A Chaotic Iteration View for Interval Constraint Solving over Lattices

Antonio J. Fernández,*
Lenguajes y Ciencias de la Computación,
Universidad de Málaga,
29071 Teatinos,
Málaga, Spain
e-mail: afdez@lcc.uma.es

Patricia M. Hill
School of Computer Studies,
University of Leeds,
Leeds, LS2 9JT,
England
e-mail: hill@scs.leeds.ac.uk

Abstract

We have defined a constraint solver for any domain that is a lattice. The approach is based on the use of a single constraint similar to the indexicals used in CLP (Constraint Logic programming) over the finite domains. In this paper, we present this constraint solver and the algorithm used by the solver for constraint propagation within the framework of chaotic iteration.

1 Introduction

During the last decade, constraint programming [Marriot and Stuckey 1998] has emerged as one of the most exciting fields for program language development. Many constraint languages are based on logic programming (CLP) [Jaffar and Maher 1994] and these have been applied on domains such as the integers [Codognet and Diaz 1996b], reals [Refalo and Van Hentenryck 1996], sets [Gervet 1997] or Booleans [Codognet and Diaz 1996a]. One particular approach to the constraint propagation on finite domains, called the indexical approach [Codognet and Diaz 1996b], has been proved to be very efficient and to have good expressivity. Indexicals are constraints on intervals of integers originally developed for constraint logic programming over an ordered finite domain and are now available in commercial languages such as SICStus Prolog [Carlsson *et al.*, 1994].

Most of these applications of CLP are independent in the sense that usually a different mechanism of constraint propagation is defined on each particular domain and no relation can be derived from them. Recently several proposals for finding general principles for constraint propagation, independently of the computation domain, have appeared in the literature. Among these proposals,

*This work was partly supported by EPSRC grants GR/L19515 and GR/M05645 and by CICYT grant TIC98-0445-C03-03.

we have developed [Fernández and Hill 1999] a theoretical framework to solve constraints on any domain with lattice structure. This proposal is independent from the nature of the domain and its cardinality so that it can be catalogued as a generic framework. Moreover, the indexical approach to constraint solvers together with a schematic procedure for computing solutions can be viewed as a special case of the generic framework [Codognet and Diaz 1996b]. Our framework is designed for similar constraints to the finite domain constraints but for any domain (finite or infinite) which has a lattice structure. Moreover, as it is based on lattice theory, starting with existing domains (system and user-defined), it is straightforward using standard lattice constructions such as the direct product of lattices to obtain new domains and new constraint solvers for these domains (as it was shown in [Fernández and Hill 1999, Section 5]).

Based on an idea, originally used in numerical analysis, for computing limits of iterations of finite sets of functions, there is in [Apt 1997],[Apt 1999a], a new proposal for a generic framework for constraint propagation that is based on chaotic iteration (CI) algorithms for partially ordered domains. The key idea is that most constraint propagation algorithms presented in the literature can be expressed as direct instances of these algorithms. These papers also show how new constraint propagation algorithms for certain domains may be obtained as instances of this framework.

The frameworks described in [Fernández and Hill 1999] and [Apt 1999a] appear to have a number of analogies; in particular, both consider schemas for constraint propagation procedures for partially ordered domains, finite or infinite. The main contribution of this paper is to establish the relationship between these approaches so that the properties of the algorithms shown in each of them can be generalised to the other; indicating the limitations of the current generic algorithms presented by Apt and how our work can exploit the results already established elsewhere.

The paper is structured as follows. Section 2 recalls some concepts used in the paper. Section 3 presents a slight variation of the interval lattice-based constraint solving framework already described in [Fernández and Hill 1999] and an operational schema for computing a solution to these interval constraints. In Section 4, a theory of the chaotic iteration based on that described in [Apt 1999a] is presented. A chaotic iteration algorithm based on the CI algorithm given in [Apt 1999a] is presented together with the necessary termination and correctness properties. In Section 5, we relate this schema to the chaotic iteration algorithm in 4. The paper ends with the conclusions and some considerations about further work.

2 Preliminaries

Suppose $D_1, \dots, D_n$ are domains, then the domain $D = D_1 \times \cdots \times D_n$ is their Cartesian product. Then if $d = (d_1, \dots, d_n) \in D$, we define $d[i] \stackrel{\text{def}}{=} d_i$, to be the i'th component of d.

If s is an infinite sequence, then s[1] *eventually stabilises at* d if, for some $j \geq 0$, we have $d_i = d$ for all $i \geq j$. A partial order satisfies the *finite chain property* if every increasing sequence of its elements eventually stabilises.

Definition 1 *($\curlyvee$-po) Let $(D, \preceq)$ a partial order. $(D, \preceq)$ is called a $\curlyvee$-po if the least element $\perp$ of D exists; for all $a, b \in D$ the least upper bound of $\{a, b\}$ exists; and for every increasing sequence $\langle d_0, d_1, d_2 \ldots \rangle$ of elements in D, the least upper bound $lub\{d_n \mid n \geq 0\}$ exists.*

Definition 2 *(Functions on D) Let $(D, \preceq_D)$ be a partial order. A function is* on D *if it is a self-map on D. A function f on D is* extensive *if, for each $d \in D$, $d \preceq_D f(d)$;* monotonic *if, for each $d_1, d_2 \in D$, whenever $d_1 \preceq_D d_2$, we have $f(d_1) \preceq_D f(d_2)$ and;* idempotent *if, for each $d \in D$, $f(f(d)) = f(d)$.*

Definition 3 *(run) A* run *on a (finite) sequence s is an infinite sequence of elements of s. We denote the i'th element of a run r as $r[i]$. A run is* fair *if each element of s appears in it infinitely often.*

Definition 4 *(Direct product) Let C_1 and C_2 be ordered sets. The* direct product *$C = \langle C_1, C_2 \rangle$ is an ordered set with domain the Cartesian product of C_1 and C_2 and ordering defined by: $\langle x_1, x_2 \rangle \preceq_C \langle y_1, y_2 \rangle \iff x_1 \preceq_{C_1} y_1$ and $x_2 \preceq_{C_2} y_2$*

Definition 5 *(Lexicographic product) Let C_1 and C_2 be ordered sets. The* lexicographic product *$C = (C_1, C_2)$ is an ordered set with domain the Cartesian product of C_1 and C_2 and ordering defined by:*

$$(x_1, x_2) \preceq_C (y_1, y_2) \iff x_1 \prec_{C_1} y_1 \textit{ or } x_1 = y_1 \textit{ and } x_2 \preceq_{C_2} y_2$$

Definition 6 *(Lattice) Let L be an ordered set. L is a* lattice *if $lub_L(x, y)$ and $glb_L(x, y)$ exist for any two elements $x, y \in L$. If $lub_L(S)$ and $glb_L(S)$ exist for all $S \subseteq L$, then L is a* complete lattice.

Definition 7 *(Dual of a lattice) Given a lattice L we can form a lattice $\hat{L}$ (called the* dual of L*) which contains the same elements as L and $glb_{\hat{L}}(S) = a$ if and only if $lub_L(S) = a$.*

Proposition 1 *(Products of lattices) Let L_1 and L_2 be two (lifted) lattices. Then the direct product $\langle L_1, L_2 \rangle$ and the lexicographic product (L_1, L_2) are lattices when we define:*

$$\begin{aligned}
glb(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) = \quad & \langle glb_{L_1}(x_1, y_1), glb_{L_2}(x_2, y_2) \rangle \\
glb((x_1, x_2), (y_1, y_2)) = \quad & \textit{if } x_1 = y_1 \textit{ then } (x_1, glb_{L_2}(x_2, y_2)) \\
& \textit{elsif } x_1 \prec y_1 \textit{ then } (x_1, x_2) \\
& \textit{elsif } x_1 \succ y_1 \textit{ then } (y_1, y_2) \\
& \textit{else } (glb_{L_1}(x_1, y_1), \top_{L_2})
\end{aligned}$$

lub is defined dually to glb[2].

More information on lattice theory can be found in [Davey and Priestley 1990].

[1] This is the same as the definition in [Apt 1999a] apart from the fact that we do not require the sequence to be increasing.

[2] Note that $\top_{L_2}$ must be also changed to its dual $\perp_{L_2}$.

3 Interval Constraints

3.1 The Computation Domain

The underlying domain for the constraints, denoted here by L_0, is a lattice called the *computation domain*. It is assumed that L_0 has been lifted to include top and bottom elements, $\top_{L_0}$ and $\perp_{L_0}$ respectively.

The domain that is actually used for the constraint solving is a set of intervals on the computation domain and called the *interval domain*. We allow for the bounds of the interval to be either open or closed and denote these bounds with open and closed brackets, respectively. Thus, we first need to define an ordering between the open and closed right brackets ')', ']' so that the domain of right brackets is itself a lattice.

Definition 8 *(Bracket domain) The* bracket domain B *is the lattice of two elements ')' and ']' with ordering ')'* $\prec_B$ *']'. Any element of B is denoted by '}'.*

Definition 9 *(Simple bounded computation domain) The* simple bounded computation domain L *is the lexicographic product* (L_0, B).

By Proposition 1, L is a lattice. For clarity we write a} to express (a,'}') in L. For example, if L_0 is the integer domain, then in $L = (L_0, B)$, $3) \preceq_L 3]$, $4] \preceq_L 7]$, $glb_L(3], 5]) = 3]$ and $lub_L(3], 3)) = 3]$. Note that $\perp_L = \perp_{L_0})$ and that $\top_L = \top_{L_0}]$.

Definition 10 *(Mirror of L) The* mirror of L *is the lexicographic product* $(\hat{L_0}, B)$ *and is denoted by $\overline{L}$. The mirror of an element $t \in L$ is denoted by $\overline{t}$. By Proposition 1, $\overline{L}$ is a lattice. For convenience, we write $\{a$ to express $\overline{a\}}$.*

Note that if $t_1 = a_1\}_1, t_2 = a_2\}_2 \in L$ where $a_1 \neq a_2$ we have:
(1) $\overline{t_2} \preceq_{\overline{L}} \overline{t_1} \Leftrightarrow t_1 \preceq_L t_2$;
(2) $glb_{\overline{L}}(\overline{t_1}, \overline{t_2}) = \overline{lub_L(t_1, t_2)}$ and $lub_{\overline{L}}(\overline{t_1}, \overline{t_2}) = \overline{glb_L(t_1, t_2)}$;
(3) $\perp_{\overline{L}} = \overline{\top_{L_0})} = (\top_{L_0}$, $\top_{\overline{L}} = \overline{\perp_{L_0}]} = [\perp_{L_0}$.

For example, if $L_0 = \Re$, $\overline{3.1]} = [3.1$ and $\overline{6.7)} = (6.7$, $[5.2 \preceq_{\overline{L}} (3.1 \preceq_{\overline{L}} [3.1 \preceq_{\overline{L}} [2.2$, $glb_{\overline{L}}([5.0, [7.2) = [7.2$ and $lub_{\overline{L}}([5.0, [7.2) = [5.0$.

3.2 Constraint Operators

Let $L = (L_0, B)$ be the simple bounded computation domain for L_0.

Definition 11 *(Unary Constraint operators) A* unary constraint operator (for L) *is a function $\delta :: L_1 \to L$ where $L_1 \in \{L, \overline{L}\}$. Given a unary constraint operator δ, the* mirror operator $\overline{\delta} :: \overline{L_1} \to \overline{L}$ *is defined, for each $t_1 \in L_1$, to be $\overline{\delta}(\overline{t_1}) = \overline{\delta(t_1)}$.*

Definition 12 *(Binary Constraint operators) A* bynary constraint operator (for L) *is a function $\circ :: L_1 \times L_2 \to L$ where $L_1, L_2 \in \{L, \overline{L}\}$. Given a binary constraint operator $\circ$, the* mirror operator $\overline{\circ} :: \overline{L_1} \times \overline{L_2} \to \overline{L}$ *is defined, for each $t_1 \in L_1$ and $t_2 \in L_2$, to be $\overline{t_1} \overline{\circ} \overline{t_2} = \overline{t_1 \circ t_2}$.*

Definition 13 *(Monotonicity of operators) Suppose* $L_1, L_2 \in \{L, \overline{L}\}$, $\delta :: L_1 \to L$ *is a unary constraint operator and* $\circ :: L_1 \times L_2 \to L$ *is a binary constraint operator. Then* δ *is* monotonic *if, for all* $t_1, t_1' \in L_1$ *such that* $t_1 \preceq_{L_1} t_1'$ *we have* $\delta(t_1) \preceq_L \delta(t_1')$ *and* $\circ$ *is* monotonic *if, for all* $t_1, t_1' \in L_1$ *and* $t_2, t_2' \in L_2$ *such that* $t_1 \preceq_{L_1} t_1'$ *and* $t_2 \preceq_{L_2} t_2'$ *we have*

$$\begin{aligned} (t_1 \circ t_2) &\preceq_L (t_1' \circ t_2) \\ (t_1 \circ t_2) &\preceq_L (t_1 \circ t_2'). \end{aligned}$$

Lemma 1 *A (unary or binary) constraint operator is monotonic if and only if its mirror operator is monotonic.*

We impose the following restriction on the constraint operators.

Law of monotonicity for constraint operators.

- Each constraint operator must be monotonic.

Normally, a constraint operator will be defined by the user or system on L_0 and B separately. The value of a unary operator $\delta :: L_1 \to L$ where $L_1 \in \{L, \overline{L}\}$ is then inferred from its value on L_0 and B so that, if $t_1 = a_1\}_1$ is a term in L_1 then $\delta(t_1) = \delta(a_1)\delta(\}_1)$. Respectively, the value of a binary operator $\circ :: L_1 \times L_2 \to L$ where $L_1, L_2 \in \{L, \overline{L}\}$ is also inferred from its value on L_0 and B so that, if $t_1 = a_1\}_1, t_2 = a_2\}_2$ are terms in L_1, L_2, respectively, then $t_1 \circ t_2 = (a_1 \circ a_2)(\}_1 \circ \}_2)$. Then, if the law of monotonicity is to hold in L, it has to hold for the definitions of δ and $\circ$ on each of L_0 and B.

For example, if $L_0 = \Re$, $3.0) + 4.0] = 7.0)$ where $)+] =)$ and $3.0 + 4.0 = 7.0$.

3.3 Indexicals

We now introduce variables to the domain (L_0, B) and its dual by means of *indexicals*. To distinguish between the simple bounded computation domain already defined and the same domain but augmented with indexicals, we denote (L_0, B) by L^s. We assume that there is both a set O_L of constraint operators defined on L^s and a (possibly infinite) set V_{L_0} of variables associated with the domain L_0.

Definition 14 *(Bounded computation domain) If* L^s *is a simple bounded computation domain for* L_0*, then the* bounded computation domain L for L_0 *and its mirror* $\overline{L}$ *are defined*

$$\begin{aligned} L = & L^s \cup \{max(x) \mid x \in V_{L_0}\} \cup \{t_1 \circ t_2 \mid \circ :: L_1^s \times L_2^s \to L^s \in O_L, t_1 \in L_1, t_2 \in L_2\}, \\ & \cup \{\delta(t_1) \mid \delta :: L_1^s \to L^s \in O_L, t_1 \in L_1\}, \\ \overline{L} = & \overline{L^s} \cup \{min(x) \mid x \in V_{L_0}\} \cup \{\overline{t_1 \circ t_2} \mid \circ :: L_1^s \times L_2^s \to L^s \in O_L, t_1 \in L_1, t_2 \in L_2\} \\ & \cup \{\overline{\delta(t_1)} \mid \delta :: L_1^s \to L^s \in O_L, t_1 \in L_1\} \end{aligned}$$

where, for each $x \in V_{L_0}$,

$$\overline{max(x)} = min(x),$$

for each $\delta :: L_1^s \to L^s \in O_L$, *if* $t_1 \in L_1$

$$\overline{\delta(t_1)} = \overline{\delta}(\overline{t_1}),$$

and, for each $\circ :: L_1^s \times L_2^s \to L^s \in O_L$, *if* $t_1 \in L_1$ *and* $t_2 \in L_2$,

$$\overline{t_1 \circ t_2} = \overline{t_1} \overline{\circ} \overline{t_2}.$$

The expressions $max(x), min(x)$ *are called* indexicals. *Elements of* L *and* $\overline{L}$ *are called* indexical terms.

The bounded computation domain L *is also a lattice inheriting its ordering from* L^s *and where* $\top_L = \top_{L^s}$ *and* $\bot_L = \bot_{L^s}$. *Thus, if* $t_1, t_2 \in L$, *then* $t_1 \preceq_L t_2$ *if and only if* $t_1, t_2 \in L^s$ *and* $t_1 \preceq_{L^s} t_2$ *or* $t_1 = t_1' \circ t_1'', t_2 = t_2' \circ t_2''$ *and* $t_1' \preceq_L t_2', t_1'' \preceq_L t_2''$ *or* $t_1 = \delta(t_1'), t_2 = \delta(t_2')$ *and* $t_1' \preceq_L t_2'$.

3.4 Interval Domain

Definition 15 *(Interval domain) Let* L *be the bounded computation domain for* L_0. *The* interval domain R_L *over* L_0 *is the lattice* $\langle \overline{L}, L \rangle$. *The* simple interval domain R_L^s *over* L_0 *is the lattice* $\langle \overline{L^s}, L^s \rangle$. *A* range *is an element of* R_L^s. *A* range expression *is an element of* $R_L \backslash R_L^s$.

Therefore, for any $r_1 = \langle \overline{s_1}, t_1 \rangle, r_2 = \langle \overline{s_2}, t_2 \rangle \in R_L$ *where* $s_1, s_2, t_1, t_2 \in L$,

$$\begin{aligned}
r_1 \preceq_{R_L} r_2 &\iff (\overline{s_1} \preceq_{\overline{L}} \overline{s_2}) \text{ and } (t_1 \preceq_L t_2),\\
glb_{R_L}(r_1, r_2) &= \langle glb_{\overline{L}}(\overline{s_1}, \overline{s_2}), glb_L(t_1, t_2) \rangle,\\
lub_{R_L}(r_1, r_2) &= \langle lub_{\overline{L}}(\overline{s_1}, \overline{s_2}), lub_L(t_1, t_2) \rangle,\\
\top_{R_L} &= [\bot_{L_0}, \top_{L_0}],\\
\bot_{R_L} &= (\top_{L_0}, \bot_{L_0}).
\end{aligned}$$

For simplicity, $\langle \overline{s}, t \rangle \in R_L$ is written as $\overline{s}, t$. Thus an element $\langle \overline{a\}}, b\} \rangle$ is written as $\{a, b\}$. As examples of the definitions shown above and considering the real domain we have that $[2.3, 8.9)$ is a range, $[1.4, max(x) + 4.9]$ is a range expression, $[3.0, 4.0) \preceq_{R_L} (1.8, 4.5]$, $glb_{R_L}([3.2, 6.7], (1.8, 4.5]) = [3.2, 4.5]$ and $lub_{R_L}([3.2, 6.7], (1.8, 4.5]) = (1.8, 6.7]$. It is important to note that $\preceq_{R_L}$ simulates the interval inclusion.

Figure 1 graphically illustrates the relationship between any two elements of the interval domain R_L^s over any domain L_0.

A range $r = \overline{s}, t \in R_L^s$ is *inconsistent* if
(1) $s \not\preceq_L t$ (note that this means that $\overline{s}, t$ is inconsistent if $s \not\prec_L t$) or
(2) $s = a)$ and $t = a\}$.
Otherwise r is *consistent*. Note that this means that $\bot_{R_L^s}$ is inconsistent.

Figure 1 illustrates the ordering relation for any simple interval domain R_L^s where the computation domain L_0 contains the elements a, b, c and $a \prec_{L_0} c \prec_{L_0} b$. Note that the part of the interval domain that corresponds to inconsistent

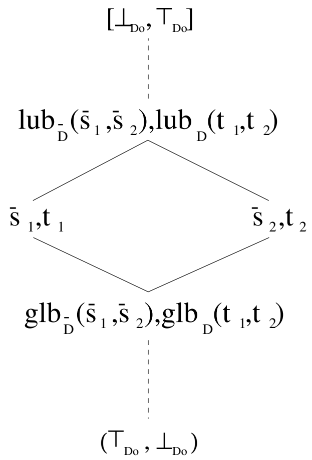

Figure 1: Relationship between any two ranges $\overline{s_1}, t_1$ and $\overline{s_2}, t_2$ in R_L^s

ranges is identified exactly. The zone marked with a square subsumes the inconsistent ranges whereas the nodes marked with a circle correspond also to inconsistent ranges in the finite domain in the cases in which c is either the immediate successor of a or the immediate predecessor b (see [Fernández and Hill 1999, Section 3.8]).

3.5 Interval Constraints

Let R_L denote the interval domain over L_0. An interval constraint assigns an element in R_L to a variable in V_{L_0}.

Definition 16 *(Interval constraint) Suppose $r \in R_L$ and $x \in V_{L_0}$. Then*

$$x \sqsubseteq r$$

is called an interval constraint *for x where x is called the* constrained variable*. The interval constraint $x \sqsubseteq \top_{R_L}$ is called a* type constraint *for x and denoted by $x::'L_0$. If $r \in R_L^s$, then $x \sqsubseteq r$ is said to be* simple*. A simple interval constraint $x \sqsubseteq r$ is* consistent *(resp.* inconsistent*) if r is consistent (resp. inconsistent).*

To illustrate these definitions:

$$y, x::'Integer, \qquad b::'Bool, \qquad w, r, t::'Real, \text{ and } \qquad n::'Natural$$

are examples of type constraints;

$$y \sqsubseteq [1, 4), \qquad b \sqsubseteq [True, True], \text{ and } \qquad n \sqsubseteq [zero, suc(suc(zero))]$$

are examples of simple interval constraints;

$$x \sqsubseteq min(y), max(y) + 3], \qquad t \sqsubseteq (1.21 \overline{*} min(w), 4.56) \text{ and}$$
$$r \sqsubseteq [\overline{sqrt}(min(w)), sqrt(max(w))]$$

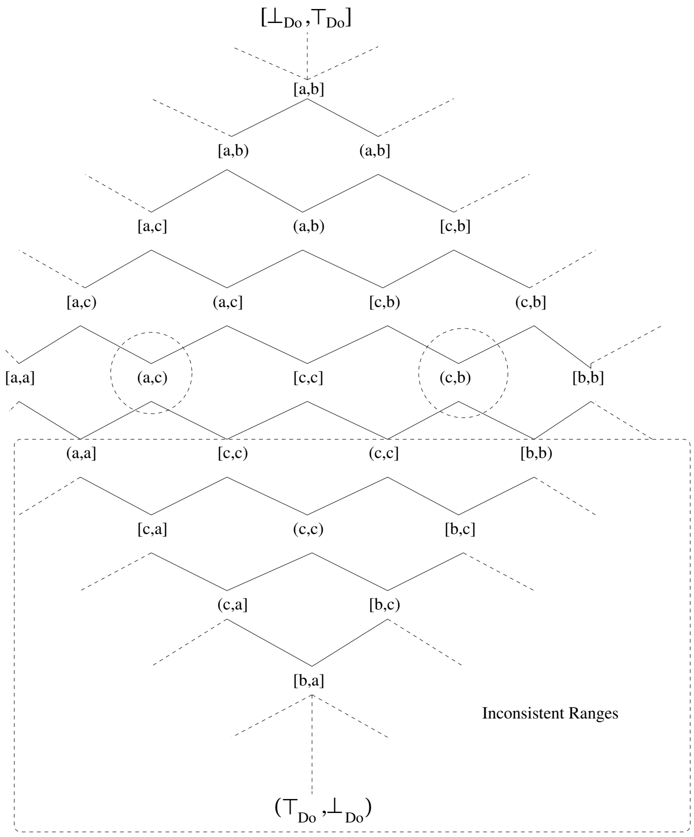

Figure 2: Structure of the simple interval domain R_L^s for L_0 where $a, b, c \in L_0$ and $a \prec_{L_0} c \prec_{L_0} b$

are examples of (non-simple) interval constraints where $+$ and $*$ are, respectively, binary constraint operators for the *Integer* and *Real* domains and *sqrt* is a unary constraint operator for the *Real* domain.

Definition 17 *(A partial ordering on interval constraints) Let C_L^x be the set of all interval constraints over L_0 with constrained variable x. Suppose $c_1 = x \sqsubseteq r_1, c_2 = x \sqsubseteq r_2 \in C_L^x$. Then $c_1 \preceq_{C_L^x} c_2$ if and only if $r_1 \preceq_{R_L} r_2$. As R_L is a lattice, C_L^x is also a lattice. Note that $glb_{C_L^x}(c_1, c_2) = x \sqsubseteq glb_{R_L}(r_1, r_2)$.*

Definition 18 *(Constraint store) If R_L is an interval domain and $X \subseteq V_{L_0}$, then* constraint store *for X is a finite set of interval constraints for the variables in X. A constraint store S for X is in a* stable form *(or is* stable*) if, for each $x \in X$, there is exactly one simple constraint c for x in S. A store is* inconsistent *if it contains at least one inconsistent simple interval constraint. A constraint store S for X is said to be* simple *if it contains only simple constraints.*

Definition 19 *(Evaluating Indexical Terms) Let S be a simple stable constraint store for a set of variables $X \subseteq V_{L_0}$. Then the (overloaded) evaluation functions are defined:*

$$eval_S :: L \to L^s, \qquad eval_S :: \overline{L} \to \overline{L^s}$$

$$\begin{array}{rll}
eval_S(t) &= t & \text{if } t \in L^s \cup \overline{L^s}, \\
eval_S(max(x)) &= t & \text{if } x \sqsubseteq \overline{s}, t \in S, \\
eval_S(max(x)) &= \top_L & \text{if } x \notin X, \\
eval_S(min(x)) &= \overline{s} & \text{if } x \sqsubseteq \overline{s}, t \in S, \\
eval_S(min(x)) &= \bot_L & \text{if } x \notin X, \\
eval_S(\delta(t_1)) &= \delta(eval_S(t_1)), & \\
eval_S(t_1 \circ t_2) &= eval_S(t_1) \circ eval_S(t_2), &
\end{array}$$

An indexical term is a generalisation of the indexical terms provided by CLP finite domain languages [Codognet and Diaz 1996b] and allows for infinite as well as finite ranges.

Remark 1 *(Monotonicity of interval constraints) Note that the law of monotonicity for constraint operators shown in Section 3.2 prevents the definition of non-monotonic interval constraints. For example, to satisfy the monotonicity condition, the binary operator '$-$' has to be declared as $- :: L \times \overline{L} \to L$. Therefore, since $max(y) \notin \overline{L}$, a 'constraint' such as*[3] *$x \sqsubseteq [10, 20] - max(y)$ is not allowed. This expression is non-monotonic since, as the range of y decreases, $max(y)$ decreases. However, as $max(y)$ decreases, $20] - max(y)$ increases, so that the range of x increases. Observe that a range such as $[10, 20] - max(y)$ could not contribute to constraint propagation. Note also that a correct definition of the monotonicity of an operator can avoid non-wanted inconsistencies. For example, a unary operator such as inv, which calculates the inverse of a number, should be declared as $inv :: \overline{L} \to L$. This declaration avoids 'wrong' constraints such as $x \sqsubseteq \overline{inv}(min(y)), inv(max(y))$ that leads to an inconsistency if $\overline{min(y)}$ is not equal to $max(y)$.*

3.6 Execution Model

The execution model is based on a particular intersection of simple interval constraints and on two processes: the stabilisation of a constraint store and the constraint propagation.

3.6.1 Intersection of simple interval constraints

Definition 20 *($\cap_L$) The intersection in the domain L of two simple interval constraints $c_1 = x \sqsubseteq r_1$ and $c_2 = x \sqsubseteq r_2$ for the same* constrained variable *x is defined as follows:*

[3] It is easier to understand this constraint when written as $x \sqsubseteq [10, -max(y) + 20]$.

$$
\begin{aligned}
c_1 \ \cap_L \ c_2 &= x \sqsubseteq glb_{R_L}(r_1, r_2) \\
&= glb_{C_L^x}(c_1, c_2)
\end{aligned}
$$

The following properties of $\cap_L$ are direct consequences of the definition.

Proposition 2 *($\cap_L$ Properties) Suppose $x \in V_{L_0}$ and c_1, c_2, c_3 are consistent constraints defined on the variable x where $c_3 = c_1 \cap_L c_2$. Then $\cap_L$ has the following properties:*

(1) Contractance: $c_3 \preceq c_1$ and $c_3 \preceq c_2$.

(2) Correctness: Only values which can't be part of any feasible solution, are removed. If $c \preceq c_1$ and $c \preceq c_2$, then $c \preceq c_3$.

(3) Commutativity: $(c_1 \ \cap_L \ c_2) = (c_2 \ \cap_L \ c_1)$

(4) Idempotence: The final constraint c_3 has to be computed once: $(c_1 \ \cap_L \ c_3) = c_3$ and $(c_3 \ \cap_L \ c_2) = c_3$.

If S_x is a set of simple constraints for a variable $x \in V_{L_0}$, then we define $\bigcap_L S_x = glb_{C_L^x}(S_x)$. As a result of the contractance property (1) in Proposition 2 we have $\cap_L S_x \preceq c$, for each $c \in S_x$.

Definition 21 *(Stabilised store) Let S be a simple interval constraint store for X and, for each $x \in X$, S_x the set of constraints for x in S. Then, the* stabilised store *S' of S is defined:*

$$S' = \{\cap_L(S_x) \mid x \in X\}$$

Note that, by Definition 17, if $S_x = \emptyset$ then $\cap_L(S_x) = x \sqsubseteq \top_{R_L}$. This ensures that S' is stable for X.

We write $S \mapsto S'$ to express that S' is the stabilised store of S.

Definition 22 *(A partial ordering on simple stable constraint stores) Let S, S' be simple stable stores for X where c_x, c'_x denotes the (simple) constraint for x in S and S', respectively. Then $S \preceq_s S'$ if and only if, for each $x \in X$, $c_x \preceq_{C_L^x} c'_x$. Thus, the simple stable constraint stores for a finite set of variables X forms a lattice with ordering $\preceq_s$, top element the set of type constraints for X and bottom element the set $\{x \sqsubseteq \bot_{R_L} \mid x \in X\}$.*

Proposition 3 *Let S' and S be simple stores for X such that $S \mapsto S'$. Then, $\forall c \in S : \exists c' \in S' \,.\, c' \preceq c$ and, if $S_1 \subseteq S$ is a simple stable store, then $S_1 \preceq S'$.*

Proof 1 *This is a direct consequence of Definition 21.* $\square$

Definition 23 *(Propagation of a constraint) Let X be a finite set of variables, $x \in X$, $c = x \sqsubseteq \overline{s}, t$ and S a simple stable constraint store for X. We say that c* is propagated (using *S*) *to the simple interval constraint $x \sqsubseteq \overline{s_1}, t_1$ if $eval_S(\overline{s}) = \overline{s_1}$ and $eval_S(t) = t_1$. We write $c \rightsquigarrow^S c'$ to express that constraint c has been propagated using S to c'.*

Definition 24 *(Store propagation) Let S be a simple stable constraint store and C a set (possibly empty) of interval constraints. Then C* is propagated to *the simple constraint store C'* using *S, denoted by $C \leadsto^S C'$, if $C' = \{c' \mid \exists c \in C \;.\; c \leadsto^S c'\}$.*

Example. By simplicity, consider the Real domain and let $S = \{x \sqsubseteq [1.0, 15.0], x \sqsubseteq (8.3, 20.4], y \sqsubseteq [1.2, 10.5), y \sqsubseteq (5.6, 15.3)\}$. Then $S \mapsto S'$ where $S' = \{x \sqsubseteq (8.3, 15.0], y \sqsubseteq (5.6, 10.5)\}$ and $\{x \sqsubseteq min(y), 20.4]\} \leadsto^{S'} \{x \sqsubseteq (5.6, 20.4]\}$.

3.7 Operational Schema

In this section we present, as a schema, an outline procedure for the execution model. This schema differs slightly from that shown in [Fernández and Hill 1999]. Let C be a set of interval constraints to be solved and let X be the set of all the variables constrained or indexed in C. Let S be a simple stable constraint store for X that contains just a type constraint for each variable in X.

Definition 25 *(Solution) A* solution for *C is a consistent simple stable constraint store R for X where,*

$$\forall c \in C : \exists c^r \in R \;.\; c^r \preceq c', \textit{where } c \leadsto^R c'$$

We provide here a schema for computing a solution for a constraint store C defined over a set of variables X. Suppose S is the top element of the lattice of simple stable constraint stores for X and S' the bottom element. The *operational schema* is as follows:

(1) *while* S *is consistent and* $S \neq S'$ *do*
(2) $C \leadsto^S \hat{C}$; %% Constraint Propagation
(3) $S' := S$;
(4) $S' \cup \hat{C} \mapsto S$; %% Store stabilisation
(5) *endwhile*

We do not discuss possible efficiency improvements here since the main aim here is to provide the basic methodology, showing how the execution method of CLP(FD) may be generalised for constraint solving on any domain with a lattice structure. If a solution exists, the solution is the simple stable store S.

Theorem 1 *(Correctness) If no inconsistency is detected, a termination execution of the operational schema computes in S a solution for C.*

Proof 2 (Proof of Correctness) *Suppose the procedure terminates after k iterations. Then, for each i where $0 < i \leq k$, let S_i and S'_i denote the constraint stores S and S', respectively, at the start of the of the $i+1$'st iteration of the while loop, and $\hat{C}_i$ the store obtained by the step in line (2) of the i'th iteration. Then, for each i where $0 < i \leq k$, we have from line(2) that $C \leadsto^{S_i} \hat{C}_i$; from line (3) that $S'_i = S_{i-1}$; and, from line (4), $S_{i-1} \cup \hat{C}_i \mapsto S_i$.*

However, when $i = k$, $S_k = S_{k-1}$ so that $S_k \cup \hat{C}_k \mapsto S_k$. Thus, by Proposition 3, $\forall c' \in \hat{C}_k : \exists c^s \in S_k \,.\, c^s \preceq c'$. Moreover, since $i = k$, we have $C \leadsto^{S_k} \hat{C}_k$ and thus, by Definitions 24 and 23, $\forall c \in C : \exists c' \in \hat{C}_k \,.\, c \leadsto^{S_k} c'$. Hence, $\forall c \in C : \exists c^s \in S_k \,.\, c^s \preceq c'$ and $c \leadsto^{S_k} c'$. Therefore, S_k is a solution for C. $\Box$

New constraints, created by the propagation step (line 2), are added to the set of constraints before the stabilisation step (line 4). Thus, with infinite domains, the algorithm may not terminate (note that the constraints can be indefinitely contracted in the stabilisation step). To avoid it, in the following we introduce the notion of *approximation to a solution*.

Approximations Let the domain $\Re\mathcal{N}$ denote the lexicographic product $(\Re, \{0, 1, 2\})$. We introduce the function *precision* $:: R_L^s \to \Re\mathcal{N}$ which must satisfy the following properties:

(i) $precision(r_2) <_{\Re\mathcal{N}} precision(r_1)$ if $r_2 <_{R_L} r_1$ (*Monotonicity*)

(ii) $precision([a, a]) = (0.0, 2)$.

(iii) for some $\delta \in \Re$, $precision(\{_1 a, b\}_2) = (\delta, \{_1 \circ\}_2)$ where the operator $\circ :: \overline{B} \times B \to \{0, 1, 2\}$ is

$$'['\circ']' \stackrel{\text{def}}{=} 2 \qquad '['\circ')' \stackrel{\text{def}}{=} 1$$

$$'('\circ']' \stackrel{\text{def}}{=} 1 \qquad '('\circ')' \stackrel{\text{def}}{=} 0.$$

The actual definition of *precision* depends on the computation domain. For Example:

- On the integer and $\Re$ domains: $precision(\{_1 a, b\}_2) = (b - a, \{_1 \circ\}_2)$.
- On the $\Re^2$ domain:

  if $\{_1 (x_1, y_1), (x_2, y_2)\}_2$ is consistent then

  $$precision(\{_1 (x_1, y_1), (x_2, y_2)\}_2) = (\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \{_1 \circ\}_2).$$

  otherwise

  $$precision(\{_1 (x_1, y_1), (x_2, y_2)\}_2) = (-\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \{_1 \circ\}_2).$$

- On the set domain: $precision(\{_1 s_1, s_2\}_2) = (\#(s_2) - \#(s_1), \{_1 \circ\}_2)$.

Note that in these examples that the binary constraint operators $-$, $+$ and $\backslash$ as well as the unary constraint operators for the $-$, *square root*, *squared* and $\#$ need to be complete functions over the whole computational domain and therefore need to include the lifted bounds.

We overload $precision/1$ and define the precision of a simple interval constraint $c = x \sqsubseteq r$ as $precision(c) = precision(r)$ and the precision of a simple stable store S for X as

$$precision(S) = \sum_{x \in X, x \sqsubseteq r \in S} precision(x \sqsubseteq r)$$

where the sum in $\Re\mathcal{N}$ is defined as $(a_1, a_2)+(b_1, b_2) = (a_1+b_1, a_2+b_2)$. This sum is monotonic in $\Re\mathcal{N}$. To maintain the second component in the natural domain the subtraction is defined as $(a_1, a_2)-(b_1, b_2) = (a_1-b_1, maximum(0, a_2-b_2))$. This definition does not affect to the correct execution of the procedure as it is shown below.

Proposition 4 *Let S and S' be two simple stable constraint stores for X. If $S \preceq_s S'$ then $precision(S) \leq_{\Re\mathcal{N}} precision(S')$.*

Proof 3 *Let $c_x^s = x \sqsubseteq r_1$ and $c_x^{s'} = x \sqsubseteq r_2$ be the simple constraints for x in S and S' respectively. By hypothesis $S \preceq_s S'$, and by Definition 22, this means that $\forall x \in X \;.\; c_x^s \preceq_{C_L^x} c_x^{s'}$. Then, by Definition 17, $r_1 \preceq_{R_L} r_2$ and, by monotonicity condition (i) of the precision/1 function, $precision(r_1) \leq_{\Re\mathcal{N}} precision(r_2)$. As, by the overloaded definition for simple interval constraints shown above, $precision(c_x^s) = precision(r_1)$ and $precision(c_x^{s'}) = precision(r_2)$ we have*

$$\sum_{x \in X, c_x^s \in S} precision(c_x^s) \leq_{\Re\mathcal{N}} \sum_{x \in X, c_x^{s'} \in S'} precision(c_x^{s'})$$

giving the required result that $precision(S) \leq_{\Re\mathcal{N}} precision(S')$. $\square$

By defining a computable[4] bound $\varepsilon \in \Re$, we can check if the precision of ranges for the simple constraints in a constraint store S were reduced by a significant amount in the stabilisation process. If the change is large enough then the propagation procedure continues. Otherwise the set of simple constraints in the store S is considered a "good enough" solution and the procedure terminates. This "solution" is an approximation to the concept of solution shown in Definition 25.

Definition 26 *Approximated solution) Let R be a solution for a constraint store C and $(\delta, \phi) \in \Re\mathcal{N}$. An* approximated solution via $\delta - \phi$ for C *is a simple stable constraint store denoted as $app_{\delta-\phi}(R)$ where, $precision(app_{\delta-\phi}(R)) - precision(R) \leq_{\Re\mathcal{N}} (\delta, \phi)$ and $R \prec_s app_{\delta-\phi}(R)$.*

The function $precision/1$ and bound ε are user or system defined for each computational domain. To use $precision/1$ and ε, the operational schema needs to be extended by replacing line (1) as follows:

$(1^\star)$ *while S is consistent* and $precision(S') - precision(S) >_{\Re\mathcal{N}} (\varepsilon, 0)$ *do*

[4]That is, representable in the machine which is being used - the computation machine.

The case $S = S'$ is equivalent to the case $precision(S') - precision(S) = (0.0, 0)$. Note that the test on the element $(\varepsilon, 0)$ controls if the change in the ranges has been large enough to continue the iterations. Also note that the case $precision(S') - precision(S) = (0.0, \delta)$ with $\delta > 0$ correspond, in the examples of the *precision* predicate that we have defined above, to a situation in which only a propagation or change in the brackets of the ranges were done.

Then any procedure based on the extended operational schema has the following two properties.

1. *Termination.* If $\varepsilon > 0.0$ or the computational domain is finite, then the procedure terminates.

2. *Correctness.* Let S^* be a solution for C. Then, if no inconsistency is detected, any termination execution of the extended operational schema computes in S an approximated solution (or a solution) that does not depend on the order in which constraints are chosen and that depends on the value of ε in the sense that

$$\lim_{\varepsilon \to 0.0} precision(S) = precision(S^*)$$

Proof 4 (Proof of termination property) *Suppose that there are at least $i > 0$ iterations of the while loop and let S_i and S'_i denote the constraint stores S and S', respectively, at the start of the $i+1$'st iteration of the while loop. Then, $S'_0 = \{x \sqsubseteq \bot_{R_L} \mid x \in V\}$ and $S_0 = \{x \sqsubseteq \top_{R_L} \mid x \in V\}$. Also, for $i \geq 1$, it follows from line (3) that $S'_i = S_{i-1}$ and from line (4) that $S_{i-1} \cup \hat{C} \mapsto S_i$. Thus, by Proposition 3, $S_i \preceq_s S_{i-1}$ and, by Proposition 4, $precision(S_i) \leq_{\Re\mathcal{N}} precision(S_{i-1})$.*

If there is an $i+1$'st iteration, $precision(S_{j-1}) >_{\Re\mathcal{N}} precision(S_j) + (\varepsilon, 0)$, for each $j \leq i$, and S_i is consistent. Thus $precision(S_0) >_{\Re\mathcal{N}} precision(S_i) + (i \times \varepsilon, 0)$. However, for some constant K, $precision(S_0) <_{\Re\mathcal{N}} (K \times \varepsilon, 0)$, so that $i < K$ and the procedure terminates after no more than K iterations. $\Box$

Proof 5 (Proof of correctness property) *Suppose the procedure terminates after k iterations. Let S_k and S_{k-1} denote respectively the constraint stores S and S' after k iterations of the while loop and S^* is the approximate solution via $\delta - \phi$ for C.*

By hypothesis, S_k is consistent and $precision(S_{k-1}) - precision(S_k) \leq_{\Re\mathcal{N}} (\varepsilon, 0)$. Using Definition 25, it follows by induction on the number of iterations of the while loop that $S^* \preceq_s S_k$. Thus, for some $(\delta, \phi) \in \Re\mathcal{N}$, $precision(S_k) - precision(S^*) = (\delta, \phi)$. As in the proof of termination, $precision(S_0) >_{\Re\mathcal{N}} (k \times \varepsilon, 0)$,

Without loss of generality, we can assume that $\varepsilon < precision(\top_D)$. Let $\varepsilon_1 = \varepsilon/2$. Then we have $precision(S_0) >_{\Re\mathcal{N}} ((k+1) \times \varepsilon_1, 0)$. Hence, $precision(S_{k-1}) - precision(S_k) >_{\Re\mathcal{N}} (\varepsilon_1, 0)$. Suppose that the procedure terminates using the new bound ε_1 at the k_1'st iteration. Then $k_1 \geq k+1$ because $precision(S_{k_1-1}) - precision(S_{k_1}) <_{\Re\mathcal{N}} (\varepsilon_1, 0)$. Thus, as $precision(S_m) \leq_{\Re\mathcal{N}} precision(S_{m-1})$ for *all* $k \leq m \leq k_1$, *we have* $precision(S_k) - precision(S_{k_1}) >_{\Re\mathcal{N}} (\varepsilon_1, 0)$ *and hence,* $precision(S_{k_1}) - precision(S^*) <_{\Re\mathcal{N}} (\delta - \varepsilon_1, 0)$. *As a consequence,*

$$\lim_{\varepsilon \to 0} precision(S) = precision(S^*).$$

□

4 Chaotic Iteration

In this section, we introduce a schema for iteration algorithms based on similar schemas defined in [Apt 1999a, Apt 1999b]. In fact, as we are concerned with termination over infinite domains, we use chaotic iterations, adapting the method given in [Apt 1999a] so that we can apply it to our interval constraints.

We need some extra concepts taken from [Apt 1999a].

Definition 27 *Suppose that* $(D, \preceq)$ *is a* Υ*-po and let* F *be a set of functions on* D. *By an* iteration of F associated with a run r on F and starting with d *we mean a sequence of values* $\langle d_0, d_1, \dots \rangle$ *such that* $d_0 = d$ *and, for each* $j \geq 1$, $d_j = r[j](d_{j-1})$. *If the starting element is not explicit, then it is assumed that it is* $\perp_D$. *The iteration is* chaotic *if the associated run is fair. We define the least upper bound of* F *to be the function* $h \colon D \to D$ *which is defined for each* $d \in D$:

$$h(d) \stackrel{\text{def}}{=} lub\{f(d) \mid f \in F\}.$$

The following result is proved in [Apt 1999a, Theorem 7].

Theorem 2 *Suppose that* $(D, \preceq)$ *is a* Υ*-po and let* F *be a set of extensive and monotonic functions on* D. *Then the limit of every chaotic iteration exists and coincides with*

$$lub\{h^j(\perp_D) \mid j \geq 0\}$$

where h *is the least upper bound for* F.

The next result is also proven in [Apt 1999a, Corollary 8].

Theorem 3 *Suppose that* $(D, \preceq)$ *is a* Υ*-po and let* F *be a set of extensive and monotonic functions on* D *and* h *the least upper bound for* F. *If some chaotic iteration of* F *eventually stabilises, then every chaotic iteration of* F *eventually stabilises at the least fixed point of* h.

4.1 Chaotic Iteration for Compound Domains

The basic iteration schemas are given in [Apt 1999a, Apt 1999b] and we do not repeat them here. This is because, we will need to have some knowledge concerning the structure of the domain so as to apply the schema to our constraint solver. In particular, we need a copy of the interval domain for each variable of interest. To accommodate this, we specialise the algorithm for a domain that is a Cartesian product. Therefore we assume that there are n $\curlyvee$-po domains $(D_i, \preceq_i)$ for $i = 1, \ldots, n$ and that $(D, \preceq)$ is the Cartesian product $(D_1, \preceq_1) \times \cdots \times (D_n, \preceq_n)$. It can be easily shown that D is also a $\curlyvee$-po. We will use the following terminology.

Definition 28 *Given a function f on D and some $j \in \{1, \ldots, n\}$, we say that f* affects *j if, for some $d \in D$, $f(d)[j] \neq d[j]$. We say that f* depends on *j if, for some $d, d' \in D$, $f(d) \neq f(d')$ and, for all $i \neq j$, $d[i] = d'[i]$.*

Suppose that F is a set of functions that are extensive and monotonic self-maps on D. The following interation schema for compound domains, called the CI algorithm, is used to compute the limit of the iterations of F.

$d := (\bot_{D_1}, \ldots, \bot_{D_n})$;
$G := F$;
while $G \neq \emptyset$ **do**
    choose $g \in G$;
    $G := G \setminus \{g\}$;
    $d' := g(d)$;
    if $d \neq d'$ **then**
        $G := G \cup \{f \in F \mid \exists i \,.\, f$ depends on i and g affects $i\}$;
        $d := d'$
    endif
endwhile

Suppose that h is the least fixed point function for F. Then we have the following two results.

Theorem 4 *Every terminating execution of the* CI *algorithm computes in d the least fixed point of the function h.*

Theorem 5 *If all $(D_i, \preceq_i)$, $i \in \{1, \ldots, n\}$, satisfy the finite chain property, then every execution of the* CI *algorithm terminates.*

Proofs can be found in [Apt 1999a, Theorem 11].

5 Chaotic Constraint Solving

In this section we relate the interval constraints and the operational schema defined in Section 3 to the CI algorithm defined above.

5.1 A modified iteration chaotic algorithm

Theorem 4 requires the domain (and hence, its components) to satisfy the finite chain property. However, for our particular application, we need to allow for any domain that is a lattice and adapt the CI algorithm to use an approximation function to ensure termination. The adapted algorithm then only tests for equality with respect to the approximate values.

Suppose that there is an *approximation function* $\alpha_D : D \to \Re\mathcal{N}$, where $\Re\mathcal{N}$ is the lexicographic product of $\Re$ and $\mathcal{N}$, such that, if $d \prec d'$ then $\alpha_D(d') <_{\Re\mathcal{N}} \alpha_D(d)$.

We define a bound $\varepsilon \in \Re_+$ to 'measure' how much constraint propagation has been done and replace the equality test in line 7 by the following.

$$(7^\star) \qquad \textbf{if } (\mathrm{trunc}(\delta/\varepsilon), \phi) >_{\Re\mathcal{N}} (\mathrm{trunc}(\delta'/\varepsilon), \phi') \textbf{ then}$$

where $\alpha_D(d) = (\delta, \phi)$, $\alpha_D(d') = (\delta', \phi')$ and trunc $:: \Re \to \mathcal{N}$ returns the integer part of its real argument. This procedure we call the *modified chaotic iteration algorithm (*CI^α*)*. We have the following results.

Theorem 6 *If the least fixed point of the function h on D is d^* and a terminating execution of the* CI^α *algorithm, using bound ε, computes the value d_ε, then*

$$\lim_{\varepsilon \to 0.0} \alpha_D(d_\varepsilon) = \alpha_D(d^*)$$

Proof 6 *In the following,* (δ_d, ϕ_d),$(\delta_{f(d)}, \phi_{f(d)})$, $(\delta_{d_\varepsilon}, \phi_{d_\varepsilon})$ *and* $(\delta_{h(d_\varepsilon)}, \phi_{h(d_\varepsilon)})$ *denote respectively* $\alpha_D(d)$, $\alpha_D(f(d))$, $\alpha_D(d_\varepsilon)$ *and* $\alpha_D(h(d_\varepsilon))$.

Note that the formula

$$I := (\forall f \in F - G \, . \, f(d) = d \vee (\mathrm{trunc}(\delta_d/\varepsilon), \phi_d) \geq_{\Re\mathcal{N}} (\mathrm{trunc}(\delta_{f(d)}/\varepsilon), \phi_{f(d)})$$

is an invariant of the **while loop** *of the* CI^α *algorithm. However, upon termination*

$$(G = \emptyset)$$

and

$$(\forall f \in F \, . \, f(d_\varepsilon) = d_\varepsilon \vee (\mathrm{trunc}(\delta_{d_\varepsilon}/\varepsilon), \phi_{d_\varepsilon}) = (\mathrm{trunc}(\delta_{f(d_\varepsilon)}/\varepsilon), \phi_{f(d_\varepsilon)})$$

Suppose $h(d_\varepsilon) = d_\varepsilon$ holds. Then the sequence of values computed in d by the CI^α *algorithm eventually stabilises at d_ε and, by Theorem 3, d_ε is the least fixpoint of h. This means that $d_\varepsilon = d^*$ and hence $\alpha_D(d_\varepsilon) = \alpha_D(d^*)$.*

Suppose now that $h(d_\varepsilon) < d_\varepsilon$. Then,

$$(\mathrm{trunc}(\delta_{d_\varepsilon}/\varepsilon), \phi_{d_\varepsilon}) = (\mathrm{trunc}(\delta_{h(d_\varepsilon)}/\varepsilon), \phi_{h(d_\varepsilon)})$$

and hence, $\phi_{d_\varepsilon} = \phi_{h(d_\varepsilon)}$,

$$\alpha_D(d_\varepsilon) - \alpha_D(h(d_\varepsilon)) <_{\Re\mathcal{N}} (\varepsilon, 0).$$

Thus, $\lim_{\varepsilon \to 0.0} \alpha_D(d_\varepsilon) - \alpha_D(h(d_\varepsilon)) = (0.0, 0)$ *and hence, as* $\varepsilon \to 0.0$, $d_\varepsilon \to h(d_\varepsilon)$. *However, if* $d = h(d)$, *then* $d = d^*$ *and* $\alpha_D(d) = \alpha_D(d^*)$. *It therefore follows that*

$$\lim_{\varepsilon \to 0.0} \alpha_D(d) = \alpha_D(d^*).$$

□

The second result is a modified version of the second part of Theorem 11 in [Apt 1999a]

Theorem 7 *Every execution of the* CI^α *algorithm terminates.*

Proof 7 *We follow a similar reasoning to the proof for Theorem 11 in [Apt 1999a].*

Each function $g \in F$ *is extensive so that in each iteration of the* **while** *loop* d *never decreases on* D *and thus*

$$(\mathrm{trunc}(\delta/\varepsilon), \phi)$$

strictly decreases in $\mathcal{N} \times \mathcal{N}$ *(note that* $\alpha_D(d) = (\delta, \phi)$*). The domain* $\mathcal{N}$ *is well-founded and consequently there is always termination.* □

5.2 From interval constraints to chaotic iterations

To apply the chaotic iteration theory and algorithm of Section 4 to the interval domain and constraints defined in Section 3, we need to relate the compound domain D to the interval domains and the extensive and monotonic functions on D to the interval constraints.

In this subsection, L_0 denotes a lattice which is the computation domain and, for some $n \in \mathcal{N}_0$, there is a given set of n variables of interest $V = \{v_1, \ldots, v_n\}$. For each $i = 1, \ldots, n$, L_i denotes a bounded computation domain for L_0 and R_i a simple interval domain over L_0.

Definition 29 *The* iteration domain $(\mathcal{I}, \preceq_\mathcal{I})$ *is the direct product* $R_1 \times \ldots \times R_n$.

Let $S_\mathcal{I}^V = \{c_1, \ldots, c_n\}$ *be a simple stable constraint store for* V *and, for each* $i = 1, \ldots, n$, *let* $c_i = v_i \sqsubseteq r_i$. *Suppose, for some* $i = 1, \ldots, n$, $c = v_i \sqsubseteq r$ *is an interval constraint where* $c \leadsto^{S_\mathcal{I}^V} c'$ *and* $glb(c_i, c') = v_i \sqsubseteq r'_i$. *Then, the* iteration function for c *is* $f_c \colon \mathcal{I} \to \mathcal{I}$ *where*

$$f_c(r_1, \ldots, r_i, \ldots, r_n) = (r_1, \ldots, r'_i, \ldots, r_n).$$

If C *is a constraint store for* V, *we define the* iteration function for C *to be*

$$F_C = \{f_c \mid c \in C\}.$$

The theory of chaotic iterations is based on increasing sequences and the constraints are defined on decreasing ranges. So in the chaotic iteration algorithm, the domain D is replaced, not by $\mathcal{I}$ but by its dual $\overline{\mathcal{I}}$. Then F_C is a set of extensive and monotonic functions on $\overline{\mathcal{I}}$. Therefore the CI$^{\alpha}$algorithm can be used directly.

To apply the CI$^{\alpha}$algorithm, we also need the approximation function for the domain $\overline{\mathcal{I}}$. First we show how the precision function *precision*/1 with bound ε can be reformulated as a function $\alpha_{\overline{\mathcal{I}}}$ from the domain $\mathcal{I}$ to the domain $\Re\mathcal{N}$. This translation is almost direct. For each $i = 1, \ldots, n$, and for each $r_i \in R_i$. we have

$$\alpha_{\mathcal{I}}(\top_{\mathcal{I}}) \stackrel{\text{def}}{=} precision(\top_s)$$
$$\alpha_{\mathcal{I}}((r_1, \ldots, r_n)) \stackrel{\text{def}}{=} \sum_{i=1}^{n} precision(r_i).$$

where $\top_s$ is the top element of the lattice of simple stable stores for V, that is, $\top_s = \{v_1 \sqsubseteq \top_{R_1}, \ldots, v_n \sqsubseteq \top_{R_n}\}$.

Since *precision* and $\sum$ are monotonic, $\alpha_{\mathcal{I}}$ is monotonic as well and hence $\alpha_{\overline{\mathcal{I}}}$ is anti-monotonic. Thus, $\alpha_{\overline{\mathcal{I}}}$ can be used as the approximation function for the CI$^{\alpha}$algorithm.

Therefore with the domain $\overline{\mathcal{I}}$, the set of iteration functions F_C for C and the approximation function $\alpha_{\overline{\mathcal{I}}}$, the CI$^{\alpha}$algorithm will terminate with a solution to the constraint store C if one exists and will fail if C is inconsistent.

5.3 Example 1

Let R_0 be the interval domain for the real domain. Consider two constrained variables x and y and let $c_1 = x \sqsubseteq [0.0, 9.4], c_2 = y \sqsubseteq (1.3, 10.2]$ and $c_3 = x \sqsubseteq min(y), max(y)$ be interval constraints defined on the interval domain R_0. Let $R = R_0 \times R_0$ be a complete lattice and hence $(R, \succeq_R)$ is a Υ-po. As shown in Section 5.2 we can define a set of functions $F_C = \{f_{c_1}, f_{c_2}, f_{c_3},\}$ where f_{c_1}, f_{c_2} and f_{c_3}, are defined as follows:

$$f_{c_1}((r_x, r_y)) = (glb_{R_0}(r_x, [0.0, 9.4]), r_y)$$
$$f_{c_2}((r_x, r_y)) = (r_x, glb_{R_0}(r_y, (1.3, 10.2]))$$
$$f_{c_3}((r_x, r_y)) = (glb_{R_0}(r_x, r_y), r_y)$$

It is straightforward to check that any chaotic iteration of F_C associated with the run $\{1, 2, 3\}$ on F_C leads to the same solution $((1.3, 9.4], (1.3, 10.2])$ (note that in this case the value of ε is irrelevant) . For example:

$$f_{c_1}([-\infty, \infty], [-\infty, \infty]) = ([0.0, 9.4], [-\infty, \infty])$$
$$f_{c_2}([0.0, 9.4], [-\infty, \infty]) = ([0.0, 9.4], (1.3, 10.2])$$
$$f_{c_3}([0.0, 9.4], (1.3, 10.2]) = ((1.3, 9.4], (1.3, 10.2])$$

Note also that the starting element $\perp_R$ is $([-\infty, \infty], [-\infty, \infty])$.

5.4 Example 2

Let R_0 be the interval domain for integers and let $c_1 = x \sqsubseteq [1,4], c_2 = y \sqsubseteq [1,4], c_3 = z \sqsubseteq [0,10]$ and $c_4 = z \sqsubseteq min(x)\overline{+}min(y), max(x)+max(y)$ be interval constraints defined on the domain R_0. Analogously as shown in preceding subsection we can define a set of functions $F_C = \{f_{c_1}, f_{c_2}, f_{c_3}, f_{c_4}\}$ where each function is defined as follows:

$$
\begin{aligned}
f_{c_1}((r_x, r_y, r_z)) &= (glb_{R_0}(r_x, [1,4]), r_y, r_z)\\
f_{c_2}((r_x, r_y, r_z)) &= (r_x, glb_{R_0}(r_y, [1,4]), r_z)\\
f_{c_3}((r_x, r_y, r_z)) &= (r_x, r_y, glb_{R_0}(r_z, [0,10]))\\
f_{c_4}((r_x, r_y, r_z)) &= (r_x, r_y, glb_{R_0}(r_x +' r_y, r_z))
\end{aligned}
$$

where $+'$: $R_0 \times R_0 \to R_0$ is defined as:

$$\overline{s_1}, t_1 \ +' \ \overline{s_2}, t_2 \ = \overline{s_1}\overline{+}\overline{s_2}, t_1 + t_2$$

It is straightforward to check that any chaotic iteration of F_C associated with the run $\{1,2,3,4\}$ on F_C leads to the same solution $([1,4],[1,4],[2,8])$ (note that in this case the value of ε is irrelevant). For example:

$$
\begin{aligned}
f_{c_1}([0,\infty],[0,\infty],[0,\infty]) &= ([1,4],[0,\infty],[0,\infty])\\
f_{c_2}([1,4],[0,\infty],[0,\infty]) &= ([1,4],[1,4],[0,\infty])\\
f_{c_3}([1,4],[1,4],[0,\infty]) &= ([1,4],[1,4],[0,10])\\
f_{c_4}([1,4],[1,4],[0,10]) &= ([1,4],[1,4],[2,8])
\end{aligned}
$$

Note that the starting element $\perp_R$ is $([0,\infty],[0,\infty],[0,\infty])$.

5.5 Comparing Constraint Propagation Algorithms

To compare the CI$^\alpha$algorithm with our operational schema we first reformulate the operational schema in a form which separates the propagation for a set of constraints into one that only deals with single constraints. Note though that in practice, propagating one constraint at a time is expected to be less efficient.

Also, to compare both algorithms we adapt our operational schema to the chaotic iteration approach by writing it in a similar way to the CI$^\alpha$algorithm and maintain a similar notation. To distinguish variables from our schema operational and the CI$^\alpha$algorithm we use a subscript c to denote 'referent to our constraint algorithm'. For example, the initial set of interval constraints to be solved in our operational schema will be denoted by F_c whereas in the CI$^\alpha$algorithm F denotes the original set of monotonic and extensive functions used in the chaotic iteration process; similarly in the modified operational schema (see below) d_c denotes a simple and stable constraint store and g_c and f_c denotes interval constraints whereas in the CI$^\alpha$algorithm d denotes an element of D, and g and f functions in F.

Definition 30 *Let $c = x \sqsubseteq r$ be an interval constraint defined on the variable x. Then, we say that c* affects *x. Also, we say that c* depends on *y if $y \in \mathcal{V}(r)$ where $\mathcal{V}(r)$ denotes the set of free variables in r.*

Let $V = \{x_1, \ldots, x_n\}$ be a set of variables of interest and F_c the set of interval constraints, defined over V, to be solved. We assume that each variable $x_i \in V$ is constrained in the interval domain R_i. If d_c is a simple stable constraint store for V, we denote $d_c[i]$ the simple interval constraint $x_i \sqsubseteq r_i$ in d_c where $x_i \in V$ and $i \in \{1, \ldots, n\}$.

The *revised operational schema* can be expressed in a chaotic form as follows:

(1) $d_c := \top_s$;
(2) $G_c := F_c$;
(3) **while** $G_c \neq \emptyset$ and d_c *is consistent* **do**
(4) choose $g_c \in G_c$; %%assume g_c is a simple constraint affecting x_k
(5) $G_c := G_c \backslash \{g_c\}$;
(6) $g_c \rightsquigarrow^{d_c} \hat{g}_c$;
(7) **if** $precision(d_c[k]) - precision(\hat{g}_c) > (\varepsilon, 0)$ **then**
(8) $G_c := G_c \cup \{f_c \in F_c \mid f_c$ depends on x_k and g_c affects to $x_k\}$;
(9) $d_c \cup \{\hat{g}_c\} \mapsto d_c$
(10) **endif**
(11) **endwhile**

where $\top_s$ is the top element of the lattice of simple stable stores for V, that is, $\top_s = \{x_1 \sqsubseteq \top_{R_1}, \ldots, x_n \sqsubseteq \top_{R_n}\}$. On termination, the store d_c will contain an (approximated) solution.

Note this algorithm and the CI$^\alpha$algorithm are very similar and that several analogies are noted. In the following we will relate both algorithms.

- *Constraint Propagation vs. Function evaluation*

  Let $c = x \sqsubseteq \overline{s}, t$. By abusing the notation we write $eval_s(c)$ to denote $x \sqsubseteq eval_s(\overline{s}), eval_s(t)$. We observe in the procedure shown above that $g_c \rightsquigarrow^{d_c} \hat{g}_c$. Assuming that g_c affects x_i, by Definition 23 $\hat{g}_c$ also affects x_i and is equal to $eval_{d_c}(g_c)$.

  Then, it is observed that the step of propagation $g_c \rightsquigarrow^{d_c} \hat{g}_c$ evaluates the constraint g_c (i.e. its range) on the store d_c. This means that there is a clear analogy with the step $d' := g(d)$ in the CI$^\alpha$algorithm (line 6) that evaluates the function g on the element d.

- *Constraint Stabilisation vs Narrowing*

  Certain analogies can also be observed between the stabilisation step (line 9) in our revised operational schema and the assignment step (line 9; also called 'narrowing' step since $d \preceq_D d'$ after its execution) in the CI$^\alpha$algorithm.

In the operational schema, $d_c^* \cup \{\hat{g}_c\} \mapsto d_c$ (stabilisation step) where d_c^* corresponds to the store d_c before executing this sentence. By Proposition 3, $d_c \preceq_s d_c^*$. Analogously in the CI$^\alpha$algorithm, if d^* denotes the element $d \in D$ before executing the loop iteration, the narrowing step $d := d'$ (line 9) in the CI$^\alpha$algorithm is executed whenever $d \neq d'$ (line 7). Then, before executing the assignment of line 9, d' is evaluated as $g(d^*)$ (line 6) and then $d^* \sqsubseteq_D d'$ as consequence of the fact that g is extensive. After executing the assignment of line 9, $d^* \preceq_D d$.

Thus there is an (reverse) analogy between the store ordering in our operational schema and the ordering in the computational domain D in the CI$^\alpha$algorithm.

More similarities can be found between our interval lattice based constraint solving framework and the Apt's chaotic iteration setting. For example, whereas Apt requires the original set of functions to be monotonic on D, we impose the monotonicity of constraints (which in our setting are the equivalent to chaotic functions; see Subsection 5.2) by means of imposing the monotonicity of operators (see Remark 1).

There are two main differences between the CI$^\alpha$algorithm and this operational schema. One is that the simple stable constraint store d_c is only updated if there is a change to the approximate value of the simple constraint for x_i. The other is that we have a test for inconsistency of the constraint store. Clearly neither of these differences would affect the termination and correctness results of the CI$^\alpha$algorithm.

6 Conclusion

This paper was motivated by our desire to see how the indexical approach to constraints and constraint solving could be expressed in the framework presented in [Apt 1999a].

We have presented a modified version of the chaotic iteration algorithm based on one presented in [Apt 1999a]. This modified algorithm relaxes the requirement that domains satisfy the finite chain property and can be used for any domain that is a lattice. Furthermore, we have presented a new formalisation for the indexical approach used in CLP(FD), generalised for arbitrary lattices, and provided an operational schema for solving constraints using this formalisation. The chaotic iteration approach of [Apt 1999a] is used to derive a constraint propagation algorithm from the interval-lattice based constraint solving framework presented in [Fernández and Hill 1999]. By establishing the connection between these two theories, we have been able to deduce correctness and termination results for algorithms based on the operational schema directly from the results about soundness and termination for the chaotic iteration algorithm.

The main similarities between both approaches have been shown by identifying the common elements in the algorithms described in [Apt 1999a] and

[Fernández and Hill 1999]. This shows that the process of constraint propagation in the operational schema can be viewed as a process of function evaluation in the chaotic iteration algorithm.

There are two main differences between the approach in [Apt 1999a] and its application to the interval constraints procedure presented here. In [Apt 1999a], a formalism for solving a constraint satisfaction problem is presented. It is then shown how the chaotic iteration algorithm can be applied to such a formalism. However, we have used the chaotic iteration algorithm directly rather than via this formalism. The reason was that our interval constraints do not correspond to the constraints as defined there. These represented sets of possible solutions whereas our interval constraints embody more; the intended dependence between the variables for the constraint propagation. Thus, our interval constraints correspond to functions that are used in the chaotic iteration algorithm. The second difference concerns the chaotic iteration algorithm itself. Since our domains (such as the reals) do not necessarily possess the finite chain property, to ensure termination we have changed the main test in the algorithm to a test where domain elements are approximated.

There remain a number of related issues that need to be explored. For example, in [Apt 1999b] it is studied how the chaotic iteration algorithms can be improved when the set of functions, already required to be monotonic and extensive, satisfy further algebraic properties such as commutativity or semi-commutativity. We have shown in this paper how the monotonicity of constraints as well constraint operators affect our framework. In this sense it would be useful to study (and detect) additional properties of the interval constraints as well the constraint operators and see if these affect the execution of the operational schema. We require the computation domain to be a lattice. Work should to be done to see how this requirement could be relaxed to semi-lattices or other partially ordered structures.

An alternative generic framework for constraint satisfaction is presented in [Bistarelli *et al.* 1997]. This is designed for constraint solving on domains that have defined with a level of preference (i.e. cost or degree). However, unlike the proposal in [Fernández and Hill 1999], the computational domain is required to be finite. Further work is needed to see if we could adopt ideas from this work and extend our approach to include such preferential levels.

References

[Apt 1997] Apt K.R., From Chaotic Iteration to Constraint Propagation. In Proc. of the *24th International Colloquium on Automata, Languages and Programming (ICALP'97)* (invited lecture), LNCS 1256, pp:36-55, 1997.

[Apt 1999a] Apt K.R.,The essence of constraint propagation. In *Theoretical Computer Science*, 221(1–2):179–210, 1999.

[Apt 1999b] Apt K.R.,The rough guide to constraint propagation. In Proceedings of *5th International Conference on Principles and Practice of Constraint Programming (CP'99)*, J.Jaffar editor, LNCS 1713, Springer-Verlag, pp:1-23, invited lecturer, 1999.

[Bistarelli *et al.* 1997] Bistarelli S., Montanari U. and Rossi F., Semiring-based constraint satisfaction and optimization. In *Journal of the ACM*, 44(2):201-236, 1997.

[Carlsson *et al.*, 1994] Carlsson M., Ottosson G. and Carlson B., An Open-Ended Finite Domain Constraint Solver. In Proceedings of the *9th International Symposium on Programming Languages: Implementations, Logics and Programs (PLILP'97)*, LNCS 1292, Springer Verlag, pp:191-206, 1994.

[Codognet and Diaz 1996a] Codognet P. and Diaz D., Local Propagation Methods for Solving Boolean Constraints in Constraint Logic Programming. In *The Journal of Automated Reasoning*, 17(1), 1996.

[Codognet and Diaz 1996b] Codognet P. and Diaz D., Compiling constraints in *clp(FD)*. In *The Journal of Logic Programming*, 27:185-226, 1996.

[Davey and Priestley 1990] Davey B.A. and Priestley H.A., Introduction to Lattices and Order. Cambridge University Press, England, 1990.

[Fernández and Hill 1999] Fernández A.J. and Hill P.M., An interval lattice-based constraint solving framework for lattices. In Proceedings of *4th International Symposium on Functional and Logic Programming (FLOPS'99)*, editors Aart Middeldorp and Taisuke Sato, LNCS 1722, Springer-Verlag, pp:194-208,Tsukuba, Japan, November, 1999.

[Gervet 1997] Gervet C., Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. In *Constraints: An International Journal*, 1(3):191-244, 1997.

[Jaffar and Maher 1994] Jaffar J. and Maher M.J., Constraint Logic Programming: A survey. In *Journal of Logic Programming*, 19/20:503-581, 1994.

[Marriot and Stuckey 1998] Marriot K. and Stuckey J., Programming with constraints. MIT Press, 1998.

[Refalo and Van Hentenryck 1996] Refalo P. and Van Hentenryck P., CLP($\Re_{lin}$) Revised. In Proceedings of Joint International Conference and Symposium on Logic Programming, JICSLP'96, editor M. Maher, pps:22-36, 1996.

[Sidebottom and Havens 1992] Sidebottom G. and Havens, W.S., Hierarchical arc consistency for disjoint real intervals in Constraint logic programming. In Computational Intelligence 8(4), 1992.