From Chaotic Iteration to Constraint Propagation

Krzysztof R. Apt
CWI
P.O. Box 94079, 1090 GB, The Netherlands
and
Dept. of Mathematics, Computer Science, Physics & Astronomy
University of Amsterdam, The Netherlands

"*Don't express your ideas too clearly.* Most people think little of what they understand, and venerate what they do not."

(The Art of Worldly Wisdom,
Baltasar Gracián, 1647.)

ABSTRACT
We show how the constraint propagation process can be naturally explained by means of chaotic iteration.

1991 Mathematics Subject Classification: 68Q60, 68N05
1991 Computing Reviews Classification System: F.3.1, I.2.2
Keywords and Phrases: chaotic iterations, constraint propagation algorithms.
Note: This paper will appear as an invited lecture in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97). Work carried out under project PNA1.2, CIP.

1. Introduction

1.1 Motivation

Over the last ten years constraint programming emerged as an interesting and viable approach to programming. In this approach the programming process is limited to a generation of requirements ("constraints") and a solution of these requirements by means of general and domain specific methods. The techniques useful for finding solutions to sets of constraints were studied for some twenty years in the field of Constraint Satisfaction. One of the most important of them is *constraint propagation*, the elusive process or reducing a constraint satisfaction problem to another one that is equivalent but "simpler".

The algorithms that achieve such a reduction usually aim at reaching some "local consistency", which denotes some property approximating in some loose sense "global consistency", so the consistency of the whole constraint satisfaction problem. (In fact, most of the notions of local consistency are neither implied by nor imply global consistency.)

For some constraint satisfaction problems such an enforcement of local consistency is already sufficient for finding a solution or for determining that none exists. In some other cases this process substantially reduces the size of the search space which makes it possible to solve the original problem more efficiently by means of some search algorithm.

The aim of this paper is to show that the constraint propagation algorithms can be naturally explained by means of *chaotic iteration*, a basic technique used for computing limits of iterations of finite sets of functions that originated from numerical analysis (see Chazan and Miranker (1969)) and was adapted for computer science needs by Cousot and Cousot (1977). In fact, several constraint propagation algorithms proposed in the literature turn out to be instances of generic chaotic iteration algorithms studied here.

Moreover, by characterizing a given notion of a local consistency as a common fixed point of a finite set of monotonic and inflationary functions we can automatically generate an algorithm achieving this notion of consistency by "feeding" these functions into a generic chaotic iteration algorithm.

1.2 Preliminaries

Definition 1.1 Consider a sequence of domains $\mathcal{D} := D_1, \ldots, D_n$.

- By a *scheme* (on n) we mean a sequence of different elements from $[1..n]$.
- We say that C is a *constraint (on $\mathcal{D}$) with scheme* $i_1, \ldots, i_l$ if $C \subseteq D_{i_1} \times \cdots \times D_{i_l}$.
- Let $\mathbf{s} := s_1, \ldots, s_k$ be a sequence of schemes. We say that a sequence of constraints $C_1, \ldots, C_k$ on $\mathcal{D}$ is an $\mathbf{s}$-*sequence* if each C_i is with scheme s_i.
- By a *Constraint Satisfaction Problem* $\langle \mathcal{D}; \mathcal{C} \rangle$, in short CSP, we mean a sequence of domains $\mathcal{D}$ together with an $\mathbf{s}$-sequence of constraints $\mathcal{C}$ on $\mathcal{D}$. We call then $\mathbf{s}$ the *scheme* of $\langle \mathcal{D}; \mathcal{C} \rangle$. □

Given an n-tuple $d := d_1, \ldots, d_n$ in $D_1 \times \ldots \times D_n$ and a scheme $s := i_1, \ldots, i_l$ on n we denote by $d[s]$ the tuple $d_{i_1}, \ldots, d_{i_l}$. In particular, for $j \in [1..n]$ $d[j]$ is the j-th element of d. By a *solution* to a CSP $\langle \mathcal{D}; \mathcal{C} \rangle$, where $\mathcal{D} := D_1, \ldots, D_n$, we mean an n-tuple $d \in D_1 \times \ldots \times D_n$ such that for each constraint C in $\mathcal{C}$ with scheme s we have $d[s] \in C$.

Consider now a sequence of schemes $s_1, \ldots, s_k$. By its *union*, written as $\langle s_1, \ldots, s_k \rangle$ we mean the scheme obtained from the sequences $s_1, \ldots, s_k$ by removing from each s_i the elements present in some s_j, where $j < i$, and by concatenating the resulting sequences. For example,

$$\langle (3,7,2), (4,3,7,5), (3,5,8) \rangle = (3,7,2,4,5,8).$$

Recall that for an $s_1, \ldots, s_k$-sequence of constraints $C_1, \ldots, C_k$ their *join*, written as $C_1 \bowtie \cdots \bowtie C_k$, is defined as the constraint with scheme $\langle s_1, \ldots, s_k \rangle$ and such that

$$d \in C_1 \bowtie \cdots \bowtie C_k \text{ iff } d[s_i] \in C_i \text{ for } i \in [1..k].$$

Further, given a constraint C and a subsequence s of its scheme, we denote by $\Pi_s(C)$ the constraint with scheme s defined by

$$\Pi_s(C) := \{d[s] \mid d \in C\},$$

and call it *the projection of C on s*. In particular, for a constraint C with scheme s and an element j of s, $\Pi_j(C) = \{a \mid \exists d \in C \; a = d[j]\}$.

Given a CSP $\langle \mathcal{D}; \mathcal{C} \rangle$ we denote by $Sol(\langle \mathcal{D}; \mathcal{C} \rangle)$ the set of all solutions to it. If the domains are clear from the context we drop the reference to $\mathcal{D}$ and just write $Sol(\mathcal{C})$. The following observation is useful.

Note 1.2 *Consider a CSP $\langle \mathcal{D}; \mathcal{C} \rangle$ with $\mathcal{D} := D_1, \ldots, D_n$ and $\mathcal{C} := C_1, \ldots, C_k$ and with scheme* $\mathbf{s}$.

(i) $$Sol(\langle \mathcal{D}; \mathcal{C} \rangle) = C_1 \bowtie \cdots \bowtie C_k \bowtie_{i \in I} D_i,$$

where $I := \{i \in [1..n] \mid \; i \text{ does not appear in } \mathbf{s}\}$.

(ii) For every $\mathbf{s}$*-subsequence* $\mathbf{C}$ *of* $\mathcal{C}$ *and* $d \in Sol(\langle \mathcal{D}; \mathcal{C} \rangle)$ *we have* $d[\langle \mathbf{s} \rangle] \in Sol(\mathbf{C})$.

□

Finally, we call two CSP's *equivalent* if they have the same set of solutions. Note that we do not insist that these CSP's have the same sequence of domains or the same scheme.

2. Chaotic Iterations

As already mentioned in the introduction, one of the corner stones of constraint programming is *constraint propagation.* In general, two basic approaches fall under this name:

- reduce the domains while maintaining equivalence;
- reduce the constraints while maintaining equivalence.

In what follows we study these two processes in full generality.

2.1 Chaotic Iterations on Simple Domains

In general, chaotic iterations are defined for functions that are projections on individual components of a specific function with several arguments. In our approach we study a more elementary situation in which the functions are unrelated but satisfy certain properties. These functions are defined on specific partial orders. We need the following concepts.

Definition 2.1 We call a partial order $(D, \sqsubseteq)$ an $\sqcup$-*po* if

- D contains the least element, denoted by $\perp$,
- for every increasing sequence

  $$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \ldots$$

  of elements from D, the least upper bound of the set

  $$\{d_0, d_1, d_2, \ldots\},$$

  denoted by $\bigsqcup_{n=0}^{\infty} d_n$ and called the *limit of* $d_0, d_1, \ldots$, exists,
- for all $a, b \in D$ the least upper bound of the set $\{a, b\}$, denoted by $a \sqcup b$, exists.

Further, we say that

- an increasing sequence $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \ldots$ *eventually stabilizes at* d if for some $j \geq 0$ we have $d_i = d$ for $i \geq j$,
- a partial order satisfies the *finite chain property* if every increasing sequence of its elements eventually stabilizes. □

Definition 2.2 Consider a set D, an element $d \in D$ and a set of functions $F := \{f_1, \ldots, f_k\}$ on D.

- By a *run* (of the functions $f_1, \ldots, f_k$) we mean an infinite sequence of numbers from $[1..k]$.
- A run $i_1, i_2, \ldots$ is called *fair* if every $i \in [1..k]$ appears in it infinitely often.
- By an *iteration of* F *associated with a run* $i_1, i_2, \ldots$ *and starting with* d we mean an infinite sequence of values $d_0, d_1, \ldots$ defined inductively by

  $$d_0 := d,$$

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

  When d is the least element of D in some partial order clear from the context, we drop the reference to d and talk about an *iteration of* F.

- An iteration of F is called *chaotic* if it is associated with a fair run. □

Definition 2.3 Consider a partial order $(D, \sqsubseteq)$. A function f on D is called

- *inflationary* if $x \sqsubseteq f(x)$ for all x,
- *monotonic* if $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$ for all x, y,
- *idempotent* if $f(f(x)) = f(x)$ for all x.

□

The following observation can be easily distilled from a more general result due to Cousot and Cousot (1977). To keep the paper self-contained we provide a direct proof.

Theorem 2.4 (Chaotic Iteration) *Consider an $\sqcup$-po $(D, \sqsubseteq)$ and a set of functions $F := \{f_1, \ldots, f_k\}$ on D. Suppose that all functions in F are inflationary and monotonic. Then the limit of every chaotic iteration of F exists and coincides with*

$$\bigsqcup_{j=0}^{\infty} f \uparrow j,$$

where the function f on D is defined by:

$$f(x) := \bigsqcup_{i=1}^{k} f_i(x)$$

and $f \uparrow j$ is an abbreviation for $f^j(\bot)$, the j-th fold iteration of f started at $\bot$.

Proof. First notice that f is inflationary, so $\bigsqcup_{j=0}^{\infty} f \uparrow j$ exists. Fix a chaotic iteration $d_0, d_1, \ldots$ of F associated with a fair run $i_1, i_2, \ldots$. Since all functions f_i are inflationary, $\bigsqcup_{j=0}^{\infty} d_j$ exists. The result follows directly from the following two claims.

Claim 1 $\forall j \, \exists m \; f \uparrow j \sqsubseteq d_m$.

Proof. We proceed by induction on j.

Base. $j = 0$. As $f \uparrow 0 = \bot = d_0$, the claim is obvious.

Induction step. Assume that for some $j \geq 0$ we have $f \uparrow j \sqsubseteq d_m$ for some $m \geq 0$. Since

$$f \uparrow (j+1) = f(f \uparrow j) = \bigsqcup_{i=1}^{k} f_i(f \uparrow j),$$

it suffices to prove

$$\forall i \in [1..k] \, \exists m_i \; f_i(f \uparrow j) \sqsubseteq d_{m_i}. \tag{2.1}$$

Indeed, we have then by the fact that $d_l \sqsubseteq d_{l+1}$ for $l \geq 0$

$$\bigsqcup_{i=1}^{k} f_i(f \uparrow j) \sqsubseteq \bigsqcup_{i=1}^{k} d_{m_i} \sqsubseteq d_{m'}$$

where $m' := max\{m_i \mid i \in [1..k]\}$.

So fix $i \in [1..k]$. By fairness of the considered run $i_1, i_2, \ldots$, for some $m_i > m$ we have $i_{m_i} = i$. Then $d_{m_i} = f_i(d_{m_i - 1})$. Now $d_m \sqsubseteq d_{m_i - 1}$, so by the monotonicity of f_i we have

$$f_i(f \uparrow j) \sqsubseteq f_i(d_m) \sqsubseteq f_i(d_{m_i-1}) = d_{m_i}.$$

This proves (2.1). □

Claim 2 $\forall m \; d_m \sqsubseteq f \uparrow m$.

Proof. The proof is by a straightforward induction on m. Indeed, for $m = 0$ we have $d_0 = \perp = f \uparrow 0$, so the induction base holds.

To prove the induction step suppose that for some $m \geq 0$ we have $d_m \sqsubseteq f \uparrow m$. For some $i \in [1..k]$ we have $d_{m+1} = f_i(d_m)$, so by the monotonicity of f we get

$$d_{m+1} = f_i(d_m) \sqsubseteq f(d_m) \sqsubseteq f(f \uparrow m) = f \uparrow (m+1).$$

□

□

In many situations some chaotic iteration studied in the Chaotic Iteration Theorem 2.4 eventually stabilizes. This is for example the case when $(D, \sqsubseteq)$ satisfies the finite chain property. In such cases the limit of every chaotic iteration can be characterized in an alternative way.

Corollary 2.5 (Chaotic Iteration) *Suppose that under the assumptions of the Chaotic Iteration Theorem 2.4 some chaotic iteration of F eventually stabilizes. Then every chaotic iteration of F eventually stabilizes at the least fixed point of f.*

Proof. It suffices to note that if some chaotic iteration $d_0, d_1 \ldots$ of F eventually stabilizes at some d_m then by Claims 1 and 2 $f \uparrow m = d_m$, so

$$\bigsqcup_{j=0}^{\infty} f \uparrow j = f \uparrow m. \tag{2.2}$$

Then, again by Claims 1 and 2, every chaotic iteration of F stabilizes at $f \uparrow m$ and it is easy to see that by virtue of (2.2) $f \uparrow m$ is the least fixed point of f. □

2.2 Chaotic Iterations on Compound Domains

Not much more can be deduced about the process of the chaotic iteration unless the structure of the domain D is further known. So assume now that $(D, \sqsubseteq)$ is the Cartesian product of the $\sqcup$-po's $(D_i, \sqsubseteq_i)$, for $i \in [1..n]$, defined in the expected way. It is straightforward to check that $(D, \sqsubseteq)$ is then an $\sqcup$-po, as well. In what follows we consider a modification of the situation studied in the Chaotic Iteration Theorem 2.4 in which each function f_i affects only certain components of D.

Consider the partial orders $(D_i, \sqsubseteq_i)$, for $i \in [1..n]$ and a scheme $s := i_1, \ldots, i_l$ on n. Then by $(D_s, \sqsubseteq_s)$ we mean the Cartesian product of the partial orders $(D_{i_j}, \sqsubseteq_{i_j})$, for $j \in [1..l]$.

Given a function f on D_s we say that f is *with scheme* s. Instead of defining iterations for the case of the functions with schemes, we rather reduce the situation to the one studied in the previous subsection. To this end we canonically extend each function f on D_s to a function f^+ on D as follows. Suppose that $s = i_1, \ldots, i_l$ and

$$f(d_{i_1}, \ldots, d_{i_l}) = (e'_{i_1}, \ldots, e'_{i_l}).$$

Let for $j \in [1..n]$

$$e_j := \begin{cases} e'_j & \text{if } j \text{ is an element of } s, \\ d_j & \text{otherwise.} \end{cases}$$

Then we set

$$f^+(d_1, \ldots, d_n) := (e_1, \ldots, e_n).$$

Suppose now that $(D, \sqsubseteq)$ is the Cartesian product of the $\sqcup$-po's $(D_i, \sqsubseteq_i)$, for $i \in [1..n]$, and $F := \{f_1, \ldots, f_k\}$ is a set of functions with schemes that are all inflationary and monotonic. Then the following algorithm can be used to compute the limit of the chaotic iterations of $F^+ := \{f_1^+, \ldots, f_k^+\}$. We say here that a function f *depends on* i if i is an element of its scheme.

Generic Chaotic Iteration Algorithm (CI)

$d := \underbrace{(\bot, \ldots, \bot)}_{n \text{ times}};$
$d' := d;$
$G := F;$
while $G \neq \emptyset$ **do**
  choose $g \in G$; suppose g is with scheme s;
  $G := G - \{g\};$
  $d'[s] := g(d[s]);$
  if $d[s] \neq d'[s]$ **then**
    $G := G \cup \{f \in F \mid f$ depends on some i in s such that $d[i] \neq d'[i]\};$
    $d[s] := d'[s]$
  fi
od

The following observation will be useful in the proof of correctness of this algorithm.

Note 2.6 *Consider the partial orders $(D_i, \sqsubseteq_i)$, for $i \in [1..n]$, a scheme s on n and a function f with scheme s. Then*

(i) f is inflationary iff f^+ is,

(ii) f is monotonic iff f^+ is.

□

The following result summarizes the properties of the CI algorithm.

Theorem 2.7

(i) Every terminating execution of the CI *algorithm computes in d the least fixed point of the function f on D defined by*

$$f(x) := \bigsqcup_{i=1}^{k} f_i^+(x).$$

(ii) If all $(D_i, \sqsubseteq_i)$, where $i \in [1..n]$, satisfy the finite chain property, then every execution of the CI *algorithm terminates.*

Proof. It is simpler to reason about a modified, but equivalent, algorithm in which the assignments $d'[s] := g(d[s])$ and $d[s] := d'[s]$ are respectively replaced by $d' := g^+(d)$ and $d := d'$ and the test $d[s] \neq d'[s]$ by $d \neq d'$.
(i) Note that the formula

$$I := \forall f \in F - G \; f^+(d) = d$$

is an invariant of the **while** loop of the modified algorithm. Thus upon its termination

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

holds, that is

$$\forall f \in F\ f^{+}(d) = d.$$

Consequently, some chaotic iteration of F^{+} eventually stabilizes at d. Hence d is the least fixpoint of the function f defined in item (i) because the Chaotic Iteration Corollary 2.5 is applicable here by virtue of Note 2.6(i) and (ii).

(ii) Consider the lexicographic order of the partial orders $(D, \sqsupseteq)$ and $(N, \leq)$, defined on the elements of $D \times N$ by

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

We use here the inverse order $\sqsupset$ and N denotes the set of natural numbers.

By Note 2.6(i) all functions f_i^{+} are inflationary, so with each **while** loop iteration of the modified algorithm the pair

$$(d, card\, G)$$

strictly decreases in this order $\leq_{lex}$. Howver, in general the lexicographic order $(D \times N, \leq_{lex})$ is not well-founded and in fact termination is not guaranteed. But assume now additionally that each partial order $(D_i, \sqsubseteq_i)$ satisfies the finite chain property. Then so does their Cartesian product $(D, \sqsubseteq)$. This means that $(D, \sqsupseteq)$ is well-founded and consequently so is $(D \times N, \leq_{lex})$ which implies termination. □

When all considered functions f_i are also idempotent, we can reverse the order of the two assignments to G, that is to put the assignment $G := G - \{g\}$ after the **if-then-fi** statement, because after applying an idempotent function there is no use in applying it immediately again. Let us denote by `CII` the algorithm resulting from this movement of the assignment $G := G - \{g\}$.

More specialized versions of the `CI` and `CII` algorithms can be obtained by representing G as a queue. To this end we use the operation **enqueue**(F, Q) which for a set F and a queue Q enqueues in an arbitrary order all the elements of F in Q, denote the empty queue by **empty**, and the head and the tail of a non-empty queue Q respectively by **head**(Q) and **tail**(Q). The following algorithm is then a counterpart of the `CI` algorithm.

Generic Chaotic Iteration Algorithm with a Queue (`CIQ`)

$d := \underbrace{(\bot, \ldots, \bot)}_{n \text{ times}};$
$d' := d;$
$Q :=$ **empty**;
enqueue(F, Q);
while $Q \neq$ **empty do**
  $g :=$ **head**(Q); suppose g is with scheme s;
  $Q :=$ **tail**(Q);
  $d'[s] := g(d[s])$;
  if $d[s] \neq d'[s]$ **then**
    enqueue$(\{f \in F \mid f$ depends on some i in s such that $d[i] \neq d'[i]\}, Q)$;
    $d[s] := d'[s]$
  fi
od

Denote by `CIIQ` the modification of the `CIQ` algorithm that is appropriate for the idempotent functions, so the one in which the assignment $Q := \textbf{tail}(Q)$ is performed after the **if-then-fi** statement.

It is easy to see that the claims of Theorem 2.7 also hold for the `CII`, `CIQ` and `CIIQ` algorithms. A natural question arises whether for the specialized versions `CIQ` and `CIIQ` some additional properties can be established. The answer is positive. Namely, for these two algorithms the following result holds which shows that the nondeterminism present in these algorithms has no bearing on their termination.

Theorem 2.8 *If some execution of the* `CIQ` *algorithm terminates, then all the executions of the* `CIQ` *algorithm terminate.*

Proof. We first establish the following observation.

Claim 1 *If some chaotic iteration of* F^+ *eventually stabilizes, then all the executions of the* `CIQ` *algorithm terminate.*

Proof. We prove the contrapositive. Consider an infinite execution of the `CIQ` algorithm algorithm. Let $i_1, i_2, \ldots$ be the run associated with it and $\xi := d_0, d_1, \ldots$ the iteration of F^+ associated with this run. By the structure of this algorithm

$$\xi \text{ does not stabilize.} \tag{2.3}$$

Let A be the set of the elements of $[1..k]$ that appear finitely often in the run $i_1, i_2, \ldots$. For some $m \geq 0$ we have $i_j \notin A$ for $j > m$. This means by the structure of this algorithm that after m iterations of the **while** loop no function f_i for $i \in A$ is ever present in the queue Q.

By virtue of the invariant I used in the proof of Theorem 2.7 we then have $f_i^+(d_j) = d_j$ for $i \in A$ and $j \geq m$. This allows us to transform the iteration ξ to a chaotic one by repeating each element d_j for $j \geq m$ $\ card\, A$ times.

Assume now that a chaotic iteration of F^+ eventually stabilizes. Then by the Chaotic Iteration Corollary 2.5 the just constructed chaotic iteration stabilizes, as well. So the original iteration ξ also stabilizes which contradicts (2.3). □

Construct now a chaotic iteration of F^+ the initial prefix of which corresponds with a terminating execution of the `CIQ` algorithm. By virtue of the invariant I this iteration eventually stabilizes. This concludes the proof thanks to Claim 1. □

An analogous result holds for the `CIIQ` algorithm. On the other hand, it is easy to see that this result does not hold for the `CI` and `CII` algorithms.

3. Constraint Propagation

Let us return now to the study of CSP's. We show here how the results of the previous section can be used to explain the constraint propagation process.

3.1 Domain Reduction

In this subsection we study the domain reduction process. First we associate with each CSP an $\sqcup$-po that "focuses" on the domain reduction.

Consider a CSP $\mathcal{P} := \langle D_1, \ldots, D_n; \mathcal{C}\rangle$. Let for $X, Y \subseteq D_i$

$$X \sqsubseteq_i Y \text{ iff } X \supseteq Y.$$

Then for $i \in [1..n]$ $(\mathcal{P}(D_i), \sqsubseteq_i)$ is an $\sqcup$-po with $\bot_i = D_i$ and $X \sqcup_i Y = X \cap Y$. Consequently, the Cartesian product $(DO, \sqsubseteq)$ of $(\mathcal{P}(D_i), \sqsubseteq_i)$, where $i \in [1..n]$, is also an $\sqcup$-po. We call $(DO, \sqsubseteq)$ *the domain* $\sqcup$*-po associated with* $\mathcal{P}$.

As in in Subsection 2.2, for a scheme $s := i_1,\ldots,i_l$ we denote by $(DO_s,\ \sqsubseteq_s)$ the Cartesian product of the partial orders $(\mathcal{P}(D_{i_j}),\ \sqsubseteq_{i_j})$, where $j \in [1..l]$.

Note that $DO_s = \mathcal{P}(D_{i_1}) \times \cdots \times \mathcal{P}(D_{i_l})$. Because we want now to use constraints in our analysis and constraint are sets of tuples, we identify DO_s with the set

$$\{X_1 \times \cdots \times X_l \mid X_j \subseteq D_{i_j} \text{ for } j \in [1..l]\}.$$

In this way we can write the elements of DO_s as Cartesian products $X_1 \times \cdots \times X_l$, so as (specific) sets of l-tuples, instead of as $(X_1,\ldots,X_l)$, and similarly with DO.

Note that because of the use of the inverse subset order $\supseteq$ we have for $X_1 \times \cdots \times X_l \in DO_s$ and $Y_1 \times \cdots \times Y_l \in DO_s$

$$\begin{array}{rcl} X_1 \times \cdots \times X_l \sqsubseteq_s Y_1 \times \cdots \times Y_l & \text{iff} & X_1 \times \cdots \times X_l \supseteq Y_1 \times \cdots \times Y_l \\ & (\text{iff} & X_i \supseteq Y_i \text{ for } i \in [1..l]), \\ (X_1 \times \cdots \times X_l) \sqcup_s (Y_1 \times \cdots \times Y_l) & = & (X_1 \times \cdots \times X_l) \cap (Y_1 \times \cdots \times Y_l) \\ & (= & (X_1 \cap Y_1) \times \cdots \times (X_l \cap Y_l)). \end{array}$$

Moreover, $D_1 \times \cdots \times D_n$ is the least element of DO.

So far we have defined an $\sqcup$-po associated with a CSP. Next, we introduce functions by means of which chaotic iterations will be generated. These functions are associated with constraints. Constraints are arbitrary sets of k-tuples for some k, while the $\sqsubseteq_s$ order and the $\sqcup_s$ operation are defined only on Cartesian products. So to define these functions we use the set theoretic counterparts $\supseteq$ and $\cap$ of $\sqsubseteq_s$ and $\sqcup_s$ which are defined on arbitrary sets.

Definition 3.1 Consider a sequence of domains $D_1,\ldots,D_n$ and a scheme s on n. By a *domain reduction function* for a constraint C with scheme s we mean a function f on DO_s such that for all $\mathbf{D} \in DO_s$

- $\mathbf{D} \supseteq f(\mathbf{D})$,
- $C \cap \mathbf{D} = C \cap f(\mathbf{D})$. □

The first condition states that f reduces the "current" domains associated with the constraint C (so no solution to C is "gained"), while the second condition states that during this domain reduction process no solution to C is "lost". In particular, the second condition implies that if $C \subseteq \mathbf{D}$ then $C \subseteq f(\mathbf{D})$.

Note that for the partial order $(DO_s,\ \sqsubseteq_s)$ a function f on DO_s is inflationary iff $\mathbf{D} \supseteq f(\mathbf{D})$ and f is monotonic iff it is monotonic w.r.t. the set inclusion.

Example 3.2 As a simple example of a domain reduction functions consider a binary constraint $C \subseteq D_1 \times D_2$. Define now the functions f_1 and f_2 on $DO_{1,2} := \mathcal{P}(D_1) \times \mathcal{P}(D_2)$ as follows:

$$f_1(X \times Y) := X' \times Y,$$

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

$$f_2(X \times Y) := X \times Y',$$

where $Y' = \{b \in Y \mid \exists a \in X\ (a, b) \in C\}$. It is straightforward to check that f_1 and f_2 are indeed domain reduction functions. Further, these functions are monotonic w.r.t. the set inclusion and idempotent. □

Take now a CSP $\mathcal{P} := \langle D_1,\ldots,D_n; \mathcal{C}\rangle$ and a sequence of domains $D'_1,\ldots,D'_n$ such that $D'_i \subseteq D_i$ for $i \in [1..n]$. Consider a CSP $\mathcal{P}'$ obtained from $\mathcal{P}$ by replacing each domain D'_i by D_i and by restricting each constraint in $\mathcal{C}$ to these new domains. We say then that $\mathcal{P}'$ *is determined by* $\mathcal{P}$ *and* $D'_1 \times \ldots \times D'_n$.

Consider now a CSP $\mathcal{P} := \langle D_1,\ldots,D_n; \mathcal{C}\rangle$ and a domain reduction function f for a constraint C of $\mathcal{C}$. Suppose that

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

where f^+ is the canonic extension of f to DO defined in Subsection 2.2. We now define $f(\mathcal{P})$ to be the CSP determined by $\mathcal{P}$ and $D'_1 \times \ldots \times D'_n$. The following observation holds.

Lemma 3.3 *Consider a CSP $\mathcal{P}$ and a domain reduction function f. Then $\mathcal{P}$ and $f(\mathcal{P})$ are equivalent.*

Proof. Suppose that $D_1, \ldots, D_n$ are the domains of $\mathcal{P}$ and assume that f is a domain reduction function for C with scheme $i_1, \ldots, i_l$. Let

$$f(D_{i_1} \times \cdots \times D_{i_l}) = D'_{i_1} \times \cdots \times D'_{i_l}.$$

Take now a solution d to $\mathcal{P}$. Then $d[i_1, \ldots, i_l] \in C$, so by the definition of f also $d[i_1, \ldots, i_l] \in D'_{i_1} \times \cdots \times D'_{i_l}$. So d is also a solution to $f(\mathcal{P})$. The converse implication holds by the definition of a domain reduction function. $\Box$

When dealing with a specific CSP we have in general several domain reduction functions. To study their interaction we can use the Chaotic Iteration Theorem 2.4 in conjunction with the above Note. After translating the relevant notions into set theoretic terms we get the following direct consequence of these results. (In this translation DO_s corresponds to D_s and DO to D.)

Theorem 3.4 (Domain Reduction) *Consider a CSP $\mathcal{P} := \langle D_1, \ldots, D_n; \mathcal{C}\rangle$. Let $F := \{f_1, \ldots, f_k\}$, where each f_i is a domain reduction function for some constraint in $\mathcal{C}$. Suppose that all functions f_i are monotonic w.r.t. the set inclusion. Then*

- *the limit of every chaotic iteration of $F^+ := \{f_1^+, \ldots, f_k^+\}$ exists;*
- *this limit coincides with*

$$\bigcap_{j=0}^{\infty} f^j(D_1 \times \cdots \times D_n),$$

where the function f on DO is defined by:

$$f(\mathbf{D}) := \bigcap_{i=1}^{k} f_i^+(\mathbf{D}),$$

- *the CSP determined by $\mathcal{P}$ and this limit is equivalent to $\mathcal{P}$.* $\Box$

Informally, this theorem states that the order of the applications of the domain reduction functions does not matter, as long as none of them is indefinitely neglected.

Consider now a CSP $\mathcal{P}$ and suppose that the domain $\sqcup$-po associated with it satisfies the finite chain property. Then we can use the `CI`, `CII`, `CIQ` and `CIIQ` algorithms to compute the limits of the chaotic iterations considered in the above Theorem. We shall explain in Subsection 4.1 how by instantiating these algorithms with specific domain reduction functions we obtain specific algorithms considered in the literature. In each case, by virtue of Theorem 2.7 and its reformulations for the `CII`, `CIQ` and `CIIQ` algorithms, we can conclude that these algorithms compute the greatest common fixpoint w.r.t. the set inclusion of the functions from F^+.

3.2 Constraint Reduction

We now study the constraint reduction process. As in the previous subsection we begin by associating with each CSP an $\sqcup$-po that "focuses" on the constraint reduction.

Consider a CSP $\mathcal{P} := \langle \mathcal{D}; C_1, \ldots, C_k \rangle$. Let for $X, Y \subseteq C_i$

$$X \sqsubseteq_i Y \text{ iff } X \supseteq Y.$$

Let now $(CO, \sqsubseteq)$ be the Cartesian product of the $\sqcup$-po's $(\mathcal{P}(C_i), \sqsubseteq_i)$, where $i \in [1..n]$. We call $(CO, \sqsubseteq)$ *the constraint $\sqcup$-po associated with $\mathcal{P}$*.

Following the notation of the previous subsection, for a scheme $s := i_1, \ldots, i_l$ on k we denote by $(CO_s, \sqsubseteq_s)$ the Cartesian product of the partial orders $(\mathcal{P}(C_{i_j}), \sqsubseteq_{i_j})$, where $j \in [1..l]$, and identify CO_s with the set

$$\{X_1 \times \cdots \times X_l \mid X_j \subseteq C_{i_j} \text{ for } j \in [1..l]\},$$

and similarly with CO.

Next, we define functions that will be used to generate chaotic iterations.

Definition 3.5 Consider a CSP $\langle \mathcal{D}; C_1, \ldots, C_k \rangle$ and a scheme s on k. By a *constraint reduction function with scheme s* we mean a function g on CO_s such that for all $\mathbf{C} \in CO_s$

- $\mathbf{C} \supseteq g(\mathbf{C})$,
- $Sol(\mathbf{C}) = Sol(g(\mathbf{C}))$. □

$\mathbf{C}$ is here a Cartesian product of some constraints and in the second condition and in the example below we identified it with the sequence of these constraints, and similarly with $g(\mathbf{C})$. The first condition states that g reduces the constraints C_i, where i is an element of s, while the second condition states that during this constraint reduction process no solution to $\mathbf{C}$ is lost.

Example 3.6 As an example of a constraint reduction function consider the following function g on some CO_s:

$$g(C \times \mathbf{C}) := C' \times \mathbf{C},$$

where $C' = \Pi_t(Sol(C, \mathbf{C}))$ and t is the scheme of C. To see that g is indeed a constraint reduction function, first note that by the definition of Sol we have $C' \subseteq C$, so $C \times \mathbf{C} \supseteq g(C \times \mathbf{C})$. Next, note that for $d \in Sol(C, \mathbf{C})$ we have $d[t] \in \Pi_t(Sol(C, \mathbf{C}))$, so $d \in Sol(C', \mathbf{C})$. This implies that $Sol(C, \mathbf{C}) = Sol(g(C, \mathbf{C}))$.

Note also that g is monotonic w.r.t. the set inclusion and idempotent. □

Example 3.7 As another example that is of importance for the discussion in Subsection 4.1 consider a CSP $\langle D_1, \ldots, D_n; \mathcal{C} \rangle$ of binary constraints such that for each scheme i, j on n there is exactly one constraint, which we denote by $C_{i,j}$.

Define now for each scheme k, l, m on n the following function $g^m_{k,l}$ on CO_s, where s is the triple corresponding to the positions of the constraints $C_{k,l}, C_{k,m}$ and $C_{m,l}$ in $\mathcal{C}$:

$$g^m_{k,l}(X_{k,l} \times X_{k,m} \times X_{m,l}) := (X_{k,l} \cap \Pi_{k,l}(X_{k,m} \bowtie X_{m,l})) \times X_{k,m} \times X_{m,l}.$$

To prove that the functions $g^m_{k,l}$ are constraint reduction functions it suffices to note that by simple properties of the $\bowtie$ operation and by Note 1.2(i) we have

$$\begin{aligned} X_{k,l} \cap \Pi_{k,l}(X_{k,m} \bowtie X_{m,l}) &= \Pi_{k,l}(X_{k,l} \bowtie X_{k,m} \bowtie X_{m,l}) \\ &= \Pi_{k,l}(Sol(X_{k,l}, X_{k,m}, X_{m,l})), \end{aligned}$$

so these functions are special cases of the functions defined in Example 3.6. □

Take now a CSP $\mathcal{P} := \langle \mathcal{D}; C_1, \ldots, C_k \rangle$ and a sequence of constraints $C'_1, \ldots, C'_k$ such that $C'_i \subseteq C_i$ for $i \in [1..k]$. Let $\mathcal{P}' := \langle \mathcal{D}; C'_1, \ldots, C'_k \rangle$. We say then that $\mathcal{P}'$ *is determined by* $\mathcal{P}$ *and* $C'_1 \times \ldots \times C'_k$.

Consider now a CSP $\mathcal{P} := \langle \mathcal{D}; C_1, \ldots, C_k \rangle$ and a constraint reduction function g with scheme s. Suppose that

$$g^+(C_1 \times \cdots \times C_k) = C'_1 \times \cdots \times C'_k,$$

where g^+ is the canonic extension of g to CO defined in Subsection 2.2. We now define

$$g(\mathcal{P}) := \langle \mathcal{D}; C'_1, \ldots, C'_k \rangle.$$

We have the following observation.

Lemma 3.8 *Consider a CSP $\mathcal{P}$ and a constraint reduction function g. Then $\mathcal{P}$ and $g(\mathcal{P})$ are equivalent.*

Proof. Suppose that s is the scheme of the function g and let $\mathbf{C}$ be an element of CO_s. $\mathbf{C}$ is a Cartesian product of some constraints. As before we identify it with the sequence of these constraints. For some sequence of schemes $\mathbf{s}$, $\mathbf{C}$ is the $\mathbf{s}$-sequence of the constraints of $\mathcal{P}$.

Let now d be a solution to $\mathcal{P}$. Then by Note 1.2(ii) we have $d[\langle \mathbf{s} \rangle] \in Sol(\mathbf{C})$, so by the definition of g also $d[\langle \mathbf{s} \rangle] \in Sol(g(\mathbf{C}))$. Hence for every constraint C' in $g(\mathbf{C})$ with scheme s' we have $d[s'] \in C'$ since $d[\langle \mathbf{s} \rangle][s'] = d[s']$. So d is a solution to $g(\mathcal{P})$. The converse implication holds by the definition of a constraint reduction function.

□

As in the case of the domain reduction we can now apply the results of Section 2 to study the outcome of the constraint reduction process. To this end it suffices to translate the relevant notions into set theoretic terms. (In this translation CO_s corresponds to D_s and CO to D.) We get then the following counterpart of the Domain Reduction Theorem 3.4.

Theorem 3.9 (Constraint Reduction) *Consider a CSP $\mathcal{P} := \langle \mathcal{D}; C_1, \ldots, C_k \rangle$. Let $F := \{g_1, \ldots, g_k\}$, where each g_i is a constraint reduction function. Suppose that all functions g_i are monotonic w.r.t. the set inclusion. Then*

- *the limit of every chaotic iteration of $F^+ := \{g_1^+, \ldots, g_k^+\}$ exists;*
- *this limit coincides with*

  $$\bigcap_{j=0}^{\infty} g^j(C_1 \times \cdots \times C_k),$$

  where the function g on CO is defined by:

  $$g(\mathbf{C}) := \bigcap_{i=1}^{k} g_i^+(\mathbf{C}),$$

- *the CSP determined by $\mathcal{P}$ and this limit is equivalent to $\mathcal{P}$.* □

When the constraint $\sqcup$-po associated with a CSP $\mathcal{P}$ satisfied the finite chain property, we can use the algorithms discussed in Subsection 2.2 to compute the limits of the chaotic iterations considered in the above Theorem. We return to this issue in Subsection 4.1. Also here, as in the previous subsection, we can conclude by virtue of Theorem 2.7 that these algorithms compute the greatest common fixpoint w.r.t. the set inclusion of the functions from F^+. So the limit of the constraint propagation process could be added to the collection of important greatest fixpoints presented in Barwise and Moss (1996).

Next, we show how specific provably correct algorithms for achieving a local consistency notion can be automatically derived. As it is difficult to define local consistency formally, we illustrate the idea on an example.

Example 3.10 We consider here the notion of relational consistency proposed recently in Dechter and van Beek (1997).

To define it need to introduce some auxiliary concepts first. Consider a CSP $\langle D_1, \ldots, D_n; \mathcal{C}\rangle$. Take a scheme $t := i_1, \ldots, i_l$ on n. We call $d \in D_{i_1} \times \cdots \times D_{i_l}$ a tuple of *type* t and say that d is *consistent* if for every subsequence s of t and a constraint $C \in \mathcal{C}$ with scheme s we have $d[s] \in C$.

A CSP $\mathcal{P}$ is called *relationally* m*-consistent* if for any $\mathbf{s}$-sequence $C_1, \ldots, C_m$ of different constraints of $\mathcal{P}$ and a subsequence t of $\langle \mathbf{s} \rangle$, every consistent tuple of type t belongs to $\Pi_t(C_1 \bowtie \cdots \bowtie C_m)$.

As the first step we characterize this notion as a common fixed point of a finite set of monotonic and inflationary functions.

Consider a CSP $\mathcal{P} := \langle D_1, \ldots, D_n; C_1, \ldots, C_k\rangle$. Assume for simplicity that for every scheme s on n there is a unique constraint with scheme s. Each CSP is trivially equivalent with such a CSP — it suffices to replace for each scheme s the set of constraints with scheme s by their intersection and to introduce "universal constraints" for the schemes without a constraint.

Consider now a scheme $i_1, \ldots, i_m$ on k. Let $\mathbf{s}$ be such that $C_{i_1}, \ldots, C_{i_m}$ is an $\mathbf{s}$-sequence of constraints and let t be a subsequence of $\langle \mathbf{s} \rangle$. Further, let C_{i_0} be the constraint of $\mathcal{P}$ with scheme t. Put $s := \langle (i_0), (i_1, \ldots, i_m)\rangle$. (Note that if i_0 does not appear in $i_1, \ldots, i_m$ then $s = i_0, i_1, \ldots, i_m$ and otherwise s is the permutation of $i_1, \ldots, i_m$ obtained by transposing i_0 with the first element.)

Define now a function g_s on CO_s by

$$g_s(C \times \mathbf{C}) := (C \cap \Pi_t(\bowtie \mathbf{C})) \times \mathbf{C}.$$

It is easy to see that if for each function g_s of the above form we have

$$g_s^+(C_1 \times \cdots \times C_k) = C_1 \times \cdots \times C_k,$$

then $\mathcal{P}$ is relationally m-consistent. (The converse implication is in general not true). Note that the functions g_s are inflationary and monotonic w.r.t. the inverse subset order $\supseteq$ and also idempotent.

Consequently, by virtue of Theorem 2.7 reformulated for the `CII` algorithm, we can now use the `CII` algorithm to achieve relational m-consistency for a CSP with finite domains by "feeding" into this algorithm the above defined functions. The obtained algorithm improves upon the (authors' terminology) brute force algorithm proposed in Dechter and van Beek (1997) since the useless constraint modifications are avoided.

As in Example 3.7, by simple properties of the $\bowtie$ operation and by Note 1.2(i) we have

$$C \cap \Pi_t(\bowtie \mathbf{C}) = \Pi_t(C \bowtie (\bowtie \mathbf{C})) = \Pi_t(sol(C, \mathbf{C})).$$

Hence, by virtue of Example 3.6, the functions g_s are all constraint reduction functions. Consequently, by the Constraint Reduction Theorem 3.9 we conclude that the CSP computed by the just discussed algorithm is equivalent to the original one. $\square$

It is perhaps worthwhile to note that the domain reduction process can be seen as a special case of the constraint reduction process. To this end it suffices to introduce unary constraints each of which coincides with a different domain of the given CSP and replace the reduction of the domains by the reduction of these unary constraints followed by the restriction of the other constraints to these reduced unary constraints. So the domain reduction functions can be seen as special cases of the constraint reduction functions.

We decided to consider the domain reduction process separately, because, as we shall see in the next section, it has been extensively studied, especially in the context of CSP's with binary constraints and of interval arithmetic. Consequently, it is useful to analyze it directly, without any introduction of new constraints.

4. Concluding Remarks

4.1 Related Work

It is illuminating see how 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.

As already stated in the introduction, the aim of the constraint propagation algorithms is most often to achieve some form of local consistency. As a result these algorithms are usually called in the literature "consistency algorithms" or "consistency enforcing algorithms".

To start with, in Mackworth (1977) a unified framework was proposed to explain the so-called arc- and path-consistency algorithms. Also the arc-consistency algorithm `AC-3` and the path-consistency algorithm `PC-2` were proposed and the latter algorithm was obtained from the former one by pursuing the analogy between both notions of consistency.

The `AC-3` consistency algorithm can be obtained by instantiating the `CII` algorithm with the domain reduction functions defined in Example 3.2, whereas the `PC-2` algorithm can be obtained by instantating this algorithm with the domain reduction functions defined in Example 3.7.

In Dechter and Pearl (1988) the notions of arc- and path-consistency were modified to directional arc- and path-consistency, versions that take into account some total order $<_d$ of the domain indices, and the algorithms for achieving these forms of consistency were presented. These algorithms can be obtained as instances of the `CIQ` algorithm as follows.

For the case of directional arc-consistency the queue in this algorithm should be instantiated with the set of the domain reduction functions f_1 of Example 3.2 for the constraints the scheme of which is consistent with the $<_d$ order. These functions should be ordered in such a way that the domain reduction functions for the constraint with the $<_d$-large second index appear earlier. This order has the effect that the **enqueue** operation within the **if-then-fi** statement has always the empty set as the first argument, so it can be deleted. Consequently, the algorithm can be rewritten as a simple **for** loop that processes the selected domain reduction functions f_1 in the appropriate order.

For the case of directional path-consistency the constraint reduction functions $g^m_{k,l}$ should be used only for $k, l <_d m$ and the queue in the `CIQ` algorithm should be initialized in such a way that the functions $g^m_{k,l}$ with the $<_d$-large m index appear earlier. As in the case of directional arc-consistency this algorithm can be rewritten as a simple **for** loop.

In Montanari and Rossi (1991) a general study of constraint propagation was undertaken by defining the notion of a relaxation rule and by proposing a general relaxation algorithm. The notion of a relaxation rule coincides with our notion of a constraint propagation function instantiated with the functions defined in Example 3.6 and the general relaxation algorithm is the corresponding instance of our `CI` algorithm.

In Montanari and Rossi (1991) it was also shown that the notions of arc-consistency and path-consistency can be defined by means of relaxation rules and that as a result arc-consistency and path-consistency algorithms can be obtained by instantiating with these rules their general relaxation algorithm.

Van Hentenryck, Deville and Teng (1992) presented a generic arc consistency algorithm, called `AC-5`, that can be specialized to the known arc-consistency algorithms `AC-3` and `AC-4` and also to new arc-consistency algorithms for specific classes of constraints.

In Benhamou, McAllester and Hentenryck (1994) and Benhamou and Older (1997) specific functions, called narrowing functions, were associated with constraints in the context of interval arithmetic for reals and some properties of them were established that in our terminology mean that these are idempotent domain reduction functions. As a consequence the algorithms proposed in these papers, called respectively a fixpoint algorithm and a narrowing algorithm, become respectively the instances of our `CIIQ` algorithm and `CII` algorithm.

The importance of fairness for the study of constraint propagation was noticed in Montanari and

Rossi (1991), while the relevance of the chaotic iteration was independently noticed in Fages, Fowler and Sola (1996) and van Emden (1996). In the latter paper the generic chaotic iteration algorithm `CII` was formulated and proved correct for the domain reduction functions defined in Benhamou and Older (1997) and it was shown that the limit of the constraint propagation process for these functions is their greatest common fixpoint.

The idea that the meaning of a constraint is a function (on a constraint store) with some algebraic properties was put forward in Saraswat, Rinard and Panangaden (1991), where the properties of being inflationary (called there extensive), monotonic and idempotent were singled out.

It is unrealistic to expect that all constraint propagation algorithms presented in the literature can be expressed as direct instances of the algorithms discussed in this paper. For example the `AC-4` algorithm of Mohr and Henderson (1986) associates with each domain element some information concerning its links with the elements of other domains. As a result this algorithm operates on some "enhancement" of the original domains.

We noted, however, that even in this case the analysis here provided can be used to explain this algorithm. To this end one needs to reason about the translation of the original CSP to a CSP defined on the enhanced domains. This analysis allows us to reduce the proof of the correctness of this algorithm to the proof that specific functions are monotonic domain reduction functions.

4.2 Idempotence

In each of the above papers the (often implicitly) considered semantic, domain or constraint reduction functions are idempotent, so we now comment on the relevance of this assumption.

To start with, in our study Apt (1997) of linear constraints on finite integer intervals we found that natural domain reduction functions are not idempotent. Secondly, as noticed in Older and Vellino (1993), another paper on constraints for interval arithmetic on reals, we can always replace each non-idempotent inflationary function f by

$$f^*(x) := \bigsqcup_{i=1}^{\infty} f^i(x).$$

The following is now straightforward to check.

Note 4.1 *Consider an* $\sqcup$*-po* $(D, \sqsubseteq)$ *and a function* f *on* D.

- *If* f *is inflationary, then so is* f^*.
- *If* f *is monotonic, then so* f^*.
- *If* f *is inflationary and* $(D, \sqsubseteq)$ *has the finite chain property, then* f^* *is idempotent.*
- *If* f *is idempotent, then* $f^* = f$.
- *Suppose that* $(D, \sqsubseteq)$ *has the finite chain property. Let* $F := \{f_1, \ldots, f_k\}$ *be a set of inflationary, monotonic functions on* D *and let* $F^* := \{f_1^*, \ldots, f_k^*\}$. *Then the limits of all chaotic iterations of* F *and of* F^* *exist and always coincide.* □

Consequently, under the conditions of the last item, every chaotic iteration of F^* can be modeled by a chaotic iteration of F, though not conversely. In fact, the use of F^* instead of F can lead to a more limited number of chaotic iterations. This may mean that in some specific algorithms some more efficient chaotic iterations of F cannot be realized when using F^*.

4.3 Semi-chaotic Iterations

The results of this paper can be slightly strengthened by considering the following generalization of the chaotic iterations.

Definition 4.2 Consider a set of functions $F := \{f_1, \ldots, f_k\}$ on a domain D.

- We say that an element $i \in [1..k]$ is *eventually irrelevant for an iteration* $d_0, d_1, \ldots$ *of* F if $\exists m \geq 0 \, \forall j \geq m \; f_i(d_j) = d_j$.
- An iteration of F is called *semi-chaotic* if every $i \in [1..k]$ that appears finitely often in its run is eventually irrelevant for this iteration. □

So every chaotic iteration is semi-chaotic but not conversely. Now, in all the results of this paper chaotic iterations can be replaced by semi-chaotic iterations. The reason is that, as shown in the proof of Theorem 2.8, every semi-chaotic iteration ξ can be transformed into a chaotic iteration ξ' with the same limit and such that ξ eventually stabilizes at some d iff ξ' does. The proof of Theorem 2.8 also shows that every infinite execution of the `CIQ` algorithm is associated with a semi-chaotic iteration of F^+.

However, the property of being a semi-chaotic iteration cannot be determined from the run only. So, for simplicity, we decided to limit our exposition to chaotic iterations.

Acknowledgements

This work was prompted by our study of van Emden (1996). Rina Dechter helped us to clarify (most of) our initial confusion about constraint propagation. Discussions with Eric Monfroy helped us to better articulate various points put forward here. Nissim Francez provided us with helpful comments.

References

Apt, K. (1997). A proof theoretic view of constraint programming, *Technical report*, CWI, Amsterdam. In preparation.

Barwise, J. and Moss, L. (1996). *Vicious Circles: on the mathematics of circular phenomena*, CSLI–Lecture Notes, Center for the Study of Language and Information, Stanford, California.

Benhamou, F. and Older, W. (1997). Applying interval arithmetic to real, integer and Boolean constraints, *Journal of Logic Programming*. Technical report 1994. To appear.

Benhamou, F., McAllester, D. and Hentenryck, P. V. (1994). CLP(intervals) revisited, *in* M. Bruynooghe (ed.), *Proceedings of the 1994 International Logic Programming Symposium*, pp. 124–138.

Chazan, D. and Miranker, W. (1969). Chaotic relaxation, *Linear Algebra and its Applications* **2**: 199–222.

Cousot, P. and Cousot, R. (1977). Automatic synthesis of optimal invariant assertions: mathematical foundations, *ACM Symposium on Artificial Intelligence and Programming Languages*, SIGPLAN Notices 12 (8), pp. 1–12.

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

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

Fages, F., Fowler, J. and Sola, T. (1996). Experiments in reactive constraint logic programming, *Technical report*, DMI - LIENS CNRS, Ecole Normale Supérieure. Submitted for publication.

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

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

Montanari, U. and Rossi, F. (1991). Constraint relaxation may be perfect, *Artificial Intelligence* **48**: 143–170.

Older, W. and Vellino, A. (1993). Constraint arithmetic on real intervals, *in* F. Benhamou and A. Colmerauer (eds), *Constraint Logic Programming: Selected Research*, MIT Press, pp. 175–195.

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

van Emden, M. H. (1996). Value constraints in the CLP scheme, *Technical Report CS-R9603*, CWI, Amsterdam. To appear in the Constraints journal.

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