The Role of Commutativity in Constraint Propagation Algorithms

KRZYSZTOF R. APT
CWI and University of Amsterdam

Constraint propagation algorithms form an important part of most of the constraint programming systems. We provide here a simple, yet very general framework that allows us to explain several constraint propagation algorithms in a systematic way. In this framework we proceed 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 to obtain specific constraint propagation algorithms. In particular, using the notions commutativity and semi-commutativity, we show that the `AC-3`, `PC-2`, `DAC`, and `DPC` algorithms for achieving (directional) arc consistency and (directional) path consistency are instances of a single generic algorithm. The work reported here extends and simplifies that of Apt [1999a].

Categories and Subject Descriptors: D.3.3 [**Language Constructs and Features**]: Constraints; I.1.2 [**Algorithms**]: Analysis of Algorithms; I.2.2 [**Automatic Programming**]: Program Synthesis

General Terms: Algorithms, Languages, Verification

Additional Key Words and Phrases: Constraint propagation, generic algorithms, commutativity

1. INTRODUCTION

1.1 Motivation

A constraint satisfaction problem, in short CSP, is a finite collection of relations (constraints), each on some variables. A solution to a CSP is an assignment of values to all variables that satisfies all constraints. Constraint programming in a nutshell consists of generating and solving CSP's means of general or domain-specific methods.

This approach to programming became very popular in the eighties and led to a creation of several new programming languages and systems. Some of the more known examples include a constraint logic programming system ECL^{i}PS^e (see Aggoun et al. [1995]), a multiparadigm programming language Oz (see, e.g., Smolka [1995]), and the ILOG Solver that is the core C++ library of the ILOG Optimization Suite (see ILOG [1998]).

One of the most important general purpose techniques developed in this area is constraint propagation that aims at reducing the search space of the considered

This is a full, revised and corrected version of our article Apt [1999b].
Author's address: CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2001 ACM $5.00

CSP while maintaining equivalence. It is a very widely used concept. For instance on Google, `http://www.google.com/` on March 21st, 2001, the query "constraint propagation" yielded 6,840 hits. For comparison, the query "NP completeness" yielded 14,400 hits. In addition, in the literature several other names have been used for the constraint propagation algorithms: consistency, local consistency, consistency enforcing, Waltz, filtering, or narrowing algorithms.

The constraint propagation 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 20 few years many useful notions of local consistency were identified, and for each of them one or more constraint propagation algorithms were proposed.

Many of these algorithms were built into the existing constraint programming systems, including the above three ones. These algorithms can be triggered either automatically, e.g., each time a new constraint is generated (added to the "constraint store"), or by means of specific instructions available to the user.

In Apt [1999a] we introduced a simple framework that allowed 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. We stated there 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 [1996] and Telerman and Ushakov [1996].

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 et al. [1991] and Benhamou and Older [1997]). Functions that satisfy these properties are called closures (see Gierz et al. [1980]). Here we show that also the notions of commutativity and so-called semi-commutativity are important.

As in Apt [1999a], to explain the constraint propagation algorithms, we proceed here in two steps. First, we introduce a generic iteration algorithm that aims to compute the least common fixpoint of a set of functions on a partial ordering 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 [1999a] because we concentrate here on constraint propagation algorithms that always terminate. This allows us to dispense with the notion of fairness. Moreover, we prove here stronger results by taking into account the commutativity and semi-commutativity information.

1.2 Example

To illustrate the problems here studied consider the following puzzle from Mackworth [1992]. Take the crossword grid of Figure 1 and suppose that we are to fill it

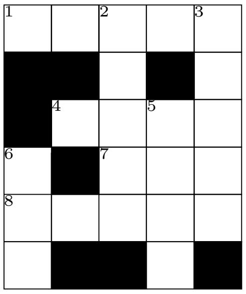

Fig. 1. A crossword grid.

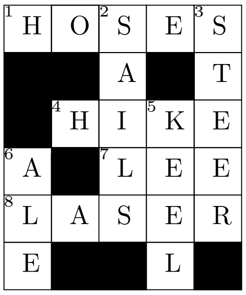

Fig. 2. A solution to the crossword puzzle.

with the words from the following list:

—HOSES, LASER, SAILS, SHEET, STEER,

—HEEL, HIKE, KEEL, KNOT, LINE,

—AFT, ALE, EEL, LEE, TIE.

This problem has a unique solution depicted in Figure 2.

This puzzle can be solved by systematically considering each crossing and eliminating the words that cannot be used. Consider for example the crossing of the positions 2 and 4, in short (2,4). Neither word HOSES nor LASER can be used in position 2 because no four-letter word (for position 4) exists with S as the second letter. Similarly, by considering the crossing (2,8) we deduce that none of the words LASER, SHEET, and STEER can be used in position 2.

The question now is what "systematically" means. For example, after considering the crossings (2,4) and (2,8) should we reconsider the crossing (2,4)? Our approach clarifies that the answer is "No" because the corresponding functions $f_{2,4}$ and $f_{2,8}$ that remove impossible words, here for position 2 on account of the crossings (2,4) and (2,8), commute. In contrast, the functions $f_{2,4}$ and $f_{4,5}$ do not commute, so after considering the crossing (4,5) the crossing (2,4) needs to be reconsidered.

In Section 6 we formulate this puzzle as a CSP and discuss more precisely the problem of scheduling of the involved functions and the role commutativity plays here.

1.3 Plan of the Article

This article is organized as follows. First, in Section 2, drawing on the approach of Monfroy and Réty [1999], we introduce a generic iteration algorithm, with the difference that 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. These last two sections essentially follow Apt [1999a], but because we started here with the generic iteration algorithms on arbitrary partial orderings we built now a framework in which we can also discuss the role of commutativity.

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 draw conclusions and discuss recent and possible future work.

We deal here only with the classic algorithms that establish (directional) arc consistency and (directional) path consistency and that are more than 20, respectively 10, years old. However, several more "modern" constraint propagation algorithms can also be explained in this framework. In particular, in Apt [1999a, page 203] we derived from a generic algorithm a simple algorithm that achieves the notion of relational consistency of Dechter and van Beek [1997]. In turn, by mimicking the development of Sections 10 and 11, we can use the framework of Section 9 to derive the adaptive consistency algorithm of Dechter and Pearl [1988]. Now, Dechter [1999] showed that the latter 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.

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 2.1. 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 2.2. 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 2.3 (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 finite set of functions F on D. We are interested in computing the least common fixpoint of the functions from F. To this end we study the following algorithm inspired by a similar, though more complex, algorithm of Monfroy and Réty [1999] defined on a Cartesian product of component partial orderings.

Generic Iteration Algorithm (`GI`)

$d := \perp;$
$G := F;$
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 $g \in update(G, g, d)$.

The above conditions on $update(G, g, d)$ look somewhat artificial and unnecessarily complex. In fact, an obviously simpler alternative exists according to which we just postulate that $\{f \in F - G \mid f(g(d)) \neq g(d)\} \subseteq update(G, g, d)$, i.e., that we add to G at least all functions from $F - G$ for which the "new value," $g(d)$, is not a fixpoint.

The problem is that for each choice of the *update* function we wish to avoid the computationally expensive task of computing the values of $f(d)$ and $f(g(d))$ for the functions f in $F-G$. Now, when we specialize the above algorithm to the case of a Cartesian product of the partial orderings we shall be able to avoid this computation of the values of $f(d)$ and $f(g(d))$ by just analyzing for which components d and $g(d)$ differ. This specialization cannot be derived by adopting the above simpler choice of the *update* function.

Intuitively, assumption **A** states that $update(G, g, d)$ at least contains all the functions from $F - G$ for which the "old value", d, is a fixpoint but the "new value," $g(d)$, 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 $g \in \{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\}$ does not hold, since we cannot have both $g(d) = d$ and $g(g(d)) \neq g(d)$. 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**, **B**, and **C** 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, again, 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 $f(d)$.

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

THEOREM 2.4 (`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, i.e.,

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

(ii) This is a direct consequence of (i) and the Stabilization Lemma 2.3.

(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. $\Box$

In particular, we obtain the following conclusion.

Corollary 2.5 (`GI`). *Suppose $(D, \sqsubseteq)$ is a finite partial ordering with the least element $\bot$. 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 2.6. 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.
—We call a function f on a partial ordering $(D, \sqsubseteq)$ a *closure* if f is inflationary, monotonic, and itempotent.

Closures were studied in Gierz et al. [1980]. They play an important role in mathematical logic and lattice theory. We shall return to them in Section 4.

The following result holds.

Theorem 2.7 (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 $\perp_i$. 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 3.1. 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 jth 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 3.2 (UPDATE). *Suppose that each function in F is of the form f^+. Then the following function update satisfies assumptions **A**, **B**, and **C**:*

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

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 $f^+ \in \mathit{update}(G, g^+, d)$.

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 $g^+ \in \mathit{update}(G, g^+, d)$. □

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 2.4 and the Update Note 3.2 summarizes the correctness of this algorithm. It corresponds to Theorem 11 of Apt [1999a] where the iteration algorithms were introduced immediately on compound domains.

COROLLARY 3.3 (CD). *Suppose that $(D, \sqsubseteq)$ is a finite partial ordering that is a Cartesian product of n partial orderings, each with the least element $\perp_i$ 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 in which the function *update* defined in the Update Theorem 2.7(i) is used.
—CDC *algorithm*. This is the version of the CD algorithm in which all the functions are idempotent and in which the combined effect of the functions *update* defined in the Update Theorem 2.7 is used for some function *Comm*.

For both algorithms the counterparts of the CD Corollary 3.3 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 CSPs $\mathcal{P}_1$ 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 CSPs 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 CSPs. 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 CSPs that satisfy the considered notion of local consistency.

Let us be now more specific about items (i) and (ii).

Re: (i)

To deal with the local consistency notions considered in this paper we shall introduce two specific partial orderings on the CSPs. In each of them the considered CSPs 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 notion of 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 [1999a, pages 193-194]).

Next, given two CSPs, $\phi := \langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle$ 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 CSPs, $\phi := \langle C_1, \ldots, C_k \ ; \ \mathcal{DE} \rangle$ 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

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

Re: (ii)

The domain reduction ordering and the constraint reduction ordering are not directly amenable to the analysis given in Section 3. Therefore, we shall rather use equivalent partial orderings defined on compound domains. To this end 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 equivalence 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 $\mathcal{P}$. Indeed, by the definition of this ordering the constraints of such a CSP are restrictions of the constraints of $\mathcal{P}$ to the domains of this CSP.

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.

The reason we use these compound orderings is that we can now employ functions with schemes, as used in Section 3. Each such function f is defined on a sub-Cartesian product of the constituent partial orderings. Its canonic extension f^+, introduced in Section 3, is then defined on the "whole" Cartesian product.

Suppose now that we are dealing with the domain reduction ordering with the least (initial) CSP $\mathcal{P}$ and that

$$f^+(D_1 \times \cdots \times D_n) = D'_1 \times \cdots \times D'_n.$$

Then the sequence of the domains $(D_1, \ldots, D_n)$ and $\mathcal{P}$ uniquely determine a CSP in this ordering and the same for $(D'_1, \ldots, D'_n)$ and $\mathcal{P}$. Hence f^+, and a fortiori f, can be viewed as a function on the CSPs that are elements of this domain reduction ordering. In other words, f can be viewed as a function on CSPs.

The same considerations apply to the constraint reduction ordering. We shall use these observations when arguing about the equivalence between the original and the final CSPs for various constraint propagation algorithms.

The considered functions with schemes will be now used in presence of the componentwise ordering $\supseteq$. The following observation will be useful.

Consider 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.

Let us discuss now briefly the functions used in our considerations. In the preceding sections we clarified which of their properties account for specific properties of the studied algorithms. It is tempting then to confine one's attention to closures, i.e., functions that are inflationary, monotonic, and itempotent. The importance of closures for concurrent constraint programming was recognized by Saraswat et al. [1991] and for the study of constraint propagation by Benhamou and Older [1997].

However, as shown in Apt [1999a], some known local consistency notions are characterized as common fixpoints of functions that in general are not itempotent. Therefore when studying constraint propagation in full generality it is preferrable not to limit one's attention to closures. On the other hand, in the sections that follow we only study notions of local consistency that are characterized by means of closures. Therefore, from now on the closures will be prominently present in our exposition.

5. A HYPER-ARC CONSISTENCY ALGORITHM

We begin by considering the notion of hyper-arc consistency of Mohr and Masini [1988] (we use here the terminology of Marriott and Stuckey [1998]). The more known notion of arc consistency of Mackworth [1977] is obtained by restricting one's attention to binary constraints. Let us recall the definition.

Definition 5.1.

—Consider a constraint C on the variables $x_1, \ldots, x_n$ with the respective domains $D_1, \ldots, D_n$, i.e., $C \subseteq D_1 \times \cdots \times D_n$. We call C *hyper-arc consistent* if for every $i \in [1..n]$ and $a \in D_i$ there exists $d \in C$ such that $a = d[i]$.

—We call a CSP *hyper-arc consistent* if all its constraints are hyper-arc consistent.

Intuitively, a constraint C is hyper-arc consistent if for every involved domain each element of it participates in a solution to C.

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 jth 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 ith component.

Re: (iii) Common fixpoints.

Their use is clarified by the following lemma that also lists the relevant properties of the functions π_i (see Apt [1999a, pages 197 and 202]).

LEMMA 5.2 (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 a closure w.r.t. the componentwise ordering $\supseteq$.

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 the following properties.

THEOREM 5.3 (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 $\mathcal{P}$.

Due to the definition of the $\sqsubseteq_d$ ordering the item (i) can be rephrased as follows. Consider all hyper-arc consistent CSPs 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 CSPs $\mathcal{P}'$ has the largest domains.

PROOF. The termination and (i) are immediate consequences of the counterpart of the CD Corollary 3.3 for the CDI algorithm and of the Hyper-arc Consistency Lemma 5.2.

To prove (ii) note that the final CSP $\mathcal{P}'$ can be obtained by means of repeated applications of the projection functions π_i starting with the initial CSP $\mathcal{P}$. (Conforming to the discussion at the end of Section 4 we view here each such function as a function on CSPs). As noted in Apt [1999a, pages 197 and 201] each of these functions transforms a CSP into an equivalent one. $\square$

6. AN IMPROVEMENT: THE AC-3 ALGORITHM

In the HYPER-ARC algorithm each time a π_i function associated with a constraint C on the variables $y_1, \ldots, y_k$ is applied and modifies its arguments, all projection functions associated with a constraint that involves the variable y_i are added to the set G. In this section we show how we can exploit information about the commutativity to add less projection functions to the set G. Recall that, in Section 3, we modified the notion of commutativity for the case of functions with schemes.

First, it is worthwhile to note that not all pairs of the π_i and π_j functions commute.

Example 6.1. (i) We consider the case of two binary constraints on the same variables. Consider two variables, x and y with the corresponding domains $D_x := \{a, b\}$, $D_y := \{c, d\}$ and two constraints on x, y: $C_1 := \{(a, c), (b, d)\}$ and $C_2 := \{(a, d)\}$.

Next, consider the π_1 function of C_1 and the π_2 function of C_2. Then applying these functions in one order, namely $\pi_2\pi_1$, to (D_x, D_y) yields D_x unchanged, whereas applying them in the other order, $\pi_1\pi_2$, yields D_x equal to $\{b\}$.

(ii) Next, we show that the commutativity can also be violated due to sharing of a single variable. As an example take the variables x, y, z with the corresponding domains $D_x := \{a, b\}$, $D_y := \{b\}$, $D_z := \{c, d\}$, and the constraint $C_1 := \{(a, b)\}$ on x, y and $C_2 := \{(a, c), (b, d)\}$ on x, z.

Consider now the π_1^+ function of C_1 and the π_2^+ function of C_2. Then applying these functions in one order, namely $\pi_2^+\pi_1^+$, to (D_x, D_y, D_z) yields D_z equal to $\{c\}$, whereas applying them in the other order, $\pi_1^+\pi_2^+$, yields D_z unchanged.

The following lemma clarifies which projection functions do commute.

LEMMA 6.2 (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.

PROOF. See the Appendix. □

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 $\bot_i$ equal to D_i. Additionally we employ the following function *Comm*, where π_i is associated with a constraint C and where E differs from C:

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

By virtue of the Commutativity Lemma 6.2 each set $Comm(g)$ satisfies the assumptions of the Update Theorem 2.7(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 3.3 for the `CDC` algorithm we conclude that the above algorithm enjoys the same properties as the `HYPER-ARC` algorithm, i.e., the counterpart of the `HYPER-ARC` Algorithm Theorem 5.3 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

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

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 [1977].

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

AC-3 Algorithm

$S_0 := \{C \mid C$ is a binary constraint from $\mathcal{C}\}$
$\quad \cup\ \{C^T \mid C$ is a binary constraint from $\mathcal{C}\}$;
$S := S_0$;
while $S \neq \emptyset$ **do**
  choose $C \in S$; suppose C is on x_i, x_j;
  $D_i := \{a \in D_i \mid \exists\, b \in D_j\ (a, b) \in C\}$;
  if D_i changed **then**
    $S := S \cup \{C' \in S_0 \mid C'$ is on the variables y, x_i where $y \not\equiv x_j\}$
  fi;
  $S := S - \{C\}$
od

It is useful to mention that the corresponding reformulation of the HYPER-ARC algorithm for binary constraints 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 6.2.

Second, no constraint or relation on x_j, x_i is added to S. Here we exploited part (i) of the Commutativity Lemma 6.2, 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 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.

To illustrate the considerations of this section let us return now to the crossword puzzle introduced in Section 1.2.

As pointed out by Mackworth [1992] this problem can be easily formulated as a CSP as follows. First, associate with each position $i \in [1..8]$ in the grid of Figure 1 a variable. Then associate with each variable the domain that consists of the set of words that can be used to fill this position. For example, position 6 needs to be filled with a three-letter word, so the domain of the variable associated with position 6 consists of the above set of five three-letter words.

Finally, we define constraints. They deal with the restrictions arising from the fact that the words that cross share a letter. For example, the crossing of the positions 1 and 2 contributes the following constraint:

$$C_{1,2} := \{(\text{HOSES, SAILS}), (\text{HOSES, SHEET}), (\text{HOSES, STEER}), (\text{LASER, SAILS}), (\text{LASER, SHEET}), (\text{LASER, STEER})\}.$$

This constraint formalizes the fact that the third letter of position 1 needs to be the same as the first letter of position 2. In total there are 12 constraints.

Each projection function π_1 associated with a constraint C or its transpose C^T corresponds to a crossing, for example (8,2). It removes impossible values from the current domain of the variable associated with the first position, here 8.

The above Commutativity Lemma 6.2 allows us to conclude, that for any pairwise different $a, b, c \in [1..8]$, the projection functions π_1 associated with the crossings (a, b) and (b, a) commute and also the projection functions π_1 associated with the crossings (a, b) and (a, c) commute. This explains why in the `AC-3` algorithm applied to this CSP after considering a crossing (a, b), for example (2,4), neither the crossing (4,2) nor the crossings (2,7) and (2,8) are added to the set of examined crossings.

To see that the `AC-3` algorithm applied to this CSP yields the unique solution depicted in Figure 2 it is sufficient to observe that this solution viewed as a CSP is arc consistent and that it is obtained by a specific execution of the `AC-3` algorithm, in which the crossings are considered in the following order:

(1,2), (2,1), (1,3), (3,1), (4,2), (2,4), (4,5), (5,4), (4,2), (2,4),
(7,2), (2,7), (7,5), (5,7), (8,2), (2,8), (8,6), (6,8), (8,2), (2,8).

The desired conclusion now follows by the counterpart of the `CD` Corollary 3.3 according to which every execution of the `AC-3` algorithm yields the same outcome.

7. A PATH CONSISTENCY ALGORITHM

The notion of path consistency was introduced in Montanari [1974]. It is defined for a special type of CSPs. For simplicity we ignore here unary constraints that are usually present when studying path consistency.

Definition 7.1. We call a CSP $\mathcal{P}$ *standardized* if for each pair x, y of its variables there exists exactly one constraint on x, y in $\mathcal{P}$. We denote this constraint by $C_{x,y}$.

Every CSP is trivially equivalent to a standardized CSP. Indeed, it suffices for each pair x, y of the variables of $\mathcal{P}$ first to add the "universal" constraint on x, y that consists of the Cartesian product of the domains of the variables x and y and then to replace the set of all constraints on x, y by their intersection.

At the cost of some notational overhead our considerations about path consistency can be generalized in a straightforward way to the case of CSPs such that for each pair of variables x, y at most one constraint exists on x, y, i.e., to the CSPs that satisfy assumption (1).

To simplify the notation 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 of two variables of a standardized CSP we introduce a "supplementary" relation $C_{y,x}$ defined by

$$C_{y,x} := C^T_{x,y}.$$

Recall that the relation C^T was introduced in the previous section. The supplementary relations are not parts of the considered CSP, as none of them is defined on a subsequence of its variables, but they allow us a more compact presentation. We now introduce the following notion.

Definition 7.2. We call a standardized CSP *path consistent* if for each subset $\{x, y, z\}$ of its variables we have

$$C_{x,z} \subseteq C_{x,y} \cdot C_{y,z}.$$

In other words, a standardized CSP is path consistent if for each subset $\{x, y, z\}$ of its variables the following holds:

if $(a, c) \in C_{x,z}$, then there exists b such that $(a, b) \in C_{x,y}$ and $(b, c) \in C_{y,z}$.

To employ the `CDI` algorithm of Section 3 we again make specific choices involving the items (i), (ii), and (iii) of Section 4. First, we provide an alternative characterization of path consistency.

Note that in the above definition we used the relations of the form $C_{u,v}$ for any *subset* $\{u, v\}$ of the considered sequence of variables. If u, v is not a *subsequence* of the original sequence of variables, then $C_{u,v}$ is a supplementary relation that is not a constraint of the original CSP. At the expense of some redundancy we can rewrite the above definition so that only the constraint of the considered CSP are involved. This is the contents of the following simple observation that will be useful in a moment.

Note 7.3 (Alternative Path Consistency). *A standardized CSP is path consistent iff for each subsequence x, y, z of its variables we have*

$$C_{x,y} \subseteq C_{x,z} \cdot C_{y,z}^T,$$

$$C_{x,z} \subseteq C_{x,y} \cdot C_{y,z},$$

$$C_{y,z} \subseteq C_{x,y}^T \cdot C_{x,z}.$$

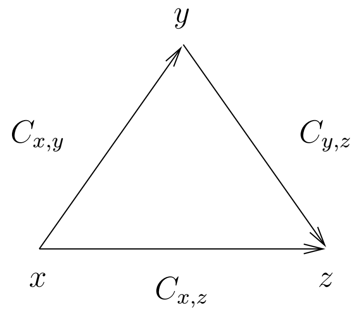

Fig. 3. Three relations on three variables.

Figure 3 clarifies this observation. For instance, an indirect path from x to y via z requires the reversal of the arc (y, z). This translates to the first formula.

Now, to study path consistency, given a standardized CSP $\mathcal{P} := \langle C_1, \ldots, C_k \ ; \ \mathcal{DE}\rangle$ we take the constraint reduction ordering of Section 4 with $\mathcal{P}$ as the least element and with the powerset function as the function $\mathcal{F}$. So, as already noted in Section 4 we can identify this ordering with the Cartesian product of the partial orderings $(\mathcal{P}(C_i), \supseteq)$, where $i \in [1..k]$. The elements of this compound ordering are thus sequences $(X_1, \ldots, X_k)$ of respective subsets of the constraints $C_1, \ldots, C_k$ ordered componentwise by the reversed subset ordering $\supseteq$.

Next, given a subsequence x, y, z of the variables of $\mathcal{P}$ we 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 $R' := R \cap P^T \cdot Q$.

In what follows, when using a function $f^z_{x,y}$ we implicitly assume that the variables x, y, z are pairwise different and that x, y is a subsequence of the variable of the considered CSP.

Finally, we relate the notion of path consistency to the common fixpoints of the above defined functions. This leads us to the following counterpart of the Hyper-arc Consistency Lemma 5.2.

Lemma 7.4 (Path Consistency).

(i) A standardized 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 closures w.r.t. the componentwise ordering $\supseteq$.

Proof. (i) is a direct consequence of the Alternative Path Consistency Note 7.3. The proof of (ii) is straightforward. These properties of the functions $f^z_{x,y}$, $f^y_{x,z}$, and $f^x_{y,z}$ were already mentioned in Apt [1999a, page 193]. $\Box$

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 $\bot_i$ equal to C_i.

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

Theorem 7.5 (`PATH` Algorithm). *Consider a standardized 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 5.3 the item (i) can be rephrased as follows. Consider all path consistent CSPs 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.

Proof. The proof is analogous to that of the `HYPER-ARC` Algorithm Theorem 5.3. The termination and (i) are immediate consequences of the counterpart of the `CD` Corollary 3.3 for the `CDI` algorithm and of the Path Consistency Lemma 7.4.

To prove (ii) we now note that the final CSP $\mathcal{P}'$ can be obtained by means of repeated applications of the functions $f^z_{x,y}$, $f^y_{x,z}$, and $f^x_{y,z}$ starting with the initial CSP $\mathcal{P}$. (Conforming to the discussion at the end of Section 4 we view here each such function as a function on CSPs). As noted in Apt [1999a, pages 193 and 195]) each of these functions transforms a CSP into an equivalent one. □

8. AN IMPROVEMENT: THE PC-2 ALGORITHM

In the `PATH` algorithm each time a $f^z_{x,y}$ function is applied and modifies its arguments, all functions associated with a triplet of variables including x and y are added to the set G. We now show how we can add fewer functions by taking into account the commutativity information. To this end we establish the following lemma.

Lemma 8.1 (Commutativity). *Consider a standardized CSP involving among others the variables* x, y, z, u. *Then the functions* $f^z_{x,y}$ *and* $f^u_{x,y}$ *commute.*

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

Proof (Sketch). The following intuitive argument may help to understand the more formal justification given in the Appendix. First, both considered functions have three arguments but share precisely one argument, the one from $\mathcal{P}(C_{x,y})$, and modify only this shared argument. Second, both functions are defined in terms of the set-theoretic intersection operation "$\cap$" applied to two, unchanged arguments. This yields commutativity since "$\cap$" is commutative. □

Fix now a standardized CSP $\mathcal{P}$. We instantiate the `CDC` algorithm with the same set of functions F_0 as in Section 7. Additionally, we use the following function $Comm$:

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

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 $m-3$ fewer elements than in the case of the corresponding instance of the `CDI` algorithm considered in the previous section.

By virtue of the Commutativity Lemma 8.1 each set $Comm(g)$ satisfies the assumptions of the Update Theorem 2.7(ii). We conclude that the above instance of the `CDC` algorithm enjoys the same properties as the original `PATH` algorithm, i.e., the counterpart of the `PATH` Algorithm Theorem 7.5 holds. To make this modification of the `PATH` algorithm easier to understand we proceed as follows.

Below we write $x \prec y$ to indicate that x, y is a subsequence of the variables of the CSP $\mathcal{P}$. 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 [1977]. 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 9.1. 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 9.2 (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 closures on* $(D, \sqsubseteq)$. *Suppose that* f_i *semi-commutes with* f_j *for* $i > j$, *i.e.,*

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

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

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

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:

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

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

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

So $f_1 f_2 \ldots f_k(\perp)$ is a common fixpoint of the functions from F. This means that any iteration of F that starts with $\perp$, $f_k(\perp)$, $f_{k-1} f_k(\perp), \ldots, f_1 f_2 \ldots f_k(\perp)$ eventually stabilizes at $f_1 f_2 \ldots f_k(\perp)$. By the Stabilization Lemma 2.3 we get the desired conclusion. □

The above lemma provides us with a simple way of computing the least common fixpoint of a finite set of 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 $\perp$.

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

SIMPLE ITERATION ALGORITHM (SI)

$d := \perp$;
for $i := k$ **to** 1 **by** -1 **do**

$d := f_i(d)$
od

The following immediate consequence of the Simple Iteration Lemma 9.2 is a counterpart of the `GI` Corollary 2.5.

COROLLARY 9.3 (`SI`). *Suppose that $(D, \sqsubseteq)$ is a partial ordering with the least element $\perp$. Let $F := f_1, \ldots, f_k$ be a finite sequence of closures on $(D, \sqsubseteq)$ such that (2) holds. Then the* `SI` *algorithm terminates and computes in d the least common fixpoint of the functions from F.*

Note that in contrast to the `GI` Corollary 2.5 we do not require here that the partial ordering is finite. 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 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 [1988]. Let us recall the definition.

Definition 10.1. Assume a linear ordering $\prec$ on the considered variables.

—Consider a binary constraint C on the variables x, y with the domains D_x and D_y. We call C *directionally arc consistent w.r.t.* $\prec$ if
  —$\forall a \in D_x \exists b \in D_y \ (a, b) \in C$ provided $x \prec y$,
  —$\forall b \in D_y \exists a \in D_x \ (a, b) \in C$ provided $y \prec x$.
  So out of these two conditions on C exactly one needs to be checked.

—We call a CSP *directionally arc consistent w.r.t.* $\prec$ if all its binary constraints are directionally arc consistent w.r.t. $\prec$.

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 $x_1, \ldots, x_n$ such that $x_1 \prec x_2 \prec \ldots \prec x_n$.

The following simple characterization holds.

LEMMA 10.2 (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. $\sqsubseteq$, i.e., if

$$f^+(g^+(d)) \sqsubseteq g^+(f^+(d))$$

for all $d \in D$.

The following lemma is crucial. To enhance the readability, we replace here the irrelevant variables by $\_$.

LEMMA 10.3 (SEMI-COMMUTATIVITY). *Consider a CSP and two binary constraints of it, C_1 on $\_, z$ and C_2 on $\_, y$, where $y \preceq z$.*

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

PROOF. See the Appendix. □

To be able to apply this lemma we order appropriately the π_1 functions of the binary constraints of $\mathcal{P}_\prec$. Namely, given two π_1 functions, f associated with a constraint on $\_, z$ and g associated with a constraint on $\_, y$, we put f before g if $y \prec z$. Then by virtue of this lemma and the Commutativity Lemma 6.2(ii) if the function f precedes the function g, then f semi-commutes with g w.r.t. the componentwise ordering $\supseteq$.

Observe that we leave here unspecified the order between two π_1 functions, one associated with a constraint on x, z and another with a constraint on y, z, for some variables x, y, z. Note that if x and y coincide then the semi-commutativity is indeed a consequence of the Commutativity Lemma 6.2(ii).

We instantiate now the refinement of the `SI` algorithm for the compound domains by the above-defined sequence of the π_1 functions and each $\perp_i$ equal to the domain D_i of the variable x_i. In this way we obtain the following algorithm, where the sequence of functions is $f_1, \ldots, f_k$.

DIRECTIONAL ARC CONSISTENCY ALGORITHM (`DARC`)

$d := (D_1, \ldots, D_n)$;
for $j := k$ **to** 1 **by** -1 **do**
    suppose f_j is with scheme s;
    $d[s] := f_j(d[s])$
od

This algorithm enjoys the following properties.

THEOREM 10.4 (`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 $\mathcal{P}$.

PROOF. The termination is obvious. (i) is an immediate consequence of the counterpart of the SI Corollary 9.3 for the SI algorithm refined for the compound domains and of the Directional Arc Consistency Lemma 10.2.

The proof of (ii) is analogous to that of the HYPER-ARC Algorithm Theorem 5.3(ii). □

Note that in contrast to the HYPER-ARC Algorithm Theorem 5.3 we do not need to assume here that each domain is finite.

Assume now that the original CSP $\mathcal{P}$ is standardized, i.e., for each pair of its variables x, y precisely one constraint on x, y exists. The same holds then for $\mathcal{P}_{\prec}$. We now specialize the DARC algorithm by ordering the π_1 functions in a deterministic way. 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 $C_{i,j}$.

Order now these constraints as follows:

$$C_{1,n}, C_{2,n}, \ldots, C_{n-1,n}, C_{2,n-1}, \ldots, C_{n-2,n-1}, \ldots, C_{1,2}.$$

That is, the constraint $C_{i',j'}$ precedes the constraint $C_{i'',j''}$ if the pair (j'', i') lexicographically precedes the pair (j', i''). Take now the π_1 functions of these constraints ordered in the same way as their constraints.

The above DARC algorithm can then be rewritten as the following double **for** loop. The resulting algorithm is known as the DAC algorithm of Dechter and Pearl [1988].

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 [1988]. Let us recall the definition.

Definition 11.1. Assume a linear ordering $\prec$ on the considered variables. We call a standardized CSP *directionally path consistent w.r.t.* $\prec$ if for each subset $\{x, y, z\}$ of its variables we have

$$C_{x,z} \subseteq C_{x,y} \cdot C_{y,z} \text{ provided } x, z \prec y.$$

This definition relies on the supplementary relations because the ordering $\prec$ may differ from the original ordering of the variables. For example, in the original ordering z can precede x. In this case $C_{z,x}$ and not $C_{x,z}$ is a constraint of the CSP under consideration.

But just as in the case of path consistency we can rewrite this definition using the original constraints only. In fact, we have the following analogue of the Alternative Path Consistency Note 7.3.

NOTE 11.2 (ALTERNATIVE DIRECTIONAL PATH CONSISTENCY). *A standardized CSP is directionally path consistent w.r.t.* $\prec$ *iff for each subsequence* x, y, z *of its variables we have*

$$C_{x,y} \subseteq C_{x,z} \cdot C^T_{y,z} \text{ provided } x, y \prec z,$$

$$C_{x,z} \subseteq C_{x,y} \cdot C_{y,z} \text{ provided } x, z \prec y,$$

$$C_{y,z} \subseteq C^T_{x,y} \cdot C_{x,z} \text{ provided } y, z \prec x.$$

Thus out of the above three inclusions precisely one needs to be checked.

As before we now characterize this local consistency notion in terms of fixpoints. To this end, as in the previous section, given a standardized CSP $\mathcal{P}$ we rather consider the equivalent CSP $\mathcal{P}_{\prec}$. The variables of $\mathcal{P}_{\prec}$ are ordered according to $\prec$, and $\mathcal{P}_{\prec}$ is standardized, as well.

The following counterpart of the Directional Arc Consistency Lemma 10.2 is a direct consequence of the Alternative Directional Path Consistency Note 11.2. We use here the functions $f^z_{x,y}$ defined in Section 7.

Lemma 11.3 (Directional Path Consistency). *Consider a standardized 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 $(f^z_{x,y})^+$, where $x \prec y \prec z$.*

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 11.4 (Semi-commutativity). *Consider a standardized CSP with a linear ordering $\prec$ on its variables. Suppose that $x_1 \prec y_1 \prec z$, $x_2 \prec y_2 \prec u$, and $u \preceq z$. 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 $\supseteq$.*

Proof. See the Appendix. □

Consider now a standardized 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 $f^z_{x,y}$ functions, where $x \prec y \prec z$, as follows.

Assume that $x_1, \ldots, x_n$ is the sequence of the variables of $\mathcal{P}_{\prec}$, i.e., $x_1 \prec x_2 \prec \ldots \prec x_n$. Let for $m \in [3..n]$ the sequence L_m consist of the functions $f^{x_m}_{x_i,x_j}$, where $i < j < m$, ordered in an arbitrary way. Consider the sequence resulting from appending the sequences $L_n, L_{n-1}, \ldots, L_3$, in that order. Then by virtue of the Semi-commutativity Lemma 11.4 if the function f precedes the function g, 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 sequence of functions $f^z_{x,y}$ and each $\perp_i$ equal to the constraint C_i. This yields the Directional Path Consistency Algorithm (`DPATH`) that apart from 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 11.5 (DPATH ALGORITHM). *Consider a standardized 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 10.4 we do not need to assume here that each domain is finite.

Let us order now each sequence L_m in such a way that the function $f^{x_m}_{x_{i'},x_{j'}}$ precedes $f^{x_m}_{x_{i''},x_{j''}}$ if the pair (j', i') lexicographically precedes the pair(j'', i''). 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 triple **for** loop. The resulting algorithm is known as the DPC algorithm of Dechter and Pearl [1988].

for $m := n$ **to** 3 **by** -1 **do**
    for $j := 2$ **to** $m - 1$ **do**
        for $i := 1$ **to** $j - 1$ **do**
            $C_{i,j} := C_{i,j} \cap C_{i,m} \cdot C^T_{j,m}$
        od
    od
od

12. CONCLUSIONS AND RECENT WORK

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. By starting the presentation with generic iteration algorithms on arbitrary partial orders we clarified the role played in the constraint propagation algorithms by the notions of commutativity and semi-commutativity. This in turn allowed us to provide rigorous and uniform correctness proofs of the AC-3, PC-2, DAC, and DPC algorithms.

The following table summarizes the results of this article.

Local Consistency Notion	Algorithm	Generic Algorithm used	Lemmata Accounting for Correctness
arc consistency	AC-3	CDC (Section 3)	Hyper-arc Consistency 5.2, Commutativity 6.2
path consistency	PC-2	CDC (Section 3)	Path Consistency 7.4, Commutativity 8.1
directional arc consistency	DAC	SI (Section 9)	Hyper-arc Consistency 5.2, Semi-commutativity 10.3
directional path consistency	DPC	SI (Section 9)	Hyper-arc Consistency 5.2, Semi-commutativity 11.4

Since the time this article was submitted for publication the line of research here presented was extended in a number of ways. First, Gennari [2000a] extended slightly the framework of this article and used it to explain the `AC-4` algorithm of Mohr and Henderson [1986], the `AC-5` algorithm of Van Hentenryck et al. [1992], and the `GAC-4` algorithm of Mohr and Masini [1988]. The complication was that these algorithms operate on some extension of the original CSP.

Then, Bistarelli et al. [2000] studied constraint propagation algorithms for soft constraints. To this end they combined the framework of Apt [1999a] and of this paper with the one of Bistarelli et al. [1997]. The latter provides a unified model for several classes of "nonstandard" constraint satisfaction problems employing the concept of a semiring.

Recently Gennari [2000b] showed how another modification of the framework here presented can be used to explain the `PC-4` path consistency algorithm of Han and Lee [1988] and the `KS` algorithm of Cooper [1989] that can achieve either k-consistency or strong k-consistency.

We noted already in Apt [1999a] that using a single framework for presenting constraint propagation algorithms makes it easier to automatically derive, verify, and compare these algorithms. In the meantime the work of Monfroy and Réty [1999] showed that this approach also allows us to parallelize constraint propagation algorithms in a simple and uniform way. This resulted in a general framework for distributed constraint propagation algorithms. As a follow up on this work Monfroy [2000] showed that it is possible to realize a control-driven coordination-based version of the generic iteration algorithm. This shows that constraint propagation can be viewed as the coordination of cooperative agents.

Additionally, as already noted to large extent in Benhamou [1996], such a general framework facilitates the combination of these algorithms, a property often referred to as "solver cooperation." For a coordination-based view of solver cooperation inspired by such a general approach to constraint propagation see Monfroy and Arbab [2000].

Let us mention also that Fernández and Hill [1999] combined the approach of Apt [1999a] with that of Codognet and Diaz [1996] to construct a general framework for solving interval constraints defined over arbitrary lattices. Finally, the generic iteration algorithm `GI` and its specializations can be used as a template for deriving specific constraint propagation algorithms in which particular scheduling strategies are employed. This was done for instance in Monfroy [1999] for the case of nonlinear constraints on reals where the functions to be scheduled were divided into two categories: "weaker" and "stronger" with the preference for scheduling the weaker functions first.

Currently we investigate whether existing constraint propagation algorithms could be improved by using the notions of commutativity and semi-commutativity.

APPENDIX

Proof of Commutativity Lemma 6.2. (i) It suffices to notice that for each k-tuple $X_1, \ldots, X_k$ of subsets of the domains of the respective variables we have

$$\begin{aligned}\pi_j(\pi_i(X_1, \ldots, X_k)) &= (X_1, \ldots, X_{i-1}, X'_i, X_{i+1}, \ldots, X_{j-1}, X'_j, X_{j+1}, \ldots, X_k)\\ &= \pi_i(\pi_j(X_1, \ldots, X_k)),\end{aligned}$$

where

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

$$X'_j := \Pi_j(C \cap (X_1 \times \cdots \times X_k)),$$

and where we assumed that $i < j$.

(ii) Let the considered CSP be of the form $\langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle$. Assume that some common variable of $y_1, \ldots, y_k$ and $z_1, \ldots, z_\ell$ is identical to the variable x_h. Further, let $Sol(C, E)$ denote the set of $d \in D_1 \times \ldots \times D_n$ such that $d[s] \in C$ and $d[t] \in E$, where s is the scheme of C and t is the scheme of E.

Finally, let f denote the π_i function of C and g the π_j function of E. It is easy to check that for each n-tuple $X_1, \ldots, X_n$ of subsets of $D_1, \ldots, D_n$, respectively, we have

$$\begin{aligned} \pi_i^+(\pi_j^+(X_1, \ldots, X_n)) &= (X_1, \ldots, X_{h-1}, X'_h, X_{h+1}, \ldots, X_n) \\ &= \pi_j^+(\pi_i^+(X_1, \ldots, X_n)), \end{aligned}$$

where

$$X'_h := \Pi_h(Sol(C, E) \cap (X_1 \times \cdots \times X_n)).$$

□

Proof of Commutativity Lemma 8.1. Note first that the "relative" positions of z and of u w.r.t. x and y are not specified. There are in total three possibilities concerning z and three possibilities concerning u. For instance, z can be "before" x, "between" x and y, or "after" y. So we have to consider in total nine cases.

In what follows we limit ourselves to an analysis of three representative cases. The proof for the remaining six cases is completely analogous. Recall that we write $x \prec y$ to indicate that x, y is a subsequence of the variables of $\mathcal{P}$.

Case 1. $y \prec z$ and $y \prec u$.
It helps to visualize these variables as in Figure 4. Informally, the functions $f^z_{x,y}$ and $f^u_{x,y}$ correspond, respectively, to the upper and lower triangle in this figure. The fact that these triangles share an edge corresponds to the fact that the functions $f^z_{x,y}$ and $f^u_{x,y}$ share precisely one argument, the one from $\mathcal{P}(C_{x,y})$.

Ignoring the arguments that do not correspond to the schemes of the functions $f^z_{x,y}$ and $f^u_{x,y}$ we can assume that the functions $(f^z_{x,y})^+$ and $(f^u_{x,y})^+$ are both defined on

$$\mathcal{P}(C_{x,y}) \times \mathcal{P}(C_{x,z}) \times \mathcal{P}(C_{y,z}) \times \mathcal{P}(C_{x,u}) \times \mathcal{P}(C_{y,u}).$$

Each of these functions changes only the first argument. In fact, for all elements P, Q, R, U, V of, respectively, $\mathcal{P}(C_{x,y}), \mathcal{P}(C_{x,z}), \mathcal{P}(C_{y,z}), \mathcal{P}(C_{x,u})$, and $\mathcal{P}(C_{y,u})$, we have

$$\begin{aligned} (f^z_{x,y})^+(f^u_{x,y})^+(P, Q, R, U, V) &= (P \cap U \cdot V^T \cap Q \cdot R^T, Q, R, U, V) \\ &= (P \cap Q \cdot R^T \cap U \cdot V^T, Q, R, U, V) \\ &= (f^u_{x,y})^+(f^z_{x,y})^+(P, Q, R, U, V). \end{aligned}$$

Case 2. $x \prec z \prec y \prec u$.

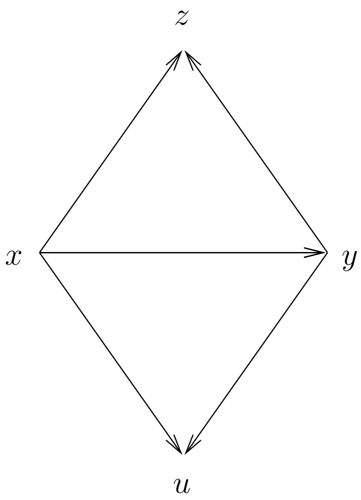

Fig. 4. Four variables connected by directed arcs.

The intuitive explanation is analogous as in Case 1. We confine ourselves to noting that $(f^z_{x,y})^+$ and $(f^u_{x,y})^+$ are now defined on

$$\mathcal{P}(C_{x,z}) \times \mathcal{P}(C_{x,y}) \times \mathcal{P}(C_{z,y}) \times \mathcal{P}(C_{x,u}) \times \mathcal{P}(C_{y,u}),$$

but each of them changes only the second argument. In fact, we have

$$\begin{aligned}(f^z_{x,y})^+(f^u_{x,y})^+(P,Q,R,U,V) &= (P, Q \cap U \cdot V^T \cap P \cdot R, R, U, V)\\ &= (P, Q \cap P \cdot R \cap U \cdot V^T, R, U, V)\\ &= (f^u_{x,y})^+(f^z_{x,y})^+(P,Q,R,U,V).\end{aligned}$$

Case 3. $z \prec x$ *and* $y \prec u$.
In this case the functions $(f^z_{x,y})^+$ and $(f^u_{x,y})^+$ are defined on

$$\mathcal{P}(C_{z,x}) \times \mathcal{P}(C_{z,y}) \times \mathcal{P}(C_{x,y}) \times \mathcal{P}(C_{x,u}) \times \mathcal{P}(C_{y,u}),$$

but each of them changes only the third argument. In fact, we have

$$\begin{aligned}(f^z_{x,y})^+(f^u_{x,y})^+(P,Q,R,U,V) &= (P, Q, R \cap U \cdot V^T \cap P^T \cdot Q, U, V)\\ &= (P, Q, R \cap P^T \cdot Q \cap U \cdot V^T, U, V)\\ &= (f^u_{x,y})^+(f^z_{x,y})^+(P,Q,R,U,V).\end{aligned}$$

□

Proof of Semi-commutativity Lemma 10.3. Suppose that the constraint C_1 is on the variables u, z and the constriant C_2 is on the variables x, y, where $y \preceq z$. Denote by $f_{u,z}$ the π_1 function of C_1 and by $f_{x,y}$ the π_1 function of C_2. The following cases arise.

Case 1. $\{u, z\} \cap \{x, y\} = \emptyset$.
Then the functions $f_{u,z}$ and $f_{x,y}$ commute since their schemes are disjoint.

Case 2. $\{u, z\} \cap \{x, y\} \neq \emptyset$.

Subcase 1. $u = x$.
Then the functions $f_{u,z}$ and $f_{x,y}$ commute by virtue of the Commutativity Lemma 6.2(ii).

Subcase 2. $u = y$.
Let the considered CSP be of the form $\langle \mathcal{C} \,;\, x_1 \in D_1, \ldots, x_n \in D_n \rangle$. We can rephrase the claim as follows, where we denote now $f_{u,z}$ by $f_{y,z}$: For all $(X_1, \ldots, X_n) \in \mathcal{P}(D_1) \times \cdots \times \mathcal{P}(D_n)$ we have

$$f^+_{y,z}(f^+_{x,y}(X_1, \ldots, X_n)) \supseteq f^+_{x,y}(f^+_{y,z}(X_1, \ldots, X_n)).$$

To prove it note first that for some $i, j, k \in [1..n]$ such that $i < j < k$ we have $x = x_i$, $y = x_j$, and $z = x_k$. We now have

$$\begin{aligned} f^+_{y,z}(f^+_{x,y}(X_1, \ldots, X_n)) &= (f_{y,z})^+(X_1, \ldots, X_{i-1}, X'_i, X_{i+1}, \ldots, X_n) \\ &= (X_1, \ldots, X_{i-1}, X'_i, X_{i+1}, \ldots, X_{j-1}, X'_j, X_{j+1}, \ldots, X_n), \end{aligned}$$

where

$$f_{x,y}(X_i, X_j) = (X'_i, X_j)$$

and

$$f_{y,z}(X_j, X_k) = (X'_j, X_k),$$

whereas

$$\begin{aligned} f^+_{x,y}(f^+_{y,z}(X_1, \ldots, X_n)) &= (f_{x,y})^+(X_1, \ldots, X_{j-1}, X'_j, X_{j+1}, \ldots, X_n) \\ &= (X_1, \ldots, X_{i-1}, X''_i, X_{i+1}, \ldots, X_{j-1}, X'_j, X_{j+1}, \ldots, X_k), \end{aligned}$$

where

$$f_{x,y}(X_i, X'_j) = (X''_i, X'_j).$$

By the Hyper-arc Consistency Lemma 5.2(ii) each function π_i is inflationary and monotonic w.r.t. the componentwise ordering $\supseteq$. By the first property applied to $f_{y,z}$ we have $X_j \supseteq X'_j$, so by the second property applied to $f_{x,y}$ we have $X'_i \supseteq X''_i$.

This establishes the claim.

Subcase 3. $z = x$.
This subcase cannot arise, since then the variable z precedes the variable y whereas by assumption the converse is the case.

Subcase 4. $z = y$.
We can assume by Subcase 1 that $u \neq x$. Then the functions $f_{u,z}$ and $f_{x,y}$ commute, since each of them can change only its first component and since this component does not appear in the scheme of the other function. □

Proof of Semi-commutativity Lemma 11.4. Recall that we assumed that $x_1 \prec y_1 \prec z$, $x_2 \prec y_2 \prec u$ and $u \preceq z$. We are supposed to prove that 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 $\supseteq$. The following cases arise.

Case 1. $(x_1, y_1) = (x_2, y_2)$.

In this and other cases by an equality between two pairs of variables we mean that both the first component variables, here x_1 and x_2, and the second component variables, here y_1 and y_2, are identical.

In this case the functions $f^z_{x_1,y_1}$ and $f^u_{x_2,y_2}$ commute by virtue of the Commutativity Lemma 8.1.

Case 2. $(x_1, y_1) = (x_2, u)$.
Then u and z differ, since $y_1 \prec z$. Ignoring the arguments that do not correspond to the schemes of the functions $f^z_{x_1,y_1}$ and $f^u_{x_2,y_2}$ we can assume that the functions $(f^z_{x_1,y_1})^+$ and $(f^u_{x_2,y_2})^+$ are both defined on

$$\mathcal{P}(C_{x_1,y_1}) \times \mathcal{P}(C_{x_1,z}) \times \mathcal{P}(C_{y_1,z}) \times \mathcal{P}(C_{x_2,y_2}) \times \mathcal{P}(C_{y_2,u}).$$

The following now holds for all elements P, Q, R, U, V of, respectively, $\mathcal{P}(C_{x_1,y_1})$, $\mathcal{P}(C_{x_1,z})$, $\mathcal{P}(C_{y_1,z})$, $\mathcal{P}(C_{x_2,y_2})$ and $\mathcal{P}(C_{y_2,u})$:

$$\begin{aligned}
(f^z_{x_1,y_1})^+(f^u_{x_2,y_2})^+(P, Q, R, U, V) &= (f^z_{x_1,y_1})^+(P, Q, R, U \cap P \cdot V^T, V) \\
&= (P \cap Q \cdot R^T, R, U \cap P \cdot V^T, V) \\
&\supseteq (P \cap Q \cdot R^T, R, U \cap (P \cap Q \cdot R^T) \cdot V^T, V) \\
&= (f^u_{x_2,y_2})^+(P \cap Q \cdot R^T, Q, R, U, V) \\
&= (f^u_{x_2,y_2})^+(f^z_{x_1,y_1})^+(P, Q, R, U, V).
\end{aligned}$$

Case 3. $(x_1, y_1) = (y_2, u)$.
In this case u and z differ as well, since $y_1 \prec z$. Again ignoring the arguments that do not correspond to the schemes of the functions $f^z_{x_1,y_1}$ and $f^u_{x_2,y_2}$ we can assume that the functions $(f^z_{x_1,y_1})^+$ and $(f^u_{x_2,y_2})^+$ are both defined on

$$\mathcal{P}(C_{x_1,y_1}) \times \mathcal{P}(C_{x_1,z}) \times \mathcal{P}(C_{y_1,z}) \times \mathcal{P}(C_{x_2,y_2}) \times \mathcal{P}(C_{x_2,u}).$$

The following now holds for all elements P, Q, R, U, V of, respectively, $\mathcal{P}(C_{x_1,y_1})$, $\mathcal{P}(C_{x_1,z})$, $\mathcal{P}(C_{y_1,z})$, $\mathcal{P}(C_{x_2,y_2})$, and $\mathcal{P}(C_{x_2,u})$:

$$\begin{aligned}
(f^z_{x_1,y_1})^+(f^u_{x_2,y_2})^+(P, Q, R, U, V) &= (f^z_{x_1,y_1})^+(P, Q, R, U \cap V \cdot P^T, V) \\
&= (P \cap Q \cdot R^T, R, U \cap V \cdot P^T, V) \\
&\supseteq (P \cap Q \cdot R^T, R, U \cap V \cdot (P \cap Q \cdot R^T)^T, V) \\
&= (f^u_{x_2,y_2})^+(P \cap Q \cdot R^T, Q, R, U, V) \\
&= (f^u_{x_2,y_2})^+(f^z_{x_1,y_1})^+(P, Q, R, U, V).
\end{aligned}$$

Case 4. $(x_1, y_1) \notin \{(x_2, y_2), (x_2, u), (y_2, u)\}$.
Then also $(x_2, y_2) \notin \{(x_1, y_1), (x_1, z), (y_1, z)\}$, since $(x_2, y_2) \neq (x_1, y_1)$ and $y_2 \neq z$ as $y_2 \prec u \preceq z$.

Thus the functions $f^z_{x_1,y_1}$ and $f^u_{x_2,y_2}$ commute since each of them can change only its first component and since this component does not appear in the scheme of the other function. □

ACKNOWLEDGMENTS

Victor Dalmau and Rosella Gennari pointed out to us that Assumptions **A** and **B** in Apt [1999b, page 4] are not sufficient to establish Theorem 1. The added now Assumption **C** was suggested to us by Rosella Gennari. The referees, the editor Alex Aiken, and Eric Monfroy made useful suggestions concerning the presentation.

REFERENCES

A. Aggoun et al. 1995. *ECL^iPS^e 3.5 User Manual.* Munich, Germany.

Apt, K. R. 1999a. The essence of constraint propagation. *Theoretical Computer Science 221,* 1–2, 179–210. Available via `http://arXiv.org/archive/cs/`.

Apt, K. R. 1999b. The rough guide to constraint propagation. In *Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, J. Jaffar, Ed. Lecture Notes in Computer Science 1713. Springer-Verlag, Alexandra, Virginia, USA, 1–23. Invited Lecture. Available via `http://arXiv.org/archive/cs/`.

Benhamou, F. 1996. Heterogeneous constraint solving. In *Proceeding of the Fifth International Conference on Algebraic and Logic Programming (ALP 96)*, M. Hanus and M. Rodriguez-Artalejo, Eds. Lecture Notes in Computer Science 1139. Springer-Verlag, Berlin, 62–76.

Benhamou, F. and Older, W. 1997. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming 32,* 1, 1–24.

Bistarelli, S., Gennari, R., and Rossi, F. 2000. Constraint propagation for soft constraint satisfaction problems: Generalization and termination conditions. In *Proc. of Constraint Programming 2000 (CP2000)*, R. Dechter, Ed. Lecture Notes in Computer Science 1894. Springer-Verlag, Berlin, 83–97.

Bistarelli, S., Montanari, U., and Rossi, F. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM 44,* 2 (Mar.), 201–236.

Codognet, P. and Diaz, D. 1996. Compiling constraints in clp(FD). *Journal of Logic Programming 27,* 3, 185–226.

Cooper, M. C. 1989. An Optimal k-Consistency Algorithm. *Artificial Intelligence 41*, 89–95.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence 113,* 1 & 2, 41–85.

Dechter, R. and Pearl, J. 1988. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence 34,* 1 (Jan.), 1–38.

Dechter, R. and van Beek, P. 1997. Local and global relational consistency. *Theoretical Computer Science 173,* 1 (20 Feb.), 283–308.

Fernández, A. and Hill, P. 1999. Interval constraint solving over lattices using chaotic iterations. In *ERCIM/COMPULOG Workshop on Constraints*, K. Apt, C. Kakas, E. Monfroy, and F. Rossi, Eds. Paphos, Cyprus. Available via `http://www.cwi.nl/ERCIM/WG/Constraints/Workshops/Workshop4/Program/index.html`.

Gennari, R. 2000a. Arc consistency via subsumed functions. In *Proc. of Computational Logic 2000 (CL2000)*, J. Lloyd, Ed. Lecture Notes in Artificial Intelligence 1861. Springer-Verlag, Berlin, 358–372.

Gennari, R. 2000b. The GIF algorithm: A general schema for constraint propagation. Manuscript. Available via `http://www.wins.uva.nl/~rgennari`.

Gierz, G., Hofmann, K., Keimel, K., Lawson, J., Mislove, M., and Scott, D. 1980. *A Compendium of Continuous Lattices.* Springer-Verlag, Berlin.

Han, C. and Lee, C. 1988. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence 36*, 125–130.

ILOG. 1998. ILOG optimization suite — white paper. Available via `http://www.ilog.com`.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence 8,* 1, 99–118.

Mackworth, A. 1992. Constraint satisfaction. In *Encyclopedia of Artificial Intelligence*, S. C. Shapiro, Ed. Wiley, 285–293. Volume 1.

Marriott, K. and Stuckey, P. 1998. *Programming with Constraints.* The MIT Press, Cambridge, Massachusetts.

Mohr, R. and Henderson, T. 1986. Arc-consistency and path-consistency revisited. *Artificial Intelligence 28*, 225–233.

Mohr, R. and Masini, G. 1988. Good old discrete relaxation. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)*, Y. Kodratoff, Ed. Pitman Publishers, 651–656.

Monfroy, E. 1999. Using "Weaker" Functions for Constraint Propagation over Real Numbers. In *Proceedings of the 14th ACM Symposium on Applied Computing, ACM SAC'99, Scientific*

Computing Track, J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. Lamont, Eds. ACM Press, San Antonio, Texas, USA, 553–559.

Monfroy, E. 2000. A Coordination-based Chaotic Iteration Algorithm for Constraint Propagation. In *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC'2000)*. ACM Press, Villa Olmo, Como, Italy, 262–270.

Monfroy, E. and Arbab, F. 2000. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, Chapter Constraints Solving as the Coordination of Inference Engines. To appear.

Monfroy, E. and Réty, J.-H. 1999. Chaotic iteration for distributed constraint propagation. In *Proceedings of the 14th ACM Symposium on Applied Computing, ACM SAC'99, Scientific Computing Track*, J. Carroll, H. Haddad, D. Oppenheim, B. Bryant, and G. Lamont, Eds. ACM Press, San Antonio, Texas, USA, 19–24.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science 7,* 2, 95–132. Also Technical Report, Carnegie Mellon University, 1971.

Saraswat, V., Rinard, M., and Panangaden, P. 1991. Semantic foundations of concurrent constraint programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages (POPL'91)*. 333–352.

Smolka, G. 1995. The Oz programming model. In *Computer Science Today*, J. van Leeuwen, Ed. Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin, 324–343.

Telerman, V. and Ushakov, D. 1996. Data types in subdefinite models. In *Artificial Intelligence and Symbolic Mathematical Computations*, J. A. C. J. Calmet and J. Pfalzgraf, Eds. Lecture Notes in Computer Science 1138. Springer-Verlag, Berlin, 305–319.

Van Hentenryck, P., Deville, Y., and Teng, C. 1992. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence 57,* 2–3 (Oct.), 291–321.

Received November 1999; accepted November 2000