Un modèle complet pour la propagation et le labeling

LIFO, EMN

Gérard Ferrand, Willy Lesaint, Alexandre Tessier

public, rapport de recherche
D1.1.2

Abstract

Ce rapport a pour but de fournir un modèle théorique pour la propagation et le labeling. Ce modèle est adapté aux solveurs sur domaines finis utilisés par chacun des partenaires du projet: GNU-Prolog (INRIA), CHIP (Cosytec) et Choco (EMN). Un calcul est formalisé par un arbre dans lequel chaque branche correspond à une itération chaotique d'opérateurs et où chaque feuille peut-être décrite en terme de clôture. Ce modèle est bien adapté aux notions d'explications qui seront utiles pour la mise au point de programmes avec contraintes. Ce rapport fait suite à la réalisation d1.1.1 qui traitait de la réduction de domaine. Il sera étendu pour inclure un langage hôte dans la prochaine réalisation.

1 Introduction

La programmation par contraintes [8] est l'un des paradigmes les plus importants de ces dernières années. Elle combine à la fois déclarativité et efficacité grâce à des solveurs de contraintes implantés pour des domaines spécifiques. Cependant, il existe peu d'outils pour la mise au point de ces programmes. Celle-ci n'a été abordé que dans peu de travaux [9], Discipl [4]. Le projet RNTL OADymPPaC tente de répondre à ce problème.

La réalisation d1.1.1 [6] a fourni un fondement théorique à la réduction de domaines. Ce modèle est bien adapté aux solveurs sur les domaines finis [10] qui sont utilisés par les partenaires du projet: GNU-Prolog [5] (INRIA), CHIP [11] (COSYTEC) et Choco [7] (EMN). Un calcul y est formalisé par une itération chaotique d'opérateurs [3] et le résultat est décrit par une clôture. Ce modèle permet la définition d'explications qui seront utiles pour la mise au point de programmes avec contraintes.

La réduction de domaine se fait selon des notions de consistance. Aux contraintes d'un problème de satisfaction de contraintes peuvent être associés des ensembles d'opérateur de consistance local qui réduisent le domaine d'une ou plusieurs variables par rapport au domaine d'autres variables du problème. En pratique l'ordre d'application de ces opérateurs se fait selon leur apparition dans une queue de propagation. Le calcul s'achevant soit quand le domaine d'une variable devient vide (il n'y a pas de solutions, on parle alors d'échec), soit quand la queue de propagation est vide. Cette queue de propagation se généralise grâce à la notion d'itération chaotique. La limite de celle-ci étant la clôture du domaine initial par l'ensemble d'opérateurs de consistance local. Une fois cette phase de propagation effectuée et si le calcul a mené, non pas à un échec, mais à un domaine réduit à la clôture, il peut être souhaitable de s'approcher davantage des solutions.

Une deuxième technique est alors nécessaire: le labeling. Le labeling sur une variable consiste à partitionner le domaine de la variable en sous domaines. On ne considère plus alors le problème initial mais un ensemble de sous-problèmes plus "proches" des solutions. Chacun de ces sous-problèmes est alors considéré de manière individuelle. Les phases de réduction de domaine par des opérateurs de consistance local et par le labeling peuvent ainsi se succéder lors d'un calcul (un labeling peut être fait alors qu'une propagation n'a pas atteint une clôture). Il est cependant nécessaire de ne pas perdre de vue le problème initial, et même si chaque sous-problème est résolu de façon individuelle, c'est à l'ensemble que l'on s'intéresse. La notion d'arbre est alors naturelle pour représenter un calcul incluant du labeling.

Dans cette réalisation, des opérateurs de labeling sont définis dans la lignée des opérateurs de consistance local. Les itérations chaotiques sont modifiées pour les prendre en compte. Un calcul incluant opérateurs de consistance local et opérateurs de labeling calcule alors une clôture. Afin de ne pas perdre de solution, les opérateurs de labeling sont considérés par ensembles, chacun traitant un sous-domaine de la variable concernée. Un arbre de recherche est la "réunion" des calculs pour chacun des sous-domaines. Ce modèle reste bien adapté aux solveurs utilisés par les partenaires du projet et permet la définition de notions d'explications de retrait de valeur qui pourront être utilisées dans une optique de mise au point de programmes avec contraintes.

Au cours d'un calcul ne reposant que sur une réduction de domaine par des d'opérateurs de consistance local (sans labeling), l'arbre de recherche ne possède qu'une branche. Une valeur ne peut y être retirée qu'au plus une fois. Si un labeling intervient dans le calcul, l'arbre possède alors plusieurs branches et une valeur peut dès lors être retirée dans différentes branches de l'arbre. Les explications doivent désormais prendre en compte ces différents retraits. Des règles sont définies pour expliquer ces retraits et des arbres de preuves sont ensuite construits à partir de ces règles. Enfin, on montre comment extraire de tels arbres de preuve à partir d'un arbre de recherche.

La section 2 fournit les notations et les définitions de base. De plus, quelques notions de la réalisation d1.1.1 [6] sont rappelées. Les arbres de recherche sont définis dans la section 3. La section 4 décrit les règles qui permettront de construire les arbres de preuve dans la section 5.

2 Préliminaires

Cette section fournit les notations et les définitions de bases utilisées dans ce rapport. De plus, des résultats importants de la réalisation d1.1.1. [6] sont également rappelés.

Dans un premier temps, la définition classique d'un problème de satisfaction de contraintes (CSP) [10] est redonnée avec des notations ensemblistes.

Un CSP est fait d'une partie syntaxique et d'une partie sémantique. La partie sémantique est composée d'un ensemble fini de symboles de variables (ou variables) V, d'un ensemble fini de symboles de contraintes (ou contraintes) C et d'une fonction var : $C \to \mathcal{P}(V)$, qui associe à chaque contrainte l'ensemble de variables qui y apparaissent.

Pour la partie sémantique, quelques préliminaires sont nécessaires. Nous allons considérer plusieurs *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* $\{(i, f_i) \mid i \in I\}$.

Chaque variable a un ensemble de valeurs possibles, nous considérons alors une famille $(D_x)_{x \in V}$ où chaque D_x est un *ensemble fini non vide.*

Dans le but d'avoir des définitions simples et uniformes d'opérateurs sur des ensembles: le *domaine global* est défini par: $\mathbb{G} = \bigcup_{x \in V}(\{x\} \times D_x)$. Des sous-ensembles d de $\mathbb{G}$ seront considérés et appelés le *domaine.* La restriction d'un ensemble $d \subseteq \mathbb{G}$ à un ensemble de variables $W \subseteq V$ sera notée $d|_W$. Autrement dit: $d|_W = \{(x, e) \in d \mid x \in W\}$. Se donner un domaine $d \subseteq \mathbb{G}$ revient à se donner une famille $(d_x)_{x \in V}$ avec $d_x \subseteq D_x$: pour $x \in V$, $d_x = \{e \in D_x \mid (x, e) \in d\}$ sera appelé le *domaine de x.*

Les mêmes notations sont utilisées pour les tuples. Un *tuple* t sur $W \subseteq V$ est un d particulier tel que chaque variable de W apparaît seulement une fois: $t \subseteq \mathbb{G}|_W$ et $\forall x \in W, \exists e \in D_x, t|_{\{x\}} = \{(x, e)\}$. Pour chaque $c \in C$, T_c est un ensemble de tuples sur $\text{var}(c)$, appelés les solutions de c.

Définition 1 *Un* Problème de Satisfaction de Contraintes *(CSP) est défini par:*

- *un ensemble fini de variables V;*
- *un ensemble fini de contraintes C;*
- *une fonction var : $C \to \mathcal{P}(V)$;*
- *la famille $(D_x)_{x \in V}$;*
- *la famille $(T_c)_{c \in C}$.*

On peut noter qu'un tuple $t \in T_c$ est équivalent à la famille $(e_x)_{x \in \text{var}(c)}$ et que t est identifié à $\{(x, e_x) \mid x \in \text{var}(c)\}$.

Définition 2 *Un tuple t sur V est une* solution *du CSP si* $\forall c \in C, t|_{var(c)} \in T_c$.

Lors de la résolution d'un CSP, le solveur réduit le domaine selon les contraintes et des notions de consistance. Ces réductions sont effectuées par les applications successives d'opérateurs. Ces opérateurs définis ci-dessous réduisent le domaine de variables selon le domaine d'autres variables.

Définition 3 *Un* opérateur de consistance local *de type* (W_{in}, W_{out})*, avec* $W_{in}, W_{out} \subseteq V$ *est une fonction monotone* $r : \mathcal{P}(\mathbb{G}) \rightarrow \mathcal{P}(\mathbb{G})$ *telle que:* $\forall d \subseteq \mathbb{G}$,

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

Ces opérateurs étant associés à des contraintes, des notions de correction sont définies (voir [6]). Pour la suite, on notera R l'ensemble des opérateurs de consistance local associés aux contraintes C du CSP.

Ces opérateurs peuvent être définis par des ensembles de règles dont on rappelle la définition.

Définition 4 *Une* règle de déduction *de type* (W_{in}, W_{out}) *est une règle* $h \leftarrow B$ *telle que* $h \in \mathbb{G}|_{W_{out}}$ *et* $B \subseteq \mathbb{G}|_{W_{in}}$.

Intuitivement, une telle règle signifie que h peut être retiré du domaine si tous les éléments de B le sont déjà. Pour chaque opérateur $r \in R$ de type (W_{in}, W_{out}), on dénote par $\mathcal{R}_r$ un ensemble de règles de déduction de type (W_{in}, W_{out}) qui le définit (voir [6]). Pour chaque opérateur, plusieurs ensembles existent mais un est naturel.

Exemple 1 On considère le CSP défini par: $V = \{x, y, z\}$, $D_x = D_y = D_z = \{0, 1\}$ et $C = \{x \neq y, x \neq z, z \neq y\}$. La fonction var et les ensembles de solutions des contraintes sont implicitement exprimés par celles-ci. A chaque contrainte on associe deux opérateurs de consistance local, ces six opérateurs sont de type $(\{x\}, \{y\})$, $(\{x\}, \{z\})$, $(\{z\}, \{y\})$, $(\{y\}, \{x\})$, $(\{z\}, \{x\})$ et $(\{y\}, \{z\})$ et sont appelés respectivement $r_1, r_2, r_3, r_4, r_5, r_6$. Par exemple, r_1 est défini par les deux règles de déductions $(y, 0) \leftarrow \{(x, 1)\}$ et $(y, 1) \leftarrow \{(x, 0)\}$.

3 Arbre de recherche

Dans la réalisation d1.1.1 [6], un calcul est formalisé par une itération chaotique d'opérateurs de consistance local, c'est à dire par une suite de domaines. Le calcul commence à partir d'un domaine initial et le passage d'un domaine au suivant se fait par l'application d'un opérateur de consistance local. La limite de ce calcul est la clôture du domaine initial par l'ensemble d'opérateurs de consistance local (l'itération chaotique assurant que chaque opérateur est appliqué une infinité de fois). On s'intéresse maintenant à un calcul incluant du labeling.

A un certain point du calcul, le domaine d'une variable est restreint de façon arbitraire. Plus exactement, le domaine d'une variable est partitionné et on va considérer un calcul différent pour chaque domaine de cette partition. La notion d'arbre de recherche devient alors naturelle pour représenter le calcul dans son ensemble.

On s'intéresse tout d'abord au calcul le long d'une branche. De la même façon que l'on a défini les opérateurs de consistance local, on introduit des opérateurs de labeling.

Définition 5 *Un* opérateur de labeling *sur $x \in V$ est une fonction monotone et idempotente $l : \mathcal{P}(\mathbb{G}) \to \mathcal{P}(\mathbb{G})$ telle que: $\forall d \subseteq \mathbb{G}$,*

- $l(d)|_{V\setminus\{x\}} = \mathbb{G}|_{V\setminus\{x\}}$,
- $l(d)|_{\{x\}} = l(\mathbb{G})|_{\{x\}}$

On notera L un ensemble d'opérateurs de labeling.

La définition suivante est empruntée à Apt [2].

Un *run* est une séquence infinie d'opérateurs de $R \cup L$, c'est à dire, un run associe à chaque $i \in \mathbb{N}$ $(i \geq 1)$ un élément de $R \cup L$ dénoté par op^i. Un run est *équitable* si chaque $op \in R \cup L$ apparaît une infinité de fois. Une *itération descendante* d'un ensemble d'opérateurs par rapport à un run est alors définie comme suit.

Définition 6 *L'*itération *d'un ensemble d'opérateurs de consistance local R et de labeling L à partir de $d \subseteq \mathbb{G}$ selon le run $op^1, op^2, \ldots$ est la séquence infinie $d^0, d^1, d^2, \ldots$ définie inductivement par:*

1. *$d^0 = d$;*
2. *pour chaque $i \in \mathbb{N}$, $d^{i+1} = d^i \cap op^{i+1}(d^i)$.*

Sa limite est dénotée par $d^\omega = \cap_{i \in \mathbb{N}} d^i$. Une itération chaotique *est une itération selon un run équitable.*

Le lemme suivant exprime la limite d'une telle itération en terme de clôture.

Lemme 1 $d^\omega = CL \downarrow (CL \downarrow (d, L), R)$

Preuve. $d^\omega = CL \downarrow (d, R \cup L)$ par le lemme 2 de d1.1.1 et $CL \downarrow (d, R \cup L) = CL \downarrow (CL \downarrow (d, L), R)$ par idempotence des opérateurs de L. $\square$

Pour que la limite d'une itération soit une clôture, il n'est pas forcément nécessaire que celle-ci soit chaotique. En effet, il suffit que chaque $r \in R$ apparaisse une infinité de fois et que chaque $l \in L$ apparaisse au moins une fois (du fait de leur idempotence).

Les opérateurs de labeling ne conservent pas les solutions d'un CSP. Pour ne pas perdre de solutions, il faut considérer un ensemble d'opérateurs de labeling qui appliqués à un domaine donnent une partition de celui-ci.

Définition 7 *Un ensemble* $\{d_i \mid 1 \leq i \leq n\}$ *est une* partition de d sur x *si:*

- $\forall i, 1 \leq i \leq n,\ d|_{V \setminus \{x\}} \subseteq d_i|_{V \setminus \{x\}}$,
- $d|_{\{x\}} \subseteq \cup_{1 \leq i \leq n} d_i|_{\{x\}}$.

En pratique, les réductions de domaine par les opérateurs de consistance local et de labeling se succèdent de manière à rendre le calcul le plus efficace possible.

Un labeling sur $x \in V$ peut aller d'un simple partage du domaine de x en plusieurs sous-domaines (splitting) à une énumération complète du domaine de x (c'est à dire que chaque domaine de la partition est réduit à un singleton). La partition vérifie toujours la propriété suivante: $\forall i, 1 \leq i \leq n, d_i|_{\{x\}} \neq \emptyset$. De plus, afin de ne calculer qu'une occurrence de chaque solution, la partition sera choisie telle que: $\forall i, j, 1 \leq i \leq n, 1 \leq j \leq n, i \neq j, d_i|_{\{x\}} \cap d_j|_{\{x\}} = \emptyset$.

Le lemme suivant assure qu'aucune solution du CSP n'est perdue lors d'un labeling (chacune d'entre elle se retrouvera dans l'une des branches de l'arbre de recherche défini plus loin).

Lemme 2 *Si $t \subseteq d$ est une solution et $\{d_i \mid 1 \leq i \leq n\}$ est une partition de d alors $t \subseteq \cup_{1 \leq i \leq n} CL \downarrow (d_i, R)$.*

Preuve. évident. □

On s'intéresse maintenant aux noeuds de l'arbre de recherche. Ceux-ci sont caractérisés par le domaine (celui qui a été calculé jusque là), la profondeur dans l'arbre de recherche (si on ne s'intéresse qu'à une branche, c'est le *step* défini dans la réalisation d1.1.1), ce qui le relie à son père (soit un opérateur de consistance local, soit un opérateur de labeling) et un contexte. Le contexte est lié à la notion de labeling: il représente le domaine initial auquel seuls les opérateurs de labeling ont été appliqués.

Définition 8 *Une* étape de recherche *est un quadruplet* $(dom, cont, op, prof)$ *avec* $dom, cont \in \mathcal{P}(\mathbb{G})$, $op \in R \cup L \cup \{\perp\}$ *et* $prof \in \mathbb{N}$.

La seule donnée de la profondeur et du contexte permet de situer le noeud dans un arbre de recherche. Dans ces arbres de recherche, il existe deux types de transitions. Les transitions dues à l'application d'un opérateur de consistance local qui assure le passage d'un noeud vers un unique fils et les transitions dues au labeling qui assurent le passage d'un noeud à plusieurs fils (autant qu'il y a de domaines dans la partition considérée).

Définition 9 *Un* arbre de recherche *est un arbre dont chaque noeud est une étape de recherche définie inductivement par:*

- $(\mathbb{G}, \mathbb{G}, \perp, 0)$ *est la racine de l'arbre,*
- *si* $(d, cont, op, p)$ *est un noeud de l'arbre alors il a:*
  - *soit aucun fils;*
  - *soit un fils:* $(d \cap r(d), cont, r, p+1)$ *avec* $r \in R$*;*
  - *soit* n *fils:* $(d \cap l_i(d), cont \cap l_i(d), l_i, p+1)$ *avec* $\{l_i(d) \mid 1 \leq i \leq n\}$ *une partition de* d *et* $l_i \in L$*.*

Exemple 2 On considère le CSP et les mêmes opérateurs de consistance local $r_1, r_2, r_3, r_4, r_5, r_6$ que dans l'exemple 1. On définit les opérateurs de labeling $L = \{l_0, l_1\}$ dont l'application forme une partition de $\mathbb{G}$ sur x: $l_0(\mathbb{G}) = \{(x,0),(y,0),(y,1),(z,0),(z,1)\}$ et $l_1(\mathbb{G}) = \{(x,1),(y,0),(y,1),(z,0),(z,1)\}$. On applique tous les opérateurs de consistance local: aucune réduction de domaine ne se produit. Puis on effectue un labeling sur la variable x. Enfin, dans chacune des deux branches (de contexte d_0 et d_1), on ré-applique les opérateurs de consistance local dans cet ordre: r_1, r_2, r_3. Ces calculs donnent l'arbre de recherche de la figure 1 où:

- $d_0^1 = \{(x,0),(y,1),(z,0),(z,1)\}$
- $d_0^2 = \{(x,0),(y,1),(z,1)\}$
- $d_0^3 = \{(x,0),(z,1)\}$
- $d_1^1 = \{(x,1),(y,0),(z,0),(z,1)\}$
- $d_1^2 = \{(x,1),(y,0),(z,0)\}$
- $d_1^3 = \{(x,1),(z,0)\}$

Définition 10 *Un arbre de recherche sera dit* complet *si chaque feuille* $(d, cont, op, p)$ *est telle que:* $d = CL \downarrow (cont, R)$*.*

Des résultats plus complets sur les arbres de recherche sont fournis dans la première partie de la réalisation.

$$
\begin{array}{ccc}
 & (\mathbb{G}, \mathbb{G}, \perp, 0) & \\
 & | & \\
 & (\mathbb{G}, \mathbb{G}, r_1, 1) & \\
 & \vdots & \\
 & (\mathbb{G}, \mathbb{G}, r_6, 6) & \\
 / & & \backslash \\
(d_0, d_0, l_0, 7) & & (d_1, d_1, l_1, 7) \\
| & & | \\
(d_0^1, d_0, r_1, 8) & & (d_1^1, d_1, r_1, 8) \\
| & & | \\
(d_0^2, d_0, r_2, 9) & & (d_1^2, d_1, r_2, 9) \\
| & & | \\
(d_0^3, d_0, r_3, 10) & & (d_1^3, d_1, r_3, 10)
\end{array}
$$

Figure 1: Arbre de recherche

4 Règles

Maintenant qu'un calcul pour la propagation et le labeling a été formalisé, on s'intéresse aux explications de retrait de valeurs. Ces explications sont construites à partir de trois types de règles définies dans cette section. Dans un premier temps, des règles sont associées aux opérateurs de consistance local. A chaque opérateur de consistance local peut en effet être associé un ensemble de règles qui justifient le retrait d'une valeur par rapport au retrait d'autres valeurs (voir règles de déductions dans [6]). Ces règles sont ici généralisées pour prendre en compte le lieu du retrait dans l'arbre de recherche. Le labeling introduit lui aussi des règles. D'une part, une valeur peut être retirée directement par un labeling (des faits caractérisent ce retrait). D'autre part, une valeur peut être retirée dans différentes branches d'un arbre de recherche. Un troisième type de règle permet alors de réunir les explications de retrait d'une même valeur dans différentes branches.

Soit un CSP fixé. Pour les notions de consistance souhaitées, un ensemble d'opérateur R lui est associé et on considère l'ensemble des règles de déductions $\mathcal{R} = \cup_{r \in R} \mathcal{R}_r$.

La notation suivante est tout d'abord introduite: $\Gamma \vdash h$ avec $\Gamma \subseteq \mathcal{P}(\mathbb{G})$ et $h \in \mathbb{G}$. Par abus de langage, Γ sera appelé *contexte*. Intuitivement, $\Gamma \vdash h$ signifiera $\forall d \in \Gamma, h \notin CL \downarrow (d, R)$. Cette notion est rendue nécessaire par le labeling: chaque $d \in \Gamma$ correspondant à une branche de l'arbre de recherche. Une valeur pouvant être retirée dans différentes branches de l'arbre de recherche, le contexte indiquera donc toutes les branches dans lesquelles elle a été retirée. Remarquons que le labeling n'étant pas traitée dans la réalisation d1.1.1 [6], ce contexte n'était pas utile (il n'aurait toujours contenu que le domaine initial).

Pour les définitions suivantes, on considère $\Gamma_1, \ldots, \Gamma_n \subseteq \mathcal{P}(\mathbb{G})$, $d, d_1, \ldots, d_n \subseteq \mathbb{G}$ et $h, h_1, \ldots, h_n \in \mathbb{G}$. Les trois types de règles nécessaires à la construction d'arbres de preuve expliquant les retraits de valeurs lors de la résolution d'un CSP peuvent maintenant être données.

Les règles de déduction définies dans la section 2 sont généralisées afin d'inclure cette notion de contexte: à un opérateur de consistance local, on associe un ensemble de règles.

Définition 11 *L'ensemble de* règles de consistance local *associé à $r \in R$ est:*

$$\mathcal{C}_r = \{ \quad \frac{\{d_1\} \vdash h_1 \ldots \{d_n\} \vdash h_n}{\{d\} \vdash h} \quad \mid h \leftarrow \{h_1, \ldots, h_n\} \in \mathcal{R}_r, d \subseteq d_1 \cap \ldots \cap d_n\}$$

On notera $\mathcal{C} = \cup_{r \in R} \mathcal{C}_r$.

Lors de l'application d'un opérateur de labeling, des valeurs peuvent être directement retirées des domaines. Ces retraits seront exprimés par des faits.

Définition 12 *L'ensemble de* règles de labeling *associé à $l \in L$ est:*

$$\mathcal{L}_l = \{ \quad \frac{}{\{d\} \vdash h} \quad \mid h \notin l(\mathbb{G}), d \subseteq l(\mathbb{G})\}$$

On notera $\mathcal{L} = \cup_{l \in L} \mathcal{L}_l$.

La dernière règle sert à collecter les informations sur une valeur retirée. Si durant une résolution, une valeur est retirée dans différentes branches de l'arbre de recherche, un arbre de preuve pourra être construit pour chacun de ces retraits. La règle suivante permet de regrouper toutes les informations concernant le retrait d'une valeur dans différentes branches de l'arbre de recherche dans un seul arbre de preuve.

Définition 13 *L'ensemble de* règles d'union *est défini par:*

$$\mathcal{U} = \{ \quad \frac{\Gamma_1 \vdash h \ldots \Gamma_n \vdash h}{\Gamma_1 \cup \ldots \cup \Gamma_n \vdash h} \quad \}$$

5 Arbres de preuve

Dans cette section, la définition d'arbre de preuve [1] est rappelée. Ces arbres de preuves sont construits à partir des règles de la section précédente. Il est prouvé qu'il existe un tel arbre pour toute valeur retirée lors d'un calcul. Enfin, il est montré comment extraire les arbres de preuves à partir d'un calcul donné, c'est à dire à partir d'un arbre de recherche.

Définition 14 *Un* arbre de preuve *est inductivement défini par: cons*$(\Gamma \vdash h, T)$ *est un arbre de preuve si* T *est un ensemble d'arbres de preuve et*

$$\frac{\{root(t) \mid t \in T\}}{\Gamma \vdash h} \in \mathcal{C} \cup \mathcal{L} \cup \mathcal{U}$$

Le théorème suivant assure qu'il existe un arbre de preuve pour tout élément retiré lors d'un calcul.

Théorème 1 $\Gamma \vdash h$ *est racine d'arbre de preuve si et seulement si* $\forall d \in \Gamma, h \notin CL \downarrow (d, R)$.

Preuve. $\Rightarrow$: par induction sur chaque type de règle:

- pour les règles de consistance local: si $\forall i, 1 \leq i \leq n, h_i \notin CL \downarrow (d_i, R)$ alors $h_i \notin CL \downarrow (d_1 \cap \ldots \cap d_n, R)$ et donc (puisque $h \leftarrow \{h_1, \ldots, h_n\} \in \mathcal{R}$) $h \notin CL \downarrow (\{d_1 \cap \ldots \cap d_n\}, R)$;
- pour les règles de labeling, $h \notin d$ donc $h \notin CL \downarrow (d, R)$;
- évident pour les règles d'union.

$\Leftarrow$: Si $\forall i, 1 \leq i \leq n, h \notin CL \downarrow (d_i, R)$ alors (d'après [6]) il existe un arbre de preuve de racine h pour chaque d_i. Donc avec la notion de contexte, $\forall i, 1 \leq i \leq n, \{d_i\} \vdash h$ est racine d'un arbre de preuve. Donc par la règle d'union, $\{d_1, \ldots, d_n\} \vdash h$ est racine d'un arbre de preuve. □

Dans une optique de mise au point de programmes avec contraintes, il est important de pouvoir obtenir de tels arbres de preuve lors d'un calcul. La dernière partie de cette section s'attache à montrer comment extraire des arbres de preuve à partir d'un arbre de recherche.

Le parcours de l'arbre de recherche se fait en profondeur d'abord. On peut considérer chaque branche à part. Le calcul est donc ramené à une itération chaotique d'opérateurs de consistance local et de labeling. Lors de cette descente, on va donc construire un ensemble d'arbres de preuve à l'aide des règles de consistance local et de labeling. Chaque noeud de cette branche est identifié par sa profondeur. On notera donc $S^i\!\downarrow$ l'ensemble des arbres associés au noeud (d_i, c_i, op_i, i).

Ces ensembles sont définis inductivement par:

- $S^0\!\downarrow = \emptyset$;
- si $op_{i+1} \in R$ alors:

$$S^{i+1}\!\downarrow = S^i\!\downarrow \cup \{\mathrm{cons}(\{c_i\} \vdash h, T) \mid T \subseteq S^i\!\downarrow, h \in d_i, \quad \frac{\{\mathrm{root}(t) \mid t \in T\}}{\{c_i\} \vdash h} \in \mathcal{C}_{op_{i+1}}\}$$

- si $op_{i+1} \in L$ alors:
$$S^{i+1}{\downarrow} = S^i{\downarrow} \cup \{\mathrm{cons}(\{c_{i+1}\} \vdash h, \emptyset) \mid h \in d_i, \quad \frac{}{\{c_{i+1}\} \vdash h} \in \mathcal{L}_{op_{i+1}}\}$$

Une fois cette descente dans l'arbre effectuée, à chaque noeud (d, c, op, p) de l'arbre de recherche est associé un ensemble d'arbres de preuve que l'on note $S{\downarrow}\,(d, c, op, p)$.

Une deuxième phase consiste à récolter ces ensembles à chaque feuille et à les faire remonter dans l'arbre. A tout noeud (d, c, op, p) est associé un nouvel ensemble d'arbres de preuve $S{\uparrow}\,(d, c, op, p)$. Cet ensemble est défini inductivement par:

- si (d, c, op, p) est une feuille alors $S{\uparrow}\,(d, c, op, p) = S{\downarrow}\,(d, c, op, p)$;
- si $r \in R$ alors $S{\uparrow}\,(d, c, op, p) = S{\downarrow}\,(d \cup r(d), c, r, p+1)$;
- si $\{l_i(d) \mid 1 \leq i \leq n\}$ est une partition de d alors $S{\uparrow}\,(d, c, op, p) = S \cup S'$ avec $S = \cup_{1 \leq i \leq n} S{\uparrow}\,(d \cap l_i(d), c \cap l_i(c), l_i, p+1)$ et
$$S' = \{\mathrm{cons}(\Gamma \vdash h, T) \mid \quad \frac{\{root(t) \mid t \in T\}}{\Gamma \vdash h} \quad \in \mathcal{U}, T \subseteq S\}.$$

Corollaire 1 *Si l'arbre de recherche de racine* $(\mathbb{G}, \mathbb{G}\bot, 0)$ *est complet alors* $\{root(t) \mid t \in S{\uparrow}\,(\mathbb{G}, \mathbb{G}\bot, 0)\} = \{\Gamma \vdash h \mid \forall d \in \Gamma, h \notin CL\downarrow(d, R)\}$.

Preuve. par le théorème 1. □

Ces arbres de preuves sont des explications pour le retrait de leur racine.

Exemple 3 Soit le CSP et l'arbre de recherche donnés dans les exemples 1 et 2. Le retrait de la valeur 0 pour y est expliqué par l'arbre de preuve donné en Figure 2.

$$\dfrac{\dfrac{\dfrac{}{\{d_0\} \vdash (x,1)}}{\{d_0\} \vdash (y,0)} \qquad \dfrac{\dfrac{\dfrac{}{\{d_1\} \vdash (x,0)}}{\{d_1\} \vdash (z,1)}}{\{d_1\} \vdash (y,0)}}{\{d_0, d_1\} \vdash (y,0)}$$

Figure 2: Arbre de preuve pour $(y, 0)$

6 Conclusion

La réalisation d1.1.1 a fourni un modèle pour la réduction de domaine des solveurs sur domaines finis. Dans ce cadre les explications de retrait de valeur ont été décrites.

En complément de la réduction de domaine par les notions de consistance, les solveurs utilisent une deuxième technique (le labeling) afin d'atteindre les solutions du CSP (du moins afin de s'en approcher davantage). La présente réalisation s'appuie sur le travail réalisé dans d1.1.1 et y inclus le labeling. Un calcul est désormais modélisé par une itération chaotique d'opérateurs de consistance local et de labeling. Ces derniers sont idempotents et peuvent donc n'être appliquer qu'une fois lors du calcul. De plus, l'application d'un tel opérateur peut éliminer des solutions du CSP. Il est donc nécessaire de considérer un ensemble d'opérateur de labeling qui une fois appliqués forment une partition du domaine. Les calculs à partir de chacun des domaines de cette partition peuvent être regroupés dans un arbre de recherche.

Une notion de contexte a été introduite, elle permet de définir un ensemble de branches d'un arbre de recherche. Trois types de règles ont été décrits. Les premières sont liées aux opérateurs de réductions et expliquent le retrait d'une valeur par le retrait d'autres valeurs. Les secondes expliquent le retrait d'une valeur par l'application d'un opérateur de labeling. Et enfin, les troisièmes permettent un regroupement d'informations pour une valeur retirée dans différentes branches de l'arbre de recherche.

A partir de ces trois types de règles peuvent être construits des arbres de preuves. Ces arbres de preuves sont des explications pour les retraits de valeurs. Il existe de tels arbres pour toute valeur retirée lors d'un calcul. Finalement, on a montré comment de tels arbres de preuve pouvaient être extraits d'un arbre de recherche.

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

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

[4] P. Deransart, M. Hermenegildo, and J. Małuszyński, editors. *Analysis and Visualisation Tools for Constraint Programming.* Springer-Verlag, 2000.

[5] D. Diaz and P. Codognet. The GNU-Prolog system and its implementation. In *ACM Symposium on Applied Computing*, volume 2, pages 728–732, 2000.

[6] G. Ferrand, W. Lesaint, and A. Tessier. A model of constraint solvers by chaotic iteration adapted to value withdrawal explanations. Outils pour l'Analyse Dynamique et la mise au Point de Programmes avec Contraintes (délivrable d1.1.1), July 2000.

[7] F. Laburthe and the OCRE project. Choco: implementing a CP kernel. In *TRICS, Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Technical report TRA9/00, Singapore, 2000.

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

[9] 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.

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

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