Mapping Inferences: Constraint Propagation and Diamond Satisfaction
Gennari, R.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 6

Constraint Propagation Functions

6.1 Introduction

6.1.1 Motivations

The relational model is one of the best-known database models, see [Ull80]: the primitive entities are relations, represented as tables, and operations on these. The relational model has at least two advantages: it is easy to grasp; it supports a high-level programming language called SQL (Structured Query Language) that allows the user to query the database, update and retrieve data stored in tables through a number of functions.

In Chapters 4 and 5 we described and analysed, via iterations of certain functions, a number of constraint propagation algorithms for classical and soft constraints, respectively. At this point, we collect all those functions in a homogeneous setting, thereby putting forward the resemblances and differences of those constraint propagation algorithms and the query programs in the relational database model.

6.1.2 Outline and Structure

In the introduction to Chapter 3, we claim that our theorisation of constraint propagation algorithms has two aims:

- one of describing and analysing constraint propagation algorithms in terms of function iterations; this is accomplished in Chapter 3;
- the other of abstracting which functions perform the task of pruning or propagation of inconsistencies in constraint propagation algorithms, in both the crisp (see Chapter 4) and soft (see Chapter 5) cases.

In Chapter 3 we do not specify which functions are used for constraint propagation — there we only study properties of functions as traced in constraint propagation
algorithms. We do so in this chapter; thus we complete our theoretical work and tackle the task in the latter item.

Section 6.2 characterises the basic and derived operations that are traced in the representation and analysis of classical constraint propagation algorithms in Chapter 4. Finally, in Section 6.3, we define functions useful in the description and analysis of soft constraint propagation algorithms as in Section 5.5.

### 6.2 Functions for CSPs

In what follows, we usually need to fix a scheme \( X \) and a domain set \( D \) on \( X \). Since structures of the form

\[
CS := \langle X, D \rangle
\]

will often recur in the remainder of this chapter, we name them constraint systems. We follow the conventions of Subsection 2.2.1, and let \( D \) denote the domain \( D_1 \times \cdots \times D_n \). Then for a constraint \( C(s) \) of \( CS \) we mean a constraint on a scheme \( s \) of \( X \), that is a subset of \( D[s] \). In the limit, \( D[s] \) is a \( CS \) constraint as well; i.e., the universal constraint on \( s \).

#### 6.2.1 Atomic Formulas

The user of a database will often query the database to select information that matches some criteria: for instance, the database could store bibliographical data, and the user, a librarian, could be interested in selecting all books in a category that were published before or after a certain year. Such criteria are usually expressed through numerical formulas such as \( 2 < 3 \), meaning that every value in the second column must be less than the one in the third column that is in the same row.

We use something similar to formulas like \( 2 < 3 \) for defining functions for constraint propagation in Chapter 4: for instance, functions for HAC-4 (see Subsection 4.2.3) are defined through a selection operation such as

\[
sel_{x_i=a}R(s)
\]

that extracts all tuples \( d \) from \( R(s) \) for which the equality \( d[x_i] = a \) holds. The following definition then aims at characterising the basic formulas, such as \( x_i = a \), that are used to express criteria for selecting data from CSP domains.

**DEFINITION 6.2.1.**

(i). Consider a constraint system \( CS := \langle X, D \rangle \) with variable scheme \( X := \langle x_1, \ldots, x_n \rangle \) and domain \( D := D_1 \times \cdots \times D_n \). The set of \( CS \) atomic formulas is defined as follows:

- \( t = d \) and \( t \neq d \) are atomic formulas for each scheme, scheme \( t \) of \( X \) and tuple \( d \in D[t] \);
6.2. Functions for CSPs

- \( t \in S \{t\} \) is an atomic formula for each scheme \( t \) of \( X \) and constraint \( S(t) \) of \( CS \);

- nothing else is an atomic formula.

(ii). Consider a scheme \( s \) of \( X \), a scheme \( t \) of \( s \), and assume that \( C(s) \) is a constraint of \( CS \) on the scheme \( s \). Given a tuple \( e \in C(s) \) and an atomic formula of the form \( t = d \), we say that \( C(s) \) satisfies \( t = d \) in \( d \) if \( e \{t\} = d \); similarly, \( C(s) \) satisfies \( t \neq d \) in \( e \) if \( e \{t\} \neq d \); it satisfies \( t \in S \) in \( e \) if \( e \{t\} \in S \).

(iii). A CS formula \( \psi(t) \) is an atomic formula or a finite conjunction of CS formulas: i.e.,

\[
\psi(t) := \bigwedge_{i=1}^{n} \psi_i(t_i)
\]

where each \( \psi_i(t_i) \) is an atomic formula and \( t \) is the join \( \bigcup_{i=1}^{n} t_i \). If \( C(s) \) is a constraint of \( CS \) and \( t \) is a scheme of \( s \), then \( C(s) \) satisfies \( \psi(t) \) in \( e \in C(s) \) if it satisfies each atomic subformula \( \psi_i(t_i) \) of \( \psi(t) \) in \( e \).

The definition of conjunctive formula is not, loosely speaking, necessary if we do not mind the order in which elements are selected: in this case, its effect can be obtained by composing a finite number of basic functions, as defined in Subsection 6.2.2 below. However, conjunctions of atomic formulas are useful to introduce derived formulas such as the following, which is itself a convenient shorthand: given a scheme \( s \) of \( X \), a finite number of schemes \( t_1, \ldots, t_k \) of \( s \), and \( t \) equal to the join of these, put

\[
t \subseteq D \{s\} := t_1 \in D \{s\} \wedge \cdots \wedge t_k \in D \{s\}.
\]

Notice also that, if \( S \) is a finite constraint over \( s \), i.e.,

\[
S = \{d_1, \ldots, d_n\},
\]

then the formula \( t \in S \{t\} \) is equivalently rewritten as the conjunctive formula

\[
t = d_1 \{t\} \wedge \cdots \wedge t = d_n \{t\}.
\]

Hence, if \( S \) is as above, we can also introduce the following shorthand

\[
t \not\in S \{t\}
\]

to denote the finite conjunction \( t \neq d_1 \{t\} \wedge \cdots \wedge t \neq d_n \{t\} \).
6.2.2 Basic Functions

In what follows, we assume that a constraint system $CS := \langle X, D \rangle$ is given. As usual, $D$ denotes the domain $D_1 \times \cdots \times D_n$ and $D[s]$ the domain over the scheme $s$. Constraints and domains over $CS$ are denoted by the letters $R$ or $S$, with additionally superscripts. At this point, we can distinguish a number of effectively computable basic functions over $CS$ as follows.

**Union.** Set-union of $R$ and $S$ over the same scheme $s$.

**Join.** Let $R$ be over the scheme $s$ and $R'$ over the scheme $t$. The join of $R$ and $S$, denoted by $R \bowtie S$, is a relation over the scheme $s \cup t$ defined as follows: $e \in R \bowtie S$ iff there exists $d \in R$ and $d' \in R'$ such that $d(s) = e(s)$ and $d'(t) = e(t)$.

**Projection.** Given $R$ over $s$ and a subsequence $t$ of $s$, the projection $\Pi_t(R)$ of $R$ over $t$ is the set of tuples $d$ for which there exists $e \in R$ and $e[t] = d$.

**Universal selection.** Consider a $CS$ constraint $R(s)$ and a $CS$ formula $\psi(t)$, for $t$ a subscheme of $s$. Then $\forall_{sel_e} R$ is the subset of all $R$ tuples $d$ such that $d[t]$ satisfies $\psi(t)$.

**Existential selection.** Consider a $CS$ constraint $R(s)$ and let $\psi(t)$ be a $CS$ formula, for $t$ subscheme of $s$. Then $\exists_{sel_e} R$ is either a singleton $\{d\}$ for $d \in R$ such that $d[t]$ satisfies $\psi(t)$, or the empty set if no tuple in $R$ satisfies $\psi(t)$.

6.2.3 Constraint Propagation Functions

The set of constraint propagation functions over $CS$ is the smallest inductive set of functions that contains the basic functions and is closed under composition.

Simple examples are the difference function and the Cartesian product.

**Example 6.2.2.**

**Intersection.** Set-intersection of $R$ and $S$, over the same scheme $s$, is defined by means of the join operation: $R \cap S := R \bowtie S$.

**Difference.** Consider two finite constraints $R$ and $S$ over the same scheme $s$. Then the difference of $R$ and $S$, denoted by $R - S$, is $\forall_{sel_{eg}} R$.

**Cartesian product.** Let $R$ be over $s$ and $S$ over $s'$ such that $s$ and $s'$ are disjoint schemes. Then $R \times S$ is the join of $R$ and $S$: i.e., $R \bowtie S$.

The two relations in Example 6.2.2, finite difference and Cartesian product, are obtained, by means of composition, from basic constraint propagation functions. The following example proposes a constraint propagation function that is instead
6.3. Functions for SCSPs

generated from a derived constraint propagation function, i.e., the Cartesian product.

**Example 6.2.3.**

**Cartesian product operator.** Given two constraint propagation functions \( f \) and \( g \), let \( f \times g \) be the function defined as

\[
f \times g(R, S) := f(R) \times g(S)
\]

for every \( R \) and \( S \). Then \( f \times g \) is the Cartesian product of \( f \) and \( g \).

### 6.3 Functions for SCSPs

In Chapter 5, we surveyed some constraint propagation algorithms for soft constraints. The definition of basic functions for those algorithms is complicated by the presence of preference structures as c-semirings.

A c-semiring constraint system \( CS \) carries over the c-semiring structure \( S := \langle S, <, \times, \perp, \top \rangle \) used to define soft constraints, see also [BMR97]:

\[
CS := \langle X, D, S \rangle,
\]

where \( X \) and \( D \) are as in the classical case. As for the rest, we follow the conventions introduced in Section 5.2, and define a soft constraint of \( CS \) as a soft constraint on a scheme \( s \) of \( X \), that maps \( D [s] \) into the c-semiring universe \( S \).

In Section 5.5, it is shown how a number of soft constraint propagation algorithms employ different functions from, in our terminology, the semiring based functions (see Definition 5.4.12). Those functions are strictly dependent on the specific c-semiring structure adopted, or depend on some extensions of it, see Subsection 5.5.2. Therefore, it is rather difficult to provide a sufficiently general definition, unless the c-semiring framework is replaced by a more general structure; such as universal algebras, with a lattice structure on the underlying universes, [Gen01a].

#### 6.3.1 Soft Constraint Propagation Functions

Here we limit ourselves to summarise the basic functions used for soft constraint propagation, without any pretence of completeness.

**Union.** Given two constraints \( R \) and \( S \) over the same scheme \( s \), the union of \( R \) and \( S \) is the constraint on \( s \) defined as follows: \( R \cup S(d) = R(d) \lor S(d) \), for each \( d \in D [s] \).
Join. Let $R$ be over the scheme $s$ and $R'$ over the scheme $t$. The join of $R$ and $S$, denoted by $R \Join S$, is a relation over the scheme $r := s \cup t$ defined as follows:

$$R \Join S(e) = R(e) \times S(e),$$

for every $e \in D[r]$; see p. 92.

Projection. Given $R$ over $s$ and a subsequence $t$ of $s$, the projection $\Pi_t(R)$ of $R$ over $t$ is the function from $D[s]$ to the c-semiring universe $S$ that maps each $d \in D[s]$ to the c-semiring value $\bigvee \{ e \in D[s] : e[t] = d \}$; see p. 92.

It is also possible to generalise, to some extent, the operations of selection from the classical to the soft case. We do it as below and then explain how these could be useful in the design of constraint propagation or solving algorithms for SCSPs.

First of all, notice that universal selection amounts to assigning $\top$ to all the tuples for which a certain formula holds true; and implicitly assigning $\perp$ to all the other ones. On the other hand, existential selection returns $\top$ only to the first tuple that is found to satisfy a certain formula; $\perp$ to all tuples if none satisfies the formula. However, a c-semiring offers us more than two values. Thus, we modify the definition of $CS$ formulas in Definition 6.2.1 as follows.

**Definition 6.3.1.**

1. Consider a constraint system $CS := (X, D, S)$ with variable scheme $X := \langle x_1, \ldots, x_n \rangle$ domain $D := D_1 \times \cdots \times D_n$, and c-semiring $S$. The set of $CS$ atomic formulas is defined as follows:
   - $t = a$ and $t \leq a$ are atomic formulas for each scheme $t$ of $X$, tuple $d \in D[t]$ and c-semiring value $a$;
   - $t \neq a$ and $t \geq a$ are atomic formulas for each scheme $t$ of $X$, tuple $d \in D[t]$ and c-semiring value $a$;
   - nothing else is an atomic formula.

2. Consider a scheme $s$ of $X$, a subsequence $t$ of $s$, and assume that $C := C(s)$ is a soft constraint of $CS$ on the scheme $s$. Given a tuple $e \in D(s)$ and an atomic formula of the form $t = a$, we say that $C$ satisfies $t = a$ in $e$ if $\Pi_t(C(e)) = a$; similarly, $C$ satisfies $t \neq a$ in $e$ if $\Pi_t(C(e)) \neq a$; it satisfies $t \leq a$ in $e$ if $\Pi_t(C(e)) \leq a$, and $t \geq a$ in $e$ if $\Pi_t(C(e)) \geq a$.

3. A $CS$ formula $\psi(t)$ is an atomic formula or a finite conjunction of $CS$ formulas: i.e.,

$$\psi(t) := \bigwedge_{i=1}^{n} \psi_i(t_i)$$

where each $\psi_i(t_i)$ is an atomic formula and $t$ is the join $\bigcup_{i=1}^{n} t_i$. If $C(s)$ is a constraint of $CS$ and $t$ is a subsequence of $s$, then $C(s)$ satisfies $\psi(t)$ in $e \in C(s)$ if it satisfies each atomic subformula $\psi_i(t_i)$ of $\psi(t)$ in $e$. 
6.3. Functions for SCSPs

Note that we choose a local definition of satisfiability: we evaluate \( \psi(t) \) true in \( C(s) \) and a tuple \( e \in D(s) \). Other choices are possible: for instance, we could define satisfiability with respect to \( C(s) \) and the maximum value returned by \( C(s) \) on \( D[\pi] \), i.e., by using the projection function \( \Pi \) on the whole domain \( D[\pi] \). However, this local definition of satisfiability is sufficient to characterise a universal selection function for soft constraints as a generalisation of the corresponding function for classical constraints.

**Universal selection.** Consider a relation \( R(s) \) and a CS formula \( \psi(t) \), for \( t \) a subscheme of \( s \). Then \( \forall_{\text{sel}_\psi} R \) is defined as follows: for each \( d \in D[\pi] \),

\[
\text{sel}_\psi R(d) := \begin{cases} 
R(d) & \text{if } R \text{ satisfies } \psi \text{ in } d, \\
\bot & \text{otherwise.}
\end{cases}
\]

The definition of selection could be more general: for each c-semiring value \( a \), we could have a selection function \( \forall_{\text{sel}_{\psi,a}} \) that maps each tuple \( d \) that does not satisfy \( \psi \) to \( a \).

### 6.3.2 On Optimal Strategies

At this point, let us revisit the definition of solution constraint for an SCSP \( P := \langle X, D, C \rangle \), cf. p. 92:

\[
\text{Sol}(P, s) := \Pi_s \bowtie C \subseteq P \overset{C}{\circ} C,
\]

Suppose that the user queries a soft constraint system and only wants to retrieve all those tuples \( d \in D[\pi] \) from the system for which \( \text{Sol}(P, s) \geq a \), where \( a \) is a certain preference value in the c-semiring universe, that the user chooses. Then \( \forall_{\text{sel}_{\geq a}} \), applied to \( \text{Sol}(P, s) \), will return the user only those tuples to which \( \text{Sol}(P, s) \) assigns a value greater than \( a \).

Incidentally, notice that the above method for computing solutions does not seem highly efficient: in fact, first we have to compute \( \bowtie \), then \( \Pi \) and, only afterwards, select those tuples. A better choice would be to apply a selection function as soon as possible, that is before applying \( \bowtie \). But then \( \bowtie \) would return a constraint on \( X \), the scheme of the CSP \( P \), that assigns \( \bot \) to all the tuples in the domain \( D \). The situation can be remedied by adopting the more general definition of selection so that this assigns to each tuple, that do not satisfy a formula, the maximum value \( \top \) of the c-semiring.

It would be interesting to see whether this abstract view on constraint propagation and solving functions could be useful to tackle optimisation tasks, as in the above case:

- which function is it better to apply first to satisfy the user's query in an "economic" manner from the viewpoint of computations?
Finally, a soft generalisation of existential selection can be given as in the case of the universal selection function; we let the reader spell out the details. This function could be applied, for instance, whenever a user wants to retrieve only a solution that satisfies a given formula, and not all of them.

6.4 Conclusions

6.4.1 Synopses

In Chapter 3, a template for constraint propagation algorithms is proposed: that is the SGI algorithm schema, which iterates functions according to a certain strategy. The present chapter collects the functions used for constraint propagation, in both the crisp and soft case, in a homogeneous setting; the class of functions in SGI iterations is thus restricted to those that are actually traced in constraint propagation algorithms in Chapters 4 and 5.

Also, this digest highlights that a number of operations are common to constraint propagation algorithms and to languages for manipulating data in relational database systems. We briefly elaborate on this issue as below.

6.4.2 Discussion

The theoretical analysis of constraint propagation algorithms, proposed in this chapter and in Chapter 3, can immediately be used for optimisation tasks. Query optimisation is already a well explored topic in the database community. Most optimisation strategies involve transforming algebraic expressions; for instance, if two operations commute, then the order in which these are applied is not relevant; therefore, the optimal sequence of applications is obtained by applying first the least expensive operation. We encounter something similar in Chapter 3: a number of properties of functions, such as stationarity or commutativity, are proved to optimise the performance of the basic SGI schema; these properties are traced in a number of constraint propagation algorithms in Chapter 4.

Therefore, a closer investigation of the similarities and differences between the two worlds, that of constraint algorithms and that of database query languages, appears to be promising for optimisation tasks. The fact that there is a common language, that of operations on relations as explained in this chapter, helps to clarify further the connections between database theory and CSP algorithms, and should help in transferring the acquired knowledge and the developed strategies from one field to the other.

In [Var00], the author showed how certain classes of finite CSPs can be reduced, in polynomial time, to Datalog programs or view-based query answering, and vice versa; notice that, there, the aim is to identify tractable classes of CSPs; not to analyse the behaviour of each single algorithm, as instead we do. Also,
the analyses in [GLS99] have the same objective as those of [Var00]. However, the theoretical analysis of constraint propagation algorithms, proposed in this chapter and in Chapter 3, could also help in this respect.

In the following part, relations continue to be the protagonists of this thesis. Relations and relational structures are in fact at the basis of modal logics; these make use of restricted formal languages for describing relations and relational structures: loosely speaking, properties of these are expressed as theorems of modal logics. Thus we shall also see how constraint programming, which manipulates relations, can be used for automated theorem proving in modal logics.