# A model of constraint solvers by chaotic iteration adapted to value withdrawal explanations

LIFO, EMN

Gérard Ferrand, Willy Lesaint, Alexandre Tessier

public, rapport de recherche

D1.1.1

**Abstract**

The aim of this report is to provide the theoretical foundations of domain reduction. The model is well suited to the solvers on finite domains which are used on the respective platforms of each partner of the project: GNU-Prolog (INRIA), CHIP (COSYTEC) and PaLM (EMN). A computation is formalized by a chaotic iteration of operators and the result is described as a closure. The model is well suited to the definition of traces and explanations which will be useful for the debugging of constraint programs. This report only deals with the reduction stage. It will be extended to the labeling and the host language in next reports.

## 1 Introduction

Constraint Logic Programming (CLP) [13] can be viewed as the reunion of two programming paradigms : logic programming and constraint programming. Declarative debugging of constraints logic programs has been treated in previous works and tools have been produced for this aim during the DiSCiPl (Debugging Systems for Constraint Programming) ESPRIT Project [9, 10]. But these works deal with the clausal aspects of CLP. This report focus on the constraint level alone. The tools used at this level strongly depend on the constraint domain and the way to solve constraints. Here we are interested in a wide field of applications of constraint programming: *finite domains* and *propagation*.

The aim of constraint programming is to solve Constraint Satisfaction Problems (CSP) [16], that is to provide an instantiation of the variables which is correct with respect to the constraints.

The solver goes toward the solutions combining two different methods. The first one (labeling) consists in partitioning the domains until to obtain singletons and, testing them. The second one (domain reduction) reduces the domains eliminating some values which cannot be correct according to the constraints. Labeling provides exact solutions whereas domain reduction simply approximates them. In general, the labeling alone is very expensive and a good combination of the two methods is more efficient. In this paper labeling is not really treated. We consider only one branch of the search tree: the labeling part is seen as additional constraint to the CSP. In future work, we plan to extend our framework in order to fully take labeling and the whole search tree (instead of a single branch) into account.

This kind of computation is not easy to debug because CSP are not algorithmic programs [14]. The constraints are re-invoked according to the domain reductions until a fix-point is reached. But the order of invocation is not known a priori and strongly depends on the strategy used by the solver.

The main contribution of this report is to formalize the domain reduction in order to provide a notion of explanation for the basic event which is "the withdrawal of a value from a domain". This notion of explanation is essential for the debugging of CSP programs. Indeed, the disappearance of a value from a domain may be a symptom of an error in the program. But the error is not always where the value has disappeared and an analysis of the explanation of the value withdrawal is necessary to locate the error. [6] provides a tool to find symptoms, this paper provides a tool which could be used to find errors from symptoms. Explanations are a tool to help debugging: we extract from a (wide) computation a structured part (a proof tree or an explanation tree) which will be analyzed more efficiently.

We are inspired by a constraint programming language over finite domains, GNU-Prolog [7], because its glass-box approach allows a good understanding of the links between the constraints and the rules. But our model is sufficiently general to take the solver of each partner into account, that is GNU-Prolog (INRIA), CHIP (COSYTEC) and PaLM (EMN).

We provide explanations in the general case of hyper-arc consistency. Obviously, this definition of explanations is correct for weaker consistencies usually used in the implemented solvers. To be easily understandable, we provide examples in the arc-consistency case.

An explanation is a subset of operators used during the computation and which are responsible for the removal of a value from a domain. Several works shown that detailed analysis of explanations have a lot of applications [11, 12]. But these applications of explanations are outside the scope of this report (see [12]). Here, our definitions of explanations are motivated by applications to debugging, in particular to error diagnosis.

An aspect of the debugging of constraint programming is to understand why we have a failure (i.e. we do not obtain any solution); this problem has been raised in [2]. This case appears when a domain becomes empty, that is no value of the domain belongs to a solution. So, an explanation of why these values have disappeared provides an explanation of the failure.

Another aspect is error diagnosis. Let us assume an expected semantics for the CSP. Consider we are waiting for a solution containing a certain value for a variable, but this value does not appear in the final domain. An explanation of the value withdrawal help us to find what is wrong in our program. It is important to note that the error is not always the constraint responsible of the value withdrawal. Another constraint may have made a wrong reduction of another domain which has finally produced the withdrawal of the value. The explanation is a structured object in which this information may be founded.

The report is organized as follows. Section 2 gives some notations and basic definitions for Constraint Satisfaction Problems. Section 3 describes a model for domain reduction based on reduction operators and chaotic iteration. Section 4 associates deduction rules to this model. Section 5 uses deduction rules in order to build explanations. Next section is a conclusion.

## 2 Preliminaries

We provide the classical definition of a constraint satisfaction problem as in [16]. The notations used are natural to express basic notions of constraints involving only some subset of the set of all variables.

Here we only consider the framework of domain reduction as in [5, 7, 17, 18]. More general framework is described in [4, 15].

A *Constraint Satisfaction Problem* (CSP) is made of two parts, the syntactic part:

- a finite set of variable symbols (variables in short) $V$;
- a finite set of constraint symbols (constraints in short) $C$;
- a function $var : C \to \mathcal{P}(V)$, which associates with each constraint symbol the set of variables of the constraint;

and a semantic part.

For the semantic part, we need some preliminaries. We are going to consider various *families* $f = (f_i)_{i \in I}$. Such a family is identified with the *function* $i \mapsto f_i$, itself identified with the *set* $\{(i, f_i) \mid i \in I\}$.

We consider a family $(D_x)_{x \in V}$ where each $D_x$ is a finite non empty set called the *domain of the variable* $x$ (domain of $x$ in short). In order to have simple and uniform definitions of monotonic operators on a power-set, we use a set which is similar to an Herbrand base in logic programming. We define the *global domain* by $\mathbb{G} = \bigcup_{x \in V}(\{x\} \times D_x)$. We consider subsets $d$ of $\mathbb{G}$, i.e. $d \subseteq \mathbb{G}$. We denote by $d|_W$ the restriction of a set $d \subseteq \mathbb{G}$ to a set of variables $W \subseteq V$, that is $d|_W = \{(x, e) \in d \mid x \in W\}$.

We use the same notations for the tuples. A *global tuple* $t$ is a particular $d$ such that each variable appears only once: $t \subseteq \mathbb{G}$ and $\forall x \in V$, $t|_{\{x\}} = \{(x, e)\}$. A *tuple* $t$ *on* $W \subseteq V$, is defined by $t \subseteq \mathbb{G}|_W$ and $\forall x \in W$, $t|_{\{x\}} = \{(x, e)\}$. So, a global tuple is a tuple on $V$.

Then, the semantic part is defined by:

- the family $(D_x)_{x \in V}$,
- the family $(T_c)_{c \in C}$ which is defined by: for each $c \in C$, $T_c$ is a set of tuple on $var(c)$ i.e. each $t \in T_c$ is identified with a set $\{(x, e) \mid x \in var(c)\}$.

A global tuple $t$ is a *solution* of the CSP if $\forall c \in C, t|_{var(c)} \in T_c$.

For any $d \subseteq \mathbb{G}$, we need another notation: for $x \in V$, we define $d_x = \{e \in D_x \mid (x, e) \in d\}$. So, we can note the following points:

- for $d = \mathbb{G}$, $d_x = D_x$,
- $d = \bigcup_{x \in V}(\{x\} \times d_x)$;
- for $d, d' \subseteq \mathbb{G}$, $d \subseteq d' \Leftrightarrow \forall x \in V, d_x \subseteq d'_x$,
- $\forall x \in V$, $d|_{\{x\}} = \{x\} \times d_x$;
- for $W \subseteq V$, $d \subseteq \mathbb{G}|_W \Leftrightarrow \forall x \in V \setminus W, d_x = \emptyset$.

***Example 1*** *CSP*
Let us consider the CSP defined by:

- $V = \{x, y, z\}$
- $C = \{x < y, y < z, z < x\}$
- $var$ such that: $var(x < y) = \{x, y\}$, $var(y < z) = \{y, z\}$ and $var(z < x) = \{x, z\}$.
- $D_x = D_y = D_z = \{0, 1, 2\}$, that is:
  $\mathbb{G} = \{(x, 0), (x, 1), (x, 2), (y, 0), (y, 1), (y, 2), (z, 0), (z, 1), (z, 2)\}$
- $T$ such that:
  $T_{x<y} = \{\{(x, 0), (y, 1)\}, \{(x, 0), (y, 2)\}, \{(x, 1), (y, 2)\}\}$
  $T_{y<z} = \{\{(y, 0), (z, 1)\}, \{(y, 0), (z, 2)\}, \{(y, 1), (z, 2)\}\}$
  $T_{z<x} = \{\{(x, 1), (z, 0)\}, \{(x, 2), (z, 0)\}, \{(x, 2), (z, 1)\}\}$

For a given CSP, one is interested in the computation of the solutions. The simplest method consists in generating all the tuples from the initial domains, then testing them. This *generate and test* method is clearly expensive for wide domains. So, one prefers to reduce the domains first ("test" and generate).

To be more precise, to reduce the domains means to replace each $D_x$ by a subset of $D_x$. But in this context, each subset of $D_x$ can be denoted by $d_x$ for $d \subseteq \mathbb{G}$. Such $d_x$ is called the *domain* of $x$ and $d$ is called the *global domain*. $D_x$ is merely the greatest domain of $x$. In fact, the reduction of domains will be applied to all domains, but since $d = \bigcup_{x \in V}(\{x\} \times d_x)$, it amounts to the reduction of the global domain $d$.

Here, we focus on the reduction stage. Let $d$ the global domain. Intuitively, if $t$ is a solution of the CSP, then $t \subseteq d$ and we attempt to approach the smallest domain containing all the solutions of the CSP. So this domain must be an "approximation" of the solutions according to an ordering which is exactly the subset ordering $\subseteq$.

We describe in the next section a model for the computation of such approximations.

# 3 Domain reduction

We propose here a model of the operational semantics for the computation of approximations. It will be well suited to define notions of basic events necessary for trace analysis, and explanations useful for debugging. Moreover main classical results [4, 5, 15] are proved again in this model.

A set of operators (local consistency operators) is associated to each constraint. The intersection between the global domain and the domain obtained by application of a local consistency operator provides a new global domain. Finally, in order to always reach a fix-point (that is the approximation we look for), all the operators will be applied, as many time as necessary, according to a chaotic iteration.

A way to compute an approximation of the solutions is to associate with the constraints some *local consistency operators*. A local consistency operator is applied to the whole global domain. But in fact, the result only depends on a restriction of it to a subset of

variables $W_{in} \subseteq V$. The *type* of such an operator is $(W_{in}, W_{out})$ with $W_{in}, W_{out} \subseteq V$. Only the domains of $W_{out}$ are modified by the application of this operator. It eliminates from these domains some values which are inconsistent with respect to the domains of $W_{in}$.

**Definition 1** *A* local consistency operator *of type* $(W_{in}, W_{out})$*, with* $W_{in}, W_{out} \subseteq V$ *is a monotonic function* $r : \mathcal{P}(\mathbb{G}) \to \mathcal{P}(\mathbb{G})$ *such that:* $\forall d \subseteq \mathbb{G}$,

- $r(d)|_{V \setminus W_{out}} = \mathbb{G}|_{V \setminus W_{out}}$,
- $r(d) = r(d|_{W_{in}})$

We can note that:

- $r(d)|_{V \setminus W_{out}}$ is independent of $d$,
- $r(d)|_{W_{out}}$ only depends on $d|_{W_{in}}$,
- a local consistency operator is not a contracting function.

**Definition 2** *We say a domain* $d$ *is* r-consistent *if* $d \subseteq r(d)$*, that is* $d|_{W_{out}} \subseteq r(d)|_{W_{out}}$.

We provide an example in the obvious case of arc-consistency.

***Example 2*** *Arc-consistency*
Let $c \in C$ with $var(c) = \{x, y\}$ and $d \subseteq \mathbb{G}$. The property of arc-consistency for $d$ is:

(1) $\forall e \in d_x, \exists f \in d_y, \{(x, e), (y, f)\} \in T_c$,

(2) $\forall f \in d_y, \exists e \in d_x, \{(x, e), (y, f)\} \in T_c$.

The local consistency operator $r$ associated to (1) has the type $(\{y\}, \{x\})$ and is defined by: $r(d) = \mathbb{G}|_{V \setminus \{x\}} \cup \{(x, e) \in \mathbb{G} \mid \exists (y, f) \in d, \{(x, e), (y, f)\} \in T_c\}$. It is obvious that (1) $\Longleftrightarrow d \subseteq r(d)$, that is $d$ is r-consistent. We can define in the same way the operator of type $(\{x\}, \{y\})$ associated to (2).

***Example 3*** *Continuation of example 1*
Let us consider the constraint $x < y$ defined in example 1. For $d = \mathbb{G}$, the property of arc-consistency provided in the example above is associated to: $r_1(d) = \mathbb{G}|_{\{y,z\}} \cup \{(x, 0), (x, 1)\}$ and $r_2(d) = \mathbb{G}|_{\{x,z\}} \cup \{(y, 1), (y, 2)\}$.

The solver is described by a set of such operators associated with the constraints of the CSP. We can choose more or less accurate local consistency operators for each constraint (in general, the more accurate they are, the more expensive is the computation).

We associate to these operators, reduction operators in order to compute the intersection with the current global domain.

**Definition 3** *The* reduction operator *associated to the local consistency operator* $r$ *is the monotonic and contracting function* $d \mapsto d \cap r(d)$.

All the solvers proceeding by domain reduction use operators with this form. For GNU-Prolog, we associate to each constraint as many operators as variables in the constraint.

***Example 4*** *GNU-Prolog*
In GNU-Prolog, such operators are written $x$ *in* $r$ [7], where $r$ is a range dependent on domains of a set of variables. The rule `x in 0..max(y)` is the local consistency operator of type $(\{y\}, \{x\})$ which computes $\{0, 1, \ldots, max(d_y)\}$ where $max(d_y)$ is the greatest value in the domain of $y$. It is the local consistency operator defined by $r(d)|_{\{x\}} = \{(x, e) \mid 0 \leq e \leq max(d_y)\}$. The reduction operator associated to this local consistency operator computes the intersection with the domain of $x$ and is implemented by the rule `x in 0..max(y):dom(x)`.

The local consistency operators we use must not remove solutions of the CSP. We formalize it by the following definition.

**Definition 4** *A local consistency operator $r$ is* correct *if, for each $d \subseteq \mathbb{G}$, for each solution $t$, $t \subseteq d \Rightarrow t \subseteq r(d)$.*

A local consistency operator is associated to a constraint of the CSP. Such an operator must obviously keep the solutions of the constraint. This is formalized by the next definition and lemma.

**Definition 5** *Let $c \in C$ and $W_{out} \subseteq var(c)$. A local consistency operator $r$ of type $(W_{in}, W_{out})$ is* correct *with respect to the constraint $c$ if, for each $d \subseteq \mathbb{G}$, for each $t \in T_c$, $t \subseteq d \Rightarrow t \subseteq r(d)$.*

**Lemma 1** *If $r$ is correct with respect to $c$, then $r$ is correct.*

> *Proof.* Let $d \subseteq \mathbb{G}$ and $s \subseteq d$ a solution of the CSP. $s|_{var(c)} \in T_c$, so $s|_{var(c)} \subseteq r(d)$. Moreover $s|_{V \setminus var(c)} \subseteq \mathbb{G}|_{V \setminus var(c)} = r(d)|_{V \setminus var(c)}$ because $W_{out} \subseteq var(c)$. □

Note that the converse does not hold.

***Example 5*** *GNU-Prolog*
The rule $r$ : `x in 0..max(y)` is correct with respect to the constraint $c$ defined by $var(c) = \{x, y\}$ and $T_c = \{\{(x, 0), (y, 0)\}, \{(x, 0), (y, 1)\}, \{(x, 1), (y, 1)\}\}$ ($D_x = D_y = \{0, 1\}$ and $c$ is the constraint $x \leq y$).

Intuitively, the solver applies the reduction operators one by one replacing the global domain with the one it computes. The computation stops when some domain becomes empty (in this case, there is no solution), or when the reduction operators cannot reduce the global domain anymore (a common fix-point is reached).

From now on, we denote by $R$ a set of local consistency operators. The common fix-point of the reduction operators associated to $R$ from a global domain $d$ is a global domain $d' \subseteq d$ such that $\forall r \in R$, $d' = d' \cap r(d')$, that is $\forall r \in R$, $d' \subseteq r(d')$. The greatest common fix-point is the greatest $d' \subseteq d$ such that $\forall r \in R$, $d'$ is r-consistent. To be more precise:

**Definition 6** *$max\{d' \subseteq \mathbb{G} \mid d' \subseteq d \wedge \forall r \in R, d' \subseteq r(d')\}$ is the* downward closure *of $d$ by $R$ and is denoted by $CL \downarrow (d, R)$.*

The downward closure is the most accurate set which can be computed using a set of correct operators. Obviously, each solution belongs to this set. It is easy to verify that $CL \downarrow (d, R)$ exists and can be obtained by iteration of the operator $d \mapsto d \cap \bigcap_{r \in R} r(d)$. There exists another way to compute $CL \downarrow (d, R)$ called the *chaotic iteration* that we are going to recall.

The following definition is taken up to Apt [3].

A *run* is an infinite sequence of operators of $R$, that is, a run associates to each $i \in \mathbb{N}$ ($i \geq 1$) an element of $R$ denoted by $r^i$. A run is *fair* if each $r \in R$ appears in it infinitely often, that is $\{i \mid r = r^i\}$ is infinite. Let us define a *downward iteration* of a set of operators with respect to a run.

**Definition 7** *The* downward iteration *of the set of local consistency operators $R$ from the global domain $d \subseteq \mathbb{G}$ with respect to the run $r^1, r^2, \ldots$ is the infinite sequence $d^0, d^1, d^2, \ldots$ inductively defined by:*

1. *$d^0 = d$;*
2. *for each $i \in \mathbb{N}$, $d^{i+1} = d^i \cap r^{i+1}(d^i)$.*

*Its limit is denoted by $d^\omega = \cap_{i \in \mathbb{N}} d^i$. A* chaotic iteration *is an iteration with respect to a fair run.*

The operator $d \mapsto d \cap \bigcap_{r \in R} r(d)$ may reduce several domains at each step. But the computations are more intricate and some can be useless. In practice chaotic iterations are preferred, they proceed by elementary steps, reducing only one domain at each step. The next well-known result of confluence [3, 8] ensures that any chaotic iteration reaches the closure. Note that, since $\subseteq$ is a well-founded ordering (i.e. $\mathbb{G}$ is a finite set), every iteration from $d \subseteq \mathbb{G}$ is stationary, that is $\exists i \in \mathbb{N}, \forall j \geq i, r^{j+1}(d^j) \cap d^j = d^j$, that is $d^j \subseteq r^{j+1}(d^j)$.

**Lemma 2** *The limit of every chaotic iteration of the set of local consistency operators $R$ from $d \subseteq \mathbb{G}$ is the* downward closure *of $d$ by $R$.*

> *Proof.* Let $d^0, d^1, d^2, \ldots$ be a chaotic iteration of $R$ from $d$ with respect to $r^1, r^2, \ldots$. Let $d^\omega$ be the limit of the chaotic iteration.
> $CL \downarrow (d, R) \subseteq d^\omega$: For each $i$, $CL \downarrow (d, R) \subseteq d^i$, by induction: $CL \downarrow (d, R) \subseteq d^0 = d$. Assume $CL \downarrow (d, R) \subseteq d^i$, $CL \downarrow (d, R) \subseteq r^{i+1}(CL \downarrow (d, R)) \subseteq r^{i+1}(d^i)$ by monotonicity. Thus, $CL \downarrow (d, R) \subseteq d^i \cap r^{i+1}(d^i) = d^{i+1}$.
> $d^\omega \subseteq CL \downarrow (d, R)$: There exists $k \in \mathbb{N}$ such that $d^\omega = d^k$ because $\subseteq$ is a well-founded ordering. The run is fair, hence $d^k$ is a common fix-point of the set of reduction operators associated to $R$, thus $d^k \subseteq CL \downarrow (d, R)$ (the greatest common fix-point). □

The fairness of runs is a convenient theoretical notion to state the previous lemma. Every chaotic iteration is stationary, so in practice the computation ends when a common

fix-point is reached. Moreover, implementations of solvers use various strategies in order to determinate the order of invocation of the operators. These strategies are used so as to optimize the computation, but this is not in the scope of this report.

In practice, when a domain becomes empty, we know that there is no solution, so an optimization consists in stopping the computation before the closure is reached. In that case, we say that we have a *failure iteration.*

We have provided in this section a model of the operational semantics for the solvers on finite domains using domain reduction. This model is language independent and enough general in order to be used for the platform of each partner: GNU-Prolog, CHIP and PaLM.

# 4 Deduction rules

The application of a local consistency operator can be considered as a basic event. But for the notion of explanation, we need to be more precise. So, in this section, we attempt to explain in detail the application of a local consistency operator.

Note that we are interested by the value withdrawal, that is when a value is not in a global domain but in its complementary. So, we consider this complementary and the "duals" of the local consistency operators. By this way, at the same time we reduce the global domain, we build its complementary. We associate natural rules to these operators. These rules will be the constructors of the explanations.

First we need some notations. Let $\overline{d} = \mathbb{G} \setminus d$. In order to help the understanding, we always use $\overline{d}$ for the complementary of a global domain and $d$ for a global domain.

**Definition 8** *Let $r$ an operator, we denote by $\widetilde{r}$ the dual of $r$ defined by:* $\forall d \subseteq \mathbb{G}, \widetilde{r}(\overline{d}) = \overline{r(d)}$.

We need to consider sets of such operators as for local consistency operators. Let $\widetilde{R} = \{\widetilde{r} \mid r \in R\}$. The upward closure of $\overline{d}$ by $\widetilde{R}$, denoted by $CL \uparrow (\overline{d}, \widetilde{R})$ exists and is the least $\overline{d'}$ such that $\overline{d} \subseteq \overline{d'}$ and $\forall r \in R$, $\widetilde{r}(\overline{d'}) \subseteq \overline{d'}$.

Next lemma ensures that the downward closure of a set of local consistency operators from a global domain $d$ is the complementary of the upward closure of the set of dual operators from the complementary of $d$.

**Lemma 3** $CL \uparrow (\overline{d}, \widetilde{R}) = \overline{CL \downarrow (d, R)}$.

*Proof.* straightforward □

By the same way we defined a downward iteration of a set of operators from a domain, we define an upward iteration.

The *upward iteration* of $\widetilde{R}$ from the global domain $\overline{d} \subseteq \mathbb{G}$ with respect to $\widetilde{r^1}, \widetilde{r^2}, \ldots$ is the infinite sequence $\overline{d^0}, \overline{d^1}, \overline{d^2}, \ldots$ inductively defined by:

1. $\overline{d^0} = \overline{d}$,
2. $\overline{d^{i+1}} = \overline{d^i} \cup \widetilde{r^{i+1}}(\overline{d^i})$.

We can rewrite the second item: $\overline{d^{i+1}} = \overline{d^i} \cup \overline{r^{i+1}(d^i)}$. It is then obvious, that we add to $\overline{d^i}$, the elements of $d^i$ removed by $r^{i+1}$.

The link between the downward and the upward iteration clearly appears by noting that: $\overline{d^{i+1}} = \overline{d^i \cap r^{i+1}(d^i)}$ and $\cup_{j\in\mathbb{N}}\overline{d^j} = CL \uparrow (\overline{d}, \widetilde{R}) = \overline{CL \downarrow (d, R)}$.

We have provided two points of view for the reduction of a global domain $d$ with respect to a run $r^1, r^2, \ldots$. In the previous section, we consider the reduced global domain, but in this section, we consider the complementary of this reduced global domain, that is the set of elements removed of the global domain. As a local consistency operator "keeps" elements in a domain, its dual "adds the others" in the complementary.

Now, we associate deduction rules to these dual operators. These rules are natural to build the complementary of the global domain and well suited to provide proof trees.

**Definition 9** *A* deduction rule *of type* $(W_{in}, W_{out})$ *is a rule* $h \leftarrow B$ *such that* $h \in \mathbb{G}|_{W_{out}}$ *and* $B \subseteq \mathbb{G}|_{W_{in}}$.

For each operator $r \in R$ of type $(W_{in}, W_{out})$, we denote by $\mathcal{R}_r$ a set of deduction rules of type $(W_{in}, W_{out})$ which defines $\widetilde{r}$, that is $\widetilde{r}$ is such that: $\widetilde{r}(\overline{d}) = \{h \in \mathbb{G} \mid \exists B \subseteq \overline{d}, h \leftarrow B \in \mathcal{R}_r\}$. For each operator, this set of deduction rules exists. There exists possibly many such sets, but one is natural.

We provide illustrations of this model on different consistency examples. Let us begin with the obvious arc consistency case.

***Example 6*** *Arc consistency*
Let us consider the local consistency operator $r$ defined in example 2 by:
$r(d) = \mathbb{G}|_{V\setminus\{x\}} \cup \{(x,e) \in \mathbb{G} \mid \exists (y,f) \in d, \{(x,e),(y,f)\} \in T_c\}$.
So, $\widetilde{r}(\overline{d}) = \overline{r(d)} = \{(x,e) \in \mathbb{G} \mid \forall (y,f) \in d, \{(x,e),(y,f)\} \notin T_c\}$.
Let $B_{(x,e)} = \{(y,f) \mid \{(x,e),(y,f)\} \in T_c\}$. Then,

$$\begin{aligned} B_{(x,e)} \subseteq \overline{d} \quad &\Leftrightarrow \quad \forall (y,f) \in \mathbb{G}, [\{(x,e),(y,f)\} \in T_c \Rightarrow (y,f) \in \overline{d}] \\ &\Leftrightarrow \quad \forall (y,f) \in \mathbb{G}, [(y,f) \in d \Rightarrow \{(x,e),(y,f)\} \notin T_c] \\ &\Leftrightarrow \quad \forall (y,f) \in d, \{(x,e),(y,f)\} \notin T_c \end{aligned}$$

So, $\widetilde{r}(\overline{d}) = \{(x,e) \in \mathbb{G} \mid B_{(x,e)} \subseteq \overline{d}\}$.
Finally, the set of deduction rules associated to $r$ is $\mathcal{R}_r = \{(x,e) \leftarrow B_{(x,e)} \mid (x,e) \in d\}$.

***Example 7*** *Continuation of example 1*
Let us consider the CSP of example 1. Two local consistency operators are associated to the constraint $x < y$: $r_1$ of type $(\{y\},\{x\})$ and $r_2$ of type $(\{x\},\{y\})$. The set of deduction rules $\mathcal{R}_{r_1}$ associated to $r_1$ contains the three deduction rules:

- $(x,0) \leftarrow \{(y,1),(y,2)\}$,
- $(x,1) \leftarrow \{(y,2)\}$, and
- $(x,2) \leftarrow \emptyset$.

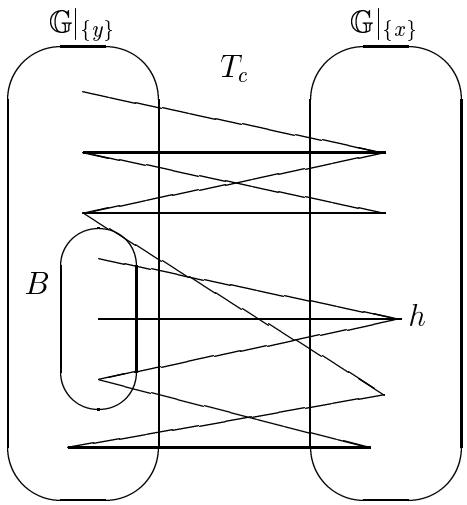


Figure 1: The particular case of arc consistency

A deduction rule $h \leftarrow B$ can be understood as follow: if all the elements of $B$ are removed from the global domain, then $h$ does not participate in any solution of the CSP and we can remove it. See for example figure 1. Note that if $(x, e) \in \mathbb{G}|_{\{x\}}$ does not appear in any tuple of $T_c$, then we have the trivial deduction rule $(x, e) \leftarrow \emptyset$.

Our formalization is also well suited to include weaker arc consistency operators. In GNU-Prolog, a full arc consistency operator $r$ of type $(\{y\}, \{x\})$ uses the whole domain of $y$, whereas, a partial arc consistency reduction operator only uses its lower and upper bounds. In that case, we need two sets of deduction rules $\mathcal{R}_{max}$ and $\mathcal{R}_{min}$, one for each bound. Then, for $\overline{d} \subseteq \mathbb{G}$, $\widetilde{r}(\overline{d}) = \{(x, e) \mid \exists B_{(x,e)} \subseteq \overline{d}, (x, e) \leftarrow B_{(x,e)} \in (\mathcal{R}_{max} \cup \mathcal{R}_{min})\}$. Note that there exists two rules with the head $(x, e)$, one for the upper bound in $\mathcal{R}_{max}$ and one for the lower bound in $\mathcal{R}_{min}$.

***Example 8*** *Partial Arc Consistency in GNU-Prolog*
Let us consider the constraint "`x #= y+c`" in GNU-Prolog where $x$, $y$ are variables and $c$ a constant. This constraint is implemented by two local consistency operators: $r_1$ of type $(\{y\}, \{x\})$ and $r_2$ of type $(\{x\}, \{y\})$. In GNU-Prolog, $r_1$ is defined by the rule `x in min(y)+c..max(y)+c`.
$\widetilde{r_1}(\overline{d}) = \{(x, e) \mid \exists B_{(x,e)} \subseteq \overline{d}, (x, e) \leftarrow B_{(x,e)} \in (\mathcal{R}_{max} \cup \mathcal{R}_{min})\}$ with:

- $\mathcal{R}_{max} = \{(x, e) \leftarrow \{(y, f) \mid f + c \geq e\} \mid (x, e) \in \mathbb{G}|_{\{x\}}\}$ and
- $\mathcal{R}_{min} = \{(x, e) \leftarrow \{(y, f) \mid f + c \leq e\} \mid (x, e) \in \mathbb{G}|_{\{x\}}\}$.

$r_2$ of type $(\{x\}, \{y\})$ is defined in the same way by the rule `y in min(x)-c..max(x)-c`.

In the framework of hyper-arc consistency, the tuples may contain more than two variables. For a constraint $c \in C$ and a variable $x \in var(c)$, if one value of each tuple containing $(x, e)$ has disappeared of the global domain, then $(x, e)$ can be removed from the global domain. For $(x, e)$, we have as much deduction rules as possibilities to take one element (except $(x, e)$) in each tuple of $T_c$ containing $(x, e)$.

***Example 9*** *Hyper-arc Consistency in GNU-Prolog*
Let us consider the constraint "`x #=# y+z`" in GNU-Prolog.
Let $\mathbb{G} = \{(x,3),(y,1),(y,2),(z,1),(z,2)\}$. The constraint is implemented by three local consistency rules $r_1$, $r_2$ and $r_3$. Let us consider $r_1$ of type $(\{y,z\},\{x\})$. $r_1$ is defined by: $\widetilde{r}_1(\overline{d}) = \{(x,e) \mid \exists B_{(x,e)} \subseteq \overline{d}, (x,e) \leftarrow B_{(x,e)} \in \mathcal{R}\}$.
We can eliminate $(x,3)$ from $d$ if for each tuple containing $(x,3)$, one value is removed from $d$. There exists two tuples containing $(x,3)$: $\{(x,3),(y,1),(z,2)\}$ and $\{(x,3),(y,2),(z,1)\}$. So, we have:

- $(y,1) \notin d \wedge (y,2) \notin d \Rightarrow (x,3) \notin r_1(d)$;
- $(y,1) \notin d \wedge (z,1) \notin d \Rightarrow (x,3) \notin r_1(d)$;
- $(y,2) \notin d \wedge (z,2) \notin d \Rightarrow (x,3) \notin r_1(d)$;
- $(z,1) \notin d \wedge (z,2) \notin d \Rightarrow (x,3) \notin r_1(d)$;

Then, $\mathcal{R}$ contains the four deduction rules:

- $(x,3) \leftarrow \{(y,1),(y,2)\}$
- $(x,3) \leftarrow \{(y,1),(z,1)\}$
- $(x,3) \leftarrow \{(y,2),(z,2)\}$
- $(x,3) \leftarrow \{(z,1),(z,2)\}$

In this section, we have considered a dual view of domain reduction. In this framework, we have introduced deduction rules. These rules explain the withdrawal of a value by the withdrawal of other values. In the next section, we construct trees with these rules, in order to have a complete explanation of a value withdrawal associated to an iteration.

# 5 Value withdrawal explanations

Sometimes, when a domain becomes empty or just when a value is removed from a domain, the user wants an explanation of this phenomenon [2, 12]. The case of failure is the particular case where all the values are removed. It is the reason why the basic event here will be a value withdrawal. Let us consider a chaotic iteration, and let us assume that at a step a value is removed from the domain of a variable. In general, all the operators used from the beginning of the iteration are not necessary to explain the value withdrawal. It is possible to explain the value withdrawal by a subset of these operators such that every chaotic iteration using this subset of operators removes the considered value. We associate a set of proof trees to a value withdrawal during a chaotic iteration. We have two notions of explanation for a value withdrawal. The first one is a set of local consistency operators responsible of this withdrawal, the second one, more precise is based on the proof trees. We recall here the definition of proof trees, then we deduce the explanation set and provide some important properties for our explanations.

First, we use the deduction rules in order to build proof trees. We consider the set of all the deduction rules for all the local consistency operators $r \in R$: let $\mathcal{R} = \cup_{r \in R} \mathcal{R}_r$.

We use the following notations: $cons(h, T)$ is a tree, $h$ is the label of its root and $T$ the set of its sub-trees. We denote by $root(t)$ the label of the root of a tree $t$. We recall the definition of a proof tree [1].

**Definition 10** *A* proof tree *with respect to $\mathcal{R}$ is inductively defined by: $cons(h, T)$ is a proof tree if $h \leftarrow \{root(t) \mid t \in T\} \in \mathcal{R}$ and $T$ is a set of proof trees with respect to $\mathcal{R}$.*

Our set of deduction rules is not complete: we must take the initial domain into account. If we compute a downward closure from the whole global domain $\mathbb{G}$, then its complementary is the empty set. In this case, $\mathcal{R}$ is complete. But if we compute a downward closure from a domain $d \subset \mathbb{G}$, then its dual upward closure starts with $\overline{d}$. We need facts in order to directly include the elements of $\overline{d}$. Let $\mathcal{R}^d = \mathcal{R} \cup \{h \leftarrow \emptyset \mid h \in \overline{d}\}$. Next lemma ensures that, with this set of deduction rules, we can build proof trees for each element of $CL \uparrow (\overline{d}, \widetilde{R})$.

**Lemma 4** *$\overline{CL \downarrow (d, R)}$ is the set of the roots of proof trees with respect to $\mathcal{R}^d$.*

*Proof.* straightforward □

It is important to note that some proof trees do not correspond to any chaotic iteration. We are interested in the proof trees corresponding to a chaotic iteration.

***Example 10*** *Continuation of example 1*
Let us consider the CSP defined in example 1. Six reduction rules are associated to the constraints of the CSP:

- $r_1$ of type $(\{y\}, \{x\})$ and $r_2$ of type $(\{x\}, \{y\})$ for $x < y$.
- $r_3$ of type $(\{z\}, \{y\})$ and $r_4$ of type $(\{y\}, \{z\})$ for $y < z$.
- $r_5$ of type $(\{z\}, \{x\})$ and $r_6$ of type $(\{x\}, \{z\})$ for $z < x$.

Figure 2 shows three different proof trees rooted by $(x, 0)$. For example, the first one says: $(x, 0)$ is removed from the global domain because $(y, 1)$ and $(y, 2)$ are removed from the global domain. $(y, 1)$ is removed from the global domain because $(z, 2)$ is removed from the global domain and so on ... The first and third proof trees correspond to some chaotic iterations. But the second one does not correspond to any (because $(x, 0)$ could not disappear twice).

We provide now the definition of an explanation set.

**Definition 11** *We call an* explanation set *for $h \in \mathbb{G}$ a set of local consistency operators $E \subseteq R$ such that $h \notin CL \downarrow (d, E)$.*

From now on, we consider a fixed chaotic iteration $d = d^0, d^1, \ldots, d^i, \ldots$ such that $d^\omega = CL \downarrow (d, R)$. In this context, to each $h \in d \setminus d^\omega$, we can associate one and only one integer $i \geq 1$ such that $h \in d^{i-1} \setminus d^i$. This integer is the step in the chaotic iteration where $h$ is removed of the global domain.

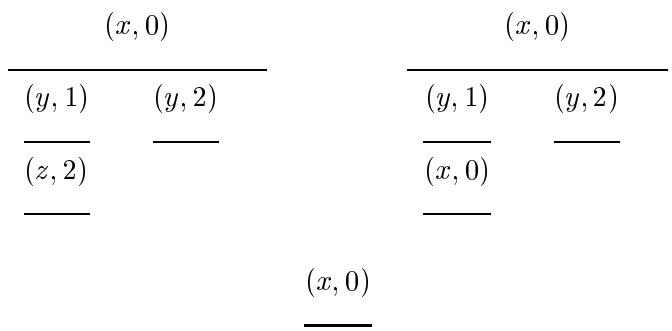


Figure 2: Proof trees for $(x, 0)$

**Definition 12** *If $h \in d \backslash d^{\omega}$, we denote by $step(h)$, the integer $i \geq 1$ such that $h \in d^{i-1} \backslash d^{i}$. If $h \notin d$ then $step(h) = 0$.*

We know that when an element is removed, there exists a proof tree rooted by this element. This proof tree uses a set of local consistency operators. These operators are responsible of this value withdrawal. We give a notation for such a set in the following definition.

**Definition 13** *Let $t$ a proof tree. We denote by $expl_set(t)$ the set of local consistency operators: $\{r^{step((x,e))} \mid (x, e)$ has an occurrence in $t\}$.*

Note that an explanation set is independent of any chaotic iteration in the sense of: if an explanation set is responsible of a value withdrawal then whatever is the chaotic iteration used, this set of operators will always remove this element.

**Theorem 1** *If $t$ is a proof tree, then $expl_set(t)$ is an explanation set for $root(t)$.*

*Proof.* By lemma 4. □

We have defined explanation sets which are independent of the computation. So, when a value is removed during a computation, we are able to obtain a set local consistency operators responsible of this removal and thus a set of constraints linked to these operators. This can be useful for failure analysis.

But we are interested in an other problem which is the debugging of constraint programs. In this framework, it is useful to have more accurate knowledge than sets of operators. So, the structure of proof trees which contains a notion of causality for the removals, provides us more information.

In order to compute incrementaly the explanations from a chaotic iteration, we define the set of proof trees $S^i$ which can be constructed at a step $i \in \mathbb{N}$ of a chaotic iteration. Obviously, before any computation, it only contains the trees without sub-trees rooted by the elements which are not in the initial domain. At each step, we construct the new trees with the trees of the previous steps and the local consistency operator used at this step. More formally:

**Definition 14** *Let the family* $(S^i)_{i\in\mathbb{N}}$ *defined by:*

- $S^0 = \{cons(h,\emptyset) \mid h \notin d\}$,
- $S^{i+1} = S^i \cup \{cons(h,T) \mid h \in d^i, T \subseteq S^i, h \leftarrow \{root(t) \mid t \in T\} \in \mathcal{R}_{r^{i+1}}\}$.

**Lemma 5** $\{root(t) \mid t \in S^i\} = \overline{d^i}$.

*Proof.* By induction on $i$: $S^0$ obviously checks this property.
We suppose $\{root(t) \mid t \in S^i\} = \overline{d^i}$ and we prove

1. $\{root(t) \mid t \in S^{i+1}\} \subseteq \overline{d^{i+1}}$. Let $h$ the root of $t$ such that $t \in S^{i+1}$. There exists two cases:
   - $t \in S^i$, then $h \in \overline{d^i}$, then $h \in \overline{d^{i+1}}$ because $\overline{d^i} \subseteq \overline{d^{i+1}}$.
   - $t \notin S^i$. There exists $h \leftarrow B \in \mathcal{R}_{r^{i+1}}, h \in d^i$ and $\forall b \in B, b = root(t_b), t_b \in S^i$. So, $b \in \overline{d^i}$, thus $h \in \overline{r^{i+1}(d^i)} \cup \overline{d^i} = \overline{d^{i+1}}$.
2. $\overline{d^{i+1}} \subseteq \{root(t) \mid t \in S^{i+1}\}$. Let $h \in \overline{d^{i+1}}$. There exists two cases:
   - $h \in \overline{d^i}$, then $h \in \{root(t) \mid t \in S^i\}$, then $h \in \{root(t) \mid t \in S^{i+1}\}$.
   - $h \in d^i$, then $\exists h \leftarrow B \in \mathcal{R}_{r^{i+1}}$ and $\forall b \in B, b \in \overline{d^i}$. That is $B = \{root(t) \mid t \in T\}$ and $cons(h,T) \in S^{i+1}$, that is $h \in \{root(t) \mid t \in S^{i+1}\}$.

□

The previous lemma is reformulated in the following corollary which ensures that each element removed from $d$ during a chaotic iteration is the root of a tree of $\cup_{i\in\mathbb{N}} S^i$.

**Corollary 1** $\{root(t) \mid t \in \cup_{i\in\mathbb{N}} S^i\} = \overline{CL \downarrow (d,R)}$.

*Proof.*

$$\begin{array}{rcl} \{root(t) \mid t \in \cup_{i\in\mathbb{N}} S^i\} & = & \cup_{i\in\mathbb{N}}\{root(t) \mid t \in S^i\} \\ & = & \cup_{i\in\mathbb{N}}\overline{d^i} \text{ by lemma 5} \\ & = & \overline{d^\omega} \\ & = & \overline{CL \downarrow (d,R)} \end{array}$$

□

$$\dfrac{\dfrac{\dfrac{(x,0,r_1)}{(y,1,r_3)}}{(z,2,r_6)} \qquad \dfrac{(x,0,r_1)}{(y,2,r_3)}}{}$$

Figure 3: Explanation tree for $(x,0)$

From a proof tree, we can obtain the local consistency operators used with the function *step*. It could be interesting to have this information directly in the tree. So, we consider an *explanation tree* as a proof tree such that, to each element $h$ of the tree, we add the

local consistency operator corresponding to $step(h)$. For example, the first proof tree of Figure 2 provides the explanation tree of Figure 3. The corresponding explanation set is $\{r_1, r_3, r_6\}$.

In this last section, we have provided the theoretical foundations of value withdrawal explanations. We will use them to explain failures and error diagnosis but this is not in the scope of this report.

# 6 Conclusion

This report has given the theoretical model for the solvers on finite domains by domain reduction. This model is language independent and can be applied to each platform of the partners: GNU-Prolog, CHIP and PaLM. This model takes several consistencies (partial and full hyper-arc consistency of GNU-Prolog for example) into account and is well suited to define explanations and traces.

This model rests on reduction operators and chaotic iteration. Reduction operators are defined from the constraint and the consistency used. These operators are applied among a chaotic iteration which ensures to take all of them into account. The order of invocation of the operators depends on the strategy used by the solver and is out of the scope of this report.

In the fourth part of the report, we were motivated by the explanations, that is to be able to answer to the question: Why this value does not appear in any solution ? So, we did not have to consider the global domain but its complementary, that is the set of removed values. A deduction rule is able to explain the propagation mechanism. It says us: these values are not in the current global domain, so this one can be removed too.

The linking of these rules defines proof trees. These trees explain the withdrawal of a value from the beginning of the computation to the value withdrawal. Only the elements responsible of the value withdrawal appears in these trees. We have defined explanations sets, that is sets of operators responsible of a value withdrawal, which can be sufficient for several applications.

Narrowing and labeling are interleaved during a resolution. So the next step of our work will include the labeling stage in this model. Finally we will take the language into account.

This report has benefited from works and discussions with EMN.

# References

[1] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland Publishing Company, 1977.

[2] A. Aggoun, F. Bueno, M. Carro, P. Deransart, M. Fabris, W. Drabent, G. Ferrand, M. Hermenegildo, C. Lai, J. Lloyd, J. Małuszyński, G. Puebla, and A. Tessier. CP debugging needs and tools. In M. Kamkar, editor, *International Workshop on Automated Debugging*, volume 2 of *Linköping Electronic Articles in Computer and Information Science*, 1997.

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

[4] K. R. Apt. The role of commutativity in constraint propagation algorithms. *ACM TOPLAS*, 22(6):1002–1034, 2000.

[5] F. Benhamou. Heterogeneous constraint solving. In M. Hanus and M. Rofríguez-Artalejo, editors, *International Conference on Algebraic and Logic Programming*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76. Springer-Verlag, 1996.

[6] F. Benhamou and F. Goualard. A visualization tool for constraint program debugging. In *International Conference on Automated Software Engineering*, pages 110–117. IEEE Computer Society Press, 1999.

[7] P. Codognet and D. Diaz. Compiling constraints in `clp(fd)`. *Journal of Logic Programming*, 27(3):185–226, 1996.

[8] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions mathematical foundation. In *Symposium on Artificial Intelligence and Programming Languages*, volume 12(8) of *ACM SIGPLAN Not.*, pages 1–12, 1977.

[9] G. Ferrand and A. Tessier. Positive and negative diagnosis for constraint logic programs in terms of proof skeletons. In M. Kamkar, editor, *International Workshop on Automated Debugging*, volume 2 of *Linköping Electronic Articles in Computer and Information Science*, 1997.

[10] G. Ferrand and A. Tessier. Declarative diagnosis in the clp scheme. In P. Deransart, M. Hermenegildo, and J. Małuszyński, editors, *Analysis and Visualisation Tools for Constraint Programming*, volume 1870 of *Lecture Notes in Computer Science*, chapter 5, pages 151–176. Springer-Verlag, 2000.

[11] C. Guéret, N. Jussien, and C. Prins. Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. *European Journal of Operational Research*, 2000.

[12] N. Jussien. *Relaxation de Contraintes pour les Problèmes dynamiques.* PhD thesis, Université de Rennes 1, 1997.

[13] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, 1998.

[14] M. Meier. Debugging constraint programs. In U. Montanari and F. Rossi, editors, *International Conference on Principles and Practice of Constraint Programming*, volume 976 of *Lecture Notes in Computer Science*, pages 204–221. Springer-Verlag, 1995.

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

[16] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[17] M. H. Van Emden. Value constraints in the clp scheme. In *International Logic Programming Symposium, post-conference workshop on Interval Constraints*, 1995.

[18] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming. MIT Press, 1989.