Chapter 5

Soft Constraint Propagation

5.1 Introduction

5.1.1 Motivations

The standard approach to constraint programming, as in Chapters 2 and 4, assumes as constraints only the so-called crisp, hard, or classical constraints. As stated in Chapter 2 on p. 11, in the classical constraint programming paradigm a constraint can only assign one out of two values to an instantiation: either true, the instantiation is consistent with the constraint, or false, it is inconsistent. While this approach leads to efficient constraint solving and propagation algorithms, in some real life situations, it can be too restrictive, and a more expressive framework is regarded as more adequate and natural.

For instance, suppose a user is uncertain whether a constraint should either allow or forbid a certain instantiation; namely the user cannot decide, on the evidence of the available information, whether a certain set of values is consistent or inconsistent with the constraint. Then the user has only two alternatives, either yes or no, in a crisp constraint setting. Instead, in a soft constraint environment the user would be able to choose to what extent an instantiation can satisfy a constraint. This is the case of systems for user interface applications, based on constraint hierarchies (see [BFBW92, WB93, SMFBB93, BAFB96, Jam96, BMSX97, Bar97a, Bar97b, Hos98, Bar98, BFB98]): a user can set preferences on geometrical constraints, for example, between the cursor and the lines that the user wants to draw on a computer screen; some constraints will have a higher priority over the others, so that any optimal partial solution to the problem has always to satisfy them.

In general, soft constraints allow users to model, naturally, those real life problems which possess features like preferences, uncertainties, costs, levels of importance or absence of solutions.
5.1.2 Outline

There are several formalisations of soft constraint problems. In this thesis, we consider the one based on semirings [BMR97]. A semiring structure provides:

- a non-empty set of elements that represent the desired features, like uncertainties, preferences or others;
- a partial order to compare those and thus constraints;
- two operations for combining features and hence constraints: one returns the least upper bound with respect to the semiring order; whereas the other returns a lower bound with respect to the same order.

This formalism is a generalisation of existing approaches to soft CSPs. For instance, the semiring based framework is a generalisation of the following others: crisp CSPs, see Chapter 2; weighted or optimisation CSPs, see [DKL01]; valued CSPs, see [SFV95, dGVS97, Sch00]; some fuzzy ([DFP93, Rut94]) and probabilistic ([FL93]) CSPs.

In the literature, some of the standard constraint propagation algorithms for crisp CSPs were successfully extended, and adapted, to semiring based CSPs. This has lead to an algorithm schema, based on rules, for soft constraint propagation. At each step of that schema, a subproblem of the original input problem is solved, see [BMR97]. In this chapter, we briefly recall this rule based schema and show how it can be recast through iterations of functions, see Section 5.4 below.

Constraint propagation over crisp constraints was studied in depth in Chapter 4 by means of the SGI schema presented in Chapter 3. In this chapter, we prove that the SGI schema can also be instantiated to a number of soft constraint propagation algorithms: all those that are instances of the rule based schema, since this is generalised by the SGI schema, see Section 5.4: a series of algorithms which the original rule based schema cannot account for, see Section 5.5. In order to prove this, it is sufficient to define appropriate partial orders between soft CSPs, see Section 5.3.

Moreover, by analysing the types of functions that SGI iterates, we shall prove in Section 5.4 that soft constraint propagation can be enforced by means of functions which are not necessarily idempotent, as instead originally demanded by the rule based schema. Thus ours is a double generalisation: in fact, we neither require that functions for soft constraint propagation should solve a subproblem; nor that they should be idempotent, see Section 5.4. The relaxation of these assumptions allows us to account for further notions of soft constraint propagation that are not expressible through the rule based schema, see Section 5.5.

Therefore, the SGI algorithm provides a general schema for soft constraint propagation as well. Properties of the schema are applicable to all its instances.
In particular in Section 5.4 we apply SGI to soft CSPs and attack not so easy tasks, like the following, in the context of soft constraint propagation:

- Is it true that each execution of a soft constraint propagation algorithm always return the same result? When does it happen (see Subsection 5.4.3)?

- Under which conditions does every execution of a soft constraint propagation algorithm terminate (see Subsection 5.4.4)?

In Chapter 4, we prove that the SGI algorithm always terminates whenever it iterates inflationary functions over finite domains or constraints; this finiteness hypothesis turn out to be common to all the analysed constraint propagation algorithms in Chapter 4. However, when we deal with semiring based constraints the set of preferences (i.e., the semiring universe) can be infinite, although the domain and constraint set are finite: for example, probabilistic constraints can be defined through a semiring that contains all the real numbers in the ordered interval \([0, 1]\). Furthermore, the semiring structure for weighted constraints contains either all real or all natural numbers. Thus it is not realistic to assume that semiring based CSPs have finite constraints, hence we need to find another property than finiteness to guarantee the termination of the SGI schema.

Our first termination result is mainly concerned with the partial order over soft constraint satisfaction problems. It can be used to prove the termination of soft constraint propagation algorithms over weighted constraints, if preferences range over natural numbers and suitable functions are used for soft propagation. Our second result for the termination of SGI is related to the two semiring operations and its order. In turn, the latter result can be used to guarantee the termination of soft constraint propagation algorithms over probabilistic constraints.

Both the aforementioned results guarantee termination in a number of cases, however their hypotheses may be difficult to check. Nevertheless, when the second semiring operation coincides with the greatest lower bound operation, the semiring is also a distributive lattice; then all finitely generated sets, via the semiring operations, are finite; thereby, functions for constraint propagation, expressed through the semiring operations, can only return finitely many values. For instance, this is the case of crisp constraints, and also that of fuzzy constraints, e.g. functions for constraint propagation are combinations of max and min on the interval \([0, 1]\) of real numbers. Thus in such cases we obtain a third termination result, whose range of applicability is the most restricted, but whose hypotheses are the easiest to check.

### 5.1.3 Structure

The chapter is organised as follows. First, Section 5.2 introduces semirings, the semiring based formalism for soft constraints, and its basic operations on constraints. Section 5.3 treats some orders among semirings, constraints, and problems, necessary for defining soft constraint propagation via rules and SGI. So, in
Section 5.4, the SGI algorithm schema is extended to soft CSPs, and is proved to encompass the rule based schema for soft constraint propagation. Subsection 5.4.4 is concerned with the termination of the SGI schema. Finally, in Section 5.5 we display some arc constraint propagation algorithms and study them via SGI, and in Section 5.6 we discuss some limitations of our approaches and possible future directions.

5.2 Soft Constraints

In the semiring based formalism of [BMR97], a soft constraint is like a classical constraint, namely a relation, such that each of its tuples gets assigned a preference value. So, if we recast relations through their respective characteristic functions, passing from crisp to soft constraints means allowing the characteristic function of a constraint to range over more values than just $T$ (true) and $\bot$ (false). Once additional values are provided for constraints, suitable operations for their combination and comparison have to be provided as well. Semiring structures, as characterised below, give us all those ingredients.

5.2.1 Constraint Semirings

Constraint semirings and lattices

A constraint semiring, briefly c-semiring, is a structure $\mathcal{S} := \langle S, <_S, \times, \bot, T \rangle$ that enjoys the following properties:

- $(S, <_S, \bot, T)$ is a complete lattice, with bottom $\bot$, and top $T$;
- $\times$ is a binary operation on $S$, which is commutative, associative, has $\bot$ as absorbing element and $T$ as unit one. Moreover, $\times$ distributes over the least upper bound operation, denoted with $\lor$.

Notice that the above definition is not the original one that is adopted in ib., nevertheless ours is equivalent to the latter. In this thesis, we prefer the lattice characterisation of c-semirings because it directly highlights the partial order relation, which plays a central role in the study of termination for soft constraint propagation.

Some useful properties of c-semirings

The greatest lower bound operation and $\times$ are related. In fact, if $\land$ denotes the greatest lower bound operation of $S$, then

$$a \times b \leq_S a \land b,$$
for every pair of elements $a$ and $b$ in $S$. The above relation holds if both the following ones do:

$$a \times b \leq_S a \text{ and } a \times b \leq_S b.$$  

To prove the above relations, it is sufficient to ascertain that $\times$ enjoys the following property:

$$a \times c \leq_S a \quad (5.1)$$

for every pair of elements $a$ and $c$ in $S$; this amounts to saying that $\times$ is inflationary with respect to the reverse $\geq_S$ order relation. Now, the relation (5.1) holds iff $\times$ is monotone with respect to $\leq_S$. This can be proved to hold iff the following relation holds:

$$a \times c \leq_S a \times b \text{ if } c \leq_S b. \quad (5.2)$$

Then we exploit the fact that the top element $T$ is the unit of $\times$ and infer (5.1). So we are left to prove the monotonicity relation as in (5.2). But this is easy, because the hypothesis $c \lor b = b$ yields the equality

$$a \times (c \lor b) = a \times b,$$

and because $\times$ distributes over $\lor$.

We collect all the above results and some well-known facts concerning the least upper bound operation $\lor$ in the following lemma, as they will be used over and over in this chapter; see also \textit{ib}.

\textbf{Lemma 5.2.1.}

- For every pair of elements $a$ and $b$ in $S$, we have that $a \times b \leq_S a \land b$.
- The $\times$ operation is inflationary with respect to $\geq_S$, whereas $\lor$ is inflationary with respect to $\leq_S$: i.e., $a \leq_S a \lor c \text{ and } a \times c \leq_S c$, for every $a$ and $c$ in $S$.
- Both $\times$ and $\lor$ are monotone with respect to $\leq_S$: i.e., $a \times b \leq_S c \times d$, $a \lor b \leq_S c \lor d$ whenever $a \leq_S c$ and $b \leq_S d$.
- The $\lor$ operation is idempotent: i.e., $a \lor a = a$, for every $a \in S$. If $\times$ is idempotent as well, then this coincides with $\land$ and the $c$-semiring is a complete distributive lattice.

\textbf{Proof.} We are only left to prove the second part of the last item. So, let us assume that $\times$ is idempotent. Then the relations

$$\begin{align*}
\begin{cases}
a \land b \leq_S a, \\
a \land b \leq_S b,
\end{cases}
\end{align*}$$

and the monotonicity of $\times$ with respect to $\leq_S$ entail the relation

$$(a \land b) \times (a \land b) \leq_S a \times b. \quad (5.3)$$
By hypothesis, $\times$ is idempotent, hence $(a \land b) \times (a \land b)$ is equal to $a \land b$. This and (5.3) yield the relation

$$a \land b \leq_S a \times b.$$ 

Thus the equality $a \land b = a \times b$ follows now from the last relation above and its reverse as in the first item. \hfill \Box

### 5.2.2 Soft Constraints

Given a c-semiring $S := \langle S, <_S, \times, \bot, \top \rangle$, a domain scheme $X$ and a domain $D$ over $X$ (see Chapter 2), we can now define an $S$ soft constraint $C(s)$, over a scheme $s$ of $X$ and the domain $D$ on $X$, as a function

$$C(s) : D [s] \to S.$$ 

Intuitively, $C(s)$ provides each tuple $d \in D [s]$ with a preference value in the c-semiring universe $S$.

A soft constraint satisfaction problem (SCSP) on $S$ is a structure $P = \langle X, D, C \rangle$ that is defined as follows:

- $X$ is a variable scheme and $D$ is a domain set over $X$,
- $C$ is a set of $S$ soft constraints over schemes in $X$.

Whenever the semiring $S$ to which we refer is clear from the context, we shall not mention it; hence we shall usually write soft constraints and SCSP.

### 5.2.3 Examples

#### Crisp CSPs

Crisp CSPs are SCSPs for which the chosen c-semiring is the Boolean algebra

$$\text{Bool} = \langle \{ \top, \bot \}, \leq, \land, \top, \bot \rangle,$$

in which $\leq$ is defined via the Boolean disjunction: $a \leq b$ iff $a \lor b = b$.

By means of $\text{Bool}$ we can associate a Boolean value, either $\bot$ (false) or $\top$ (true), with each tuple of elements in $D$. So here a constraint over $\text{Bool}$ corresponds to the characteristic function of a crisp constraint as in Chapter 2.

#### Weighted CSPs

Weighted or optimisation CSPs over natural numbers (see [DKL01]) can be defined by means of the c-semiring

$$\text{Weight} = \langle \mathbb{N}, \leq_N, +, 0, +\infty \rangle,$$

in which $\leq_N$ is the standard total order over the set of natural numbers.
5.2. Soft Constraints

Example 5.2.2. Reconsider the map colourability problem as in Subsection 2.3.1 and suppose now to have exactly two colours, *aqua* and *blue*, for every variable $x_1$, $x_2$ and $x_3$ of the problem. Clearly, the problem does not admit any solution if expressed as a crisp CSP. But suppose now that we want to find an optimal solution; in that, we want to maximize the number of satisfied constraints. Then it is sufficient to recast the same constraints we had in Subsection 2.3.1 as weighted constraints. Each of them will assign 0 to the pairs *(aqua, aqua)* and *(blue, blue)*, and 1 to all the remaining pairs. In this case, given a total assignment, we shall count the number of constraints it satisfies (via $\sum$), and if this this number is the maximum (with respect to $\leq$) we shall regard this assignment as a "solution" to this weighted map colourability problem.

Probabilistic and fuzzy CSPs

Probabilistic CSPs with minimum and maximum (see [FL93]) can be defined via the c-semiring

$$\text{Prob}(\text{min}, \text{max}) = ([0,1], \geq_\mathbb{R}, \text{max}, 0, 1),$$

in which $\geq_\mathbb{R}$ is the standard linear order over real numbers and max the corresponding maximum operator.

On the other hand, fuzzy CSPs with maximum and minimum (see [DFP93, Rut94]) can be defined via the c-semiring

$$\text{Fuzzy}(\text{max}, \text{min}) = ([0,1], \leq_\mathbb{R}, \text{min}, 0, 1),$$

in which $\leq_\mathbb{R}$ is the standard linear order over real numbers and min the corresponding minimum operator.

Example 5.2.3. Let us consider the map colourability problem as in Example 5.2.2 again, and suppose that now we have a slight preference towards any assignment that gives $x_3$ the colour *aqua*; that is, we are not happy with maximising the number of satisfied constraints, but we have a preference towards a specific colour for a specific country. Then we could express this preference by recasting $C(x_1, x_3)$ and $C(x_2, x_3)$ as fuzzy constraints, so that these assign a greater value in the interval $[0,1]$ to the tuple *(blue, aqua)* than to *(aqua, blue).* A "solution" is such that the returned value for the assignment *aqua* to $x_3$ is the maximum over all the minimum values for the assignments in which the colour *aqua* is given to $x_3$.

5.2.4 Basic Operations on Soft Constraints

Having defined soft constraints, we can now extend the basic operations for crisp constraints, namely projection and join, to analogous operations for soft ones. These are easily provided by means of the c-semiring operation $\times$ and the least upper bound one; see [BMR97], where join is called combination.
Join

Given two constraints, $C_1 := C_1(s_1)$ and $C_2 := C_2(s_2)$, their join $C_1 \Join C_2$ is the constraint over the scheme $t := s_1 \cup s_2$ such that

$$C_1 \Join C_2 (d) := C_1(d[s_1]) \times C_2(d[s_2]),$$

for every $d \in D[t]$ (remember that $d[s_i]$ is the projection of the tuple $d$ over $s_i$).

In other words, the join of two constraints assigns, to each tuple, a value that is the product (via the c-semiring $\times$) of the respective values returned by the two joint constraints.

The join operation is associative, since $\times$ is. Therefore $\Join$, which is defined as a binary operation, is easily extended to an operation over any finite number of constraints.

**Example 5.2.4.** In Example 5.2.3 the join of the two fuzzy constraints $C_1 := C(x_1, x_3)$ and $C_2 := C(x_2, x_3)$ is a constraint on $C(x_1, x_2, x_3)$ that assigns, for instance, to $(aqua, blue, aqua)$ the minimum between the values $C_1(aqua, aqua)$ and $C_2(blue, aqua)$. While, in the case of Example 5.2.2, the join of the constraints $C_1$ and $C_2$, now interpreted over the weighted c-semiring with natural numbers, is the sum of $C_1(aqua, aqua)$ and $C_2(blue, aqua)$.

Projection

Given a constraint $C := C(s)$ and a scheme $t$ of $s$, the projection of $C$ over $t$, denoted by $\Pi_t(C)$, is the unique constraint over $t$ such that

$$\Pi_t(C)(d) := \bigvee \{C(e) : e \in D[s] \text{ and } e[t] = d\}.$$ 

So, the projection of a constraint over $s$ assigns, to each tuple $d$, the least upper bound of the values assigned, by the original constraint, to all the tuples that are equal to $d$ when projected over $s$.

**Example 5.2.5.** Let us consider Example 5.2.3 and assume that $C := C(x_1, x_3)$ assigns $1/2$ to $(aqua, blue)$, then $1$ to the most preferred assignment $(blue, aqua)$, and $0$ to all the other ones. Then the projection of $C$ over $x_3$ is the constraint $C(x_3)$ on $x_3$ that assigns $1$ to $aqua$, and $1/2$ to $blue$. If $C$ assigns $+\infty$ to $(blue, aqua)$, $1$ to $(aqua, blue)$, $0$ to the remaining pairs, and is hence recast as a weighted constraint as in Example 5.2.2, then the projection of $C$ over $x_3$ will assign $+\infty$ to $aqua$ and $1$ to $blue$.

### 5.2.5 Solutions and Equivalent Problems

**Solutions**

In case of a crisp CSP $P := (X, D, C)$, a solution to $P$ is a tuple of $D$ that belongs to each constraint in $C$, see Chapter 2. In other words, a solution is
5.2. Soft Constraints

a tuple to which every $P$ constraint assigns the value true, namely $T$, in the Boolean structure $\text{Bool}$, see Subsection 5.2.3.

We repeat the same procedure for computing soft solutions. First, we need to complete the soft problem $P$ and obtain its completion $\bar{P}$, like in Subsection 2.4.2 in case of crisp CSPs:

- if there is more than one constraint on a scheme $s$, say $C_1, \ldots, C_k$, then we replace them with the constraint on $s$ equal to $\bigwedge_{i=1}^{k} C_i$ — that corresponds to intersection in the crisp case;

- if there are no constraints on a scheme $s$ in $P$, then we create a new constraint $C := C(s)$ that assigns the top value to every tuple in $D_s$, namely $C(d) := T$, for every $d \in D_s$.

Consider the completion $\bar{P} = (X, D, C)$ of an SCSP $P$. Then the solution to $P$ is the constraint

$$\text{Sol}(P) = \bigwedge_{C \in P} C,$$

that is the constraint which assigns, to each tuple, the product via $\times$ of the values assigned by the constraints in the completed problem.

If we are only interested in a solution over a scheme $s$ of $X$, then we obtain a solution to $P$ over $s$ by projecting $\text{Sol}(P)$ over $s$; so we define it as

$$\text{Sol}(P, s) = \Pi_s \text{Sol}(P).$$

Clearly, $\text{Sol}(P) = \text{Sol}(P, X)$.

Equivalence

In the case of crisp constraints, two CSPs on the same scheme are equivalent if they have the same set of solutions; see Definition 2.4.1. A similar characterisation of equivalence among problems is found in the semiring based case, with the requirement that the problems share the same c-semiring and domain set.

Two SCSPs on the same c-semiring $S$, scheme $X$ and domain set $D$, say $P_1 := (X, D, C_1)$ and $P_2 := (X, D, C_2)$, are equivalent if $\text{Sol}(P_1) = \text{Sol}(P_2)$. In this case, we write $P_1 \equiv P_2$.

Since two equivalent SCSPs are defined on the same c-semiring and domain set, they also have equal solutions on each scheme of $X$: i.e., their equivalence yields

$$\text{Sol}(P_1, s) := \text{Sol}(P_2, s) \quad (5.4)$$

for every scheme $s$ of $X$. Notice that, if the two domains were different, there would be no guarantee for (5.4) to hold. Indeed, the concept of equivalence and, more specifically, that of solution in case of SCSPs are interesting and subtle issues, see [BCR00, Gen01a, Gen01b].
Chapter 5. Soft Constraint Propagation

Example 5.2.6. The solution to the max-min fuzzy CSP $P$ of Example 5.2.3 assigns, to each state $x_i$ and colour for $x_i$, a value in $[0,1]$. For instance, let us consider the tuple of colours $(aqua, blue, aqua)$ for $x_1, x_2, x_3$. If $C(x_1, x_2)$ assigns $1/2$ to $(aqua, blue)$, $C(x_2, x_3)$ assigns $1$ to $(blue, aqua)$ and $C(x_1, x_3)$ assigns $1/3$ to $(aqua, aqua)$, then $Sol(P)$ will assign the minimum of all these values, i.e. $1/3$, to $(aqua, blue, aqua)$. Suppose that $Sol(P)$ assigns $1/4$ to $(aqua, blue, blue)$. Then $Sol(P, (x_1, x_2))$ will assign $1/3$, i.e. the maximum between $1/3$ and $1/4$, to $(aqua, blue)$. 

5.3 Soft Orders

As in the case of crisp CSPs, we need partial orders on SCSPs, to compare both them and the iterations of functions defined on them. In the crisp case, partial orders are based on the subset relation, see Subsection 2.5.2. For instance, in the crisp case, we can compare the constraint $C_1(s)$ and $C_2(s)$ if $C_1(s) \subseteq C_2(s)$. If we recast the subset relation in terms of characteristic functions, we can rewrite it as follows:

$$\chi_{C_1}(d) \leq \chi_{C_2}(d),$$

for every $d$ in the domain on $s$, where $\leq$ can only compare $\top$ and $\bot$.

Now, there is an obvious candidate for $\leq$ in (5.5) when we generalise crisp to semiring based constraints: that is, the partial order relation $\leq_S$ of the $c$-semiring.

Following this idea, we first introduce a partial order relation among constraints via $\leq_S$, see Subsection 5.3.1; then we lift such order to a partial order on constraint sets in Subsection 5.3.2, and finally to problems, see Subsection 5.3.3.

In order to simplify the discussion, we adopt a common convention when dealing with soft constraints: we always assume that SCSPs are complete. If necessary, incomplete SCSPs can first be completed.

Convention 5.3.1. In the remainder of this chapter, we always assume that every SCSP $P$ is complete.

The above assumption, strictly speaking, is not necessary for the results in the remainder of this chapter. However, it simplifies our discussion and notation.

5.3.1 Constraint Order

Given the partial ordering $\leq_S$ of a $c$-semiring $S$, we can define a new partial order relation among constraints, as follows.

Definition 5.3.1. Let $S := \langle S, \leq_S, \times, \bot, \top \rangle$ be a $c$-semiring, $X$ a scheme and $D$ a domain set over $X$. Then consider two constraints $C_1 := C_1(s)$ and $C_2 := C_2(s)$ over the scheme $s$ of $X$. We write $C_1 \sqsubseteq_S C_2$ if the following conditions are both satisfied:
1. for all the tuples \( d \in D[s] \), \( C_2(d) \leq_s C_1(d) \);

2. there exists a tuple \( d \in D[s] \) for which \( C_2(d) <_s C_1(d) \).

We write \( C_1 \sqsubseteq_s C_2 \) in case only the first relation holds.

In other words, the constraint \( C_1 \) is smaller than or equal to \( C_2 \) in the order \( \sqsubseteq_s \) if the former constraint assigns, to each tuple, either the same value as \( C_2 \), or a greater value with respect to \( <_s \). Loosely speaking, \( C_2 \) is preferred to \( C_1 \) if the former constraint is possibly more restrictive than the latter, loosely speaking.

**Theorem 5.3.2.** The relation \( \sqsubseteq_s \) is a partial order among constraints defined on the same c-semiring \( S := \langle S, <_S, \times, \bot, \top \rangle \), scheme \( X \) and domain set \( D \).

**Proof.** We need to prove that \( \sqsubseteq_s \) is a reflexive, antisymmetric and transitive relation. Reflexivity holds trivially. To prove antisymmetry, suppose that \( C_1 \sqsubseteq_s C_2 \) and \( C_2 \sqsubseteq_s C_1 \); this yields that both constraints share the same scheme, say \( s \). Now, for all tuples \( d \in D[s] \), we have both \( C_1(d) \leq_s C_2(d) \) and \( C_2(d) \leq_s C_1(d) \), hence \( C_1(d) = C_2(d) \). Therefore, \( C_1 = C_2 \). The transitivity of \( \sqsubseteq_s \) follows from the transitivity of \( \leq_s \).

### 5.3.2 Constraint Set Order

We can now easily extend the order \( \sqsubseteq_s \) over constraints to a new order between *constraint sets* as follows.

**Definition 5.3.3.** Let \( S := \langle S, <_S, \times, \bot, \top \rangle \) be a c-semiring, \( X \) a scheme and \( D \) a domain set over \( X \). Consider two sets of constraints, \( C_1 \) and \( C_2 \), over \( X, D \) and \( S \). We write \( C_1 \sqsubseteq_C C_2 \) if the following two properties both hold:

1. there exist precisely one constraint \( C_1(s) \) in \( C_1 \) on \( s \), and precisely one constraint \( C_2(s) \) in \( C_2 \) on \( s \), for each scheme \( s \) of \( X \);

2. for each scheme \( s \) on \( X \), we have that the relation \( C_1(s) \sqsubseteq_s C_2(s) \) holds between the constraints of \( C_1 \) and \( C_2 \) on \( s \).

The intuitive reading of \( C_1 \sqsubseteq_C C_2 \) is that the problems that \( C_2 \) yield are at least as constraining as those of \( C_1 \), because \( C_2 \) has at least as many more or equally restrictive constraints as \( C_1 \) has, loosely speaking.

**Theorem 5.3.4.** Consider a collection \( \mathcal{C} \) of constraint sets on the same c-semiring \( S := \langle S, <_S, \times, \bot, \top \rangle \), scheme \( X \) and domain set \( D \). Then the relation \( \sqsubseteq_C \) is a partial order between constraint sets in \( \mathcal{C} \).
Chapter 5. Soft Constraint Propagation

Proof. Reflexivity trivially holds. As for antisymmetry, suppose that both \( C_1 \sqsubseteq C_2 \) and \( C_2 \sqsubseteq C_1 \) hold. Thus both the following relations hold for each scheme \( \mathcal{S} \) on \( X \): \( C_1(s) \sqsubseteq_{\mathcal{S}} C_2(s) \) and \( C_1(s) \sqsubseteq_{\mathcal{S}} C_2(s) \), for \( C_1 \in C_1 \) and \( C_2 \in C_2 \). Hence \( C_1(s) = C_2(s) \) for each scheme \( s \) on \( X \), because \( \sqsubseteq_{\mathcal{S}} \) is a partial order relation, see Theorem 5.3.2. Transitivity follows similarly, by exploiting the transitivity of \( \sqsubseteq_{\mathcal{S}} \).

5.3.3 Problem Orderings

At this point, we know two partial orders for SCPS: a partial order among constraints \( (\sqsubseteq_{\mathcal{S}}) \); this lifted to constraint sets \( (\sqsubseteq_{\mathcal{C}}) \). However, constraint propagation algorithms have SCSPs in input; therefore, we need a partial order relation among SCSPs to enforce soft constraint propagation by means of the SGI algorithm.

**Definition 5.3.5.** Given a c-semiring \( S := \langle S, <_S, x, \bot, \top \rangle \), consider two problems over it, with the same scheme \( X \) and domain set \( D \); say \( P_1 = (X, D, C_1) \) and \( P_2 = (X, D, C_2) \). Then we write \( P_1 \sqsubseteq_P P_2 \) if \( C_1 \sqsubseteq_{\mathcal{C}} C_2 \).

**Note 5.3.6.** Notice how the above relation does not involve the domain set \( D \) at all; the comparison of semiring-based problems takes place at the level of constraint sets. Compare it to the definition of partial orders on CSPs, see Subsection 2.5.2. There are at least two reasons for such a difference, the latter following from the former:

1. the definition of equivalence among SCSPs is meaningful only if the problems share the same variable domains, see Subsection 5.2.5;

2. arc consistency for SCSPs modifies unary constraints instead of variable domains; in general, constraint propagation algorithms for SCSPs do not alter the domains of variables.

We are in the position to define a partially ordered structure that contains all of the SCSPs that constraint propagation algorithms step-by-step compute, starting from their input SCSP. The following definition is a generalisation of Definition 2.5.2 to the case of semiring based soft constraints.

**Definition 5.3.7.** Consider a c-semiring \( S := \langle S, <_S, x, \bot, \top \rangle \) and an SCSP \( P := (X, D, C) \) over it. Then the soft closure of \( P \), denoted by \( P^\uparrow \), is the class of all problems \( P' \) on \( S, X \) and \( D \) such that the relation \( P \sqsubseteq_P P' \) holds.

**Theorem 5.3.8.** Consider a c-semiring \( S := \langle S, <_S, x, \bot, \top \rangle \), and an SCSP \( P := (X, D, C) \) over it. Then the following statements hold:

- \( \langle P^\uparrow, \sqsubseteq_P \rangle \) is a partial ordering;
5.3. Soft Orders

- the bottom of \((P \uparrow, \subseteq_P)\) is \(P\) itself.

**Proof.** We prove the first claim, the other one is immediate from the definition of \(P \uparrow\). As usual, we only prove that the relation is antisymmetric, because transitivity can be proved similarly and reflexivity is trivial. Hence, suppose that both \(P_1 \subseteq_P P_2\) and \(P_2 \subseteq_P P_1\) hold between two SCSPs in \(P \uparrow\). So both those SCSPs are defined on the same scheme \(X\) and domain set \(D\). Moreover, \(C_1 \subseteq_C C_2\) and \(C_2 \subseteq_C C_1\). From the two last relations and Theorem 5.3.4, we infer the identity \(C_1 = C_2\). Thus \(P_1 = P_2.\)

Again, we have an analogous result for crisp and soft CSPs.

**Fact 5.3.9.** Consider an SCSP \(P\) over a c-semiring \(S\), and its soft closure \(P \uparrow\). Then the following statements hold:

1. if \(P_1 \subseteq_P P_2\) and \(P_1 \in P \uparrow\), then \(P_2 \in P \uparrow\);
2. if \(P_1 \subseteq_P P_2\), then \(P_2 \uparrow \subseteq P_1 \uparrow.\)

The above statement has a counterpart in the crisp case: Fact 2.5.3.

We have now obtained a partial ordering \((P \uparrow, \subseteq, P)\), for every problem \(P\). The problems in \(P \uparrow\) differ at most in their constraint components. However, such a partial ordering on a SCSP \(P\) may contain too many problems with respect to those that specific soft constraint propagation algorithms step-by-step compute, starting from the given SCSP \(P\).

We faced a similar situation with crisp CSPs: as our analysis in Chapter 4 clarifies, the closure of a problem carries over too many subproblems, that are not iterated by certain constraint propagation algorithms. So, in Subsection 2.5.2, we carved out those orderings — each based on a different subfamily of \(P \uparrow\) — that are of interest in the analysis of constraint propagation algorithms as in Chapter 4.

In the case of crisp CSPs, we distinguished between domain orderings (see p. 24) and constraint orderings (see p. 25) on CSPs. We do not have this distinction for SCSPs, since soft problem orderings are based on the comparison of constraint sets only; see also Note 5.3.6. So all we shall need in the analysis of soft constraint propagation is the following refinement of the problem ordering on SCSPs.

**Definition 5.3.10.** Let \(P\) be a problem over a c-semiring \(S\), and consider its soft closure \(P \uparrow\). Assume that \(\mathcal{F}(P)\) is a family of problems \(P' \in P \uparrow\). If \(P\) belongs to \(\mathcal{F}(P)\), then the structure

\[ \langle \mathcal{F}(P), \subseteq, P \rangle \]

is called a **problem ordering** on \(P\).

Given the above definition, any subset of \(P \uparrow\), to which the complete problem \(P\) belongs, gives rise to a problem ordering on \(P\). This generality will prove helpful when studying termination conditions, see Subsection 5.4.4 below.
5.4 Soft Constraint Propagation via SGI

Our goal in this section is to analyse soft constraint propagation. First we define a schema for constraint propagation which is based on functions that are called “rules”, as originally defined in [BMR97], see Subsection 5.4.1; each of those rules is limited to a combination of projection and join, in that they solve a subproblem of the input problem. Then we extend soft constraint propagation based on rule iterations by means of SGI iterations — see also Subsection 5.4.2 — and conclude by tackling the issue of termination for soft constraint propagation algorithms in Subsection 5.4.4.

5.4.1 Soft Constraint Propagation via Rules

Several traditional constraint propagation algorithms for crisp CSPs can be extended to SCSPs. For this purpose, the notion of constraint propagation rule was introduced in ib.

Rule iterations

Let us consider a c-semiring \( S := \langle S, \cdot, \cdot, \perp, \top \rangle \), a domain \( D \) over \( X \) and a constraint \( C'(s) \) on a scheme \( s \) of \( X \) and the domain \( D \). Assume that \( P := \langle X, D, C \rangle \) is a problem over \( S \), and \( C(s) \) is the constraint on \( s \) of \( P \). Then \( P[C(s)/C'(s)] \) denotes the problem that differs from \( P \) only in the constraint on \( s \), which is set equal to \( C'(s) \).

Let \( P \) be an SCSP as above; then consider a scheme \( s \) of \( X \) and a scheme \( t \) of \( s \) itself. A constraint propagation rule \( r_t \) for \( P \), on \( s \) and \( t \), is a function on \( P^\top \) that is defined as follows:

\[
r_t^r(P') := P'[C(t)/\Pi_t \text{Sol}(P, s)],
\]

for any \( P' \in P^\top \).

In other words, the application of \( r_t^r \) to \( P' \in P^\top \) adds the constraint \( \Pi_t \text{Sol}(P', s) \) over the scheme \( t \) to \( P' \). That constraint is obtained by combining all the constraints of \( P' \) on \( s \), and then projecting the resulting constraint over \( t \).

Once a c-semiring \( S := \langle S, \cdot, \cdot, \perp, \top \rangle \) and an SCSP \( P := \langle X, D, C \rangle \) are given, a rule based on \( X \) is any rule \( r_t \) for \( P \), on two schemes, \( s \) of \( X \), and \( t \) of \( s \).

The application of a sequence of rules to a problem is obtained by composing the rules in their order of occurrence in the sequence. An infinite sequence of rules from a finite set \( R \) is fair if each rule of \( R \) occurs in the sequence infinitely often.

So, given a problem \( P := \langle X, D, C \rangle \) and a finite set of rules \( R \) on \( X \), rule-based constraint propagation \( R \) on \( X \), starting from \( P \), is defined as a fair iteration of rules from \( R \) so that the first rule in the sequence is applied to \( P \). The following statement concerns itself with some properties of constraint propagation via rules.
5.4. Soft Constraint Propagation via SGI

Theorem 5.4.1 ([BMR97]). Consider a rule-based constraint propagation \( R \), starting from an SCSP \( P \). If \( R \) stabilises at \( P' \), then \( P' \) is a common fixpoint of all the rules from \( R \). This common fixpoint \( P' \) is both unique and equivalent to \( P \), provided that \( \times \) is idempotent.

Notice that fairness is assumed to ensure that the problem at which the iteration stabilises is a common fixpoint of all the rules; and the idempotency of \( \times \) is used to ensure both the uniqueness of the result and the equivalence.

However, the additional hypothesis of fairness is redundant for the SGI algorithm schema, as its task is accomplished by the update operator, see Subsection 5.4.2. So we shall drop the assumption of fairness and only restrict our attention to bare iterations of rules. Still, we shall be able to prove that the stabilisation point is always unique due to the monotonicity of rules, see Subsection 5.4.3. In the following part, we list the main properties of rules — as functions — as they are used in Subsection 5.4.3.

Order-related Properties of Soft Constraint Propagation Rules

We recall, from Subsection 5.2.5, that two problems \( P_1 \) and \( P_2 \), defined on the same scheme and domain, are equivalent if they have the same solution constraint; if this is the case, we write \( P_1 \equiv P_2 \).

The following lemma lists the main properties of soft constraint propagation rules that are related to the SCSP order, see Definition 5.3.5.

Lemma 5.4.2. Consider a problem \( P := \langle X, D, C \rangle \) on a c-semiring \( S \), and a rule \( r \) based on \( X \).

- If \( P' \in P\downarrow \), then \( r(P') \) is still in \( P\downarrow \). So \( r \), defined on \( X \), is a function on \( P\downarrow \). Moreover, \( P' \equiv r(P') \) if \( \times \) is idempotent.

- Furthermore, \( P' \sqsubseteq_P r(P') \), for every \( P' \in P\uparrow \).

- Consider two SCSPs, \( P_1 \) and \( P_2 \) in \( P\uparrow \). If \( P_1 \sqsubseteq_P P_2 \), then \( r(P_1) \sqsubseteq_P r(P_2) \).

Proof. The second property states that rules are inflationary functions over any constraint ordering on \( P \). This is based on the inflationarity of \( \times \) and \( \vee \), as in Lemma 5.2.1, and the definition of SCSP order. Whereas the last property is concerned with the monotonicity of rules, and again follows from Lemma 5.2.1 and the definition of SCSP order. \( \square \)
5.4.2 Soft Constraint Propagation via the SGI Schema

The functions that are iterated in a rule based constraint propagation are of a specific form: they are obtained from join and projection, so that each of them solves a subproblem of the input problem. Whereas the SGI algorithm does not impose any specific request on the iterated functions. Functions can be of any sort in the SGI schema; so we are able to generalise rule-based constraint propagation as follows.

**DEFINITION 5.4.3.** Consider an SCSP problem $P$ over a c-semiring $S$. A soft constraint function for $P$ is a function $f : P^\uparrow \rightarrow P^\uparrow$.

Notice that a function $f$ on a family $\mathcal{F}(P)$ of $P^\uparrow$, i.e.,

$$f : \mathcal{F}(P) \rightarrow \mathcal{F}(P),$$

(5.6)

can be uniquely extended to a function on the whole set $P^\uparrow$: that is to a constraint function as in Definition 5.4.3. We only need to define it equal to the identity on every problem that does not belong to $\mathcal{F}(P)$. Vice versa, if $f$ is a constraint function and $\mathcal{F}(P)$ is closed for it as in (5.6), then $f$ can be restricted to a function over $\mathcal{F}(P)$. The distinction is only notational in this context, therefore we shall usually ignore it, and freely refer to any function for which (5.6) holds as a constraint function over $\mathcal{F}(P)$.

**Functions versus rules.** The characterisation of soft constraint function given in Definition 5.4.3 abandons a number of assumptions and attributes of rules and rule-based constraint propagations (see Subsection 5.4.1):

- soft constraint functions do not necessarily compute solutions to subproblems;
- soft constraint functions are neither assumed to be monotone and inflationary, nor idempotent if $\times$ is;
- the fairness assumption on iterations of soft constraint functions is not required.

The first generalisation is the most rewarding one, in that a number of constraint propagation algorithms do not exactly solve subproblems, but compute an approximation of its solutions; see, for example, the basic algorithm for bound-consistency in [MS98], or generalised arc consistency for SCSPs as in Section 5.5 below. Thus SGI for SCSPs allows us to instantiate more algorithms than the original rule based schema.

As for the fairness hypothesis, there is no loss in abandoning it, as update takes its role. In fact, as far as the update operator satisfies Axiom 3.3.1, see Chapter 3,
5.4. Soft Constraint Propagation via SGI

the SGI schema computes a common fixpoint of all the iterated functions. We rewrite that axiom and the SGI algorithm below for the reader’s convenience, and specialise them to SCSPs. Remember that $F_P$ is the $F$ subset that collects all the $F$ functions which could alter the input problem $P$.

**Axiom 5.4.1 (Common Fixpoint).** Let $F$ be a set of functions on $P^\uparrow$. Suppose that $g(P') \neq P'$, for $g \in F$ and $P' \in P$. Then the update operator adds to $G$ all the $F - G$ functions $f$ such that $f(P') = P'$ while $fg(P') \neq g(P')$.

**Algorithm 5.4.1: SGI($P, F_P, F$)**

\[\begin{align*}
o &:= P \% \text{ complete input problem;}
G &:= F_P;
\text{while } G \neq \emptyset \text{ do}
    & \quad \text{choose } g \in G;
    & \quad G := G - \{g\};
    & \quad o' := g(o);
    & \quad \text{if } o' \neq o \text{ then}
    & \quad \quad G := G \cup \text{update}(G, F, g, o);
    & \quad o := o'
\end{align*}\]

The following result is a corollary of Theorem 3.3.3 in Chapter 3, and it states the every execution of the SGI schema on SCSPs computes a common fixpoint of the iterated functions.

**Corollary 5.4.4.** Consider an SCSP $P$ on $S := (S, <_S, \times, \bot, T)$ and a finite set $F$ of functions on a family $\mathcal{F}(P)$ of $P^\uparrow$ problems, to which $P$ belongs. Then every terminating execution of the SGI algorithm that satisfies the Common Fixpoint Axiom 5.4.1, with input $P$ and $F$, computes a common fixpoint of all the functions from $F$.

5.4.3 The Role of Monotonicity

As in the case of crisp constraints, if an instance of SGI for soft SCSPs iterates functions which are monotone, with respect to a certain problem order, than that instance of SGI always returns the same common fixpoint of the iterated functions; i.e., the least problem, with respect to the assigned problem order, that is a common fixpoint of all the iterated functions. So let us first recast Axiom 3.3.2 for generic problem orderings — as for these, see Definition 5.3.10.

**Axiom 5.4.2 (Least Fixpoint).** The finite set $F$ only contains monotone constraint functions over a problem ordering $(\mathcal{F}(P), \sqsubseteq, P)$. 

\[\]
Given the above axiom, we infer the following corollary of Theorem 3.3.8.

**Corollary 5.4.5.** Consider an SCSP $P$ on $S := \langle S, <_S, \times, \bot, \top \rangle$, a finite set $F$ of functions on a problem ordering $\langle \mathcal{F}(P), \sqsubseteq_P, P \rangle$, and assume the Common Fixpoint Axiom 5.4.1 and the Least Fixpoint Axiom 5.4.2. Then every terminating execution of SGI, with input $P$ and $F$, computes the same problem: i.e., the least common fixpoint of all the $F$ functions with respect to the problem ordering. $\Box$

The above corollary is interesting in that it highlights the role of monotonicity in the confluence of SGI on SCSPs. This property for rule based constraint propagation was originally restricted in [BMR97] to the case of $c$-semirings with an idempotent $\times$ operation, see Theorem 5.4.1. Instead, Corollary 5.4.5 allows us to extend this to all executions of SGI with rules, independently of the idempotency of $\times$, or the fairness hypothesis.

**Corollary 5.4.6.** Let $R$ be a finite set of rules for an SCSP $P := \langle X, D, C \rangle$. Then all the terminating executions of SGI, with $R$ rules and $P$ as input, compute the same problem. This is the least common fixpoint of all the rules of $F$ with respect to the problem ordering on $P$.

**Proof.** Lemma 5.4.2 yields that any rule based on $X$ is an inflationary and monotone soft constraint propagation function on $\langle P^+, \sqsubseteq_P, P \rangle$. Corollary 5.4.5 implies now our claim. $\Box$

We now turn our efforts towards the issue of termination of SGI with SCSPs and soft constraint functions. This is not an easy task in case of SCSPs; in fact, even if SCSP domains are finite, the $c$-semiring may be infinite, which is obviously a source of possible non-termination. So, whereas for crisp CSPs inflationarity of functions and finite domains are feasible and sufficient assumptions to ensure the termination of SGI (see Corollary 3.3.10), we shall see, in the following subsection, that this is not the case for all SCSPs.

### 5.4.4 Termination

As remarked in the preceding subsection, the presence of an infinite $c$-semiring universe may lead to a constraint propagation algorithm which does not always terminate. In this subsection we focus on the issue of termination and investigate under which conditions the SGI schema *always* terminates. We obtain general termination conditions, that yield the termination of every soft constraint propagation algorithm that is an instance of SGI.
Problem orderings and termination

Our first termination result below, concerns itself with the problem order (see Definition 5.3.5): instead of demanding the finiteness of the ordering, we assume that its ascending chains have finite length. Then Theorem 3.3.9 guarantees the termination of the SGI schema, in case of soft constraint propagation functions which are inflationary and computable. In other words, if Axiom 3.3.3 is satisfied. We rewrite that axiom as follows, and specialise it to the case of SCSPs and soft constraint propagation functions.

**Axiom 5.4.3 (termination).**

- Each soft constraint function \( f \in F \) is computable over a problem ordering \( \langle \mathcal{F}(P), \sqsubseteq_p, P \rangle \).
- The \( F \) functions are inflationary with respect to the assigned problem order.
- The ordering \( \langle \mathcal{F}(P), \sqsubseteq_p, P \rangle \) satisfies the ascending chain condition (ACC):
  - i.e., each \( \sqsubseteq_p \)-chain in \( P \uparrow \) is finite.

The following statement is an immediate consequence of Theorem 3.3.9.

**Corollary 5.4.7 (termination 1).** *Let us consider an SCSP problem \( P \) on a c-semiring \( S \) and a problem ordering \( \mathcal{FP} := \langle \mathcal{F}(P), \sqsubseteq_p, P \rangle \) on \( P \). Let us instantiate SGI with \( P \) and a finite set \( F \) of functions on \( \mathcal{FP} \). Furthermore, let us assume the Common Fixpoint Axiom 5.4.3. Then every execution of this instance of SGI terminates, computing a common fixpoint of the \( F \) functions.*

The above corollary can be used to prove termination in some cases, like the ones discussed in Section 5.5. However, it is a highly general result, so its hypothesis might be sometimes difficult to check. In turn, in some circumstances, specific properties of the adopted functions or c-semiring, easier to verify, can imply the assumptions of the above corollary, and thus immediately ensure the termination of SGI. This is the case of problem orderings step-by-step computed from the c-semiring operations, as made precise in the remainder of this section.

**Semiring based functions and termination**

The termination result in Corollary 5.4.7 refers to any inflationary functions on a problem ordering. In the following part, we explore and characterise some problem orderings and functions that satisfy Axiom 5.4.3.

The main motivation for relaxing the assumptions of Axiom 5.4.3 to a generic family of problems in \( P \uparrow \) is that, in general, the whole family \( P \uparrow \) contains too many problems with respect to those generated by specific constraint propagation algorithms. Above all, this is the case if we analyse constraint propagation algorithms to establish whether and under which conditions they termination. The
following simple example, that pertains to a type of SCSPs of primary interest, explains our concern.

**Example 5.4.8.** We consider the fuzzy c-semiring \( ([0, 1], \leq, \min, 0, 1) \), and a fuzzy CSP \( P \) over it. The problem \( P \) has variable domains equal to \( \{a\} \) for both of its variables, \( x \) and \( y \); \( P \) has just the trivial constraint \( c = \langle 1, \{x, y\} \rangle \), where \( 1 \) is the constant function that assigns the value 1 to each possible instantiation of \( x \) and \( y \) in \( D \). Then \( P^\uparrow \) is the class of all problems on \( x \) and \( y \). It is evident that the problem order on \( P^\uparrow \) cannot satisfy the ACC.

Consider, for instance, the set of problems \( P_n \) in \( P^\uparrow \), the constraints of which are defined as follows:

- both unary constraints on \( x \) and \( y \) assign 1 to \( a \), their only possible instantiation;
- the constraint of \( P_n \) on \( x \) and \( y \) assigns the value \( 1/n + 1 \) to the pair \( (a, a) \).

The relation \( P_n \sqsubseteq P_{n+1} \) is of strict order for every \( n \in \mathbb{N} \), since \( \langle 1/n : n \in \mathbb{N} \rangle \) is an infinite \( \sqsubseteq \)-descending chain. So we have an infinite \( \sqsubseteq \)-chain in \( P^\uparrow \).

A similar example applies to probabilistic CSPs with max instead of min.

However, if we restrict our attention to soft constraint functions and families of problems that are, in some sense, finitely generated via those functions, we may avoid the flaws of the above example. First we characterise such families and then study SGI with them.

**Definition 5.4.9.** Consider an SCSP \( P := \langle X, D, C \rangle \) on a c-semiring \( S \), and a finite set \( F \) of constraint functions on \( P^\uparrow \). Define \( F(P) \) to be the following subset of \( P^\uparrow \):

- \( P \) belongs to \( F(P) \);
- if \( P' \) belongs to \( F(P) \), then there are finitely many \( F \) functions, say \( g_1, \ldots, g_k \), such that \( P' \) is equal to \( g_1 \cdots g_k(P) \);
- nothing else belongs to \( F(P) \).

Then \( \langle F(P), \subseteq, P \rangle \) is the problem ordering that is *finitely generated* by \( F \) on \( P \).

This is a typical construction in set theory, see [DP90].

The following statement is only Corollary 5.4.7 rephrased for orderings generated by \( F \) functions: there is nothing really new about it but the choice of orderings. However, we shall use it as follows in the remainder of this section.
LEMMA 5.4.10. Given a c-semiring $S$ and an SCSP problem $P$ on it, instantiate the SGI algorithm with a finite set $F$ of constraint functions and $P$. Suppose that Axioms 5.4.1 and 5.4.3 hold for $(F(P), \subseteq_P, P)$ and the $F$ functions. Then every execution of the SGI algorithm terminates, computing a common fixpoint of the $F$ functions.

In case of crisp constraints, such lemma could be applied to arc consistency problems and the ordering $F(P)$ would be contained in the domain ordering over $P$, see Section 4.2. However, in the soft case it may not be trivial to envisage $F(P)$, or a family that contains it and for which Axiom 5.4.3 holds. However, we do have the c-semiring ordering and operations, and so the question is whether these can be used to construct $F(P)$ as in the above lemma.

In the following, we focus on the c-semirings operations and partial order, and introduce the notion of semiring closure of a soft constraint problem; this mirrors the notion of ordering generated by functions.

**DEFINITION 5.4.11.** Consider an SCSP $P := \langle X, D, C \rangle$ over a c-semiring $S := \langle S, \preceq_S, \times, \bot, T \rangle$, and the set $C(P)$ of c-semiring values that occur in $P$: i.e.,

$$C(P) := \bigcup_{C \in C} \{ C(d) : d \in D[s] \}.$$

Then the **semiring closure** of $P$, denoted by $\bar{C}(P)$, is the smallest (with respect to set inclusion) of all sets $B$ that enjoy the following properties:

1. $C(P) \subseteq B \subseteq S$;
2. $B$ is closed under $\vee$ and $\times$.

Notice that the previous definition is meaningful, since there always is a set that satisfies the two properties as above; that is, the c-semiring universe $S$ itself.

What we need now is to single out those constraint functions that, applied to a problem $P$, compute values that are in the semiring closure of $P$. Such functions, intuitively, are defined via the two semiring operations, $\vee$ and $\times$. So, given an SCSP $P$, those functions will return values that are either in $C(P)$ or obtained from elements of $C(P)$ by means of $\vee$, $\times$, or their composition.

**DEFINITION 5.4.12.** Consider an SCSP $P$ over $S$, and a soft constraint function $g : P'[\mathrel{\uparrow}] \rightarrow P'[\mathrel{\uparrow}]$. Then $g$ is a **semiring based function** if, for every $P' \in P'[\mathrel{\uparrow}]$, it enjoys the following property:

$$\bar{C}(g(P')) \subseteq \bar{C}(P').$$
In other words, the c-semiring values that a semiring based function associates with a problem \( P' \) can all be found in the semiring closure of \( P' \).

The line of our argument should be clear by now: since semiring based functions can only return values in the semiring closure of their input problem, the termination of SGI with such functions can be established by studying semiring closures.

So, let us turn our attention to semiring closures and semiring based functions as in the following lemma.

**Lemma 5.4.13.** Consider an SCSP \( P \) on \( S \) and a finite set of semiring based functions \( F \). Assume also that the c-semiring order \( <_S \) satisfies the Descending Chain Condition (DCC), when restricted to \( C(P) \): i.e., there are no infinite descending \( <_S \)-chains of \( C(P) \) elements. Then the problem ordering \( F(P) \) satisfies the ACC.

**Proof.** Suppose that the thesis of this lemma does not hold. That is, suppose that there is an infinite chain of \( F(P) \) problems,

\[
P = P_0 \sqsubseteq_P \cdots \sqsubseteq_P P_n \sqsubseteq_P P_{n+1} \sqsubseteq_P \cdots
\]

(5.7)

where, for each \( n \geq 1 \), \( P_n := g_1 \cdots g_k(P) \), for some \( g_1, \ldots, g_k \in F \). Since all the \( F \) functions \( g \) are semiring based, we also have

\[
\bar{C}(P_n) \subseteq \bar{C}(P),
\]

(5.8)

for every \( n \in \mathbb{N} \). We now aim at proving that an infinite descending \( <_S \)-chain can be extracted from (5.7), thereby contradicting our assumption on \( <_S \).

In fact, (5.7) yields the existence of a constraint \( C_n \in P_n \) that is strictly greater than \( C_{n+1} \in P_{n+1} \); that is for which \( C_n \sqsubseteq_S C_{n+1} \) holds. This is true for every \( n \in \mathbb{N} \). As the chain in (5.7) is infinite, while the scheme \( X \) is finite, we can extract (at least) an infinite chain of constraints, all on the same scheme \( s \), from the chain (5.7):

\[
\cdots \sqsubseteq_S C_m(s) \sqsubseteq_S C_{m+1}(s) \sqsubseteq_S \cdots
\]

(5.9)

Now, \( C_m(s) \sqsubseteq_S C_{m+1}(s) \) means that there exists a tuple \( d \in D[s] \) for which the following relation holds:

\[
C_m(s)(d) >_S C_{m+1}(s)(d).
\]

As the variable domain set \( D \) is finite while the chain in Equation (5.9) is infinite, we can extract an infinite chain of semiring elements from the chain (5.9) of the following form:

\[
\cdots >_S C_k(d') >_S C_{k+1}(d') >_S \cdots
\]

(5.10)
5.4. Soft Constraint Propagation via SGI

All the c-semiring elements that occur in (5.10) belong to $\mathcal{C}(P)$ due to (5.8). Therefore, the restriction of the order $<_S$ to the set $\mathcal{C}(P)$ does not satisfy the DCC, which contradicts our hypothesis.

The following corollary follows now from Lemma 5.4.10 (which is Corollary 5.4.7, specialised to finitely generated orderings) via Lemma 5.4.13.

**Corollary 5.4.14 (termination 2).** Consider an SCSP $P$ over $S$ and a finite set $F$ of semiring based functions. Assume the Common Fixpoint Axiom 5.4.1. Suppose that the $F$ functions are computable and inflationary over a problem ordering of the form $(F(P), \subseteq_P, P)$. Assume, also, that the c-semiring order $<_S$ satisfies the Descending Chain Condition (DCC) when restricted to $\mathcal{C}(P)$; namely, there are no infinite descending $<_S$-chains of $\mathcal{C}(P)$ elements. Then every execution of the SGI algorithm terminates, computing a common fixpoint of the $F$ functions.

Nevertheless, even the assumptions of Corollary 5.4.14 may be difficult to check. In fact, it might not always be trivial to determine the semiring closure of a given SCSP $P$; furthermore, we should also check that the restriction of the semiring order to the closure satisfies the DCC. Nevertheless, if the multiplicative operation of the semiring is idempotent, then the semiring closure of any given problem is always finite, and hence satisfies the DCC.

**Corollary 5.4.15 (termination 3).** Consider an SCSP $P$ on $S$ and a finite set of $F$ of semiring based functions. Assume the Common Fixpoint Axiom 5.4.1. Suppose that these are inflationary on a problem ordering defined on $F(P)$. Assume, also, that the $\times$ operation of $S$ is idempotent.

- Then the semiring closure of $P$ is finite.
- Thus every execution of the SGI algorithm terminates, computing a common fixpoint of the $F$ functions.

**Proof.** We only need to prove the first item, and the second follows then from Corollary 5.4.14. First, let us recall that a c-semiring with an idempotent $\times$ operation is a distributive complete lattice, see Lemma 5.2.1. Furthermore, every finitely generated sublattice of a distributive lattice is finite, see [DP90]. Thus every finitely generated sublattice of a c-semiring with idempotent $\times$ operation is finite.

Now, the set $\mathcal{C}(P)$ of all semiring elements in $P$ is finite, since every SCSP has finitely many constraints, finite domains and finitely many variables. So the semiring closure of $P$ is finitely generated. Therefore the semiring closure of $P$ is finite, due the aforementioned result of lattice theory.
Notice that the above two results concerning terminations are the analogues of Theorem 4.14 in [BMR97]. However there the set $C(s)$ is assumed to be finite in order to guarantee the termination of a constraint propagation algorithm. This hypothesis is much more restrictive, and implies ours in Corollary 5.4.14. Thus Theorem 4.14 in ib. is a special case of our Corollaries 5.4.14 and 5.4.15.

**Finale**

Our results concerning termination can suggest various strategies for proving the termination of SGI instances, like the following ones.

- If the constraint functions are semiring based and the multiplicative operation $\times$ of the $c$-semiring is idempotent, then we resort to Corollary 5.4.15.

- If $\times$ is not idempotent, but the soft constraint functions are semiring based, we can check whether the restriction of the semiring order to the semiring closure of $P$ satisfies the DCC and appeal to Corollary 5.4.14.

- If the restriction of the semiring order to the semiring closure of $P$ does not satisfy the DCC or the soft constraint functions are not all semiring based, then we can try to prove that the problem order on $P^*$, or a smaller problem ordering on $P$ satisfies the ACC. If the $F$ functions are inflationary with respect to that ordering, then SGI terminates by Corollary 5.4.7.

In Section 5.5 below, we investigate some examples of constraint propagation algorithms for SCSPs, and briefly analyse them by means of the theoretical results of the present section.

### 5.5 Soft Constraint Propagation Algorithms

In this section, we briefly survey several soft constraint propagation algorithms and show how the general results concerning SGI over soft CSPs can be used to both describe and analyse those specific algorithms. We restrict our attention to arc consistency algorithms over SCSPs; for these are the most used constraint propagation algorithms over SCSPs and easier to explain.

Our account of constraint propagation algorithms in this section has no pretence of completeness. However, it proves that a number of constraint propagation algorithms for SCSPs are extensions of their corresponding crisp counterparts, since all of them are instances of SGI. By these claims, we mean that the basic process of "prune-and-propagate" is carried over, and only slightly modified by passing from crisp to soft CSPs:

- pruning of domain or constraint values is transformed as reduction to more restrictive constraint problems, so to speak;
whereas the propagation phase is characterised as in the crisp case, because update is not substantially different.

We start our survey and analysis with the fuzzy and probabilistic cases, based on the \( \leq_R \) total order over real numbers, and the max or min operator. Then we deal with a form of generalised arc consistency for soft constraint problems, that is defined through functions which are not finitely generated by means of the two c-semiring operations. Finally, we briefly present the partial arc consistency counter algorithm by Freuder and Wallace, and show that this simple constraint propagation algorithm is an instance of SGI too.

### 5.5.1 Probabilistic and Fuzzy Arc Consistency Algorithms

Let us consider, as in Subsection 5.2.3, a fuzzy CSP \( P := \langle X, D, C \rangle \) based on the c-semiring \( ([0,1], \leq_R, \min, 0, 1) \). Notice that \( \times \) is the minimum operator, so it is idempotent and the underlying c-semiring is a distributive lattice.

Then, given a unary constraint \( C_i \), we define, for each \( x_j \) such that \( i \neq j \), a function of the form \( f(x_i, s) \); the scheme \( s \) is equal either to \( \langle x_i, x_j \rangle \) if \( i < j \), or to \( \langle x_j, x_i \rangle \) if \( j < i \). Then, \( f(x_i, s) \), if applied to \( P \), returns a problem \( P' \) that differs from \( P \) at most in its unary constraint \( C'(x_i) \), that is defined as follows:

\[
E(s)(d) := \min \{ C(x_i)(d[x_i]), C(s)(d), C(x_j)(d[x_j]) \}, \quad \text{for every } d \in D[s],
\]

\[
C'(x_i)(a) := \max \{ E(s)(d) : d \in D[s] \text{ and } d[x_i] = a \}, \quad (5.11)
\]
in which \( C(x_i), C(x_j) \) and \( C(s) \) are all \( P \) constraints. Each \( f(x, s) \) function is a soft constraint function on \( F(P) \), that collects all problems \( P' \) such that \( P \subseteq P' \), and \( P \) differs from \( P' \) at most in the unary constraints.

Thus, Corollary 5.4.15 can be directly employed to prove that any constraint propagation algorithm with \( f(x_i, s) \) functions such as (5.11) always terminates, whenever it is an instance of SGI. The returned problem is a common fixpoint of the iterated functions as in (5.11).

Since those functions are also monotone with respect to the problem order \( \subseteq \) on \( F(P) \), then all the executions of the SGI algorithm with them computes the same problem; namely, the least common fixpoint with respect to the problem order, that is more constraining than \( P \). This is due to Corollary 5.4.5.

Notice that each function \( f := f(x_i, s) \) such as (5.11) preserves equivalence:

\[
\text{Sol}(P) = \text{Sol}(f(P)).
\]

It is not a difficult exercise to verify that the above relation holds, since max and min are both idempotent. We refer the reader to [Gen01a, Gen01b] for a deeper analysis on this topic.

Finally, let us consider a probabilistic CSP \( P := \langle X, D, C \rangle \) based on the c-semiring \( ([0,1], \geq_R, \max, 0, 1) \). Notice that \( \times \) is the maximum operator, so it is idempotent and the underlying c-semiring is a distributive lattice.
As in the fuzzy case, arc consistency is enforced by means of functions of the form \( f(x_i, s) \). When this function is applied to \( P \), it returns a problem \( P' \) that differs from \( P \) at most in its unary constraint \( C'(x_i) \), that is defined as follows:

\[
E(s)(d) := \max \{C(x_i)(d[i]), C(s)(d), C(x_i)(d[j])\}, \quad \text{for every } d \in D[s],
\]

\[
C'(x_i)(a) := \min \{E(s)(d) : d \in D[s] \text{ and } d[x_i] = a\},
\]

(5.12)
in which \( C(x_i), C(x_j) \) and \( C(s) \) are all \( P \) constraints. As in the case of fuzzy CSPs, we conclude that any constraint propagation algorithm with \( f(x_i, s) \) functions as in (5.12) always terminates, whenever it is an instance of SGI (by Corollary 5.4.15). The returned problem is the least common fixpoint of the iterated functions as in (5.12), with respect to the problem order (by Corollary 5.4.5).

### 5.5.2 Generalised Arc Consistency Algorithms

The above functions in Equations (5.11) and (5.12) are semiring based functions, as characterised in Definition 5.4.12. However, the user may profit from having at her/his disposal further functions than those provided by the chosen c-semiring.

To our knowledge, it was first Schiex to underline the faults of soft arc consistency for weighted CSPs and propose a solution to it. In fact, as shown for instance in [Sch00], if only addition is used to enforce soft constraint propagation, equivalence gets lost. So, in [Sch00, CS01], arc consistency is refined by introducing a sort of inverse operation to addition; this new operation is not — in our terminology — semiring based.

![Algorithm 5.5.1: AC-proj and AC-join](image)

The resulting algorithm in [CS01] iterates two sorts of functions, illustrated as in Algorithms 5.5.1. The algorithm of [CS01] can be easily turned into an instance of SGI. Hence this explains how the algorithm computes a common fixpoint of
all the iterated functions. The algorithm of [CS01] preserves equivalence, but neither its confluence nor its termination can be ensured in the general case; in fact, functions for this are neither inflationary nor monotone over the soft constraint orderings.

5.5.3 Maximal CSPs

An interesting case of weighted CSPs, see Subsection 5.2.3, is constituted by maximal CSPs, briefly max-CSPs. Given an over constrained CSP, namely a CSP that does not admit solutions, Freuder et al. (see [FW92]) devised a series of algorithms to maximise the number of satisfied constraints, regardless of their importance. In other words, each constraint can assign one out of two levels of preference to each constraints; either 0 (yes), or 1 (no). Then, at each step of the basic algorithm in [FW92], the number of constraints, unsatisfied by the extension $d'$ of the current assignment $d$, gets computed; then this number is compared with the stored number of unsatisfied constraints by a previously computed total assignment — the initial stored number being 0 or another lower bound, chosen by the programmer. If the stored number is less than the new computed value, then search is abandoned along the path of $d'$; else it continues to extend it to another variable. This basic branch-and-bound algorithm terminates when the number of violated constraints is minimised, and a total assignment that does so is returned.

In [FW92], Freuder and Wallace devise also a form of arc consistency for max-CSPs, named partial arc consistency. Their partial arc consistency counter (PACC) algorithm computes the number of satisfied constraints for each variable instantiation. The PACC algorithm is extended, by means of SGI, to its hyper arc version, namely the Partial Generalised Arc Consistency Counter (PGACC) Algorithm 5.5.2 displayed as below. This is a naive algorithm, and it could be improved in a number of way, for instance by ordering variables. However, we only aim at showing that PACC is an instance of the more general schema, SGI, for soft constraint propagation.

First we regard the given max-CSP as a weighted CSP, so that its constraints can only assign either 0 or 1 to their tuples. Therefore our extension of the partial arc consistency counter algorithm becomes straightforward. In fact, instead of assuming only binary constraints, we check that a variable instantiation is consistent with constraints of any arity. As for this, we introduce new unary constraints in the original problem; namely we define a "counter constraint" $CC(i)$ for each variable $x_i$ of the problem. So each $CC(i)$ assigns a natural number to each value of the original CSP domain $D_i$; whereas all the original problem constraints can only assign either 0 or 1 to their tuples. In other words, $CC(i)$ is assumed to count the number of satisfied constraints for each assignment $a$ of the variable $x_i$. If no prior knowledge is assumed, each $CC(i)$ of the extended input problem, denoted by $P$, assigns 0 to each value in $D_i$. 
The functions, iterated by the PGACC algorithm, are defined as follows: for each value $a$ in $D_i$, the input problem domain, we define a function $f(x_i, a)$ of the form $\min \{\ldots, \prod \sum \ldots\}(a)$. So, given a problem $P \supseteq P_k$, the computed problem $P_{k+1} := f(x_i, a)(P_k)$ differs from $P_k$ at most in the added constraint $CC_{k+1}(x_i)$ that is characterised as follows:

$$CC_{k+1}(x_i)(a) := \min \left\{ C_k(x_i), \prod_i \sum \left\{ C_k(s) : x_i \in s, s \neq x_i \right\} \right\}(a).$$

Notice that each selected function is removed after being applied in PGACC, and that $update$ is always empty; in fact the adopted functions $f(i, a)$ are all idempotent and commutative. Therefore, there is no propagation as, commonly, in the case of classical crisp constraints.

**Algorithm 5.5.2: PGACC($P, F$)**

- $P :=$ given problem, extended with $GC$;
- $G := F$;
- while $G \neq \emptyset$ do
  - choose $f(x_i, a) \in G$;
  - $P' := f(x_i, a)(P)$;
  - $G := G - \{f(x_i, a)\}$

The above algorithm can only count the number of constraint violations for each assignment. It could be modified so as to incorporate a propagation phase, as suggested in see [FW92].

The fact that PGACC terminates is obviously true, since there is no propagation phase; i.e., $update$ is empty. The termination of the PGACC algorithm is also a trivial consequence of Corollary 5.4.7, since the generated problem order on $P$ is always finite, and hence it satisfies the ACC.

### 5.6 Conclusions

#### 5.6.1 Synopsis

In the present chapter, constraints with preference values, i.e., soft constraints, are introduced and a number of constraint propagation algorithms for these are studied. So, the SGI schema is used to represent and analyse semiring-based constraint propagation algorithms, as we did in Chapter 4 for classical constraints. Therefore, the SGI algorithm provides a general schema for soft constraint propagation as well. Again, the results obtained for the schema are applicable to all its soft instances as straightforward corollaries.
5.6. Conclusions

In particular, termination is not an easy issue in the context of soft constraint propagation algorithms: yet our analysis stresses the role that, in this, inflationarity of soft constraint functions and well-founded orders play. This results in three general termination properties that can be applied in different soft frameworks, as we summarise at the end of Subsection 5.4.4.

5.6.2 Discussion

The SGI algorithm schema iterates functions according to a certain strategy; however these are not required to have special properties for the algorithm to compute a common fixpoint of theirs. Due to this generality, we are able to instantiate this schema to a larger class of soft constraint propagation algorithms than the rule based schema of [BMR97] can account for, see Section 5.5.

Also, properties of functions, i.e. monotonicity and inflationarity, are studied as separate issues in Chapter 3. Thus their respective roles in connection with certain behaviours of soft constraint propagation algorithms are differentiated. This is already an achievement: in the soft constraint literature, often, the two properties are studied together and the role of each in the analysis of soft constraint propagation thus gets lost. This distinction can help in the design of new algorithms: e.g., it is often the case that in planning we do not want an algorithm to terminate, and an optimal solution is the one that maximises the number of satisfied constraints; hence inflationarity is a property that should be overlooked in the design of algorithms for such problems. On the other hand, the capability to predict the outcome of the algorithms' computations could be essential; thus monotonicity becomes an important requirement.

In the remainder of this part, we conclude our theoretical analysis of constraint propagation algorithms: in the following chapter, we summarise and specify all the functions used to characterise them.