The Rough Guide to Constraint Propagation

Krzysztof R. Apt[1,2]

[1] CWI
P.O. Box 94079, 1090 GB Amsterdam, the Netherlands
K.R.Apt@cwi.nl
[2] University of Amsterdam, the Netherlands

Abstract. We provide here a simple, yet very general framework that allows us to explain several constraint propagation algorithms in a systematic way. In particular, using the notions commutativity and semi-commutativity, we show how the well-known `AC-3`, `PC-2`, `DAC` and `DPC` algorithms are instances of a single generic algorithm. The work reported here extends and simplifies that of Apt [1].

1 Introduction

Constraint programming in a nutshell consists of formulating and solving so-called constraint satisfaction problems. One of the most important techniques developed in this area is constraint propagation that aims at reducing the search space while maintaining equivalence.

We call the corresponding algorithms constraint propagation algorithms but several other names have also been used in the literature: consistency, local consistency, consistency enforcing, Waltz, filtering or narrowing algorithms. These algorithms usually aim at reaching some form of "local consistency", a notion that in a loose sense approximates the notion of "global consistency".

Over the last twenty few years several constraint propagation algorithms were proposed and many of them are built into the existing constraint programming systems. In Apt [1] we introduced a simple framework that allows us to explain many of these algorithms in a uniform way. In this framework the notion of chaotic iterations, so fair iterations of functions, on Cartesian products of specific partial orderings played a crucial role. In Monfroy and Réty [13] this framework was modified to study distributed chaotic iterations. This resulted in a general framework for distributed constraint propagation algorithms.

We stated in Apt [1] that "the attempts of finding general principles behind the constraint propagation algorithms repeatedly reoccur in the literature on constraint satisfaction problems spanning the last twenty years" and devoted three pages to survey this work. Two references that are perhaps closest to our work are Benhamou [2] and Telerman and Ushakov [16].

These developments led to an identification of a number of mathematical properties that are of relevance for the considered functions, namely monotonicity, inflationarity and idempotence (see, e.g., Saraswat, Rinard and Panangaden [15] and Benhamou and

Older [3]). Here we show that also the notions of commutativity and so-called semi-commutativity are important.

As in Apt [1], to explain the constraint propagation algorithms, we proceed here in two steps. First, we introduce a generic iteration algorithm on partial orderings and prove its correctness in an abstract setting. Then we instantiate this algorithm with specific partial orderings and functions. The partial orderings will be related to the considered variable domains and the assumed constraints, while the functions will be the ones that characterize considered notions of local consistency in terms of fixpoints.

This presentation allows us to clarify which properties of the considered functions are responsible for specific properties of the corresponding algorithms. The resulting analysis is simpler than that of Apt [1] because we concentrate here on constraint propagation algorithms that always terminate. This allows us to dispense with the notion of fairness. On the other hand, we can now prove stronger results by taking into account the commutativity and semi-commutativity information.

This article is organized as follows. First, in Section 2, drawing on the approach of Monfroy and Réty [13], we introduce a generic algorithm for the case when the partial ordering is not further analyzed. Next, in Section 3, we refine it for the case when the partial ordering is a Cartesian product of component partial orderings and in Section 4 explain how the introduced notions should be related to the constraint satisfaction problems.

In the next four sections we instantiate the algorithm of Section 2 or some of its refinements to obtain specific constraint propagation algorithms. In particular, in Section 5 we derive algorithms for arc consistency and hyper-arc consistency. These algorithms can be improved by taking into account information on commutativity. This is done in Section 6 and yields the well-known `AC-3` algorithm. Next, in Section 7 we derive an algorithm for path consistency and in Section 8 we improve it, again by using information on commutativity. This yields the `PC-2` algorithm.

In Section 9 we clarify under what assumptions the generic algorithm of Section 2 can be simplified to a simple **for** loop statement. Then we instantiate this simplified algorithm to derive in Section 10 the `DAC` algorithm for directional arc consistency and in Section 11 the `DPC` algorithm for directional path consistency. Finally, in Section 12 we briefly discuss possible future work.

So we deal here only with the classical algorithms that establish (directional) arc consistency and (directional) path consistency and that are more than twenty, respectively ten, years old. However, several more "modern" constraint propagation algorithms can also be explained in this framework. In particular, in Apt [1, page 203] we derived from a generic algorithm a simple algorithm that achieves the notion of relational consistency of Dechter and van Beek [7]. In turn, we can use the framework of Section 9 to derive the adaptive consistency algorithm of Dechter and Pearl [6]. Now, Dechter [5] showed that this algorithm can be formulated in a very general framework of bucket elimination that in turn can be used to explain such well-known algorithms as directional resolution, Fourier-Motzkin elimination, Gaussian elimination, and also various algorithms that deal with belief networks.

Due to lack of space we do not define here formally the considered local consistency notions and refer the interested reader instead to the original papers or to Tsang [17].

2 Generic Iteration Algorithms

Our presentation is completely general. Consequently, we delay the discussion of constraint satisfaction problems till Section 4. In what follows we shall rely on the following concepts.

Definition 1. *Consider a partial ordering* $(D, \sqsubseteq)$ *with the least element* $\perp$ *and a finite set of functions* $F := \{f_1, \ldots, f_k\}$ *on* D.

- *By an* iteration of F *we mean an infinite sequence of values* $d_0, d_1, \ldots$ *defined inductively by*

$$d_0 := \perp,$$

$$d_j := f_{i_j}(d_{j-1}),$$

  where each i_j *is an element of* $[1..k]$.
- *We say that an increasing sequence* $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \ldots$ *of elements from* D eventually stabilizes at d *if for some* $j \geq 0$ *we have* $d_i = d$ *for* $i \geq j$. □

In what follows we shall consider iterations of functions that satisfy some specific properties.

Definition 2. *Consider a partial ordering* $(D, \sqsubseteq)$ *and a function* f *on* D.

- f *is called* inflationary *if* $x \sqsubseteq f(x)$ *for all* x.
- f *is called* monotonic *if* $x \sqsubseteq y$ *implies* $f(x) \sqsubseteq f(y)$ *for all* x, y. □

The following simple observation clarifies the role of monotonicity. The subsequent result will clarify the role of inflationarity.

Lemma 1 (Stabilization). *Consider a partial ordering* $(D, \sqsubseteq)$ *with the least element* $\perp$ *and a finite set of monotonic functions* F *on* D.

Suppose that an iteration of F *eventually stabilizes at a common fixpoint* d *of the functions from* F. *Then* d *is the least common fixed point of the functions from* F.

Proof. Consider a common fixpoint e of the functions from F. We prove that $d \sqsubseteq e$. Let $d_0, d_1, \ldots$ be the iteration in question. For some $j \geq 0$ we have $d_i = d$ for $i \geq j$.

It suffices to prove by induction on i that $d_i \sqsubseteq e$. The claim obviously holds for $i = 0$ since $d_0 = \perp$. Suppose it holds for some $i \geq 0$. We have $d_{i+1} = f_j(d_i)$ for some $j \in [1..k]$.

By the monotonicity of f_j and the induction hypothesis we get $f_j(d_i) \sqsubseteq f_j(e)$, so $d_{i+1} \sqsubseteq e$ since e is a fixpoint of f_j. □

We fix now a partial ordering $(D, \sqsubseteq)$ with the least element $\perp$ and a set of functions $F := \{f_1, \ldots, f_k\}$ on D. We are interested in computing the least common fixpoint of the functions from F. To this end we study the following algorithm that is inspired by a similar algorithm of Monfroy and Réty [13].

GENERIC ITERATION ALGORITHM (GI)

$d := \bot$;
$G := F$;
while $G \neq \emptyset$ **do**
    choose $g \in G$;
    $G := G - \{g\}$;
    $G := G \cup update(G, g, d)$;
    $d := g(d)$
od

where for all G, g, d the set of functions $update(G, g, d)$ from F is such that

A. $\{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \subseteq update(G, g, d)$,
B. $g(d) = d$ implies that $update(G, g, d) = \emptyset$,
C. $g(g(d)) \neq g(d)$ implies that $g \in update(G, g, d)$.

Intuitively, assumption **A** states that $update(G, g, d)$ at least contains all the functions from $F - G$ for which d is a fixpoint but $g(d)$ is not. So at each loop iteration such functions are added to the set G. In turn, assumption **B** states that no functions are added to G in case the value of d did not change.

Note that even though after the assignment $G := G - \{g\}$ we have $g \in F - G$ still $g \notin \{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\}$ holds. So assumption **A** does not provide any information when g is to be added back to G. This information is provided in assumption **C**.

On the whole, the idea is to keep in G at least all functions f for which the current value of d is not a fixpoint.

An obvious example of an $update$ function that satisfies assumptions **A** and **B** is

$$update(G, g, d) := \{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \cup \mathbf{C}(g),$$

where

$$\mathbf{C}(g) = \{g\} \text{ if } g(g(d)) \neq g(d) \text{ and otherwise } \mathbf{C}(g) = \emptyset.$$

However, this choice of the $update$ function is computationally expensive because for each function f in $F - G$ we would have to compute the values $f(g(d))$ and $f(d)$. In practice, we are interested in some approximations of the above *update* function. We shall deal with this matter in the next section.

We now prove correctness of this algorithm in the following sense.

Theorem 1 (GI) **.**

(i) *Every terminating execution of the* GI *algorithm computes in* d *a common fixpoint of the functions from* F.
(ii) *Suppose that all functions in* F *are monotonic. Then every terminating execution of the* GI *algorithm computes in* d *the least common fixpoint of the functions from* F.
(iii) *Suppose that all functions in* F *are inflationary and that* $(D, \sqsubseteq)$ *is finite. Then every execution of the* GI *algorithm terminates.*

Proof.

(i) Consider the predicate I defined by:

$$I := \forall f \in F - G\ f(d) = d.$$

Note that I is established by the assignment $G := F$. Moreover, it is easy to check that by virtue of assumptions **A**, **B** and **C** I is preserved by each **while** loop iteration. Thus I is an invariant of the **while** loop of the algorithm. (In fact, assumptions **A**, **B** and **C** are so chosen that I becomes an invariant.) Hence upon its termination

$$(G = \emptyset) \wedge I$$

holds, that is

$$\forall f \in F\ f(d) = d.$$

(ii) This is a direct consequence of (i) and the Stabilization Lemma 1.

(iii) Consider the lexicographic ordering of the strict partial orderings $(D, \sqsupset)$ and $(\mathcal{N}, <)$, defined on the elements of $D \times \mathcal{N}$ by

$$(d_1, n_1) <_{lex} (d_2, n_2) \text{ iff } d_1 \sqsupset d_2 \text{ or } (d_1 = d_2 \text{ and } n_1 < n_2).$$

We use here the inverse ordering $\sqsupset$ defined by: $d_1 \sqsupset d_2$ iff $d_2 \sqsubseteq d_1$ and $d_2 \neq d_1$.

Given a finite set G we denote by $card G$ the number of its elements. By assumption all functions in F are inflationary so, by virtue of assumption **B**, with each **while** loop iteration of the modified algorithm the pair

$$(d, card\ G)$$

strictly decreases in this ordering $<_{lex}$. But by assumption $(D, \sqsubseteq)$ is finite, so $(D, \sqsupset)$ is well-founded and consequently so is $(D \times \mathcal{N}, <_{lex})$. This implies termination. □

In particular, we obtain the following conclusion.

Corollary 1 (`GI`) **.** *Suppose that $(D, \sqsubseteq)$ is a finite partial ordering with the least element $\perp$. Let F be a finite set of monotonic and inflationary functions on D. Then every execution of the* `GI` *algorithm terminates and computes in d the least common fixpoint of the functions from F.* □

In practice, we are not only interested that the *update* function is easy to compute but also that it generates small sets of functions. Therefore we show how the function $update$ can be made smaller when some additional information about the functions in F is available. This will yield specialized versions of the `GI` algorithm. First we need the following simple concepts.

Definition 3. *Consider two functions f, g on a set D.*

- *We say that f and g* commute *if $f(g(x)) = g(f(x))$ for all x.*
- *We call f* idempotent *if $f(f(x)) = f(x)$ for all x.* □

The following result holds.

Theorem 2 (Update).

(i) If $update(G, g, d)$ satisfies assumptions **A**, **B** *and* **C**, *then so does the function*

$$update(G, g, d) - Idemp(g),$$

where

$$Idemp(g) = \{g\} \text{ if } g \text{ is idempotent and otherwise } Idemp(g) = \emptyset.$$

(ii) Suppose that for each g the set of functions $Comm(g)$ from F is such that
- *$g \notin Comm(g)$,*
- *each element of $Comm(g)$ commutes with g.*

If $update(G, g, d)$ satisfies assumptions **A**, **B** *and* **C**, *then so does the function*

$$update(G, g, d) - Comm(g).$$

Proof. It suffices to establish in each case assumption **A** and **C**. Let

$$A := \{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\}.$$

(i) After introducing the `GI` algorithm we noted already that $g \notin A$. So assumption **A** implies $A \subseteq update(G, g, d) - \{g\}$ and a fortiori $A \subseteq update(G, g, d) - Idemp(g)$.

For assumption **C** it suffices to note that $g(g(d)) \neq g(d)$ implies that g is not idempotent, i.e., that $Idemp(g) = \emptyset$.

(ii) Consider $f \in A$. Suppose that $f \in Comm(g)$. Then $f(g(d) = g(f(d)) = g(d)$ which is a contradiction. So $f \notin Comm(g)$. Consequently, assumption **A** implies $A \subseteq update(G, g, d) - Comm(g)$.

For assumption **C** it suffices to use the fact that $g \notin Comm(g)$. □

We conclude that given an instance of the `GI` algorithm that employs a specific $update$ function, we can obtain other instances of it by using $update$ functions modified as above. Note that both modifications are independent of each other and therefore can be applied together.

In particular, when each function is idempotent and the function $Comm$ satisfies the assumptions of (ii), then the following holds: if $update(G, g, d)$ satisfies assumptions **A**, **B** and **C**, then so does the function $update(G, g, d) - (Comm(g) \cup \{g\})$.

3 Compound Domains

In the applications we study the iterations are carried out on a partial ordering that is a Cartesian product of the partial orderings. So assume now that the partial ordering $(D, \sqsubseteq)$ is the Cartesian product of some partial orderings $(D_i, \sqsubseteq_i)$, for $i \in [1..n]$, each with the least element $\bot_i$. So $D = D_1 \times \cdots \times D_n$.

Further, we assume that each function from F depends from and affects only certain components of D. To be more precise we introduce a simple notation and terminology.

Definition 4. *Consider a sequence of partial orderings $(D_1, \sqsubseteq_1), \ldots, (D_n, \sqsubseteq_n)$.*

- *By a* scheme *(on n) we mean a growing sequence of different elements from $[1..n]$.*
- *Given a scheme $s := i_1, \ldots, i_l$ on n we denote by $(D_s, \sqsubseteq_s)$ the Cartesian product of the partial orderings $(D_{i_j}, \sqsubseteq_{i_j})$, for $j \in [1..l]$.*
- *Given a function f on D_s we say that f is* with scheme *s and say that f* depends on *i if i is an element of s.*
- *Given an n-tuple $d := d_1, \ldots, d_n$ from D and a scheme $s := i_1, \ldots, i_l$ on n we denote by $d[s]$ the tuple $d_{i_1}, \ldots, d_{i_l}$. In particular, for $j \in [1..n]$ $d[j]$ is the j-th element of d.* □

Consider now a function f with scheme s. We extend it to a function f^+ from D to D as follows. Take $d \in D$. We set

$$f^+(d) := e$$

where $e[s] = f(d[s])$ and $e[n-s] = d[n-s]$, and where $n - s$ is the scheme obtained by removing from $1, \ldots, n$ the elements of s. We call f^+ the *canonic extension* of f to the domain D.

So $f^+(d_1, \ldots, d_n) = (e_1, \ldots, e_n)$ implies $d_i = e_i$ for any i not in the scheme s of f. Informally, we can summarize it by saying that f^+ does not change the components on which it does not depend. This is what we meant above by stating that each considered function affects only certain components of D.

We now say that two functions, f with scheme s and g with scheme t *commute* if the functions f^+ and g^+ commute.

Instead of defining iterations for the case of the functions with schemes, we rather reduce the situation to the one studied in the previous section and consider, equivalently, the iterations of the canonic extensions of these functions to the common domain D. However, because of this specific form of the considered functions, we can use now a simple definition of the $update$ function. More precisely, we have the following observation.

Note 1 (Update). Suppose that each function in F is of the form f^+. Then the following function $update$ satisfies assumptions **A**, **B** and **C**:

$$update(G, g^+, d) := \{f^+ \in F \mid f \text{ depends on some } i \text{ in } s \text{ such that } d[i] \neq g^+(d)[i]\},$$

where g is with scheme s.

Proof. To deal with assumption **A** take a function $f^+ \in F - G$ such that $f^+(d) = d$. Then $f^+(e) = e$ for any e that coincides with d on all components that are in the scheme of f.

Suppose now additionally that $f^+(g^+(d)) \neq g^+(d)$. By the above $g^+(d)$ is not such an e, i.e., $g^+(d)$ differs from d on some component i in the scheme of f. In other words, f depends on some i such that $d[i] \neq g^+(d)[i]$. This i is then in the scheme of g and consequently $f^+ \in update(G, g^+, d)$.

The proof for assumption **B** is immediate.

Finally, to deal with assumption **C** it suffices to note that $g^+(g^+(d)) \neq g^+(d)$ implies $g^+(d)) \neq d$, which in turn implies that $g^+ \in update(G, g^+, d)$. □

This, together with the GI algorithm, yields the following algorithm in which we introduced a variable d' to hold the value of $g^+(d)$, and used $F_0 := \{f \mid f^+ \in F\}$ and the functions with schemes instead of their canonic extensions to D.

GENERIC ITERATION ALGORITHM FOR COMPOUND DOMAINS (CD)

$d := (\bot_1, \ldots, \bot_n)$;
$d' := d$;
$G := F_0$;
while $G \neq \emptyset$ **do**
  choose $g \in G$; suppose g is with scheme s;
  $G := G - \{g\}$;
  $d'[s] := g(d[s])$;
  $G := G \cup \{f \in F_0 \mid f$ depends on some i in s such that $d[i] \neq d'[i]\}$;
  $d[s] := d'[s]$
od

The following corollary to the GI Theorem 1 and the Update Note 1 summarizes the correctness of this algorithm.

Corollary 2 (CD). *Suppose that $(D, \sqsubseteq)$ is a finite partial ordering that is a Cartesian product of n partial orderings, each with the least element $\bot_i$ with $i \in [1..n]$. Let F be a finite set of functions on D, each of the form f^+.*

Suppose that all functions in F are monotonic and inflationary. Then every execution of the CD algorithm terminates and computes in d the least common fixpoint of the functions from F. □

In the subsequent presentation we shall deal with the following two modifications of the CD algorithm:

- CDI *algorithm*. This is the version of the CD algorithm in which all the functions are idempotent and the function $update$ defined in the Update Theorem $2(i)$ is used.
- CDC *algorithm*. This is the version of the CD algorithm in which all the functions are idempotent and the combined effect of the functions $update$ defined in the Update Theorem 2 is used for some function $Comm$.

For both algorithms the counterparts of the CD Corollary 2 hold.

4 From Partial Orderings to Constraint Satisfaction Problems

We have been so far completely general in our discussion. Recall that our aim is to derive various constraint propagation algorithms. To be able to apply the results of the previous section we need to relate various abstract notions that we used there to constraint satisfaction problems.

This is perhaps the right place to recall the definition and to fix the notation. Consider a finite sequence of variables $X := x_1, \ldots, x_n$, where $n \geq 0$, with respective domains $\mathcal{D} := D_1, \ldots, D_n$ associated with them. So each variable x_i ranges over the domain D_i. By a *constraint* C on X we mean a subset of $D_1 \times \ldots \times D_n$.

By a *constraint satisfaction problem*, in short CSP, we mean a finite sequence of variables X with respective domains $\mathcal{D}$, together with a finite set $\mathcal{C}$ of constraints, each on a subsequence of X. We write it as $\langle \mathcal{C} ; x_1 \in D_1, \ldots, x_n \in D_n \rangle$, where $X := x_1, \ldots, x_n$ and $\mathcal{D} := D_1, \ldots, D_n$.

Consider now an element $d := d_1, \ldots, d_n$ of $D_1 \times \ldots \times D_n$ and a subsequence $Y := x_{i_1}, \ldots, x_{i_\ell}$ of X. Then we denote by $d[Y]$ the sequence $d_{i_1}, \ldots, d_{i_\ell}$.

By a *solution* to $\langle \mathcal{C} ; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ we mean an element $d \in D_1 \times \ldots \times D_n$ such that for each constraint $C \in \mathcal{C}$ on a sequence of variables Y we have $d[Y] \in C$. We call a CSP *consistent* if it has a solution. Two CSP's $\mathcal{P}_1$ and $\mathcal{P}_2$ with the same sequence of variables are called *equivalent* if they have the same set of solutions. This definition extends in an obvious way to the case of two CSP's with the same *sets* of variables.

Let us return now to the framework of the previous section. It involved:

(i) Partial orderings with the least elements;
These will correspond to partial orderings on the CSP's. In each of them the original CSP will be the least element and the partial ordering will be determined by the local consistency notion we wish to achieve.

(ii) Monotonic and inflationary functions with schemes;
These will correspond to the functions that transform the variable domains or the constraints. Each function will be associated with one or more constraints.

(iii) Common fixpoints;
These will correspond to the CSP's that satisfy the considered notion of local consistency.

In what follows we shall discuss two specific partial orderings on the CSP's. In each of them the considered CSP's will be defined on the same sequences of variables.

We begin by fixing for each set D a collection $\mathcal{F}(D)$ of the subsets of D that includes D itself. So $\mathcal{F}$ is a function that given a set D yields a set of its subsets to which D belongs.

When dealing with the hyper-arc consistency $\mathcal{F}(D)$ will be simply the set $\mathcal{P}(D)$ of all subsets of D but for specific domains only specific subsets of D will be chosen. For example, to deal with the the constraint propagation for the linear constraints on integer interval domains we need to choose for $\mathcal{F}(D)$ the set of all subintervals of the original interval D.

When dealing with the path consistency, for a constraint C the collection $\mathcal{F}(C)$ will be also the set $\mathcal{P}(C)$ of all subsets of C. However, in general other choices may be needed. For example, to deal with the cutting planes method, we need to limit our attention to the sets of integer solutions to finite sets of linear inequalities with integer coefficients (see Apt [1, pages 193-194]).

Next, given two CSP's, $\phi := \langle \mathcal{C} ; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ and $\psi := \langle \mathcal{C}' ; x_1 \in D'_1, \ldots, x_n \in D'_n \rangle$, we write $\phi \sqsubseteq_d \psi$ iff

– $D'_i \in \mathcal{F}(D_i)$ (and hence $D'_i \subseteq D_i$) for $i \in [1..n]$,
– the constraints in $\mathcal{C}'$ are the restrictions of the constraints in $\mathcal{C}$ to the domains $D'_1, \ldots, D'_n$.

Next, given two CSP's, $\phi := \langle C_1, \ldots, C_k \ ; \ \mathcal{DE}\rangle$ and $\psi := \langle C'_1, \ldots, C'_k \ ; \ \mathcal{DE}\rangle$, we write $\phi \sqsubseteq_c \psi$ iff

– $C'_i \in \mathcal{F}(C_i)$ (and hence $C'_i \subseteq C_i$) for $i \in [1..k]$.

In what follows we call $\sqsubseteq_d$ the *domain reduction ordering* and $\sqsubseteq_c$ the *constraint reduction ordering*. To deal with the arc consistency, hyper-arc consistency and directional arc consistency notions we shall use the domain reduction ordering, and to deal with path consistency and directional path consistency notions we shall use the constraint reduction ordering.

We consider each ordering with some fixed initial CSP $\mathcal{P}$ as the least element. In other words, each domain reduction ordering is of the form

$$(\{\mathcal{P}' \mid \mathcal{P} \sqsubseteq_d \mathcal{P}'\}, \sqsubseteq_d)$$

and each constraint reduction ordering is of the form

$$(\{\mathcal{P}' \mid \mathcal{P} \sqsubseteq_c \mathcal{P}'\}, \sqsubseteq_c).$$

Note that $\langle \mathcal{C} \ ; \ x_1 \in D'_1, \ldots, x_n \in D'_n\rangle \sqsubseteq_d \langle \mathcal{C}' \ ; \ x_1 \in D''_1, \ldots, x_n \in D''_n\rangle$ iff $D'_i \supseteq D''_i$ for $i \in [1..n]$.

This means that for $\mathcal{P} = \langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n\rangle$ we can identify the domain reduction ordering $(\{\mathcal{P}' \mid \mathcal{P} \sqsubseteq_d \mathcal{P}'\}, \sqsubseteq_d)$ with the Cartesian product of the partial orderings $(\mathcal{F}(D_i), \supseteq)$, where $i \in [1..n]$. Additionally, each CSP in this domain reduction ordering is uniquely determined by its domains and by the initial $\mathcal{P}$.

Similarly,

$$\langle C'_1, \ldots, C'_k \ ; \ \mathcal{DE}\rangle \sqsubseteq_c \langle C''_1, \ldots, C''_k \ ; \ \mathcal{DE}\rangle \text{ iff } C'_i \supseteq C''_i \text{ for } i \in [1..k].$$

This allows us for $\mathcal{P} = \langle C_1, \ldots, C_k \ ; \ \mathcal{DE}\rangle$ to identify the constraint reduction ordering $(\{\mathcal{P}' \mid \mathcal{P} \sqsubseteq_c \mathcal{P}'\}, \sqsubseteq_c)$ with the Cartesian product of the partial orderings $(\mathcal{F}(C_i), \supseteq)$, where $i \in [1..k]$. Also, each CSP in this constraint reduction ordering is uniquely determined by its constraints and by the initial $\mathcal{P}$.

In what follows instead of the domain reduction ordering and the constraint reduction ordering we shall use the corresponding Cartesian products of the partial orderings. So in these compound orderings the sequences of the domains (respectively, of the constraints) are ordered componentwise by the reversed subset ordering $\supseteq$. Further, in each component ordering $(\mathcal{F}(D), \supseteq)$ the set D is the least element.

Consider now a function f on some Cartesian product $\mathcal{F}(E_1) \times \ldots \times \mathcal{F}(E_m)$. Note that f is inflationary w.r.t. the componentwise ordering $\supseteq$ if for all $(X_1, \ldots, X_m) \in \mathcal{F}(E_1) \times \ldots \times \mathcal{F}(E_m)$ we have $Y_i \subseteq X_i$ for all $i \in [1..m]$, where $f(X_1, \ldots, X_m) = (Y_1, \ldots, Y_m)$.

Also, f is monotonic w.r.t. the componentwise ordering $\supseteq$ if for all $(X_1, \ldots, X_m)$, $(X'_1, \ldots, X'_m) \in \mathcal{F}(E_1) \times \ldots \times \mathcal{F}(E_m)$ such that $X_i \subseteq X'_i$ for all $i \in [1..m]$, the following holds: if

$$f(X_1, \ldots, X_m) = (Y_1, \ldots, Y_m) \text{ and } f(X'_1, \ldots, X'_m) = (Y'_1, \ldots, Y'_m),$$

then $Y_i \subseteq Y'_i$ for all $i \in [1..m]$.

In other words, f is monotonic w.r.t. $\supseteq$ iff it is monotonic w.r.t. $\subseteq$. This reversal of the set inclusion of course does not hold for the inflationarity notion.

5 A Hyper-arc Consistency Algorithm

We begin by considering the notion of hyper-arc consistency of Mohr and Masini [12] (we use here the terminology of Marriott and Stuckey [10]). The more known notion of arc consistency of Mackworth [9] is obtained by restricting one's attention to binary constraints.

To employ the `CDI` algorithm of Section 3 we now make specific choices involving the items (i), (ii) and (iii) of the previous section.

Re: (i) Partial orderings with the least elements.

As already mentioned in the previous section, for the function $\mathcal{F}$ we choose the powerset function $\mathcal{P}$, so for each domain D we put $\mathcal{F}(D) := \mathcal{P}(D)$.

Given a CSP $\mathcal{P}$ with the sequence $D_1, \ldots, D_n$ of the domains we take the domain reduction ordering with $\mathcal{P}$ as its least element. As already noted we can identify this ordering with the Cartesian product of the partial orderings $(\mathcal{P}(D_i), \supseteq)$, where $i \in [1..n]$. The elements of this compound ordering are thus sequences $(X_1, \ldots, X_n)$ of respective subsets of the domains $D_1, \ldots, D_n$ ordered componentwise by the reversed subset ordering $\supseteq$.

Re: (ii) Monotonic and inflationary functions with schemes.

Given a constraint C on the variables $y_1, \ldots, y_k$ with respective domains $E_1, \ldots, E_k$, we abbreviate for each $j \in [1..k]$ the set $\{d[j] \mid d \in C\}$ to $\Pi_j(C)$. Thus $\Pi_j(C)$ consists of all j-th coordinates of the elements of C. Consequently, $\Pi_j(C)$ is a subset of the domain E_j of the variable y_j.

We now introduce for each $i \in [1..k]$ the following function π_i on $\mathcal{P}(E_1) \times \cdots \times \mathcal{P}(E_k)$:

$$\pi_i(X_1, \ldots, X_k) := (X_1, \ldots, X_{i-1}, X'_i, X_{i+1}, \ldots, X_k)$$

where

$$X'_i := \Pi_i(C \cap (X_1 \times \cdots \times X_k)).$$

That is, $X'_i = \{d[i] \mid d \in X_1 \times \cdots \times X_k \text{ and } d \in C\}$. Each function π_i is associated with a specific constraint C. Note that $X'_i \subseteq X_i$, so each function π_i boils down to a projection on the i-th component.

Re: (iii) Common fixpoints.

Their use is clarified by the following lemma that also lists the relevant properties of the functions π_i.

Lemma 2 (Hyper-arc Consistency).

(i) *A CSP* $\langle \mathcal{C} \; ; \; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ *is hyper-arc consistent iff* $(D_1, \ldots, D_n)$ *is a common fixpoint of all functions* π_i^+ *associated with the constraints from* $\mathcal{C}$.

(ii) *Each projection function* π_i *associated with a constraint* C *is*
- *inflationary w.r.t. the componentwise ordering* $\supseteq$,
- *monotonic w.r.t. the componentwise ordering* $\supseteq$,
- *idempotent.* $\Box$

By taking into account only the binary constraints we obtain an analogous characterization of arc consistency. The functions π_1 and π_2 can then be defined more directly as follows:

$$\pi_1(X, Y) := (X', Y),$$

where $X' := \{a \in X \mid \exists\, b \in Y \; (a, b) \in C\}$, and

$$\pi_2(X, Y) := (X, Y'),$$

where $Y' := \{b \in Y \mid \exists a \in X \; (a, b) \in C\}$.

Fix now a CSP $\mathcal{P}$. By instantiating the `CDI` algorithm with

$$F_0 := \{f \mid f \text{ is a } \pi_i \text{ function associated with a constraint of } \mathcal{P}\}$$

and with each $\perp_i$ equal to D_i we get the `HYPER-ARC` algorithm that enjoys following properties.

Theorem 3 (`HYPER-ARC` Algorithm). *Consider a CSP* $\mathcal{P} := \langle \mathcal{C} \; ; \; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ *where each* D_i *is finite.*

The `HYPER-ARC` *algorithm always terminates. Let* $\mathcal{P}'$ *be the CSP determined by* $\mathcal{P}$ *and the sequence of the domains* $D'_1, \ldots, D'_n$ *computed in d. Then*

(i) $\mathcal{P}'$ *is the* $\sqsubseteq_d$*-least CSP that is hyper-arc consistent,*

(ii) $\mathcal{P}'$ *is equivalent to* $\mathcal{P}$. $\Box$

Due to the definition of the $\sqsubseteq_d$ ordering the item (i) can be rephrased as follows. Consider all hyper-arc consistent CSP's that are of the form $\langle \mathcal{C}' \; ; \; x_1 \in D'_1, \ldots, x_n \in D'_n \rangle$ where $D'_i \subseteq D_i$ for $i \in [1..n]$ and the constraints in $\mathcal{C}'$ are the restrictions of the constraints in $\mathcal{C}$ to the domains $D'_1, \ldots, D'_n$. Then among these CSP's $\mathcal{P}'$ has the largest domains.

6 An Improvement: the `AC-3` Algorithm

In this section we show how we can exploit an information about the commutativity of the π_i functions. Recall that in Section 3 we modified the notion of commutativity for the case of functions with schemes. We now need the following lemma.

Lemma 3 (Commutativity). *Consider a CSP and two constraints of it,* C *on the variables* $y_1, \ldots, y_k$ *and* E *on the variables* $z_1, \ldots, z_\ell$.

(i) For $i, j \in [1..k]$ the functions π_i and π_j of the constraint C commute.
(ii) If the variables y_i and z_j are identical then the functions π_i of C and π_j of E commute. □

Fix now a CSP. We derive a modification of the `HYPER-ARC` algorithm by instantiating this time the `CDC` algorithm. As before we use the set of functions $F_0 := \{f \mid f \text{ is a } \pi_i \text{ function associated with a constraint of } \mathcal{P}\}$ and each $\perp_i$ equal to D_i. Additionally we employ the following function *Comm*, where π_i is associated with a constraint C:

$$\begin{aligned} Comm(\pi_i) := &\{\pi_j \mid i \neq j \text{ and } \pi_j \text{ is associated with the constraint } C\} \\ &\cup \{\pi_j \mid \pi_j \text{ is associated with a constraint } E \text{ and} \\ &\qquad \text{the } i\text{-th variable of } C \text{ and the } j\text{-th variable of } E \text{ coincide}\}. \end{aligned}$$

By virtue of the Commutativity Lemma 3 each set $Comm(g)$ satisfies the assumptions of the Update Theorem $2(ii)$.

By limiting oneself to the set of functions π_1 and π_2 associated with the binary constraints, we obtain an analogous modification of the corresponding arc consistency algorithm.

Using now the counterpart of the `CD` Corollary 2 for the `CDC` algorithm we conclude that the above algorithm enjoys the same properties as the `HYPER-ARC` algorithm, that is the counterpart of the `HYPER-ARC` Algorithm Theorem 3 holds.

Let us clarify now the difference between this algorithm and the `HYPER-ARC` algorithm when both of them are limited to the binary constraints.

Assume that the considered CSP is of the form $\langle \mathcal{C} \ ; \ \mathcal{DE} \rangle$. We reformulate the above algorithm as follows. Given a binary relation R, we put

$$R^T := \{(b, a) \mid (a, b) \in R\}.$$

For F_0 we now choose the set of the π_1 functions of the constraints or relations from the set

$$\begin{aligned} S_0 := &\{C \mid C \text{ is a binary constraint from } \mathcal{C}\} \\ &\cup \{C^T \mid C \text{ is a binary constraint from } \mathcal{C}\}. \end{aligned}$$

Finally, for each π_1 function of some $C \in S_0$ on x, y we define

$$\begin{aligned} Comm(\pi_1) := &\{f \mid f \text{ is the } \pi_1 \text{ function of } C^T\} \\ &\cup \{f \mid f \text{ is the } \pi_1 \text{ function of some } E \in S_0 \text{ on } x, z \text{ where } z \not\equiv y\}. \end{aligned}$$

Assume now that

$$\text{for each pair of variables } x, y \text{ at most one constraint exists on } x, y. \tag{1}$$

Consider now the corresponding instance of the `CDC` algorithm. By incorporating into it the effect of the functions π_1 on the corresponding domains, we obtain the following algorithm known as the `AC-3` algorithm of Mackworth [9].

We assume here that $\mathcal{DE} := x_1 \in D_1, \ldots, x_n \in D_n$.

`AC-3` ALGORITHM

```
S_0 := {C | C is a binary constraint from 𝒞}
    ∪ {C^T | C is a binary constraint from 𝒞};
S := S_0;
while S ≠ ∅ do
    choose C ∈ S; suppose C is on x_i, x_j;
    D_i := {a ∈ D_i | ∃ b ∈ D_j (a, b) ∈ C};
    if D_i changed then
        S := S ∪ {C' ∈ S_0 | C' is on the variables y, x_i where y ≢ x_j}
    fi;
    S := S − {C}
od
```

It is useful to mention that the corresponding reformulation of the `HYPER−ARC` algorithm differs in the second assignment to S which is then

$$S := S \cup \{C' \in S_0 \mid C' \text{ is on the variables } y, z \text{ where } y \text{ is } x_i \text{ or } z \text{ is } x_i\}.$$

So we "capitalized" here on the commutativity of the corresponding projection functions π_1 as follows. First, no constraint or relation on x_i, z for some z is added to S. Here we exploited part (ii) of the Commutativity Lemma 3.

Second, no constraint or relation on x_j, x_i is added to S. Here we exploited part (i) of the Commutativity Lemma 3, because by assumption (1) C^T is the only constraint or relation on x_j, x_i and its π_1 function coincides with the π_2 function of C.

In case the assumption (1) about the considered CSP is dropped, the resulting algorithm is somewhat less readable. However, once we use the following modified definition of $Comm(\pi_1)$:

$$Comm(\pi_1) := \{f \mid f \text{ is the } \pi_1 \text{ function of some } E \in S_0 \text{ on } x, z \text{ where } z \not\equiv y\}$$

we get an instance of the `CDC` algorithm which differs from the `AC−3` algorithm in that the qualification "where $y \not\equiv x_j$" is removed from the definition of the second assignment to the set S.

7 A Path Consistency Algorithm

The notion of path consistency was introduced in Montanari [14]. It is defined for special type of CSP's. For simplicity we ignore here unary constraints that are usually present when studying path consistency.

Definition 5. *We call a CSP* normalized *if it has only binary constraints and for each pair x, y of its variables exactly one constraint on them exists. We denote this constraint by $C_{x,y}$.* □

Every CSP with only unary and binary constraints is trivially equivalent to a normalized CSP. Consider now a normalized CSP $\mathcal{P}$. Suppose that $\mathcal{P} = \langle C_1, \ldots, C_k \,;\, \mathcal{DE}\rangle$.

We proceed now as in the case of hyper-arc consistency. First, we choose for the function $\mathcal{F}$ the powerset function. For the partial ordering we choose the constraint reduction ordering of Section 4, or rather its counterpart which is the Cartesian product of the partial orderings $(\mathcal{P}(C_i), \supseteq)$, where $i \in [1..k]$.

Second, we introduce appropriate monotonic and inflationary functions with schemes. To this end, given two binary relations R and S we define their composition $\cdot$ by

$$R \cdot S := \{(a, b) \mid \exists c\,((a, c) \in R, (c, b) \in S)\}.$$

Note that if R is a constraint on the variables x, y and S a constraint on the variables y, z, then $R \cdot S$ is a constraint on the variables x, z.

Given a subsequence x, y, z of the variables of $\mathcal{P}$ we now introduce three functions on $\mathcal{P}(C_{x,y}) \times \mathcal{P}(C_{x,z}) \times \mathcal{P}(C_{y,z})$:

$$f^z_{x,y}(P, Q, R) := (P', Q, R),$$

where $P' := P \cap Q \cdot R^T$,

$$f^y_{x,z}(P, Q, R) := (P, Q', R),$$

where $Q' := Q \cap P \cdot R$, and

$$f^x_{y,z}(P, Q, R) := (P, Q, R'),$$

where $R' := R \cap P^T \cdot Q$.

Finally, we introduce common fixpoints of the above defined functions. To this end we need the following counterpart of the Hyper-arc Consistency Lemma 2.

Lemma 4 (Path Consistency).
(i) *A normalized CSP $\langle C_1, \ldots, C_k\ ;\ \mathcal{DE}\rangle$ is path consistent iff $(C_1, \ldots, C_k)$ is a common fixpoint of all functions $(f^z_{x,y})^+$, $(f^y_{x,z})^+$ and $(f^x_{y,z})^+$ associated with the subsequences x, y, z of its variables.*
(ii) *The functions $f^z_{x,y}$, $f^y_{x,z}$ and $f^x_{y,z}$ are*
- *inflationary w.r.t. the componentwise ordering $\supseteq$,*
- *monotonic w.r.t. the componentwise ordering $\supseteq$,*
- *idempotent.* □

We now instantiate the `CDI` algorithm with the set of functions

$$F_0 := \{f \mid x, y, z \text{ is a subsequence of the variables of } \mathcal{P} \text{ and } f \in \{f^z_{x,y}, f^y_{x,z}, f^x_{y,z}\}\},$$

$n := k$ and each $\perp_i$ equal to C_i.

Call the resulting algorithm the `PATH` algorithm. It enjoys the following properties.

Theorem 4 (`PATH` Algorithm). *Consider a normalized CSP $\mathcal{P} := \langle C_1, \ldots, C_k\ ;\ \mathcal{DE}\rangle$. Assume that each constraint C_i is finite.*

The `PATH` algorithm always terminates. Let $\mathcal{P}' := \langle C'_1, \ldots, C'_k\ ;\ \mathcal{DE}\rangle$, where the sequence of the constraints $C'_1, \ldots, C'_k$ is computed in d. Then

(i) *$\mathcal{P}'$ is the $\sqsubseteq_c$-least CSP that is path consistent,*
(ii) *$\mathcal{P}'$ is equivalent to $\mathcal{P}$.* □

As in the case of the `HYPER−ARC` Algorithm Theorem 3 the item (i) can be rephrased as follows. Consider all path consistent CSP's that are of the form $\langle C'_1, \ldots, C'_k\ ;\ \mathcal{DE}\rangle$ where $C'_i \subseteq C_i$ for $i \in [1..k]$. Then among them $\mathcal{P}'$ has the largest constraints.

8 An Improvement: the `PC−2` Algorithm

As in the case of the hyper-arc consistency we can improve the `PATH` algorithm by taking into account the commutativity information.

Fix a normalized CSP $\mathcal{P}$. We abbreviate the statement "x, y is a subsequence of the variables of $\mathcal{P}$" to $x \prec y$. We now have the following lemma.

Lemma 5 (Commutativity). *Suppose that $x \prec y$ and let z, u be some variables of $\mathcal{P}$ such that $\{u, z\} \cap \{x, y\} = \emptyset$. Then the functions $f^z_{x,y}$ and $f^u_{x,y}$ commute.* $\square$

In other words, two functions with the same pair of variables as a subscript commute.

We now instantiate the `CDC` algorithm with the same set of functions F_0 as in Section 7. Additionally, we use the function $Comm$ defined as follows, where $x \prec y$ and where $z \notin \{x, y\}$:

$$Comm(f^z_{x,y}) = \{f^u_{x,y} \mid u \notin \{x, y, z\}\}.$$

Thus for each function g the set $Comm(g)$ contains precisely $m - 3$ elements, where m is the number of variables of the considered CSP. This quantifies the maximal "gain" obtained by using the commutativity information: at each "update" stage of the corresponding instance of the `CDC` algorithm we add up to $m - 3$ less elements than in the case of the corresponding instance of the `CDI` algorithm considered in the previous section.

By virtue of the Commutativity Lemma 5 each set $Comm(g)$ satisfies the assumptions of the Update Theorem $2(ii)$. We conclude that the above instance of the `CDC` algorithm enjoys the same properties as the original `PATH` algorithm, that is the counterpart of the `PATH` Algorithm Theorem 4 holds. To make this modification of the `PATH` algorithm easier to understand we proceed as follows.

Each function of the form $f^u_{x,y}$ where $x \prec y$ and $u \notin \{x, y\}$ can be identified with the sequence x, u, y of the variables. (Note that the "relative" position of u w.r.t. x and y is not fixed, so x, u, y does not have to be a subsequence of the variables of $\mathcal{P}$.) This allows us to identify the set of functions F_0 with the set

$$V_0 := \{(x, u, y) \mid x \prec y, u \notin \{x, y\}\}.$$

Next, assuming that $x \prec y$, we introduce the following set of triples of different variables of $\mathcal{P}$:

$$\begin{aligned} V_{x,y} := \; & \{(x, y, u) \mid x \prec u\} \cup \{(y, x, u) \mid y \prec u\} \\ \cup \; & \{(u, x, y) \mid u \prec y\} \cup \{(u, y, x) \mid u \prec x\}. \end{aligned}$$

Informally, $V_{x,y}$ is the subset of V_0 that consists of the triples that begin or end with either x, y or y, x. This corresponds to the set of functions in one of the following forms: $f^y_{x,u}$, $f^x_{y,u}$, $f^x_{u,y}$ and $f^y_{u,x}$.

The above instance of the `CDC` algorithm then becomes the following `PC−2` algorithm of Mackworth [9]. Here initially $E_{x,y} = C_{x,y}$.

`PC−2` ALGORITHM

$V_0 := \{(x, u, y) \mid x \prec y, u \notin \{x, y\}\}$;
$V := V_0$;
while $V \neq \emptyset$ **do**
  choose $p \in V$; suppose $p = (x, u, y)$;
  apply $f^u_{x,y}$ to its current domains;
  if $E_{x,y}$ changed **then**
    $V := V \cup V_{x,y}$;
  fi;
  $V := V - \{p\}$
od

Here the phrase "apply $f^u_{x,y}$ to its current domains" can be made more precise if the "relative" position of u w.r.t. x and y is known. Suppose for instance that u is "before" x and y. Then $f^u_{x,y}$ is defined on $\mathcal{P}(C_{u,x}) \times \mathcal{P}(C_{u,y}) \times \mathcal{P}(C_{x,y})$ by

$$f^u_{x,y}(E_{u,x}, E_{u,y}, E_{x,y}) := (E_{u,x}, E_{u,y}, E_{x,y} \cap E^T_{u,x} \cdot E_{u,y}),$$

so the above phrase "apply $f^u_{x,y}$ to its current domains" can be replaced by the assignment

$$E_{x,y} := E_{x,y} \cap E^T_{u,x} \cdot E_{u,y}.$$

Analogously for the other two possibilities.

The difference between the `PC-2` algorithm and the corresponding representation of the `PATH` algorithm lies in the way the modification of the set V is carried out. In the case of the `PATH` algorithm the second assignment to V is

$$V := V \cup V_{x,y} \cup \{(x, u, y) \mid u \notin \{x, y\}\}.$$

9 Simple Iteration Algorithms

Let us return now to the framework of Section 2. We analyze here when the **while** loop of the GENERIC ITERATION ALGORITHM `GI` can be replaced by a **for** loop. First, we weaken the notion of commutativity as follows.

Definition 6. *Consider a partial ordering $(D, \sqsubseteq)$ and functions f and g on D. We say that f* semi-commutes with *g (*w.r.t. *$\sqsubseteq$) if $f(g(x)) \sqsubseteq g(f(x))$ for all x.* □

The following lemma provides an answer to the question just posed. Here and elsewhere we omit brackets when writing repeated applications of functions to an argument.

Lemma 6 (Simple Iteration). *Consider a partial ordering $(D, \sqsubseteq)$ with the least element $\perp$. Let $F := f_1, \ldots, f_k$ be a finite sequence of monotonic, inflationary and idempotent functions on D. Suppose that f_i semi-commutes with f_j for $i > j$, that is,*

$$f_i(f_j(x)) \sqsubseteq f_j(f_i(x)) \text{ for all } x. \tag{2}$$

Then $f_1 f_2 \ldots f_k(\perp)$ is the least common fixpoint of the functions from F. □

Proof. We prove first that for $i \in [1..k]$ we have

$$f_i f_1 f_2 \ldots f_k(\bot) \sqsubseteq f_1 f_2 \ldots f_k(\bot).$$

Indeed, by the assumption (2) we have the following string of inclusions, where the last one is due to the idempotence of the considered functions:

$$f_i f_1 f_2 \ldots f_k(\bot) \sqsubseteq f_1 f_i f_2 \ldots f_k(\bot) \sqsubseteq \ldots \sqsubseteq f_1 f_2 \ldots f_i f_i \ldots f_k(\bot) \sqsubseteq f_1 f_2 \ldots f_k(\bot).$$

Additionally, by the inflationarity of the considered functions, we also have for $i \in [1..k]$

$$f_1 f_2 \ldots f_k(\bot) \sqsubseteq f_i f_1 f_2 \ldots f_k(\bot).$$

So $f_1 f_2 \ldots f_k(\bot)$ is a common fixpoint of the functions from F. This means that the iteration of F that starts with $\bot$, $f_k(\bot)$, $f_{k-1} f_k(\bot), \ldots, f_1 f_2 \ldots f_k(\bot)$ eventually stabilizes at $f_1 f_2 \ldots f_k(\bot)$. By the Stabilization Lemma 1 we get the desired conclusion. $\Box$

The above lemma provides us with a simple way of computing the least common fixpoint of a set of finite functions that satisfy the assumptions of this lemma, in particular condition (2). Namely, it suffices to order these functions in an appropriate way and then to apply each of them just once, starting with the argument $\bot$.

To this end we maintain the considered functions not in a set but in a list. Given a non-empty list L we denote its head by $\mathbf{head}(L)$ and its tail by $\mathbf{tail}(L)$. Next, given a sequence of elements $a_1, \ldots, a_n$ with $n \geq 0$, we denote by $[a_1, \ldots, a_n]$ the list formed by them. If $n = 0$, then this list is empty and is denoted by $[\,]$ and if $n > 0$, then $\mathbf{head}([a_1, \ldots, a_n]) = a_1$ and $\mathbf{tail}([a_1, \ldots, a_n]) = [a_2, \ldots a_n]$.

The following algorithm is a counterpart of the `GI` algorithm. We assume in it that condition (2) holds for the functions $f_1, \ldots, f_k$.

SIMPLE ITERATION ALGORITHM (`SI`)

$d := \bot$;
$L := [f_k, f_{k-1}, \ldots, f_1]$;
for $i := 1$ **to** k **do**
    $g := \mathbf{head}(L)$;
    $L := \mathbf{tail}(L)$;
    $d := g(d)$
od

The following immediate consequence of the Simple Iteration Lemma 6 is a counterpart of the `GI` Corollary 1.

Corollary 3 (`SI`) . *Suppose that $(D, \sqsubseteq)$ is a partial ordering with the least element $\bot$. Let $F := f_1, \ldots, f_k$ be a finite sequence of monotonic, inflationary and idempotent functions on D such that (2) holds. Then the `SI` algorithm terminates and computes in d the least common fixpoint of the functions from F.* $\Box$

Note that in contrast to the GI Corollary 1 we do not require here that the partial ordering is finite. Because at each iteration of the **for** loop exactly one element is removed from the list L, at the end of this loop the list L is empty. Consequently, this algorithm is a reformulation of the one in which the line

for $i := 1$ **to** k **do**

is replaced by

while $L \neq [\,]$ **do**.

So we can view the SI algorithm as a specialization of the GI algorithm of Section 2 in which the elements of the set of functions G (here represented by the list L) are selected in a specific way and in which the $update$ function always yields the empty set.

In Section 3 we refined the GI algorithm for the case of compound domains. An analogous refinement of the SI algorithm is straightforward and omitted. In the next two sections we show how we can use this refinement of the SI algorithm to derive two well-known constraint propagation algorithms.

10 DAC: a Directional Arc Consistency Algorithm

We consider here the notion of directional arc consistency of Dechter and Pearl [6]. To derive an algorithm that achieves this local consistency notion we first characterize it in terms of fixpoints. To this end, given a $\mathcal{P}$ and a linear ordering $\prec$ on its variables, we rather reason in terms of the equivalent CSP $\mathcal{P}_{\prec}$ obtained from $\mathcal{P}$ by reordering its variables along $\prec$ so that each constraint in $\mathcal{P}_{\prec}$ is on a sequence of variables $x_1, \ldots, x_k$ such that $x_1 \prec x_2 \prec \ldots \prec x_k$.

The following characterization holds.

Lemma 7 (Directional Arc Consistency). *Consider a CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables. Let $\mathcal{P}_{\prec} := \langle \mathcal{C} \,;\, x_1 \in D_1, \ldots, x_n \in D_n \rangle$. Then $\mathcal{P}$ is directionally arc consistent w.r.t. $\prec$ iff $(D_1, \ldots, D_n)$ is a common fixpoint of the functions π_1^+ associated with the binary constraints from $\mathcal{P}_{\prec}$.* □

We now instantiate in an appropriate way the SI algorithm for compound domains with all the π_1 functions associated with the binary constraints from $\mathcal{P}_{\prec}$. In this way we obtain an algorithm that achieves for $\mathcal{P}$ directional arc consistency w.r.t. $\prec$. First, we adjust the definition of semi-commutativity to functions with different schemes. To this end consider a sequence of partial orderings $(D_1, \sqsubseteq_1), \ldots, (D_n, \sqsubseteq_n)$ and their Cartesian product $(D, \sqsubseteq)$. Take two functions, f with scheme s and g with scheme t. We say that f *semi-commutes with* g (*w.r.t.* $\sqsubseteq$) if f^+ semi-commutes with g^+ w.r.t. $\sqsubseteq$, that is if

$$f^+(g^+(Q)) \sqsubseteq g^+(f^+(Q)).$$

for all $Q \in D$.

The following lemma is crucial.

Lemma 8 (Semi-commutativity). *Consider a CSP and two binary constraints of it, C_1 on u, z and C_2 on x, y, where $y \prec z$.*

Then the π_1 function of C_1 semi-commutes with the π_1 function of C_2 w.r.t. the componentwise ordering $\supseteq$. $\square$

Consider now a CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables and the corresponding CSP $\mathcal{P}_\prec$. To be able to apply the above lemma we order the π_1 functions of the binary constraints of $\mathcal{P}_\prec$ in an appropriate way. Namely, given two π_1 functions, f associated with a constraint on u, z and g associated with a constraint on x, y, we put f before g if $y \prec z$.

More precisely, let $x_1, \ldots, x_n$ be the sequence of the variables of $\mathcal{P}_\prec$. So $x_1 \prec x_2 \prec \ldots \prec x_n$. Let for $m \in [1..n]$ the list L_m consist of the π_1 functions of those binary constraints of $\mathcal{P}_\prec$ that are on x_j, x_m for some x_j. We order each list L_m arbitrarily. Consider now the list L resulting from appending $L_n, L_{n-1}, \ldots, L_1$, in that order, so with the elements of L_n in front. Then by virtue of the Semi-commutativity Lemma 8 if the function f precedes the function g in the list L, then f semi-commutes with g w.r.t. the componentwise ordering $\supseteq$.

We instantiate now the refinement of the `SI` algorithm for the compound domains by the above-defined list L and each $\bot_i$ equal to the domain D_i of the variable x_i. We assume that L has k elements. We obtain then the following algorithm.

DIRECTIONAL ARC CONSISTENCY ALGORITHM (`DARC`)

$d := (\bot_1, \ldots, \bot_n)$;
for $i := 1$ **to** k **do**
  $g := \mathbf{head}(L)$; suppose g is with scheme s;
  $L := \mathbf{tail}(L)$;
  $d[s] := g(d[s])$
od

This algorithm enjoys the following properties.

Theorem 5 (`DARC` Algorithm). *Consider a CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables. Let $\mathcal{P}_\prec := \langle \mathcal{C} \,;\, x_1 \in D_1, \ldots, x_n \in D_n \rangle$.*

The `DARC` algorithm always terminates. Let $\mathcal{P}'$ be the CSP determined by $\mathcal{P}_\prec$ and the sequence of the domains $D'_1, \ldots, D'_n$ computed in d. Then

(i) $\mathcal{P}'$ is the $\sqsubseteq_d$-least CSP in $\{\mathcal{P}_1 \mid \mathcal{P}_\prec \sqsubseteq_d \mathcal{P}_1\}$ that is directionally arc consistent w.r.t. $\prec$,
(ii) $\mathcal{P}'$ is equivalent to $\mathcal{P}$. $\square$

Note that in contrast to the `HYPER-ARC` Algorithm Theorem 3 we do not need to assume here that each domain is finite.

Assume now that for each pair of variables x, y of the original CSP $\mathcal{P}$ there exists precisely one constraint on x, y. The same holds then for $\mathcal{P}_\prec$. Suppose that $\mathcal{P}_\prec := \langle \mathcal{C} \,;\, x_1 \in D_1, \ldots, x_n \in D_n \rangle$. Denote the unique constraint of $\mathcal{P}_\prec$ on x_i, x_j by $C_{i,j}$. The above `DARC` algorithm can then be rewritten as the following algorithm known as the `DAC` algorithm of Dechter and Pearl [6]:

for $j := n$ **to** 2 **by** -1 **do**
  for $i := 1$ **to** $j-1$ **do**
    $D_i := \{a \in D_i \mid \exists\, b \in D_j\ (a,b) \in C_{i,j}\}$
  od
od

11 DPC: a Directional Path Consistency Algorithm

In this section we deal with the notion of directional path consistency defined in Dechter and Pearl [6]. As before we first characterize this local consistency notion in terms of fixpoints. To this end, as in the previous section, given a normalized CSP $\mathcal{P}$ we rather consider the equivalent CSP $\mathcal{P}_\prec$. The variables of $\mathcal{P}_\prec$ are ordered according to $\prec$ and on each pair of its variables there exists a unique constraint.

The following is a counterpart of the Directional Arc Consistency Lemma 7.

Lemma 9 (Directional Path Consistency). *Consider a normalized CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables. Let $\mathcal{P}_\prec := \langle C_1, \ldots, C_k\ ;\ \mathcal{DE}\rangle$. Then $\mathcal{P}$ is directionally path consistent w.r.t. $\prec$ iff $(C_1, \ldots, C_k)$ is a common fixpoint of all functions $(f^z_{x,y})^+$ associated with the subsequences x, y, z of the variables of $\mathcal{P}_\prec$.* □

To obtain an algorithm that achieves directional path consistency we now instantiate in an appropriate way the SI algorithm. To this end we need the following lemma.

Lemma 10 (Semi-commutativity). *Consider a normalized CSP and two subsequences of its variables, x_1, y_1, z and x_2, y_2, u. Suppose that $u \prec z$.*

Then the function $f^z_{x_1,y_1}$ semi-commutes with the function $f^u_{x_2,y_2}$ w.r.t. the componentwise ordering $\supseteq$. □

Consider now a normalized CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables and the corresponding CSP $\mathcal{P}_\prec$. To be able to apply the above lemma we order in an appropriate way the $f^t_{r,s}$ functions, where the variables r, s, t are such that $r \prec s \prec t$. Namely, we put $f^z_{x_1,y_1}$ before $f^u_{x_2,y_2}$ if $u \prec z$.

More precisely, let $x_1, \ldots, x_n$ be the sequence of the variables of $\mathcal{P}_\prec$, that is $x_1 \prec x_2 \prec \ldots \prec x_n$. Let for $m \in [1..n]$ the list L_m consist of the functions $f^{x_m}_{x_i,x_j}$ for some x_i and x_j. We order each list L_m arbitrarily and consider the list L resulting from appending $L_n, L_{n-1}, \ldots, L_1$, in that order. Then by virtue of the Semi-commutativity Lemma 9 if the function f precedes the function g in the list L, then f semi-commutes with g w.r.t. the componentwise ordering $\supseteq$.

We instantiate now the refinement of the SI algorithm for the compound domains by the above-defined list L and each $\perp_i$ equal to the constraint C_i. We assume that L has k elements. This yields the DIRECTIONAL PATH CONSISTENCY ALGORITHM (DPATH) that, apart from of the different choice of the constituent partial orderings, is identical to the DIRECTIONAL ARC CONSISTENCY ALGORITHM DARC of the previous section. Consequently, the DPATH algorithm enjoys analogous properties as the DARC algorithm. They are summarized in the following theorem.

Theorem 6 (`DPATH` **Algorithm).** *Consider a CSP $\mathcal{P}$ with a linear ordering $\prec$ on its variables. Let $\mathcal{P}_{\prec} := \langle C_1, \ldots, C_k \; ; \; \mathcal{DE} \rangle$.*

The `DPATH` *algorithm always terminates. Let $\mathcal{P}' := \langle C'_1, \ldots, C'_k \; ; \; \mathcal{DE} \rangle$, where the sequence of the constraints $C'_1, \ldots, C'_k$ is computed in d. Then*

(i) $\mathcal{P}'$ is the $\sqsubseteq_c$-least CSP in $\{\mathcal{P}_1 \mid \mathcal{P}_{\prec} \sqsubseteq_d \mathcal{P}_1\}$ that is directionally path consistent w.r.t. $\prec$,
(ii) $\mathcal{P}'$ is equivalent to $\mathcal{P}$. □

As in the case of the `DARC` Algorithm Theorem 5 we do not need to assume here that each domain is finite.

Assume now that that $x_1, \ldots, x_n$ is the sequence of the variables of $\mathcal{P}_{\prec}$. Denote the unique constraint of $\mathcal{P}_{\prec}$ on x_i, x_j by $C_{i,j}$.

The above `DPATH` algorithm can then be rewritten as the following algorithm known as the `DPC` algorithm of Dechter and Pearl [6]:

for $m := n$ **to** 3 **by** -1 **do**
  for $j := 1$ **to** $m - 1$ **do**
    for $i := 1$ **to** $j - 1$ **do**
      $C_{i,j} := C_{i,m} \cdot C^T_{j,m}$
    od
  od
od

12 Conclusions

In this article we introduced a general framework for constraint propagation. It allowed us to present and explain various constraint propagation algorithms in a uniform way. Using such a single framework we can easier verify, compare, modify, parallelize or combine these algorithms. The last point has already been made to large extent in Benhamou [2]. Additionally, we clarified the role played by the notions of commutativity and semi-commutativity.

The line of research presented here could be extended in a number of ways. First, it would be interesting to find examples of existing constraint propagation algorithms that could be improved by using the notions of commutativity and semi-commutativity.

Second, as already stated in Apt [1], it would be useful to explain in a similar way other constraint propagation algorithms such as the `AC-4` algorithm of Mohr and Henderson [11], the `PC-4` algorithm of Han and Lee [8], or the `GAC-4` algorithm of Mohr and Masini [12]. The complication is that these algorithms operate on some extension of the original CSP.

Finally, it would be useful to apply the approach of this paper to derive constraint propagation algorithms for the semiring-based constraint satisfaction framework of Bistarelli, Montanari and Rossi [4] that provides a unified model for several classes of “nonstandard” constraints satisfaction problems.

Acknowledgements

Victor Dalmau and Rosella Gennari pointed out to us that Assumptions **A** and **B** on page 4 are not sufficient to establish Theorem 1. The added now Assumption **C** was suggested to us by Rosella Gennari.

References

1. K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2):179–210, 1999. Available via `http://xxx.lanl.gov/archive/cs/`.
2. F. Benhamou. Heterogeneous constraint solving. In M. Hanus and M. Rodriguez-Artalejo, editors, *Proceeding of the Fifth International Conference on Algebraic and Logic Programming (ALP 96)*, Lecture Notes in Computer Science 1139, pages 62–76, Berlin, 1996. Springer-Verlag.
3. F. Benhamou and W. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
4. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
5. R. Dechter. Bucket elimination: A unifying framework for structure-driven inference. *Artificial Intelligence*, 1999. To appear.
6. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, January 1988.
7. R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 20 February 1997.
8. C. Han and C. Lee. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, 36:125–130, 1988.
9. A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
10. K. Marriott and P. Stuckey. *Programming with Constraints*. The MIT Press, Cambridge, Massachusetts, 1998.
11. R. Mohr and T.C. Henderson. Arc-consistency and path-consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
12. R. Mohr and G. Masini. Good old discrete relaxation. In Y. Kodratoff, editor, *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)*, pages 651–656. Pitman Publishers, 1988.
13. E. Monfroy and J.-H. Réty. Chaotic iteration for distributed constraint propagation. In J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. Lamont, editors, *Proceedings of The 1999 ACM Symposium on Applied Computing, SAC'99*, pages 19–24, San Antonio, Texas, USA, March 1999. ACM Press.
14. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):95–132, 1974. Also Technical Report, Carnegie Mellon University, 1971.
15. V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages (POPL'91)*, pages 333–352, 1991.
16. V. Telerman and D. Ushakov. Data types in subdefinite models. In J. A. Campbell J. Calmet and J. Pfalzgraf, editors, *Artificial Intelligence and Symbolic Mathematical Computations*, Lecture Notes in Computer Science 1138, pages 305–319, Berlin, 1996. Springer-Verlag.
17. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.