An Abstraction Framework for Soft Constraints, And Its Relationship with Constraint Propagation

S. Bistarelli[1], P. Codognet[2], F. Rossi[3]

1: Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy. E-mail: bista@di.unipi.it
2: University of Paris 6, LIP6, case 169, 4, Place Jussieu, 75 252 Paris Cedex 05, France. E-mail: Philippe.Codognet@lip6.fr
3: Università di Padova, Dipartimento di Matematica, Via Belzoni 7, 35131 Padova, Italy. E-mail: frossi@math.unipd.it

Abstract. Soft constraints are very flexible and expressive. However, they also are very complex to handle. For this reason, it may reasonable in several cases to pass to an abstract version of a given soft problem, and then to bring some useful information from the abstract problem to the concrete one. This will hopefully make the search for a solution, or for an optimal solution, of the concrete problem, faster.
In this paper we review the main concepts and properties of our abstraction framework for soft constraints, and we show how it can be used to import constraint propagation algorithms from the abstract scenario to the concrete one. This may be useful when we don't have any (or any efficient) propagation algorithm in the concrete setting.

1 Introduction

Soft constraints allow to model faithfully many real-life problems, especially those which possess features like preferences, uncertainties, costs, levels of importance, and absence of solutions. Formally, a soft constraint problem (SCSP) is just like a classical constraint problem (CSP), except that each assignment of values to variables in the constraints is associated to an element taken from a set (usually ordered). These elements will then directly represent the desired features, since they can be interpreted as levels of preference, or costs, or levels of certainty, or many other criteria.

SCSPs are more expressive than classical CSPs, but they are also more difficult to process and to solve. For these reasons, it may be reasonable to work on a simplified version of the given problem, trying however to not loose too much information. We propose to define this simplified version by means of the notion of abstraction, which takes an SCSP and returns a new one which is simpler to solve. Here, as in many other works on abstraction [11, 10], "simpler" may mean many things, like the fact that a certain solution algorithm finds a solution, or an optimal solution, in a fewer number of steps, or also that the abstracted problem can be processed by a machinery which is not available in the concrete context.

Once we get the abstracted version of a given problem, we 1) process the abstracted version; 2) bring back to the original problem some (or possibly all) of the information derived in the abstract context; and 3) continue the solution process on the transformed problem, which is a concrete problem equivalent to the given one. All this process has the main aim of finding an optimal solution, or an approximation of it, for the original SCSP, within the resource bounds we have. The hope is that, by following the above three steps, we get to the final goal faster than just solving the original problem.

In particular, we can prove the following:

- If the abstraction satisfies a certain property, all optimal solutions of the concrete SCSP are also optimal in the corresponding abstract SCSP. Thus, in order to find an optimal solution of the concrete problem, we could find all the optimal solutions of the abstract problem, and then just check their optimality on the concrete SCSP.
- Given any optimal solution of the abstract problem, we can find upper and lower bounds for an optimal solution for the concrete problem. If we are satisfied with these bounds, we could just take the optimal solution of the abstract problem as a reasonable approximation of an optimal solution for the concrete problem.
- If we apply some constraint propagation technique over the abstract problem, say P, obtaining a new abstract problem, say P', some of the information in P' can be inserted into P, obtaining a new concrete problem which is closer to its solution and thus easier to solve. This however can be done only if the semiring operation which describes how to combine constraints on the concrete side is idempotent.
- If instead this operation is not idempotent, still we can bring back some information from the abstract side. In particular, we can bring back the inconsistencies (that is, tuples with associated the worst element of the semiring), since we are sure that these same tuples are inconsistent also in the concrete SCSP.

In both the last two cases, the new concrete problem is easier to solve, in the sense, for example, that a branch-and-bound algorithm would explore a smaller (or equal) search tree before finding an optimal solution.

In this paper we show how to use this abstraction framework, and its properties, to import constraint propagation algorithms from the abstract scenario to the concrete one. More precisely, we show how to construct propagation rules for the concrete problem from propagation rules for the abstract problem. This may be useful when we don't have any (or any efficient) propagation algorithm in the concrete setting.

The only other abstraction scheme for soft constraint problems we are aware of is the one in [12], where *valued CSPs* [17] are abstracted in order to produce good lower bounds for the optimal solutions. The concept of valued CSPs is similar to our notion of SCSPs. In fact, in valued CSPs, the goal is to minimize the value associated to a complete assignment. In valued CSPs, each constraint has

one associated element, not one for each tuple of domain values of its variables. However, our notion of soft CSPs and that in valued CSPs are just different formalizations of the same idea, since one can pass from one formalization to the other one without changing the solutions, provided that the partial order is total [3]. However, our abstraction scheme is different from the one in [12]. In fact, we are not only interested in finding good lower bounds for the optimum, but also in finding the exact optimal solutions in a shorter time. Moreover, we don't define *ad hoc* abstraction functions but we follow the classical abstraction scheme devised in [6], with Galois insertions to relate the concrete and the abstract domain, and locally correct functions on the abstract side. We think that this is important in that it allows to inherit many properties which have already been proven for the classical case. It is however worth noticing that our notion of an order-preserving abstraction is related to their concept of aggregation compatibility, although generalized to deal with partial orders.

Another abstraction framework for constraints can be found in [5]. However, it only deals with classical "crisp" constraints, and it aims at abstracting the domains of the variables, while maintaining the same constraint combination operator.

The paper is organized as follows. Section 2 summarizes the main notions on soft constraints and soft constraint propagation. Then, Sections 3 and 4 describe how to abstract soft constraints, and Section 5 summarizes the main properties of our approach. Then, Section 6 shows the relationship between abstraction and local consistency, and finally Section 7 concludes the paper and gives some hints about possible lines of future work.

A longer, more detailed, and completely formal description of our approach to soft constraint abstraction has appeared in [2]. Here we summarize the approach by giving a more informal description and providing several examples, and we focus on the relationship between abstraction and local consistency.

2 Soft constraints

A soft constraint [4] is just a classical constraint where each instantiation of its variables has an associated value from a partially ordered set. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of semiring, which is just a set plus two operations satisfying certain properties: $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$.

If we consider the relation $\leq_S$ over A defined as $a \leq_S b$ iff $a + b = b$, then we have that:

- $\leq_S$ is a partial order;
- $+$ and $\times$ are monotone on $\leq_S$;
- $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
- $\langle A, \leq_S\rangle$ is a complete lattice and $+$ is its lub.

Moreover, if $\times$ is idempotent, then $\langle A, \leq_S \rangle$ is a complete distributive lattice and $\times$ is its glb. Informally, the relation $\leq_S$ gives us a way to compare (some of the) tuples of values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a.*

Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, a finite set D (the domain of the variables), and an ordered set of variables V, a constraint is a pair $\langle def, con \rangle$ where $con \subseteq V$ and $def : D^{|con|} \rightarrow A$. Therefore, a constraint specifies a set of variables (the ones in con), and assigns to each tuple of values of D of these variables an element of the semiring set A. This element can then be interpreted in several ways: as a level of preference, or as a cost, or as a probability, etc. The correct way to interpret such elements depends on the choice of the semiring operations.

Constraints can be compared by looking at the semiring values associated to the same tuples: Consider two constraints $c_1 = \langle def_1, con \rangle$ and $c_2 = \langle def_2, con \rangle$, with $|con| = k$. Then $c_1 \sqsubseteq_S c_2$ if for all k-tuples t, $def_1(t) \leq_S def_2(t)$. The relation $\sqsubseteq_S$ is a partial order. In the following we will also use the obvious extension of this relation to sets of constraints, and also to problems (seen as sets of constraints).

Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs [8, 15, 16] can instead be modeled in the SCSP framework by choosing the c-semiring: $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$.

Given two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con \rangle$ defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$. In words, combining two constraints means building a new constraint involving all the variables of the original ones, and which associates to each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

Given a constraint $c = \langle def, con \rangle$ and a subset I of V, the *projection* of c over I, written $c \Downarrow_I$, is the constraint $\langle def', con' \rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t/t\downarrow^{con}_{I \cap con} = t'} def(t)$. Informally, projecting means eliminating some variables. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables.

The *solution* of a SCSP problem $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$: we combine all constraints, and then project over the variables in con. In this way we get the constraint over con which is "induced" by the entire SCSP. Optimal solutions are those solutions which have the best semiring element among those associated to solutions. The set of optimal solutions of an SCSP P will be written as $Opt(P)$. In the following, we will sometimes call "a solution" one tuple of domain values for all the problem's variables (over con), plus its associated semiring element.

Figure 1 shows an example of fuzzy CSP and its solutions.

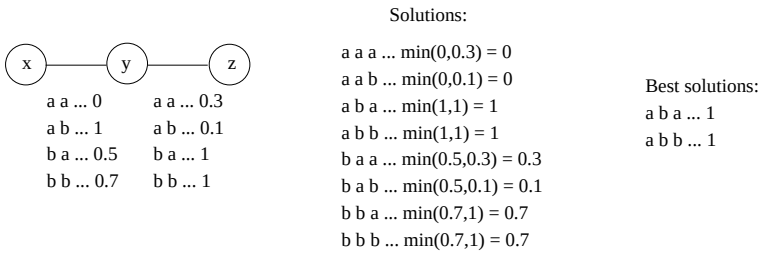

Fig. 1. A fuzzy CSP and its solutions.

Consider two problems P_1 and P_2. Then $P_1 \sqsubseteq_P P_2$ if $Sol(P_1) \sqsubseteq_S Sol(P_2)$. If $P_1 \sqsubseteq_P P_2$ and $P_2 \sqsubseteq_P P_1$, then they have the same solution, thus we say that they are equivalent and we write $P_1 \equiv P_2$.

SCSP problems can be solved by extending and adapting the technique usually used for classical CSPs. For example, to find the best solution we could employ a branch-and-bound search algorithm (instead of the classical backtracking), and also the successfully used propagation techniques, like arc-consistency [13], can be generalized to be used for SCSPs. The detailed formal definition of propagation algorithms (sometimes called also *local consistency* algorithms) for SCSPs can be found in [4]. For the purpose of this paper, what is important to say is that a *propagation rule* is a function which, taken an SCSP, solves a subproblem of it. It is possible to show that propagation rules are idempotent, monotone, and intensive functions (over the partial order of problems) which do not change the solution set. Given a set of propagation rules, a local consistency algorithm consists of applying them in any order until stability. It is possible to prove that local consistency algorithms defined in this way have the following properties if the multiplicative operation of the semiring is idempotent: equivalence, termination, and uniqueness of the result.

Thus we can notice that the generalization of local consistency from classical CSPs to SCSPs concerns the fact that, instead of deleting values or tuples, obtaining local consistency in SCSPs means changing the semiring values associated to some tuples or domain elements. The change always brings these values towards the worst value of the semiring, that is, the $\mathbf{0}$. Thus, it is obvious that, given an SCSP problem P and the problem P' obtained by applying some local consistency algorithm to P, we must have $P' \sqsubseteq_S P$.

3 Abstraction

The main idea [1, 6, 7] is to relate the concrete and the abstract scenarios by a pair of functions, the *abstraction* function α and the *concretization* function γ, which form a Galois connection.

Let $(\mathcal{C}, \sqsubseteq)$ and $(\mathcal{A}, \leq)$ be two posets (the concrete and the abstract domain). A Galois connection $\langle \alpha, \gamma \rangle : (C, \sqsubseteq) \rightleftharpoons (A, \leq)$ is a pair of maps $\alpha : \mathcal{C} \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ such that

1. α and γ are monotonic,
2. for each $x \in \mathcal{C}, x \sqsubseteq \gamma(\alpha(x))$ and
3. for each $y \in \mathcal{A}, \alpha(\gamma(y)) \leq y$.

Moreover, a Galois insertion (of $\mathcal{A}$ in $\mathcal{C}$) $\langle \alpha, \gamma \rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ is a Galois connection where $\gamma \cdot \alpha$ is the identity over $\mathcal{A}$, that is, $Id_{\mathcal{A}}$.

An example of a Galois insertion can be seen in Figure 2. Here, the concrete lattice is $\langle [0,1], \leq \rangle$, and the abstract one is $\langle \{0,1\}, \leq \rangle$. Function α maps all real numbers in $[0, 0.5]$ into 0, and all other integers (in $(0.5, 1]$) into 1. Function γ maps 0 into 0.5 and 1 into 1.

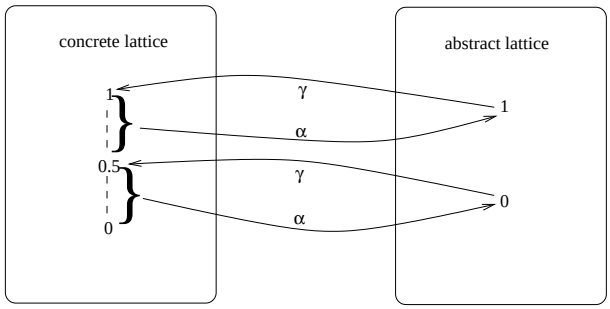

Fig. 2. A Galois insertion.

Consider a Galois insertion from $(\mathcal{C}, \sqsubseteq)$ to $(\mathcal{A}, \leq)$. Then, if $\sqsubseteq$ is a total order, also $\leq$ is so.

Most of the times it is useful, and required, that the abstract operators show a certain relationship with the corresponding concrete ones. This relationship is called *local correctness*. Let $f : \mathcal{C}^n \to \mathcal{C}$ be an operator over the concrete lattice, and assume that $\tilde{f}$ is its abstract counterpart. Then $\tilde{f}$ is locally correct w.r.t. f if $\forall x_1, \ldots, x_n \in \mathcal{C}. f(x_1, \ldots, x_n) \sqsubseteq \gamma(\tilde{f}(\alpha(x_1), \ldots, \alpha(x_n)))$.

4 Abstracting soft CSPs

The main idea is very simple: we just want to pass, via the abstraction, from an SCSP P over a certain semiring S to another SCSP $\tilde{P}$ over the semiring $\tilde{S}$, where the lattices associated to $\tilde{S}$ and S are related by a Galois insertion as shown above.

Consider the *concrete* SCSP problem $P = \langle C, con \rangle$ over semiring S, where

- $S = \langle A, +, \times, 0, 1 \rangle$ and
- $C = \{c_0, \ldots, c_n\}$ with $c_i = \langle con_i, def_i \rangle$ and $def_i : D^{|con_i|} \to A$;

we define an *abstract* SCSP problem $\tilde{P} = \langle \tilde{C}, con \rangle$ over the semiring $\tilde{S}$, where

- $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{0}, \tilde{1} \rangle$;
- $\tilde{C} = \{\tilde{c_0}, \ldots, \tilde{c_n}\}$ with $\tilde{c_i} = \langle con_i, \tilde{def_i} \rangle$ and $\tilde{def_i} : D^{|con_i|} \to \tilde{A}$;
- if $L = \langle A, \leq \rangle$ is the lattice associated to S and $\tilde{L} = \langle \tilde{A}, \tilde{\leq} \rangle$ the lattice associated to $\tilde{S}$, then there is a Galois insertion $\langle \alpha, \gamma \rangle$ such that $\alpha : L \to \tilde{L}$;
- $\tilde{\times}$ is locally correct with respect to $\times$.

Notice that the kind of abstraction we consider in this paper does not change the structure of the SCSP problem. The only thing that is changed is the semiring.

Notice also that, given two problems over two different semirings, there may exist zero, one, or also many abstractions (that is, a Galois insertion between the two semirings) between them. This means that given a concrete problem over S and an abstract semiring $\bar{S}$, there may be several ways to abstract such a problem over $\bar{S}$.

Example 1. As an example, consider any SCSP over the semiring for optimization $\langle \mathcal{R}^- \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$ and suppose we want to abstract it onto the semiring for fuzzy reasoning $\langle [0,1], max, min, 0, 1 \rangle$. In other words, instead of computing the maximum of the sum of all costs (which are negative reals), we just want to compute the maximum of their minimum vale, and we want to normalize the costs over [0..1]. Notice that the abstract problem has an idempotent $\times$ operator (which is the min). This means that in the abstract framework we can perform local consistency over the problem in order to find inconsistencies.

Example 2. Another example is the abstraction from the fuzzy semiring to the classical one:

$$S_{CSP} = \langle \{0,1\}, \vee, \wedge, 0, 1 \rangle.$$

Here function α maps each element of $[0,1]$ into either 0 or 1. For example, one could map all the elements in $[0,x]$ onto 0, and all those in $(x,1]$ onto 1, for some fixed x. Figure 2 represents this example with $x = 0.5$.

An important property of our notion of abstraction is that the composition of two abstractions is still an abstraction. This allows to build a complex abstraction by defining several simpler abstractions to be composed.

5 Properties of the abstraction

We will now summarize the main results about the relationship between a concrete problem and an abstraction of it.

Let us consider the scheme depicted in Figure 3. Here and in the following pictures, the left box contains the lattice of concrete problems, and the right one the lattice of abstract problems. The partial order in each of these lattices is shown via dashed lines. Connections between the two lattices, via the abstraction and concretization functions, is shown via directed arrows. In the following, we will call $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ the concrete semiring and $\tilde{S} = \langle \tilde{A}, \tilde{+}, \tilde{\times}, \tilde{\mathbf{0}}, \tilde{\mathbf{1}} \rangle$ the

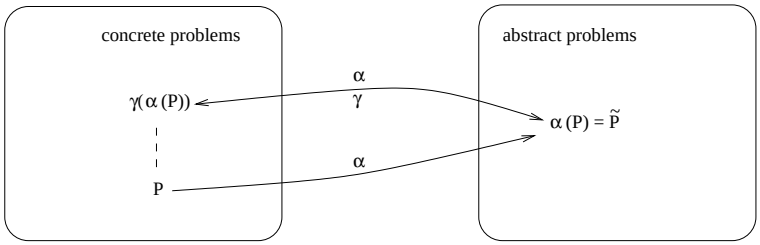

Fig. 3. The concrete and the abstract problem.

abstract one. Thus we will always consider a Galois insertion $\langle \alpha, \gamma \rangle : \langle A, \leq_S \rangle \rightleftharpoons \langle \tilde{A}, \leq_{\tilde{S}} \rangle$.

In Figure 3, P is the starting SCSP problem. Then with the mapping α we get $\tilde{P} = \alpha(P)$, which is an abstraction of P. By applying the mapping γ to $\tilde{P}$, we get the problem $\gamma(\alpha(P))$. Let us first notice that these two problems (P and $\gamma(\alpha(P))$) are related by a precise property:

$$P \sqsubseteq_S \gamma(\alpha(P)).$$

Notice that this implies that, if a tuple in $\gamma(\alpha(P))$ has semiring value $\mathbf{0}$, then it must have value $\mathbf{0}$ also in P. This holds also for the solutions, whose semiring value is obtained by combining the semiring values of several tuples. Therefore, by passing from P to $\gamma(\alpha(P))$, no new inconsistencies are introduced. However, it is possible that some inconsistencies are forgotten.

Example 3. Consider the abstraction from the fuzzy to the classical semiring, as described in Figure 2. Then, if we call P the fuzzy problem in Figure 1, Figure 4 shows the concrete problem P, the abstract problem $\alpha(P)$, and its concretization $\gamma(\alpha(P))$. It is easy too see that, for each tuple in each constraint, the associated semiring value in P is lower than or equal to that in $\gamma(\alpha(P))$.

If the abstraction preserves the semiring ordering (that is, applying the abstraction function and then combining gives elements which are in the same ordering as the elements obtained by combining only), then the abstraction is called *order-preserving*, and in this case there is also an interesting relationship between the set of optimal solutions of P and that of $\alpha(P)$. In fact,

if a certain tuple is optimal in P, then this same tuple is also optimal in $\alpha(P)$.

Example 4. Consider again the previous example. The optimal solutions in P are the tuples $\langle a, b, a \rangle$ and $\langle a, b, b \rangle$. It is easy to see that these tuples are also optimal in $\alpha(P)$. In fact, this is a classical constraint problem where the solutions are tuples $\langle a, b, a \rangle$, $\langle a, b, b \rangle$, $\langle b, b, a \rangle$, and $\langle b, b, b \rangle$.

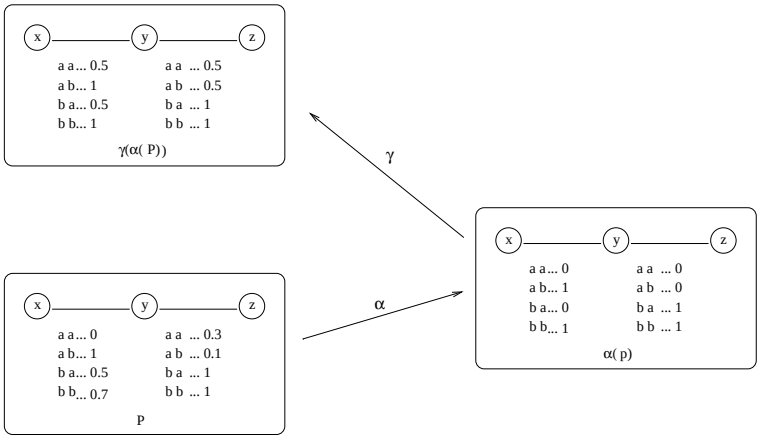

Fig. 4. An example of the abstraction fuzzy-classical.

Thus, if we want to find an optimal solution of the concrete problem, we could find all the optimal solutions of the abstract problem, and then use them on the concrete side to find an optimal solution for the concrete problem. Assuming that working on the abstract side is easier than on the concrete side, this method could help us find an optimal solution of the concrete problem by looking at just a subset of tuples in the concrete problem.

Another important property, which holds for any abstraction, concerns computing bounds that approximate an optimal solution of a concrete problem. In fact, any optimal solution, say t, of the abstract problem, say with value $\tilde{v}$, can be used to obtain both an upper and a lower bound of an optimum in P. In fact, we can prove that

> *there is an optimal solution in P with value between $\gamma(\tilde{v})$ and the value of t in P.*

Thus, if we think that approximating the optimal value with a value within these two bounds is satisfactory, we can take t as an approximation of an optimal solution of P. Notice that this theorem does not need the order-preserving property in the abstraction, thus any abstraction can exploit its result.

Example 5. Consider again the previous example. Now take any optimal solution of $\alpha(P)$, for example tuple $\langle b, b, b\rangle$. Then the above result states that there exists an optimal solution of P with semiring value v between the value of this tuple in P, which is 0.7, and $\gamma(1) = 1$. In fact, there are optimal solutions with value 1 in P.

Consider now what we can do on the abstract problem, $\alpha(P)$. One possibility is to apply an abstract function $\tilde{f}$, which can be, for example, a local consistency

algorithm (like arc-consistency or path-consistency [14]) or also a solution algorithm. In the following, we will consider functions $\tilde{f}$ which are always intensive, that is, which bring the given problem closer to the bottom of the lattice. Also, functions $\tilde{f}$ will always be locally correct with respect to any function f_{sol} which solves the concrete problem. We will call such a property *solution-correctness*. We will also need the notion of safeness of a function, which just means that it maintains all the solutions. It is easy to see that any local consistency algorithm, as defined in [4], can be seen as a safe, intensive, and solution-correct function.

From $\tilde{f}(\alpha(P))$, applying the concretization function γ, we get $\gamma(\tilde{f}(\alpha(P)))$, which therefore is again over the concrete semiring (the same as P). If $\tilde{f}$ is safe, solution-correct, and intensive, and $\times$ is idempotent, we have that

$$Sol(P) = Sol(P \otimes \gamma(\tilde{f}(\alpha(P)))).$$

Figure 5 describes such a situation.

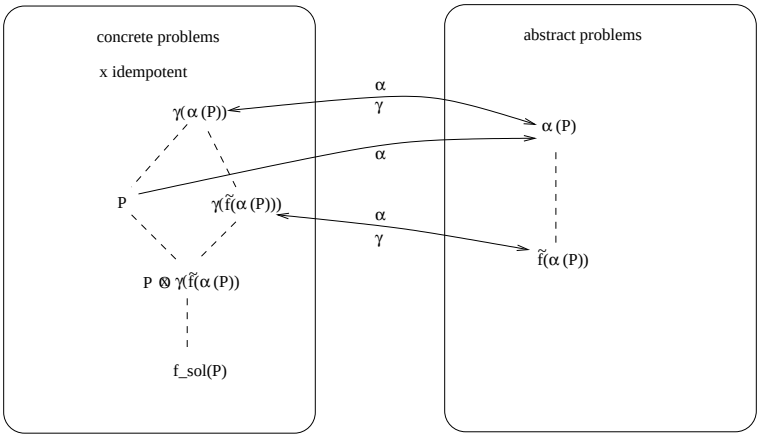

Fig. 5. The general abstraction scheme, with $\times$ idempotent.

The statement above does not say anything about the power of $\tilde{f}$, which could make many modifications to $\alpha(P)$, or it could also not modify anything. In this last case, $\gamma(\tilde{f}(\alpha(P))) = \gamma(\alpha(P)) \sqsupseteq P$ (see Figure 6), so $P \otimes \gamma(\tilde{f}(\alpha(P))) = P$, which means that we have not gained anything in abstracting P. However, we can always use the relationship between P and $\alpha(P)$ to find an approximation of the optimal solutions and of the inconsistencies of P.

Example 6. Figure 7 uses the abstraction in Figure 2 and shows a concrete problem and the result of the construction of Figure 5 over it.

If instead $\times$ is not idempotent, then we can prove something weaker. Figure 8 shows this situation. With respect to Figure 5, we can see that the possible

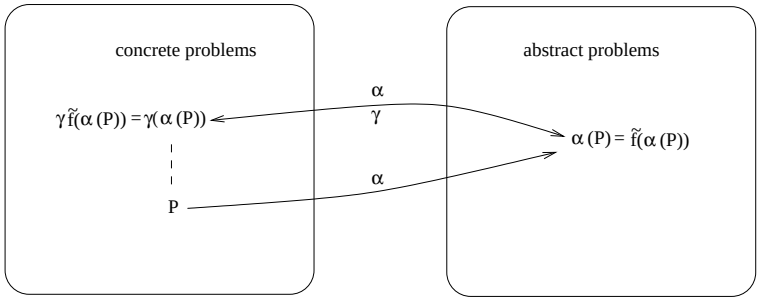

Fig. 6. The scheme when $\tilde{f}$ does not modify anything.

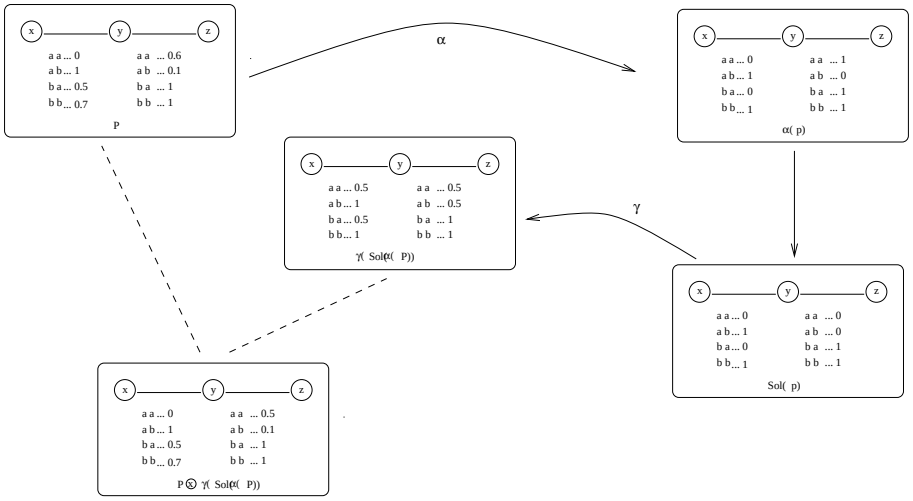

Fig. 7. An example with $\times$ idempotent.

non-idempotence of $\times$ changes the partial order relationship on the concrete side. In particular, we don't have the problem $P \otimes \gamma(\tilde{f}(\alpha(P)))$ any more, nor the problem $f_{sol}(P)$, since these problems would not have the same solutions as P and thus are not interesting to us. We have instead a new problem P', which is constructed in such a way to "insert" the inconsistencies of $\gamma(\tilde{f}(\alpha(P)))$ into P. P' is obviously lower than P in the concrete partial order, since it is the same as P with the exception of some more $\mathbf{0}$'s, but the most important point is that it has the same solutions as P.

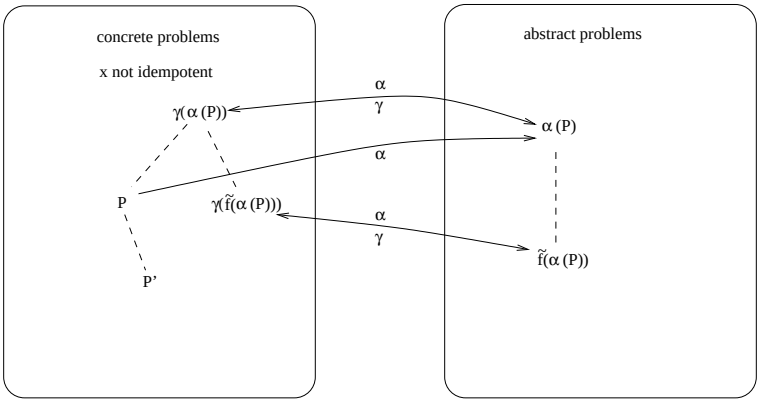

Fig. 8. The scheme when $\times$ is not idempotent.

Example 7. Consider the abstraction from the semiring $S = \langle Z^- \cup \{-\infty\}, max, +, -\infty, 0\rangle$ to the semiring $S' = \langle Z^- \cup \{-\infty\}, max, min, -\infty, 0\rangle$, where α and γ are the identity. This means that we perform the abstraction just to change the multiplicative operation, which is min instead of $+$. Then Figure 9 shows a concrete problem over S, and the construction shown in Figure 8 over it.

Summarizing, the above theorems can give us several hints on how to use the abstraction scheme to make the solution of P easier: If $\times$ is idempotent, then we can replace P with $P \otimes \gamma(\alpha(\tilde{f}(P)))$, and get the same solutions. If instead $\times$ is not idempotent, we can replace P with P'. In any case, the point in passing from P to $P \otimes \gamma(\alpha(\tilde{f}(P)))$ (or P') is that the new problem should be easier to solve than P, since the semiring values of its tuples are more explicit, that is, closer to the values of these tuples in a completely solved problem.

6 Abstraction vs. local consistency

It is now interesting to consider the relationship between our abstraction framework and the concept of local consistency.

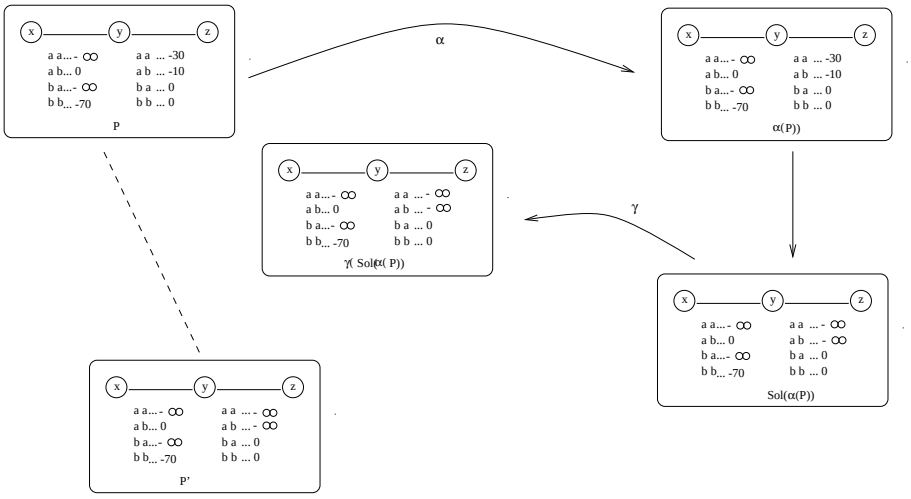

Fig. 9. An example with $\times$ is not idempotent.

In fact, it is possible to show that, given an abstraction $\langle \alpha, \gamma \rangle$ between semirings S and $\bar{S}$ and any propagation rule r in $\bar{S}$, the function $\gamma(r(\alpha(P))) \otimes P$ is a propagation rule for problem P over S. This can be convenient when S does not have any, or any efficient, propagation algorithms. In fact, in such cases, we can resort to the propagation algorithms of $\bar{S}$ to perform propagation also over S.

Notice however that, when S has a non-idempotent multiplicative operator, function $\gamma(r(\alpha(P))) \otimes P$ could change the solution of P. To avoid this problem, we just have to follow the same reasoning as in the previous section, that is, to replace such a function with a function with just inserts into P the inconsistencies of $\gamma(r(\alpha(P))$. We will denote such a function by using a different combination operator: $\otimes_0$. Thus the function to be used in these cases is $\gamma(r(\alpha(P)) \otimes_0 P$. Notice that $\otimes_0$ is a non-commutative operator, since it inserts into the right operand the zeros of the left operand.

This results however hold only when the abstraction is order-preserving. We recall that this means that applying the abstraction function and then combining gives elements which are in the same ordering as the elements obtained by combining only. In particular, if two abstract elements $\alpha(x)$ and $\alpha(y)$ are ordered, then also x and y are ordered as well, and in the same direction.

Theorem 1. *Given an order-preserving abstraction $\langle \alpha, \gamma \rangle$ between semiring S and $\bar{S}$, assume that S has an idempotent multiplicative operation and consider any propagation rule r in $\bar{S}$ and any problem P over S. Then the function $f(P) = \gamma(r(\alpha(P))) \otimes P$ is a propagation rule for P.*

Proof. By definition, a propagation rule is an intensive, monotone, and idempotent function which takes a problem and returns an equivalent problem over the

same semiring. Since $\otimes$ is intensive, also f is so. Moreover, by monotonicity of γ, r, α, and $\otimes$, also f is monotone.

For proving idempotence of f, we need the order-preserving property of the abstraction. In fact, consider what happens when applying function f to P: some tuple values in $\alpha(P)$, say $\bar{v} = \alpha(v)$, will not be changed by r, while others will receive a lower value, say $\bar{v}' = r(\bar{v})$. By order-preservation, the new tuples values in the concrete semiring (that is, $\gamma(r(\alpha(v))) \times v$), are equal or lower than the original values. Let us now apply function f again. Function α will bring these new concrete values to either $\bar{v}$ (if we start from v) or $\bar{v}'$ (if we start from $\gamma(r(\alpha(v)))$). In any case, r will bring such values to $\bar{v}'$, for Lemma 1 (see below). Thus f is idempotent. Finally, f returns an equivalent problem by the theorem depicted in Figure 5.

Lemma 1. *Consider any SCSP P over S and any propagation rule r for P, with $r(P) = P'$. Then, taken any SCSP P'' such that $P' \leq_S P'' \leq_S P$, we have $r(P'') = P'$.*

Proof. Any rule r solves a subproblem $\langle C, con\rangle$ and changes the values of the tuples connecting the variables in *con*. Thus the result of applying r is a new constraint over *con*: $(C \otimes C_{con}) \Downarrow_{con}$, where C_{con} is the original constraint connecting the variables in *con*. This can also be written as $C_{con} \otimes C \Downarrow_{con}$. Let us now take any C''_{con} such that $(C_{con} \otimes C \Downarrow_{con}) \leq_S C''_{con} \leq_S C_{con}$. We can now multiply all these three constraints by $C \Downarrow_{con}$, obtaining: $(C_{con} \otimes C \Downarrow_{con} \otimes C \Downarrow_{con}) \leq_S (C''_{con} \otimes C \Downarrow_{con}) \leq_S (C_{con} \otimes C \Downarrow_{con})$. By idempotence of $\otimes$, we get: $(C_{con} \otimes C \Downarrow_{con}) \leq_S (C''_{con} \otimes C \Downarrow_{con}) \leq_S (C_{con} \otimes C \Downarrow_{con})$. Thus we have that $(C_{con} \otimes C \Downarrow_{con}) = (C''_{con} \otimes C \Downarrow_{con})$.

We can now prove a similar result for the case of a non-idempotent multiplicative operation in the concrete semiring. However, as noted above, we cannot combine the new problem with the old one, but we can just insert the zeroes of the new problem into the new one.

Theorem 2. *Given an order-preserving abstraction $\langle \alpha, \gamma\rangle$ between semiring S and $\bar{S}$, assume that S has a non-idempotent multiplicative operation and consider any propagation rule r in $\bar{S}$ and any problem P over S. Then the function $f(P) = \gamma(r(\alpha(P))) \otimes_0 P$ is a propagation rule for P, where $\otimes_0$ inserts the zeroes of its left operand into the right one.*

The proof of this theorem is similar to the previous one, and for the equivalence it refers also to the theorem depicted in Figure 8.

By taking several propagation rules in the abstract semiring, we can thus obtain an equal number of propagation rules over the concrete semiring. This set of rules can then be used to perform constraint propagation over a concrete problem. Notice however that, while an idempotent multiplicative operation in the concrete semiring allows us to use such rules until stability, with all the desired properties (equivalence, uniqueness, and termination), in the case of a non-idempotent multiplicative operation we can just apply the various propagation rules once each to insert several zeroes into the original problem (with the same properties as above).

7 Conclusions and future work

In this paper we have presented a framework for abstracting soft constraints, and we have shown how to use it to import propagation rules from the abstract setting to the concrete one.

An experimental phase is necessary to check the real practical value of our proposal. We plan to perform such a phase within the `clp(fd,S)` system developed at INRIA [9], which can already solve soft constraints in the classical way (branch-and-bound plus propagation via partial arc-consistency).

Another line for future research concerns the generalization of our approach to include also domain and topological abstractions, as already considered for classical CSPs.

We also plan to investigate the relationship of our notion of abstraction with several other existing notions currently used in constraint solving. For example, it seems to us that many versions of intelligent backtracking search could be easily modeled via soft constraints, by associating to each constraint some information about the variables responsible for the failure. Then, it should be possible to define suitable abstractions between the more complex of these frameworks and the simpler ones.

References

1. G. Birkhoff and S. MacLane. *A Survey of Modern Algebra*. MacMillan, 1965.
2. S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Abstracting soft constraints. In K. Apt, E. Monfroy, T. Kakas, and F. Rossi, editors, *Proc. 1999 ERCIM/Compulog Net workshop on Constraints*, Springer LNAI, 2000, to appear.
3. S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. In *Over-Constrained Systems*. Springer-Verlag, LNCS 1106, 1996.
4. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
5. Y. Caseau. Abstract Interpretation of Constraints on Order-Sorted Domains. In Proc. ILPS91, MIT Press, 1991.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977.
7. P. Cousot and R. Cousot. Systematic design of program analyis. In *Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979.
8. D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE, 1993.
9. Y. Georget and P. Codognet. Compiling semiring-based constraints with clp(fd,s). In M.Maher and J-F. Puget, editors, *Proc. CP98*. Springer-Verlag, LNCS 1520, 1998.
10. F. Giunchiglia, A. Villafiorita and T. Walsh. Theories of abstraction. *AI Communication*, 1997, vol.10, n. 3-4, pp. 167-176.
11. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 56(2-3):323–390, 1992.

12. S. de Givry, G. Verfaille, , and T. Schiex. Bounding The Optimum of Constraint Optimization Problems. In G. Smolka, editor, *Proc. CP97*, pages 405–419. Springer-Verlag, LNCS 1330, 1997.
13. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
14. A.K. Mackworth. Constraint Satisfaction. *Encyclopedia of AI (second edition)*, John Wiley & Sons, Stuart C. Shapiro ed., Vol. 1, pp. 285–293, 1992.
15. Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.
16. T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.
17. T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*, pages 631–637. Morgan Kaufmann, 1995.