Explanations to Understand the Trace of a Finite Domain Constraint Solver[⋆]

Gérard Ferrand[1], Willy Lesaint[2], and Alexandre Tessier[1]

[1] Laboratoire d'Informatique Fondamentale d'Orléans
University of Orléans, France
`http://www.univ-orleans.fr/SCIENCES/LIFO/`
{`Gerard.Ferrand,Alexandre.Tessier`}`@lifo.univ-orleans.fr`

[2] Laboratoire d'Études et de Recherche en Informatique d'Angers
University of Angers, France
`http://www.info.univ-angers.fr/info/leria.html`
`Willy.Lesaint@univ-angers.fr`

Abstract. Some works in progress on finite domain constraint solvers concern the implementation of a XML trace of the computation according to the OADymPPaC DTD (for example in GNU-Prolog, PaLM, Chip). Because of the large size of traces, even for small toy problems, some tools are needed to understand this trace. Explanations of value withdrawal (or nogoods) during domain reduction are used by some solvers in finite domain constraint programming. In this paper, we use a formalization of explanations by proof trees in a fixpoint framework based on iteration of monotonic local consistency operators. Proof trees provide a declarative view of the computations by constraint propagation. We show how explanations may be naturally extracted from the OADymPPaC trace format. Explanations allow a better understanding of the domain reductions in the trace.

1 Introduction

For some years, constraint programming over finite domains [1, 2] has proved its ability efficiency to solve difficult problems. It combines declarativity of relational style and efficiency of finite domain constraint solvers. These solvers are mainly based on domain reduction by consistency notions. It consists in eliminating, from the current domains of some variables, some values which cannot belong to a solution according to the constraints and the current domains of some other variables. These removed values are characterized by a notion of local consistency.

Several works [3–5] formalize domain reduction thanks to operators. These operators reduce the variable domains. In practice, operators are applied along an iteration according to different strategies. Chaotic iterations [6] have been used in order to describe domain reduction from a theoretical general point of

[⋆] This work is partially supported by the French RNTL through the OADymPPaC project. `http://contraintes.inria.fr/OADymPPaC/`

view. It ensures confluence, that is to obtain the same reduced domain whatever the order of application of the operators is. In this framework, domain reduction can be described with notions of fixpoints and closures.

Other works in the constraint community concerns explanations[3] [7] (or no-goods). The interest of explanations in this paradigm of constraint programming over finite domains is growing. The idea of explanations is to memorize information about the removals of values. They have successfully been used for dynamic constraint satisfaction problems [7], dynamic backtracking [8], configuration problems [9, 10], constraint retraction [11], failure analysis [12], or declarative diagnosis [13].

[14] is an attempt to lay a theoretical foundation of value withdrawal explanations in the above-mentioned framework of chaotic iteration. To each local consistency operator is associated its dual operator. Each dual operator may be defined by a set of rules in the sense of inductive definitions [15]. A rule expresses a value removal as a consequence of other value removals. In this framework, an explanation is formalized by a proof tree using these sets of rules. The explanations defined provide us with a declarative view of the computation. Note that the single role of a solver is to remove values. In this paper, explanations are proofs of these removals. Explanations may be considered as the essence of domain reduction.

Some works in progress on finite domains constraint solvers concern the implementation of a XML [16] trace [17] of the computation according to the OADymPPaC[4] DTD (already implemented in GNU-Prolog [18, 19], PaLM [20] and Chip [21]). Because of their large size, even for toy problems, some tools are needed to understand traces. Since explanations only provide the necessary information explaining a value removal, they are a good way to answer this issue and particularly to understand the domain reductions described in a trace. The main contribution of this paper is to show how explanations can be naturally extracted from the OADymPPaC trace format.

The paper is organized as follows: Section 2 (Preliminaries) briefly provides a formalism for domain reduction well-suited to define explanations for the basic events which are the withdrawal of a value from a domain. Section 3 (Explanations) is dedicated to the notion of explanation. It defines explanations as proof trees built from sets of rules defining the dual operators of local consistency operators. Section 4 (Computed Explanations) first describes quickly the OADymPPaC XML trace format focusing on the parts which concern domain reduction. Next it shows how explanations can be extracted from a given trace.

2 Preliminaries

Our framework uses families instead of cartesian products because it leads to lighter notations. Indeed, the notion of monotonic operators and least or greatest

[3] See `http://www.e-constraints.net/` for more details on explanations.

[4] `http://contraintes.inria.fr/OADymPPaC/index_en.html` (OADymPPaC is a French RNTL project, consortium: COSYTEC, ILOG, EMN, INRIA, IRISA, LIFO)

fixpoints are easier in a set theoretical framework where the order is the set inclusion.

2.1 Some Notations

Let us assume fixed:

- a finite set of *variable* symbols V;
- a family $(D_x)_{x \in V}$ where each D_x is a finite non empty set, D_x is the *domain* of the variable x.

We are going to consider various *families* $f = (f_i)_{i \in I}$. Such a family can be identified with the *function* $i \mapsto f_i$, itself identified with the *set* $\{(i, f_i) \mid i \in I\}$.

In order to define monotonic operators on a power-set, we consider the *domain* $\mathbb{D} = \bigcup_{x \in V}(\{x\} \times D_x)$, i.e. $\mathbb{D}$ is the set of all possible pairs of a variable and its value.

A subset d of $\mathbb{D}$ is called *an environment.* We denote by $d|_W$ the *restriction* of d to a set of variables $W \subseteq V$, that is, $d|_W = \{(x, e) \in d \mid x \in W\}$. Note that, with $d, d' \subseteq \mathbb{D}$, $d = \bigcup_{x \in V} d|_{\{x\}}$, and $(d \subseteq d') \Leftrightarrow (\forall x \in V, d|_{\{x\}} \subseteq d'|_{\{x\}})$. $d|_{\{x\}}$ is called the *environment of the variable* x (in the environment d).

A *tuple* (or *valuation*) t is a particular environment such that each variable appears only once: $t \subseteq \mathbb{D}$ and $\forall x \in V, \exists e \in D_x, t|_{\{x\}} = \{(x, e)\}$. A *tuple* t *on* a set of variables $W \subseteq V$, is defined by $t \subseteq \mathbb{D}|_W$ and $\forall x \in W, \exists e \in D_x, t|_{\{x\}} = \{(x, e)\}$.

2.2 Constraint Satisfaction Problem and Solutions

A *Constraint Satisfaction Problem* (CSP) on $(V, \mathbb{D})$ is made of:

- a finite set of *constraint* symbols C;
- a function var : $C \to \mathcal{P}(V)$, which associates with each constraint symbol the set of variables of the constraint;
- a family $(T_c)_{c \in C}$ such that: for each $c \in C$, T_c is a set of tuples on $\mathrm{var}(c)$, T_c is the set of *solutions* of c.

From now on, we assume fixed a CSP $(C, \mathrm{var}, (T_c)_{c \in C})$ on $(V, \mathbb{D})$.

Definition 1. *A tuple t is a* solution *of the CSP if $\forall c \in C, t|_{\mathrm{var}(c)} \in T_c$. We denote by* Sol *the set of the solutions of the CSP.*

2.3 Program and Closure

A program is used to solve a CSP, (i.e to find the solutions) thanks to domain reduction techniques and labeling. Here we focus on domain reduction. In this paper, we are interested in only one branch of the search tree. In this context, labeling can be considered as additional constraints. It could be possible to distinguish these two kinds of constraints, this leads to no conceptual difficulties and is not really necessary here. It complicates the formalism and is omitted. [22] shows how to introduce labeling in the formalism.

The main idea of domain reduction is to remove from the current environment some values which cannot participate to any solution of some constraints, thus which cannot participate to any solution of the CSP. These removals are closely related to a notion of local consistency. This can be formalized by local consistency operators.

Definition 2. *A* local consistency operator *r is a monotonic function*

$$r : \mathcal{P}(\mathbb{D}) \to \mathcal{P}(\mathbb{D})$$

Note that even global constraints can be formalized by these operators.

As we want a contracting operator to reduce the environment (i.e. $r(d) \subseteq d$), next we will also consider $d \mapsto d \cap r(d)$ called a *reduction operator* . But in general, the local consistency operators are not contracting functions, as shown later to define their dual operators.

A *program* R on $(V, \mathbb{D})$ is a set of local consistency operators.

From now on, we assume fixed a program R on $(V, \mathbb{D})$.

We are interested in particular environments: the common fix-points of the *reduction operators* $d \mapsto d \cap r(d)$, $r \in R$. Such an environment d verifies $\forall r \in R$, $d = d \cap r(d)$, that is no value cannot be removed according to the operators.

Definition 3. *Let $r \in R$. We say an environment d is r-*consistent *if $d \subseteq r(d)$.*
*We say an environment d is R-*consistent *if $\forall r \in R$, d is r-consistent.*

Domain reduction from an environment d by R amounts to compute the greatest common fix-point included in d of the reduction operators $d \mapsto d \cap r(d)$, $r \in R$.

Definition 4. *The* downward closure *of d by R, denoted by $\mathrm{CL}{\downarrow}(d, R)$, is the greatest $d' \subseteq d$ such that d' is R-consistent.*

In general, we are interested in the closure of $\mathbb{D}$ by R (the computation starts from $\mathbb{D}$), but sometimes we would like to express closures of subset of $\mathbb{D}$, for example to take into account dynamic aspects or labeling.

By Definition 4, since $d \subseteq \mathbb{D}$:

Lemma 1. *If d is R-consistent then $d \subseteq \mathrm{CL}{\downarrow}(\mathbb{D}, R)$.*

2.4 Links between CSP and Program

Of course, the program is linked to the CSP. The operators are chosen to "implement" the CSP. In practice, this correspondence is expressed by the fact that the program is able to test any valuation. That is, if all the variables are bounded, the program should be able to answer to the question: "is this valuation a solution of the CSP?".

Definition 5. *A local consistency operator r* preserves the solutions *of a set of constraints C' if, for each tuple t, $(\forall c \in C', t|_{\mathrm{var}(c)} \in T_c) \Rightarrow t$ is r-consistent.*

If $C' \subseteq C''$ and if r preserves the solutions of C' then r preserves the solutions of C''. In particular, considering $C'' = C$, we have r preserves the solutions of the CSP.

For example, in the well-known case of (hyper) arc-consistency, each constraint c of the CSP is implemented by a set of local consistency operators R_c. Of course, each $r \in R_c$ preserves the solutions of $\{c\}$.

To preserve solutions is a correction property of operators. A notion of completeness is used to choose the set of operators "implementing" the CSP. It ensures to reject valuations which are not solutions of constraints. But this notion is not necessary for our purpose.

In the following lemmas, we consider $S \subseteq \mathrm{Sol}$, that is S a set of solutions of the CSP and $\bigcup S$ $(= \bigcup_{t \in S} t)$ its projection on $\mathbb{D}$.

Lemma 2. *Let* $S \subseteq \mathrm{Sol}$*, if* r *preserves the solutions of the CSP then* $\bigcup S$ *is* r*-consistent.*

Proof. $\forall t \in S, t \subseteq r(t)$ so $\bigcup S \subseteq \bigcup_{t \in S} r(t)$. Now, $\forall t \in S, t \subseteq \bigcup S$ so $\forall t \in S, r(t) \subseteq r(\bigcup S)$. $\square$

From now on, we consider that the set of local consistency operators of the fixed program R preserves the solutions of the fixed CSP.

Lemma 3. *If* $S \subseteq \mathrm{Sol}$ *then* $\bigcup S \subseteq \mathrm{CL}{\downarrow}(\mathbb{D}, R)$*.*

Proof. by Lemmas 1 and 2. $\square$

Finally, the following corollary emphasizes the link between the CSP and the program.

Corollary 1. $\bigcup \mathrm{Sol} \subseteq \mathrm{CL}{\downarrow}(\mathbb{D}, R)$*.*

The downward closure is a superset (an "approximation") of $\bigcup \mathrm{Sol}$ which is itself the projection (an "approximation") of Sol. But the downward closure is the most accurate set which can be computed using a set of local consistency operators in the framework of domain reduction without splitting the domain (without search tree).

3 Explanations

An explanation is a proof tree of a value removal ([14] provides more details about explanations).

3.1 Dual View of Domain Reduction

First we need some notations. Let $\overline{d} = \mathbb{D} \setminus d$. In order to help understanding, we always use the notation $\overline{d}$ for a subset of $\mathbb{D}$ if intuitively it denotes a set of removed values.

Definition 6. *Let r be an operator, we denote by $\widetilde{r}$ the* dual *of r defined by: $\forall d \subseteq \mathbb{D}, \widetilde{r}(\overline{d}) = \overline{r(d)}$ (see [15]).*

Definition 6 provides a dual view of domain reduction: instead of speaking about values that are kept in the environments this dual view consider the values removed from the environments.

We consider the set of dual operators of R: let $\widetilde{R} = \{\widetilde{r} \mid r \in R\}$.

Definition 7. *The* upward closure *of $\overline{d}$ by $\widetilde{R}$, denoted by $\mathrm{CL}\!\uparrow(\overline{d}, \widetilde{R})$ 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 establishes the correspondence between downward closure of local consistency operators and upward closure of their duals.

Lemma 4. $\mathrm{CL}\!\uparrow(\overline{d}, \widetilde{R}) = \overline{\mathrm{CL}\!\downarrow(d, R)}$.

Proof.
$$\begin{aligned}\mathrm{CL}\!\uparrow(\overline{d}, \widetilde{R}) &= \min\{\overline{d'} \mid \overline{d} \subseteq \overline{d'}, \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d'}) \subseteq \overline{d'}\} \\ &= \min\{\overline{d'} \mid \overline{d} \subseteq \overline{d'}, \forall r \in R, d' \subseteq r(d')\} \\ &= \overline{\max}\{d' \mid d' \subseteq d, \forall r \in R, d' \subseteq r(d')\}\end{aligned}$$
□

In particular, $\mathrm{CL}\!\uparrow(\emptyset, \widetilde{R}) = \overline{\mathrm{CL}\!\downarrow(\mathbb{D}, R)}$ is the set of values removed by the program during the computation.

3.2 Deduction Rules and Explanations

Now, we associate rules in the sense of [15] with these dual operators. These rules are well suited to provide proof trees of value removals.

Definition 8. *A* deduction rule *is a rule $h \leftarrow B$ such that $h \in \mathbb{D}$ and $B \subseteq \mathbb{D}$.*

Intuitively, a deduction rule $h \leftarrow B$ can be understood as follows: if all the elements of B are removed from the environment, then h can be removed.

A very simple case is arc-consistency where B corresponds to the well-known notion of support of h. But in general (even for hyper arc-consistency) the rules are more intricate.

Note that the whole set of rules is only a theoretical tool to define explanations. But in practice, this set does not need to be given. The rules are hidden in the algorithms which implement the solver.

For each operator $r \in R$, we denote by $\mathcal{R}_r$ the set of deduction rules $\mathcal{R}_r = \{h \leftarrow B \mid h \in \widetilde{r}(B)\}$. This set defines $\widetilde{r}$, that is, $\mathcal{R}_r$ is such that: $\widetilde{r}(\overline{d}) = \{h \in \mathbb{D} \mid \exists B \subseteq \overline{d}, h \leftarrow B \in \mathcal{R}_r\}$. $\mathcal{R}_r$ contains a lot of redundant rules. There possibly exists many other sets of rules defining $\widetilde{r}$. For classical notions of local consistency one is always smaller and natural [14]. But here it is easier to consider $\mathcal{R}_r$. Note that deduction rules clearly appear inside the algorithms of the solver.

In [23] the proposed solver is directly something similar to the set of rules (it is not exactly a set of deduction rules because the heads of the rules do not have the same shape that the elements of the body).

With the deduction rules, we have a notion of proof tree [15]. We consider the set of all deduction rules for all local consistency operators of R: let $\mathcal{R} = \bigcup_{r \in R} \mathcal{R}_r$.

We denote by $\mathrm{cons}(h, T)$ the tree defined by: h is the label of its root and T the set of its sub-trees. The label of the root of a tree t is denoted by $\mathrm{root}(t)$.

Definition 9. *An* explanation *is a proof tree* $\mathrm{cons}(h, T)$ *with respect to* $\mathcal{R}$*; it is inductively defined by:* T *is a set of explanations with respect to* $\mathcal{R}$ *and* $(h \leftarrow \{\mathrm{root}(t) \mid t \in T\}) \in \mathcal{R}$*.*

Finally we prove that the elements removed from the domain are the roots of the explanations.

Theorem 1. $\overline{\mathrm{CL}{\downarrow}(\mathbb{D}, R)}$ *is the set of the roots of explanations with respect to* $\mathcal{R}$*.*

Proof. Let E be the set of the roots of explanations wrt $\mathcal{R}$. By induction on explanations $E \subseteq \min\{\overline{d} \mid \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d}) \subseteq \overline{d}\}$. It is easy to check that $\widetilde{r}(E) \subseteq E$. Hence, $\min\{\overline{d} \mid \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d}) \subseteq \overline{d}\} \subseteq E$. So $E = \mathrm{CL}{\uparrow}(\emptyset, \widetilde{R})$. $\square$

In [14] there is a more general result which establishes the link between the closure of an environment d and the roots of explanations of $\mathcal{R} \cup \{h \leftarrow \emptyset \mid h \in \overline{d}\}$. But here, to be lighter, the previous theorem is sufficient because we do not consider dynamic aspects. All the results are easily adaptable when the starting environment is $d \subset \mathbb{D}$.

It is interesting to note that explanations are defined by rules associated with the dual operators and that the dual operators are duals of local consistency operators, they are not duals of reduction operators.

4 Computed Explanations

The solver computes $\mathrm{CL}{\downarrow}(\mathbb{D}, R)$ by *chaotic iterations* (introduced in [3]) of local consistency operators. The principle of a chaotic iteration [24] is to apply the reduction operators one after the other in a fair way, that is such that no operator is forgotten. In practice this is often implemented thanks to a propagation queue. In our framework, the reduction operator associated with a local consistency operator r is the contracting operator $d \mapsto d \cap r(d)$. That is, at each step the solver chooses a local consistency operator r and removes from the current environment d the values which are not in $r(d)$.

The well-known result of confluence [6, 3] ensures that the limit of every chaotic iteration of the set of local consistency operators R is the downward closure of $\mathbb{D}$ by R. Since $\subseteq$ is a well-founded ordering (because $\mathbb{D}$ is a finite set), every chaotic iteration is stationary. So in practice the computation ends when a common fix-point is reached (or when an environment of a variable becomes empty). Moreover, implementations of solvers use various strategies in order to determine the order of invocation of the operators. These strategies are used to optimize the computation.

4.1 The OADymPPaC XML Trace Format

Some works in progress on finite domain constraint solvers concern the description of a trace format of the computation. The trace is intended to facilitate adaptation of visualization tools and debugging tools on different finite domain solvers. It enables these tools to be defined almost independently from finite domain solvers, and conversely, tracers to be built independently from these tools. The trace format is a description of the trace events that tracers should generate when tracing execution of a finite domain solver.

Traces are encoded in an XML format using the OADymPPaC DTD. It describes all the events of the solver [17]. This trace format is already tested in GNU-Prolog, PaLM and Chip. The purpose of the trace format is to fully define the communications between solvers and tools ensuring full compatibility of all the tools with all the solvers.

In general, traces are very large with thousands of events. Thus they are neither understandable nor readable by a human. Explanations are a good abstraction (visualization) of the trace to understand domain reduction and to understand why a value has been removed from a domain. Each event of the solver [17] is described by an XML element in the trace. The event corresponding to a domain reduction by the solver is described by the `<reduce>` element in the XML trace. Since it is the only interesting element to extract explanations from the trace, we will only describe the part of the OADymPPaC DTD concerning this element. The other elements are not presented in this paper.

Note that, in the trace, the search tree is described by the two elements `<choicepoint>` and `<back-to>`. Since we only focus on one branch of the search tree, we do not need to consider all the `<reduce>` elements of the trace but only those included in this branch. Thanks to the `<choicepoint>` and `<back-to>` elements, it is easy to filter the `<reduce>` corresponding to the concerned branch but this is not in the scope of this paper.

```
<!ELEMENT reduce ( update?, explanation*, state? ) >
<!ATTLIST reduce
   %eventAttributes;
   cident CDATA #IMPLIED
   vident CDATA #IMPLIED
   algo CDATA #IMPLIED >
```

The `%eventAttributes` of the `<reduce>` element is a set of optional attributes except the `chrono` attribute which is an integer indicating the event number in the trace. The `algo` attribute provides the name of the algorithm used to reduce the environment. This corresponds exactly to the notion of local consistency operator of our framework. In the arc-consistency case, the constraint and the variable whose environment is reduced are sufficient to know this local consistency operator and this attribute is rather used for global constraints, for example in Chip. The `cident` attribute identifies a constraint. The local consistency operator (`algo` attribute) comes from this constraint. The `vident` attribute identifies the variable whose environment is reduced. The `<reduce>`

element may contain an `<update>` element, a list of `<explanation>` elements and a `<state>` element.

The `<state>` element is not useful here and is not described.

```
<!ELEMENT update %valueList; >
<!ATTLIST update
   vident CDATA #REQUIRED
   type (ground | any | min | max |
         minmax | empty | val | nothing ) #IMPLIED >
```

The `vident` attribute of the `<update>` element identifies the variable whose environment is reduced. Note that this attribute is redundant with the `vident` optional attribute of the `<reduce>` element. The optional `type` attribute is not useful here. The `<update>` element may contain some `%valueList` elements which provide the values removed from the environment.

```
<!ELEMENT explanation ( valueList, (cause)*, constraints? ) >
```

The `<explanation>` element has no attribute. It contains some `%valueList` elements, a list of `<cause>` elements and it may contain a `<constraints>` element. The `<constraints>` element is not useful here. The `<explanation>` element explains the removal of the `%valueList` elements (the `%valueList` is a subset of the `%valueList` given in the `<update>` element). The list of `<cause>` elements provide the reason why the values of the `%valueList` have been removed.

```
<!ELEMENT cause %valueList; >
<!ATTLIST cause
   vident CDATA #REQUIRED
   type (ground | any | min | max |
         minmax | empty | val ) #IMPLIED >
```

The `type` attribute is not useful here. The `vident` attribute identifies a variable and the `%valueList` elements provide a set of values removed from the environment of the `vident` variable. The values in the `%valueList` elements of the `<explanation>` element have been removed from the environment of the `vident` variable of the `<update>` element because, for each `<cause>` element, the values in the `%valueList` elements have been removed from the environment of the `vident` variable. Next, an example, together with deduction rules, will clear up the semantics of these elements.

```
<!ENTITY % valueList "( values | range )*" >

<!ELEMENT values (#PCDATA) >

<!ELEMENT range EMPTY>
<!ATTLIST range
   from CDATA #REQUIRED
   to CDATA #REQUIRED >
```

The `%valueList` entity is a sequence of `<values>` and `<range>` elements.

4.2 Extraction of Explanations from a Trace

In order to facilitate understanding, let us consider the XML trace of Fig. 1 (`[...]` indicates some removed parts of the trace).

The `<new-variable>` element declares a variable x_2 whose environment contains the integer values from 0 to 10. The `<new-constraint>` element declares a constraint $c_3(x_7, x_4, x_2)$. The `<post>` element corresponds to the addition of the constraint $c_3(x_7, x_4, x_2)$ to the store.

Deduction rules can be extracted from the `<reduce>` element using the information provided by the `<explanation>` and `<cause>` elements. As said before, `<explanation>` elements express the cause of value removals. A set of values A is removed from the environment by a local consistency operator r because other values B were removed, that is $A \subseteq \widetilde{r}(B)$.

For example, from the `<reduce>` element of the trace given by the figure, it can be concluded that (where r_5 is the local consistency operator used to reduce the environment, and d is the environment before the reduction):

$$\begin{array}{l}
\{(x_4,0),(x_4,1),(x_4,2)\} \subseteq \widetilde{r}_5\left(\left\{\begin{array}{c}(x_2,0),(x_2,1),(x_2,2),(x_7,3),\\(x_7,4),(x_7,5),(x_7,6),(x_7,7),(x_7,8)\end{array}\right\}\right)\\
\{(x_4,3),(x_4,7),(x_4,9),(x_4,10)\} \subseteq \widetilde{r}_5\left(\left\{\begin{array}{c}(x_2,2),(x_2,3),(x_2,4),(x_2,4),\\(x_2,5),(x_2,6),(x_2,7),(x_2,8)\end{array}\right\}\right)\\
\{(x_4,4)\} \subseteq \widetilde{r}_5(\emptyset)\\
\{(x_4,5)\} \subseteq \widetilde{r}_5(\mathbb{D}|_{\{x_2,x_7\}} \setminus (d|_{\{x_2\}} \cup d|_{\{x_7\}}))
\end{array}$$

Note that for the value $(x_4, 4)$, there is no `<cause>` element in the `<explanation>` element. When no `<explanation>` element is given for a removed value, for example $(x_4, 5)$, the meaning is different. In the trace format, it is not mandatory to give an explanation for each removed value. If there is no `<explanation>` for a value h, it is always true that $h \notin r(d)$, where r is the local consistency operator used by the `<reduce>` and d is the environment just before this reduction. That is to say $h \in \widetilde{r}(\overline{d})$. In the example, as r_5 comes from the constraint $c_3(x_7, x_4, x_2)$ and reduces the environment of the variable x_4, it only depends on the current environment of the variables x_7 and x_2. Thus we have $(x_4, 5) \in \widetilde{r}_5(\mathbb{D}|_{\{x_2,x_7\}} \setminus (d|_{\{x_2\}} \cup d|_{\{x_7\}}))$, where d is the environment before the `<reduce>`.

An inclusion $A \subseteq \widetilde{r}(B)$ provides the set of deduction rules: $\{h \leftarrow B \mid h \in A\}$. Note that these rules are members of $\mathcal{R}_r$. In the example, this provides the following deduction rules of $\mathcal{R}_{r_5}$:

$$\begin{array}{cl}
(x_4,0) & \leftarrow \{(x_2,0),(x_2,1),(x_2,2),(x_7,3),(x_7,4),(x_7,5),(x_7,6),(x_7,7),(x_7,8)\}\\
(x_4,1) & \leftarrow \{(x_2,0),(x_2,1),(x_2,2),(x_7,3),(x_7,4),(x_7,5),(x_7,6),(x_7,7),(x_7,8)\}\\
(x_4,2) & \leftarrow \{(x_2,0),(x_2,1),(x_2,2),(x_7,3),(x_7,4),(x_7,5),(x_7,6),(x_7,7),(x_7,8)\}\\
(x_4,3) & \leftarrow \{(x_2,2),(x_2,3),(x_2,4),(x_2,4),(x_2,5),(x_2,6),(x_2,7),(x_2,8)\}\\
(x_4,7) & \leftarrow \{(x_2,2),(x_2,3),(x_2,4),(x_2,4),(x_2,5),(x_2,6),(x_2,7),(x_2,8)\}\\
(x_4,9) & \leftarrow \{(x_2,2),(x_2,3),(x_2,4),(x_2,4),(x_2,5),(x_2,6),(x_2,7),(x_2,8)\}\\
(x_4,10) & \leftarrow \{(x_2,2),(x_2,3),(x_2,4),(x_2,4),(x_2,5),(x_2,6),(x_2,7),(x_2,8)\}\\
(x_4,4) & \leftarrow \emptyset\\
(x_4,5) & \leftarrow \mathbb{D}|_{\{x_2,x_7\}} \setminus (d|_{\{x_2\}} \cup d|_{\{x_7\}})
\end{array}$$

```
<oadymppac xmlns="http://contraintes.inria.fr/OADymPPaC">

[...]

<new-variable chrono="15" vident="x_2">
    <range from="0" to="10">
</new-variable>

[...]

<new-constraint chrono="73" cident="c_3(x_7,x_4,x_2)">
</new-constraint>

[...]

<post chrono="82" cident="c_3(x_7,x_4,x_2)">
</post>

[...]

<reduce chrono="735" cident="c_3(x_7,x_4,x_2)" algo="r_5">
    <update vident="x_4">
       <range from="0" to="5"/><values>7 9 10</values>
    </update>
    <explanation>
       <range from="0" to="2"/>
       <cause vident="x_2">
          <values>0 1 2</values>
       </cause>
       <cause vident="x_7">
          <values>7 6 8</values><range from="3" to="5"/>
       </cause>
    </explanation>
    <explanation>
       <values>3 7</values><range from="9" to="10"/>
       <cause vident="x_2">
          <range from="2" to="8"/>
       </cause>
    </explanation>
    <explanation>
       <values>4</values>
    </explanation>
</reduce>

[...]

</oadymppac>
```

Fig. 1. An extract of a XML trace

As said before we only consider a branch of the search tree. Let us consider the sequence of `<reduce>` elements corresponding to this branch. They are ordered by the number of the `chrono` attribute.

The set of computed explanations extracted from the trace is defined inductively as follows:

Let n be the `chrono` of a `<reduce>` element, the set of computed explanations extracted at the step n is S_n:

$$S_n = \left\{ \text{cons}(h,T) \;\middle\|\; \begin{array}{l} h \leftarrow \{\text{root}(t) \mid t \in T\} \text{ is a deduction rule associated} \\ \qquad\qquad\qquad\quad \text{with the } \texttt{<reduce>} \text{ element} \\ \qquad\qquad\qquad\quad \text{of } \texttt{chrono } n \\ \text{and} \\ T \subseteq \bigcup_{m<n} S_m \end{array} \right\}$$

where $\bigcup_{m<n} S_m$ denotes the set of all the computed explanations extracted from the `<reduce>` elements of `chrono` m, $m < n$ (it is also possible to consider $S_m = \emptyset$ when m is the `chrono` of an element different from a `reduce`).

Theorem 2. *Let n be the* `chrono` *of a* `<reduce>` *element, let d_n be the environment at the step n:*

$$\overline{d_n} = \bigcup_{m \le n} \{\text{root}(t) \mid t \in S_m\}$$

Proof. By induction on n. □

Let k be the `chrono` of the last `<reduce>` element of the considered branch of the search tree, it is important to note that: $\bigcup_{m \le k} S_m$ denotes the set of all the computed explanations extracted from the trace. Each $t \in \bigcup_{m \le k} S_m$ declaratively explains the withdrawal of root(t).

The link between the set of explanations $\bigcup_{m \le k} S_m$ and the closure $\text{CL}\downarrow(\mathbb{D}, R)$ is given by:

- if $\text{CL}\downarrow(\mathbb{D}, R) \neq \emptyset$ then we have

$$\overline{\text{CL}\downarrow(\mathbb{D}, R)} = \bigcup_{m \le k} \{\text{root}(t) \mid t \in S_m\}$$

- if $\text{CL}\downarrow(\mathbb{D}, R) = \emptyset$ then there exists $x \in V$ such that

$$\mathbb{D}|_{\{x\}} \subseteq \bigcup_{m \le k} \{\text{root}(t) \mid t \in S_m\}$$

This result is similar to (and can be seen as an adaptation of) a result in [14] which shows the equality between $\overline{\text{CL}\downarrow(\mathbb{D}, R)}$ and the set of root of explanations computed at the limit of every chaotic iteration.

5 Conclusion

This paper shows how to extract explanations from a constraint program trace (using the OADymPPaC trace format). The deduction rules used to build explanations can be obtained thanks to the `<explanation>` element in the trace. The linking of these rules allows to build an explanation for each element which is removed during the computation.

It is important to note that if a tracer is not able to provide the `<explanation>` element, explanations can nevertheless be obtained. Indeed, the `<reduce>` element indicates the domains of variables used by the operator at this step. The body of the deduction rule used can then be replaced (with a loss of information of course) by all the elements which have been removed of theses domains during the computation.

Explanations provide us with a declarative view of the trace and domain reduction. They can be used for debugging: in [13], an adaptation of algorithmic debugging [25] for constraint programs is proposed. This adaptation is based on explanations and their tree structure. It will be possible to provide a declarative debugging tool, using explanations and algorithmic debugging, to locate errors in a constraint program from a trace of a computation giving an erroneous result. As the tool will use in input an OADymPPaC XML trace, it could be plugged in all the solvers that use this trace format (actually GNU-Prolog, PaLM, Chip). It is also possible to adapt existing tools based on explanations to use the OADymPPaC trace format.

This paper does not take into account the search tree of the labeling. In the OADymPPaC trace format, the search tree is coded by the two elements: `<choicepoint>` and `<back-to>`. A `<choicepoint>` element defines a node of the search tree and a `<back-to>` element indicates the start of a new branch from a node previously defined by a `<choicepoint>` element. It is sufficient to remove explanations built between a `<choicepoint>` and a `<back-to>` referencing the same node in order to compute the explanations of a new branch.

References

1. Van Hentenryck, P.: Constraint Satisfaction in Logic Programming. Logic Programming. MIT Press (1989)
2. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
3. Fages, F., Fowler, J., Sola, T.: A reactive constraint logic programming scheme. In Sterling, L., ed.: Proceedings of the Twelfth International Conference on Logic Programming, ICLP 95, MIT Press (1995) 149–163
4. Benhamou, F.: Heterogeneous constraint solving. In Hanus, M., Rofríguez-Artalejo, M., eds.: Proceedings of the 5th International Conference on Algebraic and Logic Programming, ALP 96. Volume 1139 of Lecture Notes in Computer Science., Springer-Verlag (1996) 62–76
5. Apt, K.R.: The role of commutativity in constraint propagation algorithms. ACM Transactions On Programming Languages And Systems **22** (2000) 1002–1036

6. Cousot, P., Cousot, R.: Automatic synthesis of optimal invariant assertions mathematical foundation. In: Symposium on Artificial Intelligence and Programming Languages. Volume 12(8) of ACM SIGPLAN Not. (1977) 1–12
7. Jussien, N.: e-constraints: Explanation-based constraint programming. In: CP 01 Workshop on User-Interaction in Constraint Satisfaction. (2001)
8. Boizumault, P., Debruyne, R., Jussien, N.: Maintaining arc-consistency within dynamic backtracking. In: Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, CP 00. Number 1894 in Lecture Notes in Computer Science, Springer-Verlag (2000) 249–261
9. Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic CSPs—application to configuration. Artificial Intelligence **135** (2002) 199–234
10. Felfernig, A., Friedrich, G.E., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. In Horn, W., ed.: Proceedings of the 14th European Conference on Artificial Intelligence, ECAI 2000, IOS Press (2000) 146–150
11. Debruyne, R., Ferrand, G., Jussien, N., Lesaint, W., Ouis, S., Tessier, A.: Correctness of constraint retraction algorithms. In Russell, I., Haller, S., eds.: FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference, AAAI Press (2003) 172–176
12. Jussien, N., Ouis, S.: User-friendly explanations for constraint programming. In: Proceedings of the 11th Workshop on Logic Programming Environments. (2001)
13. Ferrand, G., Lesaint, W., Tessier, A.: Towards declarative diagnosis of constraint programs over finite domains. In Ronsse, M., ed.: Proceedings of the Fifth International Workshop on Automated Debugging, AADEBUG2003. (2003) 159–170
14. Ferrand, G., Lesaint, W., Tessier, A.: Theoretical foundations of value withdrawal explanations for domain reduction. Electronic Notes in Theoretical Computer Science **76** (2002)
15. Aczel, P.: An introduction to inductive definitions. In Barwise, J., ed.: Handbook of Mathematical Logic. Volume 90 of Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company (1977) 739–782
16. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (xml) 1.0. Technical report, World Wide Web Consortium (W3C) (2004) available at `http://www.w3.org/TR/REC-xml`.
17. Deransard, P., Ducassé, M., Langevine, L.: A generic trace model fo finite domain solvers. In O'Sullivan, B., ed.: Proceedings of the 2nd international Workshop on User-Interaction in Constraint Satisfaction, UICS 02. (2002) 32–46
18. Diaz, D., Codognet, P.: The GNU-Prolog system and its implementation. In: ACM Symposium on Applied Computing. Volume 2. (2000) 728–732
19. Langevine, L., Ducassé, M., Deransart, P.: A propagation tracer for gnu-prolog: from formal definition to efficient implementation. In Palamidessi, C., ed.: Proceedings of International Conference on Logic Programming. Volume 2916 of Lecture Notes in Computer Science., Springer-Verlag (2003) 269–283
20. Jussien, N., Barichard, V.: The PaLM system: Explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000. (2000) 118–133
21. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A.: The constraint logic programming language CHIP. In: International Conference on Fifth Generation Computer Systems. (1988) 249–264

22. Lesaint, W.: Value withdrawal explanations: a theoretical tool for programming environments. In Tessier, A., ed.: 12th Workshop on Logic Programming Environments. (2002) 17–33
23. Apt, K.R., Monfroy, E.: Automatic generation of constraint propagation algorithms for small finite domains. In Jaffar, J., ed.: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, CP 99. Number 1713 in Lecture Notes in Computer Science, Springer-Verlag (1999) 58–72
24. Apt, K.R.: The essence of constraint propagation. Theoretical Computer Science **221** (1999) 179–210
25. Shapiro, E.: Algorithmic Program Debugging. ACM Distinguished Dissertation. MIT Press (1982)