONOIDwORDER is
 pr BOOL .
 sort Rank.
 op one  : -> Rank  .
 op _*_ : Rank Rank -> Rank         [assoc comm id: one prec 31] .
 op _<=_ : Rank Rank -> Bool        [prec 37] .
 vars X Y Z : Rank .
 eq X <= X = true                   [nonexec] .
 ceq X <= Z = true
     if X <= Y = true /\  Y equiv Z = true [nonexec] .
endfth
```

To implement the meeting schedule example this theory is instantiated by the ordered monoid structures for the grades of the 'key-person', 'optional-person' and 'not-on-weekend' constraints, i.e. with $Nat^{\geq}_{add}$ and $Bool$. These domains can be composed to more complex rank domains by forming lexicographic and cartesian products:

```
*** cartesian product
op _x_ : Rank Rank -> Rank [ctor] .

*** lexicographic product
op _lex_ : Rank Rank -> Rank [ctor] .
```

The corresponding ordering relations are defined elementwise:

```
*** symmetric partial order
eq (R1 x R2) <= (R3 x R4) = (R1 <= R3) and (R2 <= R4) .

*** lexicographic order
eq (R1 lex R2) <= (R3 lex R4) = (R1 < R3) or (R1 == R3 and (R2 <= R4)) .
```

Then we model the lexicographic product of the example $\mathbb{N}_{\leq} \times_{lex} (\mathbb{N}_{\leq} \times Bool_{\leq})$ simply by `NatInf lex (NatInf x Bool)` where `NatInf` is the sort of natural numbers extended with an infinite element.

The flattened constraints are expressed by injecting the grades of the three types of individual constraints into the three components of the product domain. For example, one may have the following schedule constraints where the operation *in* denotes the canonical injection of natural numbers into `NatInf`.

```
*** A key person has preference 5 for meeting in Munich at 2008-02-22
[ Date Place | (2008-02-22 Munich |-> (in(5) lex (one x one)) ... ]
*** An optional person has preference 7 for meeting in Munich at 2008-02-22
```

```
[ Date Place | (2008-02-22 Munich |-> (one lex (in(7) x one)) ... ]
*** not-on-weekend constraint: 2008-02-22 is a weekday
[ Date | (2008-02-22 |-> (one lex (one x true)) ... ]
```

4.2 Search

For computing the solutions of a soft-constraint problem we use a branch-and-bound depth-first search algorithm which is based on the following idea: we search in a depth-first manner a solution of a list of constraints cl by choosing in a consistent way a valuation of each of the constraints of cl and by multiplying these valuations in order to obtain a rank of the solution. The best ranks bl of the solutions obtained so far serve as threshold for the branch-and-bound procedure. During the computation we continue to construct a partial solution only if the rank q of the partial solution is not smaller than any of the elements b of the actual threshold bl. Only in this case it will be possible to obtain a solution with a rank which is not dominated by any other element of the threshold. This follows from the fact that $\otimes$ is intensive: if $q < b$ then for any further multiplication of a with a rank—say p—we have $q \otimes p < b$.

The set of solutions of a soft-constraint problem is represented as a list of constraints. The rank of any solution is not dominated by any other solution, but in case the domain of the ranks is only partially (and not totally) ordered here may exist several solutions with incomparable ranks.

In order to restrict the search space, we also use two further optimisations where appropriate. If the grade of a constraint is totally ordered (as it is the case for all constraints of the meeting schedule example) then we can sort the valuations of each constraint according to the ranks of valuations in a descending order. Now consider the situation where we try to combine a valuation—say η of rank p—of a constraint c with the actual partial solution—of rank q say. Then in case of $p \otimes q < b$ it will not be possible to obtain a solution with a rank better or equal than the actual threshold b and therefore the valuation η can be discarded. Moreover, due to the descending ordering of the valuations of c any of the remaining valuations of c has a rank—p_i say—with $p_i \leq p$. Since multiplication is semi-monotonic w.r.t. the set of the grades of each constraint, $p_i \otimes q \leq p \otimes q < b$ holds and therefore also all remaining valuations of c can be discarded.

In some cases, it is also possible to use divide-and-conquer optimizations. If two lists of constraints $cl1$ and $cl2$ are combined lexicographically, it is possible to solve the first constraint independently and then process the second constraint for each solution of the first one. Cartesian products can also be computed independently if they do not share variables.

The implementation consists of the following two parameterized modules: a module SOLVECONSTRAINT for the search algorithms and a module CONTINUATION for storing the necessary backtracking information. A continuation $ct(cl0, sc)$ consists of a partial solution sc and a list $cl0$ of unsolved constraints with the intended property that the combination $sc \otimes \bigotimes cl0$ of sc with all constraints in $cl0$ forms again a set of possible solutions of the original constraint problem cl.

The module SOLVECONSTRAINT defines depth-first search algorithms for finding all best solutions, for finding a first solution better than a certain threshold, and for finding all such solutions (see [3] for a similar approach). For any list cl, $search(cl)$ computes the set $maxSol(cl)$ of all best solutions. $maxSol$ is implemented to take a list of constraints, not a constraint problem, to avoid the need to repeatedly embed and extract the list of constraints. For constraints $[al \mid (val_1 \mapsto r_1), \ldots, (val_m \mapsto r_m)]$ in cl with a totally ordered rank domain it assumes that all ranks $r_1, \ldots, r_m$ are in descending order. $search$ uses an auxiliary tail-recursive operation $\$search(cl, sc, cont, csol)$ where cl denotes the list of constraints to be solved, sc the actual partial solution, $cont$ the continuation, and $csol$ the constraint solutions obtained so far.

The most interesting case of the recursive definition is $cl = [al \mid (vl \mapsto p) erest]\, rest$ and $sc = [bl \mid (wl \mapsto q)]$ where vl has rank p, $erest$ denotes the map consisting of the remaining assignments of the first constraint of cl, and $rest$ denotes the tail of the list of constraints cl. (All other cases are simpler with $erest$, $rest$ or cl being empty, or sc corresponding to $noConstraint$.) Here $noConstraint$ denotes the empty constraint and plays the role of a unit element in the semiring of constraints (cf. [6]).

If the rank $p \otimes q$ of the new solution is different from $zero$ and not smaller than the rank b of an existing solution and and if the first subconstraint $c_0 = [al \mid (vl \mapsto p)]$ of cl is consistent with sc then $\$search$ computes a new partial solution $[al \mid (vl \mapsto p)] \otimes sc$ and saves the rest of the constraint in the continuation for later backtracking. Otherwise, if $p \otimes q$ is large enough but c_0 not consistent with sc, the search continues with the remaining assignments $erest$ of the first constraint. Finally, if $p \otimes q < b$ and the rank domain of the constraint is totally ordered (as in the case of our example), we can use the fact that every constraint is sorted in a descending order of ranks which implies that for all ranks r of the entries of $erest$ we have $r \otimes q \leq p \otimes q < b$ and thus also $r \otimes q < b$. Since $\otimes$ is intensive, further multiplication of $p \otimes q$ with grades $h_1, \ldots, h_k$ of the remaining other constraints preserves this property: we have $h_1 \otimes \cdots \otimes h_k \otimes p \otimes q \leq p \otimes q < b$. Therefore the partial solution sc cannot be completed to a best solution and the algorithm backtracks with the continuation. By `branch(P, ListResult)` we denote an auxiliary operation which yields `true` iff `P` is smaller than some element of `ListResult`.

```
*** search all best solutions
op search : ListConstraint -> ListConstraint .

eq search(L) = $search(L, noConstraint, nil, noConstraint ) .


op $search : ListConstraint Constraint ListContinuation
             ListConstraint -> ListConstraint .

*** total solution found
eq $search(noConstraint, SC, Cont, ListResult) =
   $backtrack(Cont, insert(SC, ListResult)) .

*** constraint cannot be combined with the current partial solution
eq $search([AL | empty] Rest, SC, Cont, ListResult) =
   $backtrack(Cont, ListResult) .

*** search for a partial solution
eq $search([AL | (VL |-> P) ERest] Rest,
           noConstraint, Cont, ListResult) =
  if branch(P, ListResult) then
```

```
     $search([AL | ERest] Rest, noConstraint, Cont, ListResult)
  else
     $search(Rest, [AL | (VL |-> P)],
             ct([AL | ERest] Rest, noConstraint) Cont, ListResult)
  fi .

*** combine remaining constraints with an existing partial solution
eq $search([AL | (VL |-> P) ERest] Rest,
           [BL | (WL |-> Q)], Cont, ListResult) =
   if branch(P * Q, ListResult) then
   *** stop searching this branch (descending order optimization)
      $backtrack(Cont, ListResult)
   else if (not consistent(AL, BL, VL, WL)) then
   *** continue on this branch
      $search([AL | ERest] Rest, [BL | (WL |-> Q)], Cont, ListResult)
   else
   *** extend partial solution and store continuations
      $search(Rest, [AL | (VL |-> P)] * [BL | (WL |-> Q)],
              ct([AL | ERest] Rest, [BL | (WL |-> Q)]) Cont, ListResult)
   fi fi .
```

4.3 Correctness of Search

It is easy to see that the search algorithm is terminating. The following lemma is the basis for the correctness proof of the search algorithm; it assumes semi-monotonicity of multiplication w.r.t. $\leq$ and the sets B_i of the soft constraints under consideration.

Lemma 4.1 (search invariant) *Let M be an ordered monoid. Let $cl = cl_1 \dots cl_n$ be a list of soft constraints such that for each cl_i, $\otimes$ is semi-monotone w.r.t $<$ and the set of ranks of cl_i, and that the valuations of each cl_i are sorted in descending order according to their ranks. Let sc be a partial constraint, $cont$ be a list of continuations of the form $ct(contl_1, scont_1), \dots, ct(contl_k, scont_k)$, and scl be a list of m possible solutions with ranks $b_1 \dots b_m$, i.e., scl is a list of simple constraints of the form $scl_i = [xl \mid (wl_i \mapsto b_i)]$.*

Then the following holds:

(i) $\$backtrack(cont, scl) = maxSol(\bigoplus_{j=1}^{k}(scont_j \otimes \bigotimes contl_j) \oplus \bigoplus(scl))$;

(ii) $\$search(cl, sc, cont, scl) = maxSol(sc \otimes \bigotimes cl \oplus \bigoplus_{j=1}^{k}(scont_j \otimes \bigotimes contl_j) \oplus \bigoplus(scl))$.

(We use $\oplus$ and $\bigoplus$ to denote list concatenation and iterated list concatenation.)

Proof. By simultaneous computational induction on both operations. □

The correctness of the search operation follows directly from termination and the invariant lemma:

Theorem 4.2 (total correctness of search) *Let $\mathcal{C}$ be a constraint problem and let $\mathcal{C}'$ be the corresponding flattened constraint problem. Let cl' be the list of constraints of $\mathcal{C}'$. Then the following holds:*

$$search(cl') = maxSol(\mathcal{C}).$$

Proof. We have

$$search(cl') = \$search(cl', noConstraint, nil, noConstraint)$$

	10	$5 \times_{\text{lex}} 5$	$3 \times_{\text{lex}} 3 \times_{\text{lex}} 4$	$3 \times_{\text{lex}} 4 \times_{\text{lex}} 3$
branch & bound	27,7	24,6	2,6	2,9
b&b, divide & conquer	27,7	14	0,5	0,9

Fig. 1. Average performance for randomly generated problems with 20 variables, 3 domain elements and 10 constraints for total preference relations. Times in seconds.

Preferences	Time
6	2.4
3×3	92.0
$3 \times_{\text{lex}} 3$	0.1

Fig. 2. Average performance for randomly generated problems with 20 variables, 3 domain elements and 6 constraints for total and partial preference relations. Branch and bound search, times in seconds.

$$
\begin{aligned}
&= maxSol(noConstraint \otimes \bigotimes cl') \\
&= maxSol(\bigotimes cl'),
\end{aligned}
$$

where the second equality uses Lemma 4.1 and the fact that the empty constraint does not contribute any constraint. The third equality follows from the unit property of *noConstraint*. Then the result follows from Lemma 3.1. □

5 Benchmarks

To evaluate the performance of the solver we used several variations of randomly generated problems with preference relations mapping either into lexical or pointwise products of finite sets, corresponding to, respectively, preference or indifference between constraints. As expected the search complexity depends strongly on the size of the given problem and on details like the ordering of the constraints and variables.

A general observation is that the introduction of totally ordered preferences tends to speed up the search significantly whereas the introduction of indifferences (which lead to partial orders) slows down the search, often several orders of magnitude. The divide and conquer optimization separates the problem into several smaller ones which can be solved independently and thereby tends to reduce the time necessary to compute the solutions. However, there are certain problems where each of the preceding statements does not hold and the reverse effect can be observed.

We generated random problems with 20 domain variables, 3 different values in each domain and 10 constraints each having 3 variables in its domain. Without constraint orders the computation of all maximal solutions to these problems took between 20 and 30 seconds. Dividing the set of constraints into two preference classes reduced the time to between 7 and 20 seconds in most cases, but some problems required nearly a minute of search time, depending on the placement of the individual constraints into the preference classes. Introducing a third hierarchy of levels clearly increased the performance of the search in all cases and led to times between 0,3 and 5 seconds, see Figure 1 for the average values. Adding the divide and conquer optimization reduced the average times significantly with an increase in performance roughly proportional to the number of preference classes. If no preferences between constraints are defined the divide and conquer optimization is not applicable and does not influence the execution time.

Indifferences between constraints which lead to partial orders heavily increase the time required to find the best solutions. Problems with 6 constraints generally take under 5 seconds to compute, but using only a single indifference increases this to more than 90 seconds. In this variation, total preference orders require only fractions of a second of execution time, see Figure 2.

6 Related Work

The most direct influence on our work was the elegant theory of semiring-based constraint satisfaction problems (SCSPs)(see e.g. [5,6,4]); the constraint solver presented in this paper is an enhanced version of the solver for semiring-based constraints presented in [21]. Semirings are not closed under lexicographic products, therefore SCSPs cannot be used to directly express the preference relations described in this paper. Other approaches to generalize SCSPs for preference relations are given in [8,7]. Our approach was also inspired by [1] where preferences are treated in an elegant algebraic way.

The separation into grades and ranks resembles the stratification of constraint problems in constraint hierarchies [10,9]. Constraint hierarchies describe a hierarchical structure of crisp constraints where a comparator function computes values for the quality of solutions which are consistent with the hierarchy. There exist many algorithms for solving particular constraint hierarchies, and a number of constraint solvers for solving particular constraint hierarchies have been implemented [19,2,16]. It is an interesting problem to try to generalize some of these algorithms to work with our framework.

Other implementations of constraint solvers for soft constraints are described in [3,15,14,18]. The standard reference for decisions with multiple objectives is [17].

7 Conclusions and Further Work

We have presented a novel approach to soft constraints that is based on well-known mathematical structures [11,12] and allows users to express general preference relations for decisions with multiple objectives in a straightforward manner, developed an implementation of a constraint solver in Maude, and proved its correctness.

The current implementation of the constraint solver is based on a branch-and-bound search. An interesting open problem is which other constraint satisfaction techniques can be applied to (specializations of) the soft constraint systems described in this paper and how their performance will relate to the relatively simple branch-and-bound search mechanism.

Using Maude as the implementation language simplifies the correctness proof for the implementation and allows us to easily integrate the solver into the Pagoda system for software-defined radios. On the other hand, using the Maude system entails some inefficiencies which are not incurred when implementing the solver in a lower-level language. We are currently working on a C# implementation which we expect to be competitive with other soft-constraint solvers.

In the Maude implementation we flatten the constraint system and thereby remove the separation between grades and ranks before trying to solve the constraints. While this simplifies the constraint solver this preprocessing removes information that might be used to improve the performance of the constraint solver. Further research is needed to find appropriate analysis methods that can exploit the two-level structure of the constraint problem to generate improved solvers.

Currently we restrict constraints to finite support. It is straightforward to generalize the theory to constraints with infinite support and therefore to address problems that go beyond constraint satisfaction problems. However, further research is needed to develop efficient methods to solve these generalized problems.

References

[1] Hajnal Andréka, Mark Ryan, and Pierre-Yves Schobbens. Operators and laws for combining preference relations. *J. Log. Comput.*, 12(1):13–53, 2002.

[2] Greg J. Badros, Alan Borning, and Peter J. Stuckey. The cassowary linear arithmetic constraint solving algorithm. *ACM Trans. Comput.-Hum. Interact.*, 8(4):267–306, 2001.

[3] Stefano Bistarelli, Thom W. Frühwirth, and Michael Marte. Soft constraint propagation and solving in chrs. In *SAC*, pages 1–5. ACM, 2002.

[4] Stefano Bistarelli and Fabio Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of ECAI 2006, 17th European Conference on Artificial Intelligence*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 63–67. IOS Press, 2006.

[5] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[6] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Soft concurrent constraint programming. *ACM Trans. Comput. Log.*, 7(3):563–589, 2006.

[7] Stefano Bistarelli, Maria Silvia Pini, Francesca Rossi, and Kristen Brent Venable. Bipolar preference problems: Framework, properties and solving techniques. In Francisco Azevedo, Pedro Barahona, François Fages, and Francesca Rossi, editors, *CSCLP*, volume 4651 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2006.

[8] Stefano Bistarelli, Maria Silvia Pini, Francesca Rossi, and Kristen Brent Venable. Uncertainty in bipolar preference problems. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 782–789. Springer, 2007.

[9] Alan Borning, Bjørn N. Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[10] Alan Borning, Bjørn N. Freeman-Benson, and Molly Wilson. Constraint hierarchies. In Michael Jampel, Eugene C. Freuder, and Michael J. Maher, editors, *Over-Constrained Systems*, volume 1106 of *Lecture Notes in Computer Science*, pages 23–62. Springer, 1995.

[11] Nicolas Bourbaki. *Algebra I, Chapters 1–3.* Élements de mathématique. English. Springer, 1989.

[12] Nicolas Bourbaki. *Algebra II, Chapters 4–7.* Élements de mathématique. English. Springer, 1990.

[13] Rocco De Nicola, Gianluigi Ferrari, Ugo Montanari, Rosario Pugliese, and Emilio Tuosto. A Basic Calculus for Modelling Service Level Agreements. In Jean-Marie Jacquet and Gian Pietro Picco, editors, *International Conference on Coordination Models and Languages*, volume 3454 of *LNCS*, pages 33 – 48, Namur (Belgium), April 2005. Springer.

[14] Alberto Delgado, Carlos Alberto Olarte, Jorge Andrés Pérez, and Camilo Rueda. Implementing semiring-based constraints using mozart. In Peter Van Roy, editor, *MOZ*, volume 3389 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 2004.

[15] Yan Georget and Philippe Codognet. Compiling semiring-based constraints with clp (fd, s). In Michael J. Maher and Jean-Francois Puget, editors, *CP*, volume 1520 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 1998.

[16] Warwick Harvey, Peter J. Stuckey, and Alan Borning. Fourier elimination for compiling constraint hierarchies. *Constraints*, 7(2):199–219, 2002.

[17] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs.* Cambridge University Press, Cambridge, 1993.

[18] Hana Rudová. Soft clp (fd). In Ingrid Russell and Susan M. Haller, editors, *FLAIRS Conference*, pages 202–207. AAAI Press, 2003.

[19] Michael Sannella, John Maloney, Bjørn N. Freeman-Benson, and Alan Borning. Multi-way versus one-way constraints in user interfaces: Experience with the deltablue algorithm. *Softw., Pract. Exper.*, 23(5):529–566, 1993.

[20] Martin Wirsing, Allan Clark, Stephen Gilmore, Matthias Hölzl, Alexander Knapp, Nora Koch, and Andreas Schroeder. Semantic-Based Development of Service-Oriented Systems. In E. Najn et al., editor, *Proc. 26th IFIP WG 6.1 Internat. Conf. on Formal Methods for Networked and Distributed Systems (FORTE'06), Paris, France*, volume 4229 of *LNCS*, pages 24–45. Springer, 2006.

[21] Martin Wirsing, Grit Denker, Carolyn Talcott, Andy Poggio, and Linda Briesemeister. A rewriting logic framework for soft constraints. In *WRLA 2006, 6th International Workshop on Rewriting Logic and its Applications*, April 2006. To appear in ENTCS, 2006.