Explications pour Comprendre la Trace d'un Solveur de Contraintes sur Domaines Finis [1]

Gérard Ferrand* — **Willy Lesaint**** — **Alexandre Tessier***

* *Laboratoire d'Informatique Fondamentale d'Orléans*
Université d'Orléans, France
http://www.univ-orleans.fr/SCIENCES/LIFO/
{Gerard.Ferrand,Alexandre.Tessier}@lifo.univ-orleans.fr

** *Laboratoire d'Études et de Recherche en Informatique d'Angers*
Université d'Angers, France
http://www.info.univ-angers.fr/info/leria.html
Willy.Lesaint@univ-angers.fr

RÉSUMÉ. *Certains travaux en cours sur les solveurs de contraintes sur domaines finis concernent l'implantation d'une trace XML du calcul, selon la DTD OADymPPaC (par exemple en GNU-Prolog, PA*LM, CHIP*). A cause de la grande taille des traces, même pour de petits exemples, des outils sont nécessaires pour comprendre cette trace. Des explications de retrait de valeurs (ou nogoods) sont utilisées pendant la réduction de domaine par certains solveurs. Dans ce papier, nous utilisons une formalisation des explications par des arbres de preuve dans un cadre point fixe basé sur l'itération d'opérateurs monotones de consistance locale. Les arbres de preuve donnent une vue déclarative des calculs par propagation de contrainte. Nous montrons comment des explications peuvent être extraites de façon naturelle du format de trace OADymPPaC. Les explications permettent une meilleure compréhension des réductions de domaine dans la trace.*

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, PA*LM, 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) are used during domain reduction by some solvers. 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.*

MOTS-CLÉS : *Explication, trace, contrainte, reduction de domaines*

KEYWORDS: *Explanation, trace; constraint, domain reduction*

1. Introduction

Depuis un certain nombre d'années la programmation par contraintes sur domaines finis [Van 89, TSA 93] a montré son efficacité pour résoudre des problèmes difficiles. Elle combine la déclarativité du style relationnel à l'efficacité des solveurs de contraintes sur domaines finis. Ces solveurs sont principalement basés sur la réduction de domaine par des notions de consistance. Cette méthode consiste à éliminer du domaine courant de certaines variables, des valeurs qui ne peuvent participer à une solution compte tenu des contraintes et du domaine courant des autres variables. Ces valeurs retirées sont caractérisées par une notion de consistance locale.

Certains travaux [FAG 95, BEN 96, APT 00] formalisent la réduction de domaine grâce à des opérateurs. Ces opérateurs réduisent les domaines de variables. En pratique ces opérateurs sont appliqués itérativement selon différentes stratégies. Les itérations chaotiques [COU 77] ont été utilisées pour décrire la réduction de domaine d'un point de vue théorique général. Elles garantissent la confluence, c'est à dire d'obtenir le même domaine réduit quel que soit l'ordre d'application des opérateurs. Dans ce cadre, la réduction de domaine peut être décrite en terme de points fixes et de clôtures.

D'autres travaux de la communauté contraintes s'intéressent aux notions d'explications[1] [JUS 01a] (ou nogoods). L'intérêt pour les explications dans ce paradigme de programmation est grandissant. Le principe des explications est de mémoriser des informations sur les retraits de valeurs. Elles ont déjà été utilisées avec succès pour les problèmes de satisfaction de contraintes dynamiques [JUS 01a], le backtracking dynamique [BOI 00], les problèmes de configurations [AMI 02, FEL 00], le retrait de contraintes [DEB 03], l'analyse d'échec [JUS 01b] ou le diagnostic déclaratif [FER 03].

Dans [FER 02] une base théorique des explications de retraits de valeur a été posée dans le cadre sus-mentionné des itérations chaotiques. A chaque opérateur de consistance locale est associé son opérateur dual. Chaque opérateur dual peut être défini par un système de règles au sens des définitions inductives [ACZ 77], une règle exprimant un retrait de valeur comme conséquence d'autres retraits de valeurs. Dans ce cadre, une explication est formalisée par un arbre de preuve utilisant ces ensembles de règles. Les explications ainsi définies fournissent une vue déclarative du calcul. Il faut noter que le seul rôle d'un solveur est de retirer des valeurs et que, ici, les explications sont des preuves de ces retraits. Les explications peuvent être considérées comme l'essence de la réduction de domaine.

Des travaux en cours sur les solveurs de contraintes sur domaines finis concernent l'implantation d'une trace [DER 02] XML [BRA 04] du calcul selon la DTD OADymPPaC[2] (déjà implantée en GNU-Prolog [DIA 00, LAN 03], PALM [JUS 00] et CHIP [DIN 88]).

---

1. Voir `http ://www.e-constraints.net/` pour plus de détails sur les explications.
2. `http ://contraintes.inria.fr/OADymPPaC/index_en.html` (OADymPPaC est un projet RNTL, consortium : COSYTEC, ILOG, EMN, INRIA, IRISA, LIFO)

A cause de leur grande taille, même pour des problèmes jouets, des outils sont requis pour comprendre les traces. Puisque les explications donnent seulement l'information nécessaire pour expliquer un retrait de valeur, elles sont un bon moyen pour contribuer à répondre à ce besoin et en particulier pour comprendre les réductions de domaines décrites dans une trace. La principale contribution du papier est de montrer comment des explications peuvent être extraites de façon naturelle à partir du format de trace OADymPPaC.

Le papier est organisé de la façon suivante : la section 2 (Préliminaires) donne brièvement un formalisme pour la réduction de domaine bien adapté pour définir les explications des événements de base que sont les retraits de valeurs d'un domaine. La section 3 (Explications) est consacrée à la notion d'explication. Elle définit les explications comme des arbres de preuve construits à partir d'ensembles de règles qui définissent les opérateurs duaux des opérateurs de consistance locale. La section 4 (Explications calculées) décrit d'abord rapidement le format de trace OADymPPaC en se concentrant sur les parties qui concernent la réduction de domaine. Ensuite il est montré comment des explications peuvent être extraites d'une trace donnée.

2. Préliminaires

Notre cadre théorique utilise des familles plutôt que des produits cartésiens car cela permet d'alléger les notations. En effet, les notions d'opérateurs monotones et de plus petit et plus grand point fixe sont plus faciles dans un cadre ensembliste où l'ordre est l'inclusion d'ensembles.

2.1. *Quelques notations*

Supposons fixés :

– un ensemble fini de symboles de *variable* V ;

– une famille $(D_x)_{x \in V}$ où chaque D_x est un ensemble fini non vide, D_x est le *domaine* de la variable x.

Nous allons considérer diverses *familles* $f = (f_i)_{i \in I}$. Une telle famille peut être identifiée à la *fonction* $i \mapsto f_i$, elle-même identifiée à l'*ensemble* $\{(i, f_i) \mid i \in I\}$.

Afin de définir des opérateurs monotones sur un ensemble de parties, on considère le *domaine* $\mathbb{D} = \bigcup_{x \in V}(\{x\} \times D_x)$, c'est à dire que $\mathbb{D}$ est l'ensemble de tous les couples possibles formés par une variable et sa valeur.

Une partie d de $\mathbb{D}$ est appelée *un environnement*. Nous notons $d|_W$ la *restriction* de d à un ensemble de variables $W \subseteq V$, c'est à dire que $d|_W = \{(x, e) \in d \mid x \in W\}$. Noter que, avec $d, d' \subseteq \mathbb{D}$, $d = \bigcup_{x \in V} d|_{\{x\}}$, et $(d \subseteq d') \Leftrightarrow (\forall x \in V, d|_{\{x\}} \subseteq d'|_{\{x\}})$. $d|_{\{x\}}$ est appelé l'*environnement de la variable* x (dans l'environnement d).

Un *tuple* (ou *valuation*) t est un environnement particulier tel que chaque variable apparaît seulement une fois : $t \subseteq \mathbb{D}$ et $\forall x \in V, \exists e \in D_x, t|_{\{x\}} = \{(x, e)\}$. Un *tuple* t *sur* un ensemble de variables $W \subseteq V$, est défini par $t \subseteq \mathbb{D}|_W$ et $\forall x \in W, \exists e \in D_x, t|_{\{x\}} = \{(x, e)\}$.

2.2. *Problèmes de satisfaction de contraintes et solutions*

Un *problème de satisfaction de contraintes* (CSP) sur $(V, \mathbb{D})$ consiste en :

– un ensemble fini de symboles de *contrainte* C ;

– une fonction $\mathrm{var} : C \to \mathcal{P}(V)$, qui associe à chaque symbole de contrainte l'ensemble des variables de la contrainte ;

– une famille $(T_c)_{c \in C}$ telle que : pour chaque $c \in C$, T_c est un ensemble de tuples sur $\mathrm{var}(c)$, T_c est l'ensemble des *solutions* de c.

Dans la suite nous supposons fixé un CSP $(C, \mathrm{var}, (T_c)_{c \in C})$ sur $(V, \mathbb{D})$.

Définition 1 *Un tuple t est une* solution *du CSP si $\forall c \in C, t|_{\mathrm{var}(c)} \in T_c$. On note* Sol *l'ensemble des solutions du CSP.*

2.3. *Programme et clôture*

Pour résoudre un CSP (c'est à dire pour trouver les solutions), un programme employant des techniques de réduction de domaine et de labeling est utilisé. Ici nous nous concentrons sur la réduction de domaine. Dans ce papier, l'intérêt est porté sur une unique branche de l'arbre de recherche. Dans ce contexte le labeling peut être considéré comme l'ajout de contraintes supplémentaires. Il serait possible de distinguer ces deux sortes de contraintes, cela ne conduit à aucune difficulté conceptuelle mais n'est pas réellement nécessaire ici (cela complique le formalisme et est donc omis). L'introduction du labeling dans ce formalisme est décrite dans [LES 02].

L'idée principale de la réduction de domaine est de retirer de l'environnement courant certaines valeurs qui ne peuvent participer à aucune solution de certaines contraintes, et qui ne peuvent donc participer à aucune solution du CSP. Ces retraits sont étroitement liés à une notion de consistance locale. Ceci peut être formalisé par des opérateurs de consistance locale.

Définition 2 *Un* opérateur de consistance locale r *est une fonction monotone*

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

Il faut noter que même les contraintes globales peuvent être formalisées par ces opérateurs.

Etant donné qu'il est nécessaire d'avoir un opérateur contractant pour réduire l'environnement (c'est à dire $r(d) \subseteq d$), nous allons aussi considérer $d \mapsto d \cap r(d)$ appelé *opérateur de reduction*. Mais, en général, les opérateurs de consistance locale ne sont pas des fonctions contractantes, comme il est montré plus loin pour définir leurs opérateurs duaux.

Un *programme* R sur $(V, \mathbb{D})$ est un ensemble d'opérateurs de consistance locale.

Dans la suite, on suppose fixé un programme R sur $(V, \mathbb{D})$.

On s'intéresse à des environnements particuliers : les point fixes communs des *opérateurs de réduction* $d \mapsto d \cap r(d)$, $r \in R$. Un tel environnement d vérifie $\forall r \in R$, $d = d \cap r(d)$, c'est à dire qu'aucune valeur ne peut être retirée par l'application des opérateurs.

Définition 3 *Soit $r \in R$. On dit qu'un environnement d est* r-consistant *si* $d \subseteq r(d)$.

Un environnement d est dit R-consistant *si* $\forall r \in R$, d *est* r*-consistant.*

La réduction de domaine à partir d'un environnement d par R revient à calculer le plus grand point fixe commun inclus dans d des opérateurs de réduction $d \mapsto d \cap r(d)$, $r \in R$.

Définition 4 *La* clôture descendante *de d par R, notée $\mathrm{CL}\downarrow(d, R)$, est le plus grand $d' \subseteq d$ tel que d' est R-consistant.*

En général, on s'intéresse à la clôture de $\mathbb{D}$ par R (le calcul débute à partir de $\mathbb{D}$), mais il est parfois nécessaire de pouvoir exprimer des clôtures de sous-ensembles de $\mathbb{D}$, par exemple pour prendre en compte les aspects dynamiques ou le labeling.

Par la définition 4, puisque $d \subseteq \mathbb{D}$:

Lemme 1 *Si d est R-consistant alors $d \subseteq \mathrm{CL}\downarrow(\mathbb{D}, R)$.*

2.4. *Liens entre CSP et programme*

Bien sûr le programme est relié au CSP, les opérateurs sont choisis pour "implanter" le CSP. En pratique cette correspondance s'exprime par le fait que le programme peut tester toute valuation. C'est à dire que, si toutes les variables sont instanciées, le programme doit être capable de répondre à la question : "cette valuation est-elle solution du CSP ?".

Définition 5 *Un opérateur de consistance locale r* préserve les solutions *d'un ensemble de contraintes C' si, pour chaque tuple t, $(\forall c \in C', t|_{\mathrm{var}(c)} \in T_c) \Rightarrow t$ est r-consistant.*

Si $C' \subseteq C''$ et si r préserve les solutions de C' alors r préserve les solutions de C''. En particulier,en considerant $C'' = C$, r préserve les solutions du CSP.

Par exemple, dans le cas classique de l'(hyper) arc-consistance, chaque contrainte c du CSP est implantée par un ensemble d'opérateurs de consistance locale R_c. Bien entendu, chaque $r \in R_c$ préserve les solutions de $\{c\}$.

Préserver les solutions est une propriété de correction des opérateurs. Une notion de complétude est utilisée pour choisir l'ensemble d'opérateurs "implantant" le CSP. Elle garantit de rejeter les valuations qui ne sont pas solutions des contraintes. Mais cette notion n'est pas nécessaire ici et n'est donc pas décrite.

Dans les lemmes suivant, on considère $S \subseteq \mathrm{Sol}$, c'est à dire que S est un ensemble de solutions du CSP et $\bigcup S$ ($= \bigcup_{t \in S} t$) sa projection sur $\mathbb{D}$.

Lemme 2 *Soit $S \subseteq \mathrm{Sol}$, si r préserve les solutions du CSP alors $\bigcup S$ est r-consistant.*

Preuve. $\forall t \in S, t \subseteq r(t)$ donc $\bigcup S \subseteq \bigcup_{t \in S} r(t)$. Mais $\forall t \in S, t \subseteq \bigcup S$ donc $\forall t \in S, r(t) \subseteq r(\bigcup S)$.

Dans la suite, on suppose que l'ensemble des opérateurs de consistance locale du programme fixé R préserve les solutions du CSP fixé.

Lemme 3 *Si $S \subseteq \mathrm{Sol}$ alors $\bigcup S \subseteq \mathrm{CL}{\downarrow}(\mathbb{D}, R)$.*

Preuve. par les lemmes 1 et 2.

Finallement, le corollaire suivant met en évidence le lien entre le CSP et le programme.

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

La clôture descendante est un sur-ensemble (une "approximation") de $\bigcup \mathrm{Sol}$ qui est lui-même la projection (une "approximation") de Sol. Mais la clôture descendante est l'ensemble le plus précis qui peut être calculé en utilisant un ensemble d'opérateurs de consistance locale dans le cadre de la réduction de domaine sans découper le domaine (sans arbre de recherche).

3. Explications

Une explication est un arbre de preuve d'un retrait de valeur ([FER 02] donne plus de détails sur les explications).

3.1. *Vue duale de la réduction de domaine*

Quelques notations sont tout d'abord nécessaires. Soit $\overline{d} = \mathbb{D} \setminus d$. Pour faciliter la compréhension, on utilisera toujours la notation $\overline{d}$ pour un sous-ensemble de $\mathbb{D}$ s'il désigne intuitivement un ensemble de valeurs retirées.

Définition 6 *Soit r un opérateur, on note $\widetilde{r}$ le* dual *de r défini pas : $\forall d \subseteq \mathbb{D}, \widetilde{r}(\overline{d}) = \overline{r(d)}$ (voir [ACZ 77]).*

La définition 6 donne une vision duale de la réduction de domaine : au lieu de parler de valeurs qui sont conservées dans les environnements cette vision duale considère les valeurs retirées de ces environnements.

On considère l'ensemble des opérateurs duaux de ceux de R : soit $\widetilde{R} = \{\widetilde{r} \mid r \in R\}$.

Définition 7 *La* clôture montante *de $\overline{d}$ par $\widetilde{R}$, notée $\mathrm{CL}{\uparrow}(\overline{d}, \widetilde{R})$ existe et est le plus petit $\overline{d'}$ tel que $\overline{d} \subseteq \overline{d'}$ et $\forall r \in R$, $\widetilde{r}(\overline{d'}) \subseteq \overline{d'}$.*

Le lemme suivant établit la correspondance entre la clôture descendante d'opérateurs de consistance locale et la clôture montante de leurs duaux.

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

Preuve.
$$\begin{array}{rcl} \mathrm{CL}{\uparrow}(\overline{d}, \widetilde{R}) & = & \min\{\overline{d'} \mid \overline{d} \subseteq \overline{d'}, \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d'}) \subseteq \overline{d'}\} \\ & = & \min\{\overline{d'} \mid \overline{d} \subseteq \overline{d'}, \forall r \in R, d' \subseteq r(d')\} \\ & = & \overline{\max}\{d' \mid d' \subseteq d, \forall r \in R, d' \subseteq r(d')\} \end{array}$$

En particulier, $\mathrm{CL}{\uparrow}(\emptyset, \widetilde{R}) = \overline{\mathrm{CL}{\downarrow}(\mathbb{D}, R)}$ est l'ensemble des valeurs retirées par le programme pendant le calcul.

3.2. *Règles de déduction et explications*

Des règles peuvent être associées, au sens de [ACZ 77], à ces opérateurs duaux. Ces règles sont adéquates pour définir les arbres de preuves de retraits de valeur.

Définition 8 *Une* règle de déduction *est une règle $h \leftarrow B$ telle que $h \in \mathbb{D}$ et $B \subseteq \mathbb{D}$.*

Intuitivement, une règle de déduction $h \leftarrow B$ peut être comprise ainsi : si tous les éléments de B sont retirés de l'environnement alors h peut être retiré.

Un cas très simple est l'arc-consistance où B correspond à la notion bien connue de support de h. Mais en général (même pour l'hyper arc-consistance), les règles peuvent être plus compliquées.

Il faut noter que l'ensemble entier des règles n'est qu'un outil théorique pour définir les explications. Mais en pratique cet ensemble n'a pas besoin d'être donné. Les règles sont cachées dans les algorithmes qui implantent le solveur.

Pour chaque opérateur $r \in R$, on note $\mathcal{R}_r$ l'ensemble des règles de déduction $\mathcal{R}_r = \{h \leftarrow B \mid h \in \widetilde{r}(B)\}$. Cet ensemble définit $\widetilde{r}$, c'est à dire que $\mathcal{R}_r$ vérifie : $\widetilde{r}(\overline{d}) = \{h \in \mathbb{D} \mid \exists B \subseteq \overline{d}, h \leftarrow B \in \mathcal{R}_r\}$. $\mathcal{R}_r$ peut contenir beaucoup de règles redondantes. Il peut exister beaucoup d'autres ensembles de règles définissant $\widetilde{r}$. Pour les notions classiques de consistance locale, il en existe toujours un qui est à la fois naturel et le plus petit [FER 02]. Cependant, il est plus facile ici de considérer $\mathcal{R}_r$. Rappelons que les règles de déduction apparaissent clairement dans les algorithmes du solveur. Dans [APT 99b], le solveur proposé est directement analogue à l'ensemble des règles (ce n'est pas exactement un ensemble de règles de déduction car les têtes des règles n'ont pas la même forme que les éléments du corps).

Grâce à ces règles de déduction, une notion d'arbre de preuve [ACZ 77] peut être obtenue. On considère l'ensemble de toutes les règles de déduction pour tous les opérateurs de consistance locale de R : soit $\mathcal{R} = \bigcup_{r \in R} \mathcal{R}_r$.

On note $\mathrm{cons}(h, T)$ l'arbre défini par : h est l'étiquette de sa racine et T l'ensemble de ses sous-arbres. L'étiquette de la racine d'un arbre t est notée $\mathrm{root}(t)$.

Définition 9 *Une* explication *est un arbre de preuve* $\mathrm{cons}(h, T)$ *par rapport à* $\mathcal{R}$ *; Celle-ci peut être défini inductivement par :* T *est un ensemble d'explications par rapport à* $\mathcal{R}$ *et* $(h \leftarrow \{\mathrm{root}(t) \mid t \in T\}) \in \mathcal{R}$.

Finalement on prouve que les éléments retirés du domaine sont les racines des explications.

Théorème 1 $\overline{\mathrm{CL}\downarrow(\mathbb{D}, R)}$ *est l'ensemble des racines des explications par rapport à* $\mathcal{R}$.

Preuve. Soit E l'ensemble des racines des explications par rapport à $\mathcal{R}$. Par induction sur les explications $E \subseteq \min\{\overline{d} \mid \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d}) \subseteq \overline{d}\}$. On vérifie facilement que $\widetilde{r}(E) \subseteq E$. D'où $\min\{\overline{d} \mid \forall \widetilde{r} \in \widetilde{R}, \widetilde{r}(\overline{d}) \subseteq \overline{d}\} \subseteq E$. Donc $E = \mathrm{CL}\uparrow(\emptyset, \widetilde{R})$.

Dans [FER 02], un résultat plus général établit le lien entre la clôture d'un environnement d et les racines des explications de $\mathcal{R} \cup \{h \leftarrow \emptyset \mid h \in \overline{d}\}$. Mais ici, pour alléger, le théorème précédent est suffisant car les aspects dynamiques ne sont pas considérées. Il faut noter que tous les résultats s'adaptent facilement pour un environnement de départ $d \subset \mathbb{D}$.

Il est intéressant de remarquer que les explications sont définies par des règles associées aux opérateurs duaux des opérateurs de consistance locale et non pas des opérateurs de réduction.

4. Explications Calculées

Un solveur calcule $\mathrm{CL}\downarrow(\mathbb{D}, R)$ par l'intermédiaire d'une *itération chaotique* (introduite dans [FAG 95]) d'opérateurs de consistance locale. Le principe d'une itération chaotique [APT 99a] est d'appliquer les opérateurs de réduction les uns après les autres de façon équitable, c'est-à-dire telle qu'aucun opérateur ne soit oublié. En pratique, elles sont souvent implantées par des queues de propagation. Dans notre cadre, l'opérateur de réduction associé à un opérateur de consistance locale r est l'opérateur contractant $d \mapsto d \cap r(d)$. A chaque pas de l'itération, le solveur choisit un opérateur de consistance locale r et retire de l'environnement courant d les valeurs qui n'apparaîssent pas dans $r(d)$.

Le résultat de confluence [COU 77, FAG 95] assure que la limite de toute itération chaotique d'un ensemble d'opérateurs de consistance locale R est la clôture descendante de $\mathbb{D}$ par R. Etant donné que $\subseteq$ est un ordre bien fondé (car $\mathbb{D}$ est un ensemble fini), toute itération chaotique est stationnaire. Ce qui entraine qu'en pratique les calculs s'arrêtent quand un point fixe commun est atteint (ou quand l'environnement d'une variable devient vide). De plus, les implémentations de solveurs utilisent diverses stratégies pour déterminer l'ordre d'invocation des opérateurs. Ces stratégies sont utilisées pour optimiser les calculs.

4.1. *Le format de trace XML OADymPPaC*

Certains travaux en cours concernant les solveurs sur domaines finis consistent à décrire un format de trace des calculs. La trace a pour but de faciliter l'adaptation des outils de visualisation et de mise au point pour différents solveurs sur domaines finis. Elle permet à ces outils d'être définis indépendemment des solveurs et inversement aux traceurs d'être construits indépendemment de ces outils. Le format de trace est une description des événements que les traceurs doivent générer pendant une résolution par un solveur sur domaines finis.

Les traces sont codées dans un format XML [BRA 04] utilisant la DTD OADymPPaC. Elle décrit tous les événements d'un solveur [DER 02]. Ce format de trace est déjà testé dans GNU-Prolog [DIA 00, LAN 03], PALM [JUS 00] et CHIP [DIN 88]. L'intérêt de ce format de trace est de définir complètement les communications entre les solveurs et les outils assurant ainsi une pleine compatibilité de tous les outils avec les solveurs.

En général, les traces sont énormes et contiennet des milliers d'événements pour des exemples simples. Elles ne sont donc pas compréhensibles ou exploitables par un humain. Les explications sont une bonne abstraction (visualisation) de la trace pour comprendre les réductions de domaines ou le retrait d'une valeur d'un domaine. Chaque événement du solveur est décrit par un élément XML de la trace [DER 02]. L'événement correspondant à une réduction de domaine par le solveur est décrit par l'élément `<reduce>` dans la trace XML. Etant donné que c'est le seul élément intéres-

sant pour extraire des explications de la trace, seule la partie de la DTD OADymPPaC concernant celui-ci sera détaillée.

Il faut noter que, dans la trace, l'arbre de recherche est décrit par les deux éléments <choicepoint> et <back-to>. Etant donné que l'on ne s'intéresse qu'à une branche de l'arbre de recherche, tous les éléments <reduce> de la trace ne seront donc pas considérés, mais seulement ceux inclus dans cette branche. Grâce aux éléments <choicepoint> et <back-to>, il est facile de filtrer les éléments <reduce> correspondant à la branche concernée mais ceci sort du cadre de ce papier.

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

%eventAttributes de l'élément <reduce> est l'ensemble des attributs optionnels excepté l'attribut chrono qui est un entier indiquant le numéro d'événement dans la trace. L'attribut algo fournit le nom de l'algorithme utilisé pour réduire l'environnement. Ceci correspond exactement à la notion d'opérateur de consistance locale dans notre cadre. Dans le cas de l'arc-consistance, la contrainte et la variable dont l'environnement est réduit sont suffisants pour connaître cet opérateur de consistance locale et cet attribut est surtout utilisé pour les contraintes globales (dans CHIP par exemple). L'attribut cident identifie la contrainte et vident la variable dont l'environnement est réduit. L'élément <reduce> peut également contenir un élément <update>, une liste d'éléments <explanation> et un élément <state>. L'élément <state> n'étant pas utile ici, il ne sera pas détaillé.

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

L'attribut vident de l'élément <update> identifie la variable dont l'environnement est réduit. On peut noter que cet attribut est redondant avec celui (optionel) de l'élément <reduce>. L'attribut optionel type n'est pas utile ici. L'élément <update> peut contenir des éléments %valueList fournissant les valeurs retirées de l'environnement.

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

L'élément <explanation> ne possède aucun attribut, il contient des éléments %valueList, une liste d'éléments <cause> et, de manière optionnelle, un élément

<constraints> qui n'est pas utile ici. L'élément <explanation> fournit l'explication du retrait des éléments %valueList (%valueList est un ensemble de valeurs donné par l'élément <update>). La liste d'éléments <cause> donne les raisons pour lesquelles les valeurs de %valueList ont été retirées.

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

L'attribut type n'est pas utile ici. L'attribut vident identifie une variable et les éléments %valueList donnent un ensemble de valeurs retirées de l'environnement de cette variable. Les valeurs des éléments %valueList de l'élément <explanation> ont été retirés de l'environnement de la variable vident de l'élément <update> car, pour chaque élément <cause>, les valeurs des éléments %valueList ont été retirées de l'environnement de la variable vident. Un exemple viendra éclairer la sémantique de ces éléments.

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

<!ELEMENT values (#PCDATA) >

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

L'entité %valueList est une séquence d'éléments <values> et <range>.

4.2. *Extraction des Explications à partir de la Trace*

Afin d'aider à la compréhension, considérons la trace XML de la figure 1 (certaines parties inutiles ici de cette trace sont remplacées par [...].

L'élément <new-variable> déclare la variable x_2 dont l'environnement contient les valeurs entières de 0 à 10. L'élément <new-constraint> déclare une contrainte $c_3(x_7, x_4, x_2)$. L'élément <post> correspond à l'ajout de la contrainte $c_3(x_7, x_4, x_2)$ dans le store.

Les règles de déduction peuvent être extraites de l'élément <reduce> en utilisant l'information fournie par les éléments <explanation> et <cause>. Comme évoqué plus tôt, les éléments <explanation> expriment la cause de retraits de valeurs. Un

```
<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>
```

Figure 1. *Un extrait de trace XML*

ensemble de valeurs A est retiré de l'environnement par un opérateur de consistance locale r car d'autres valeurs B ont été retirées, c'est-à-dire $A \subseteq \widetilde{r}(B)$.

Par exemple, à partir de l'élément `<reduce>` de la trace donnée dans la figure, il peut être conclu que : (r_5 est l'opérateur de consistance locale utilisé pour réduire l'environnement, et d est l'environnement avant la réduction)

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

On peut remarquer que pour la valeur $(x_4, 4)$, il n'existe pas d'élément `<cause>` dans l'élément `<explanation>`. Quand aucun élément `<explanation>` n'est donné pour le retrait d'une valeur, par exemple $(x_4, 5)$, la signification est différente. Dans le format de trace, il n'est pas obligatoire de fournir une explication pour chaque valeur retirée. S'il n'existe pas d'élément `<explanation>` pour une valeur h, il est toujours vrai que $h \notin r(d)$, où r est l'opérateur de consistance locale utilisé par le `<reduce>` et d est l'environnement avant la réduction. C'est-à-dire $h \in \widetilde{r}(\overline{d})$. Dans l'exemple, comme r_5 est issu de la contrainte $c_3(x_7, x_4, x_2)$ et réduit l'environnement de la variable x_4, il dépend seulement de l'environnement courant des variables x_7 et x_2. Par conséquent $(x_4, 5) \in \widetilde{r}_5(\mathbb{D}|_{\{x_2,x_7\}} \setminus (d|_{\{x_2\}} \cup d|_{\{x_7\}}))$, où d est l'environnement avant le `<reduce>`.

Une inclusion $A \subseteq \widetilde{r}(B)$ fournit l'ensemble des règles de déduction : $\{h \leftarrow B \mid h \in A\}$. On peut remarquer que ces règles appartiennent à $\mathcal{R}_r$. Dans l'exemple, Les règles de déductions suivantes peuvent être obtenues à partir de $\mathcal{R}_{r_5}$:

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

Comme déjà évoqué, on ne s'intéresse qu'à une unique branche de l'arbre de recherche. Si on considère la séquence d'éléments `<reduce>` correspondant à cette branche, ceux-ci sont ordonnés par le nombre de l'attribut `chrono`.

L'ensemble des explications calculées extraites de cette trace est alors défini inductivement comme suit :

Soit n le `chrono` d'un élément `<reduce>`, l'ensemble des explications calculées extraites à l'étape n est S_n :

$$S_n = \left\{ \mathrm{cons}(h, T) \;\middle\|\; \begin{array}{ll} h \leftarrow \{\mathrm{root}(t) \mid t \in T\} & \text{est une règle de déduction associée à l'élément } \texttt{<reduce>} \text{ de } \texttt{chrono } n \\ \text{et} & \\ T \subseteq \bigcup_{m<n} S_m & \end{array} \right\}$$

où $\bigcup_{m<n} S_m$ est l'ensemble de toutes les explications calculées extraites des éléments `<reduce>` de `chrono` m, $m < n$ (il est aussi possible de considérer $S_m = \emptyset$ quand m est le `chrono` d'un élément différent d'un `reduce`).

Théorème 2 *Soi n le* `chrono` *d'un élément* `<reduce>`*, soit d_n l'environnement à l'étape n :*

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

Preuve. Par induction sur n.

Soit k le `chrono` du dernier élément `<reduce>` de la branche considérée de l'arbre de recherche, il est important de remarquer que : $\bigcup_{m \leq k} S_m$ est l'ensemble de toutes les explications calculées extraites de la trace. Chaque $t \in \bigcup_{m \leq k} S_m$ explique déclarativement le retrait de $\mathrm{root}(t)$.

Le lien entre l'ensemble d'explications $\bigcup_{m \leq k} S_m$ et la clôture $\mathrm{CL}{\downarrow}(\mathbb{D}, R)$ est exprimé par :

– si $\mathrm{CL}{\downarrow}(\mathbb{D}, R) \neq \emptyset$ alors

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

– si $\mathrm{CL}{\downarrow}(\mathbb{D}, R) = \emptyset$ alors il existe $x \in V$ tel que

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

Ce résultat est similaire (et peut être vu comme une adaptation) à un résultat énoncé dans [FER 02] montrant l'égalité entre $\overline{\mathrm{CL}{\downarrow}(\mathbb{D}, R)}$ et l'ensemble des racines d'explications calculées à la limite de toute itération chaotique.

5. Conclusion

Ce papier montre comment extraire des explications à partir d'une trace de programme avec contraintes (utilisant le format de trace OADymPPaC). Ces explications fournissent une vue déclarative de la trace et de la réduction de domaines.

Dans [FER 03], une adaptation du diagnostic déclaratif [SHA 82] à la programmation par contraintes a été proposée. Cette adaptation est basée sur les explications et leur structure arborescente. Il sera possible d'obtenir un outil de mise au point déclaratif, en utilisant les explications et le diagnostic déclaratif, pour localiser des erreurs dans un programmme par contraintes à partir de la trace d'un calcul produisant un résultat erroné. Etant donné que l'outil utilisera une trace XML OADymPPaC en entrée, il pourra être utilisé pour tout solveur respectant ce format de trace (actuellement GNU-Prolog, PALM, CHIP). Il sera aussi possible d'adapter les outils existants basés sur les explications pour utiliser le format de trace OADymPPaC.

Ce papier n'aborde pas l'arbre de recherche utilisé pour le labeling. Dans le format de trace OADymPPaC, l'arbre de recherche est codé par les deux éléments : `<choicepoint>` et `<back-to>`. Un élément `<choicepoint>` définit un nœud de l'arbre de recherche et un élément `<back-to>` indique le commencement d'une nouvelle branche à partir d'un nœud défini précédemment par un `<choicepoint>`. Il suffit de retirer les explications construites entre les éléments `<choicepoint>` et `<back-to>` référençant le même nœud afin de pouvoir calculer les explications d'une nouvelle branche.

6. Bibliographie

[ACZ 77] ACZEL P., « An Introduction to Inductive Definitions », BARWISE J., Ed., *Handbook of Mathematical Logic*, vol. 90 de *Studies in Logic and the Foundations of Mathematics*, chapitre C.7, p. 739–782, North-Holland Publishing Company, 1977.

[AMI 02] AMILHASTRE J., FARGIER H., MARQUIS P., « Consistency Restoration and Explanations in Dynamic CSPs—Application to Configuration », *Artificial Intelligence*, vol. 135, n° 1–2, 2002, p. 199–234.

[APT 99a] APT K. R., « The Essence of Constraint Propagation », *Theoretical Computer Science*, vol. 221, n° 1–2, 1999, p. 179–210.

[APT 99b] APT K. R., MONFROY E., « Automatic Generation of Constraint Propagation Algorithms for Small Finite Domains », JAFFAR J., Ed., *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, CP 99*, n° 1713 Lecture Notes in Computer Science, Springer-Verlag, 1999, p. 58–72.

[APT 00] APT K. R., « The Role of Commutativity in Constraint Propagation Algorithms », *ACM Transactions On Programming Languages And Systems*, vol. 22, n° 6, 2000, p. 1002–1036.

[BEN 96] BENHAMOU F., « Heterogeneous Constraint Solving », HANUS M., ROFRÍGUEZ-ARTALEJO M., Eds., *Proceedings of the 5th International Conference on Algebraic and Logic Programming, ALP 96*, vol. 1139 de *Lecture Notes in Computer Science*, Springer-Verlag, 1996, p. 62–76.

[BOI 00] BOIZUMAULT P., DEBRUYNE R., JUSSIEN N., « Maintaining Arc-Consistency within Dynamic Backtracking », *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, CP 00*, n° 1894 Lecture Notes in Computer Science, Springer-Verlag, 2000, p. 249–261.

[BRA 04] BRAY T., PAOLI J., SPERBERG-MCQUEEN C. M., MALER E., YERGEAU F., « Extensible Markup Language (XML) 1.0 », rapport, 2004, World Wide Web Consortium (W3C), `http ://www.w3.org/TR/REC-xml`.

[COU 77] COUSOT P., COUSOT R., « Automatic synthesis of optimal invariant assertions mathematical foundation », *Symposium on Artificial Intelligence and Programming Languages*, vol. 12(8) de *ACM SIGPLAN Not.*, 1977, p. 1–12.

[DEB 03] DEBRUYNE R., FERRAND G., JUSSIEN N., LESAINT W., OUIS S., TESSIER A., « Correctness of Constraint Retraction Algorithms », RUSSELL I., HALLER S., Eds., *FLAIRS'03 : Sixteenth international Florida Artificial Intelligence Research Society conference*, AAAI Press, 2003, p. 172–176.

[DER 02] DERANSART P., DUCASSÉ M., LANGEVINE L., « A Generic Trace Model for Finite Domain Solvers », O'SULLIVAN B., Ed., *Proceedings of User Interaction in Constraint Satisfaction (UICS'02)*, 2002, p. 32–46.

[DIA 00] DIAZ D., CODOGNET P., « The GNU-Prolog System and its Implementation », *ACM Symposium on Applied Computing*, vol. 2, 2000, p. 728–732.

[DIN 88] DINCBAS M., VAN HENTENRYCK P., SIMONIS H., AGGOUN A., « The Constraint Logic Programming Language CHIP », *International Conference on Fifth Generation Computer Systems*, 1988, p. 249–264.

[FAG 95] FAGES F., FOWLER J., SOLA T., « A Reactive Constraint Logic Programming Scheme », STERLING L., Ed., *Proceedings of the Twelfth International Conference on Logic Programming, ICLP 95*, MIT Press, 1995, p. 149–163.

[FEL 00] FELFERNIG A., FRIEDRICH G. E., JANNACH D., STUMPTNER M., « Consistency-Based Diagnosis of Configuration Knowledge Bases », HORN W., Ed., *Proceedings of the 14th European Conference on Artificial Intelligence, ECAI 2000*, IOS Press, 2000, p. 146–150.

[FER 02] FERRAND G., LESAINT W., TESSIER A., « Theoretical Foundations of Value Withdrawal Explanations for Domain Reduction », *Electronic Notes in Theoretical Computer Science*, vol. 76, 2002, Elsevier.

[FER 03] FERRAND G., LESAINT W., TESSIER A., « Towards Declarative Diagnosis of Constraint Programs over Finite Domains », RONSSE M., Ed., *Proceedings of the Fifth International Workshop on Automated Debugging, AADEBUG2003*, 2003, p. 159–170.

[JUS 00] JUSSIEN N., BARICHARD V., « The PaLM System : Explanation-based Constraint Programming », *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, 2000, p. 118–133.

[JUS 01a] JUSSIEN N., « e-constraints : explanation-based Constraint Programming », *CP 01 Workshop on User-Interaction in Constraint Satisfaction*, 2001.

[JUS 01b] JUSSIEN N., OUIS S., « User-friendly Explanations for Constraint Programming », *Proceedings of the 11th Workshop on Logic Programming Environments*, 2001.

[LAN 03] LANGEVINE L., DUCASSÉ M., DERANSART P., « A Propagation Tracer for GNU-Prolog : from Formal Definition to Efficient Implementation », PALAMIDESSI C., Ed., *Proceedings of International Conference on Logic Programming*, vol. 2916 de *Lecture Notes in Computer Science*, Springer-Verlag, 2003, p. 269–283.

[LES 02] LESAINT W., « Value Withdrawal Explanations : a Theoretical Tool for Programming Environments », TESSIER A., Ed., *12th Workshop on Logic Programming Environments*, 2002.

[SHA 82] SHAPIRO E., *Algorithmic Program Debugging*, ACM Distinguished Dissertation, MIT Press, 1982.

[TSA 93] TSANG E., *Foundations of Constraint Satisfaction*, Academic Press, 1993.

[Van 89] VAN HENTENRYCK P., *Constraint Satisfaction in Logic Programming*, Logic Programming, MIT Press, 1989.