Une approche de résolution de problèmes en logique basée sur des techniques de satisfaction de contraintes

Jean-Michel Richer, Jean-Jacques Chabrier

CRID - Université de Bourgogne
U.F.R des Sciences et Techniques
Bâtiment Mirande - 9, Avenue Alain Savary, B.P. 400
21011 Dijon Cedex
E-mail : {richer,chabrier}@crid.u-bourgogne.fr

Résumé : *Dans cet article, nous présentons une nouvelle approche de la résolution des problèmes exprimés en logique des prédicats du premier ordre sous forme normale conjonctive. Cette approche se fonde sur une modélisation et une résolution de ces problèmes par des techniques mises en oeuvre pour la résolution des problèmes de satisfaction de contraintes. Nous illustrons cette méthode au travers de quelques exemples type et présentons les premiers résultats expérimentaux du système implanté.*

Mots-clés : *Satisfaction de contraintes, Logique des prédicats, Démonstration automatique de théorèmes, Programmation en logique.*

---

1 Introduction

De nombreuses approches ont été proposées afin de résoudre les problèmes en logique du premier ordre. Elles ont pour but d'établir la consistance d'une formule ou d'un ensemble de clauses et s'articulent autour de deux propriétés inhérentes à la logique : syntaxe et sémantique. Les méthodes syntaxiques, basées sur l'utilisation de règles d'inférence (résolution, élimination de modèles [LOV 69], tableaux sémantiques [FIT 90], ...), procèdent par déductions successives de nouvelles formules conséquences logiques du problème initial. Les méthodes sémantiques opèrent par recherche d'un modèle d'une formule. Fondées sur le théorème de Herbrand, peu de travaux leur ont été consacrés en raison de la complexité de leur mise en oeuvre : l'univers de Herbrand est infini dans le cas général. Les premières tentatives [GIL 60] ont très vite montré leurs limites. Certains systèmes, notamment le logiciel FINDER [SLA 95], se restreignent donc à des domaines finis. Plusieurs autres approches mixtes combinent syntaxe et sémantique dans le but de diriger la recherche de la procédure de preuve [MAN 88, CHU 94], afin de construire un modèle du problème. La possibilité d'écrire des méta-interpréteurs en Prolog a également conduit à étendre le cadre de la programmation en logique (formes clausales, négation logique) dans le cadre de la démonstration automatique [PLA 82, LOV 87, STI 88].

L'objectif de cet article est de proposer une approche novatrice pour la résolution des problèmes en logique des prédicats du premier ordre exprimés sous forme normale conjonctive. Cette approche se fonde en réalité sur une coopération efficace entre les techniques de programmation en logique et celles employées en démonstration automatique. Ce rapprochement repose sur une modélisation et une résolution des problèmes de la logique des prédicats en tant que problèmes de satisfaction de contraintes. La méthode qui en résulte permet de rechercher simultanément une réfutation et un modèle d'un ensemble de clauses. les problèmes sont décrits dans un langage de spécification logique exécutable, déclaratif et sans contrôle.

Dans la prochaine section nous rappelons brièvement les notions fondamentales de la programmation par contraintes. En section 3, après avoir introduit des concepts et notations préliminaires, nous présentons une méthode originale utilisant des techniques de satisfaction de contraintes pour résoudre efficacement des programmes en logique du premier ordre. La section 4 illustre cette méthode au travers de quelques exemples type. Nous présentons ensuite les résultats expérimentaux du système implanté et concluons sur les perspectives futures de la méthode.

2 Le paradigme de la programmation par contraintes

2.1 *Rappels concernant la PPC*

Le cadre des PSC[1], introduit par Montanari [MON 74], sous-jacent à la PPC, est fondé sur un ensemble de techniques d'optimisation qui ont pour but l'élagage de l'arbre de recherche. Ces techniques permettent un parcours astucieux de l'espace de recherche (retour-arrière intelligent, cohérence des domaines) mais aussi la prise en compte d'une connaissance sur le moyen le plus efficace de résoudre un problème donné (heuristiques). La modélisation des PSC est déclarative et consiste en une énumération des relations (ou contraintes) impliquant des variables prenant leurs valeurs dans des domaines préalablement définis. Nous définissons à présent, de manière plus formelle, ce qu'est un PSC :

Définition 1 (Problème de Satisfaction de Contraintes) *un PSC $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est défini par*

1. où Problèmes de Satisfaction de Contraintes.

- un ensemble $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$ de n variables;

- un ensemble $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$ de n domaines finis associés à chaque variable de $\mathcal{X}$;

- un ensemble $\mathcal{C} = \{c_1, c_2, \cdots, c_m\}$ de m contraintes. Chaque contrainte $c_i = \{x_{i1}, \cdots, x_{ik}\}$ exprime une relation entre les variables $x_{i1}, \cdots, x_{ik}$.

Définition 2 (Satisfaisabilité d'une contrainte) *Une contrainte c_i est dite satisfaite s'il existe une affectation de ses variables compatible avec la relation qu'elle exprime.*

Définition 3 *Une solution S d'un PSC $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est une instanciation des variables de $\mathcal{X}$ telle que toutes les contraintes de $\mathcal{C}$ soient satisfaites. Un PSC est dit* consistant *ou* satisfaisable *s'il admet au moins une solution.*

Pour un PSC donné on peut vouloir : vérifier l'existence d'une solution (test de consistance), exhiber une solution (recherche d'un modèle), exhiber toutes les solutions (recherche de tous les modèles), déterminer une solution optimale pour un critère donné.

Exemple 1 - Soit le PSC donné par :

$$(PSC_1)\begin{cases} \mathcal{X} = & \{x_1, x_2, x_3\}, \\ \mathcal{D} = & \{d_1, d_2, d_3\}, \text{ avec } d_1 = d_2 = d_3 = \{0, 1, 2\} \\ \mathcal{C} = & \{x_1 + x_2 = x_3\}, \end{cases}$$

Les n-uplets solutions de ce PSC sont :

$$S_c = \{\ (0,0,0), (0,1,1), (1,0,1), (1,1,2), (0,2,2), (2,0,2)\ \}$$

3 Approche méthodologique

Dans cette section, les notions du calcul des prédicats sont supposées connues. La résolution des problèmes du calcul des prédicats soulève des difficultés inconnues en calcul propositionnel : notions d'unification et de domaine de Herbrand infini. Après avoir introduit des concepts et notations, nécessaires à la compréhension de notre approche, nous présentons un procédé de traduction des problèmes du calcul des prédicats sous forme normale conjonctive (CNF) en problèmes de satisfaction de contraintes.

3.1 *Concepts et notations*

Définition 4 (Problème en Logique des Prédicats) *On appelle problème en logique des prédicats (noté PLP), une formule en calcul des prédicats du premier ordre exprimé sous forme normale conjonctive. Un PLP est défini par un ensemble de clauses qui sont des disjonctions de littéraux. Chaque littéral est de la forme $p(t)$ ou $\neg p(t)$ où p est appelé prédicat et t est une liste de termes.*

Définition 5 (Notion de domaine étendu) *La définition 1 correspond au cadre standard de résolution des PSC pour lequel les domaines sont fixés une fois pour toute lors de la description du problème. Nous introduisons la notion de PSC à* domaines étendus, *pour lesquels les domaines sont susceptibles d'être étendus par propagation des contraintes au cours de la résolution.*

Exemple 2 Considérons le PSC_1 donné exemple 1. Dans le cadre d'une résolution standard, l'affectation de $x_1 = 1$ et $x_2 = 2$ est inconsistante puisque $x_1 + x_2 = 3$ ne fait pas partie du domaine d_3 de x_3. Dans le cas d'un PSC à domaines étendus, cette affectation est légale et la valeur 3 est ajoutée au domaine d_3 de la variable x_3, fournissant ainsi l'ensemble provisoire de solutions :

$$S_e = S_c \cup \{\ (1,2,3),\ (2,1,3),\ (2,2,4)\ \}$$

En réitérant le procédé de résolution, les domaines d_1, d_2, d_3 pourront être étendus jusqu' à obtenir $d_1 = d_2 = d_3 = \mathbb{N}$.

3.2 *Modélisation PSC*

Définition 6 (Traduction PLP en PSC$_e^{\mathcal{H}}$) *Un PLP P est représenté sous forme d'un PSC à domaines étendus sur le domaine de Herbrand $\mathcal{H}$ de P (on note PSC$_e^{\mathcal{H}}$), en appliquant les règles de traduction suivantes :*

- *pour chaque prédicat p de du PLP P :*
  - *$\mathcal{X}$ est constitué des variables notées x_p^+ et x_p^- à valeurs dans $\mathcal{H}$ telles que :*
    - *$x_p^+ = \{t,\ p(t)\}$, x_p^+ représente l'ensemble des termes dont l'interprétation dans $\mathcal{H}$ de PLP donne la valeur* vrai *au littéral $p(t)$,*
    - *$x_p^- = \{t,\ \neg p(t)\}$, x_p^- représente l'ensemble des termes dont l'interprétation dans $\mathcal{H}$ donne la valeur* faux *au littéral $p(t)$ dans PLP,*

- *$\mathcal{D}$ est formé des domaines $d_{x_p^+}$ et $d_{x_p^-}$ associés aux variables x_p^+ et x_p^-, définis sur $\mathcal{H}$;*

- *$\mathcal{C}$ contient la contrainte de consistance $x_p^+ \cap x_p^- = \emptyset$ qui garantit le maintien de la consistance au niveau logique;*

- *les clauses unitaires de P définissent initialement les domaines des variables de $\mathcal{X}$:*

  - $d_{x_p^+} = d_{x_p^+} \cup \{t_0\}$, *pour toute clause $p(t_0)$ de P,*

  - $d_{x_p^-} = d_{x_p^-} \cup \{t_0\}$, *pour toute clause $\neg p(t_0)$ de P,*

- *les clauses non unitaires sont modélisées sous forme de* contraintes déductives *(cf. définition 8). Une clause de longueur k, $p_1(t_1) \vee \cdots \vee p_k(t_k)$ est représentée par k contraintes déductives comme suit :*

$$\begin{cases} [\![p_1(t_1) \# p_2(t_2), \cdots, p_k(t_k)]\!] \\ [\![p_2(t_2) \# p_1(t_1), \cdots, p_k(t_k)]\!] \\ \cdots \\ [\![p_k(t_k) \# p_1(t_1), \cdots, p_{k-1}(t_{k-1})]\!] \end{cases}$$

Par la suite, si nous ne connaissons pas le signe du littéral p, nous noterons x_p la variable associée au littéral p, et $x_{\neg p}$ la variable associée au littéral complémentaire de p, soit $\neg p$.

Définition 7 (Base d'unificateurs) *On appelle base d'unificateurs du littéral $p(t)$, et on note $\sigma^p(t)$, l'ensemble des unificateurs θ tels que $\theta(t) \in d_{x_{\neg p}}$. On dit aussi que θ* falsifie $p(t)$. *Par extension, pour toute clause de la forme $c = p_1(t_1) \vee \cdots \vee p_k(t_k)$, on notera $\Sigma(p_1(t_1) \vee \cdots \vee p_k(t_k)) = \{\sigma,\ \sigma = \sigma^{p_1}(t_1) \circ \cdots \circ \sigma^{p_k}(t_k)\ \}$ la base d'unificateurs correspondants.*

Définition 8 (Contrainte Déductive) *On appelle contrainte déductive et on note $[\![p_1(t_1) \# p_2(t_2), \cdots, p_k(t_k)]\!]$ la contrainte qui s'exprime par :*

$$x_{p_1} = x_{p_1} \cup \{\sigma(t_1),\ \forall \sigma\ \in \Sigma(p_2(t_2), \cdots, p_k(t_k))\}$$

$p_1(t_1)$ *est appelée* tête *et* $p_2(t_2), \cdots, p_k(t_k)$ corps *de la contrainte. Une contrainte déductive est dite satisfaite si l'ensemble $\Sigma(p_2(t_2), \cdots, p_k(t_k))$ est non vide.*

Exemple 3 La transformation d'un problème logique en un problème de satisfaction de contraintes est illustré figure 1.

$$
(PLP) \quad \begin{cases} p(a) \\ \neg p(X) \vee q(X) \\ \neg q(X) \end{cases}
$$

$$
(PSC_e^{\mathcal{H}}) \quad \begin{cases} \mathcal{X} = \{ \, x_p^-, x_p^+, x_q^-, x_q^+ \, \}; \\ \mathcal{D} = \{ \, d_{x_p^-} = \emptyset, \, d_{x_p^+} = \{a\}, \, d_{x_q^-} = \{X\}, \, d_{x_q^+} = \emptyset \, \} \\ \mathcal{C} = \{ \, [\![\neg p(X) \# \, q(X)]\!], \, [\![\, q(X) \# \, \neg p(X)]\!] \, \} \end{cases}
$$

Figure 1 : *Problème logique et PSC$_e^{\mathcal{H}}$ correspondant*

3.3 *Résolution*

Dons son principe, la résolution d'un $PSC_e^{\mathcal{H}}$ consiste à générer toutes les clauses unitaires conséquences logiques du PLP initial. L'algorithme de test de consistance (fig. 2) sert de modèle aux algorithmes de recherche d'un ou de tous les modèles du problème.

L'approche que nous proposons s'inscrit dans la continuité de travaux réalisés au sein de notre laboratoire [CHA 95, CHA 96]. Son intérêt réside dans la possibilité d'intégrer des techniques heuristiques utilisées par les solveurs de PSC, afin d'aboutir rapidement à une solution en tentant de générer le moins de conséquences logiques possibles. En particulier, la sélection de la variable x_p ou de la contrainte déductive à satisfaire sont paramétrables par des heuristiques fonctions de la taille des domaines. La taille du domaine d'une variable peut aussi se révéler un critère de choix pour la recherche de valeurs inconsistantes (c.f.section **4.2**). Il est également possible de paramétrer la stratégie de recherche : stratégie en profondeur ou en largeur d'abord ou par approfondissements successifs [KOR 85].

Nous expliquons à présent le comportement de l'algorithme de test de consistance. On commence par construire l'ensemble V de toutes les variables de domaine non vide, i.e. l'ensemble des variables potentiellement aptes à satisfaire des contraintes déductives. Tant que V est non vide, on choisit une variable $x_p \in V$ dont on évalue l'ensemble des contraintes déductives pour lesquelles $x_{\neg p}$ apparaît dans le corps de la contrainte, i.e. l'ensemble des contraintes susceptibles de produire de nouveaux termes. Si une contrainte déductive $[\![\, q(t) \# \, p_{i1}(t_{i1}), \cdots, \neg p(t_p), \cdots, p_{ik}(t_{ik})]\!]$ est satisfaite, i.e. il existe un unificateur σ falsifiant $p_{i1}(t_{i1})$ à $p_{ik}(t_{ik})$, le terme $\mu = \sigma(t)$ est ajouté au domaine de la variable x_q s'il n'en fait pas partie. Cependant, si μ appartient déjà au domaine du complémentaire de x_q (soit $x_{\neg q}$, on déduit alors que l'ensemble de clauses est insatisfaisable. Une telle valeur est appelée *valeur inconsistante*. Violer une contrainte de consistance revient à produire, par application du principe de résolution, les clauses $q(\mu)$ et $\neg q(\mu)$. Une nouvelle application du principe de résolution

Algorithme TESTDECONSISTANCE(P)
 entrée : un $PSC_e^{\mathcal{H}} = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$
 sortie : vrai si le problème est consistant, faux sinon.
 debut
  $V = \{x_p \in \mathcal{X}$ tels que $d_{x_p} \neq \emptyset\}$
  $Consistant = vrai$
  tant que $(V \neq \emptyset)$ et $Consistant$ **faire**
   choisir $x_p \in V$
   $V = V - \{x_p\}$
   $C(x_p) = \{$ ensemble des contraintes déductives telles que
      $\neg p$ figure dans le corps de la contrainte $\}$
   tant que $C(x_p) \neq \emptyset$ et $Consistant$ **faire**
    choisir $c \in C(x_p)$
    $C(x_p) = C(x_p) - \{c\}$
    $Consistant$ = SatisfaireContrainte(c, x_p)
   fintq
  fintq
  retourne $Consistant$
 fin

Fonction SATISFAIRECONTRAINTE(c, x_p)
 entrée : $c = [\![\, q(t) \# \, p_{i1}(t_{i1}), \cdots, \neg p(t_p), \cdots, p_{ik}(t_{ik})]\!]$, une variable x_p
 sortie : vrai si le PLP est consistant, faux sinon
 debut
  pour tout unificateur σ falsifiant $p_{i1}(t_{i1}), \cdots, \neg p(t_p), \cdots, p_{ik}(t_{ik})$ **faire**
   $\mu = \sigma(t)$
   si $\mu \in d_{x_{\neg q}}$ **alors** **retourne** $faux$ **finsi**
   $d_{x_q} = d_{x_q} \cup \{\mu\}$
   si $x_q \notin V$ **alors** $V = V \cup \{x_q\}$ **finsi**
  finpour
  retourne $vrai$
 fin

Figure 2 : *Algorithme de recherche de consistance*

entre ces deux clauses génère la clause vide. Le processus est réitéré jusqu'à ce que V soit vide (dans le cas de la recherche de modèles) où qu'une valeur inconsistante soit produite (test de consistance ou recherche de modèles).

3.4 *Complétude et terminaison*

La méthode de résolution exposée est incomplète : la satisfaction des contraintes déductives s'apparente, en termes de règle d'inférence logique, à l'utilisation de l'UR-résolution[2], complète uniquement pour les clauses de Horn[CHA 70]. En outre, le caractère semi-décidable de la logique du premier ordre pose le problème de la terminaison dans le cas de PLP consistants dont l'univers de Herbrand est non fini (ex. $\{p(a), \neg p(X) \vee p(f(X))\}$). Il est cependant envisageable de fixer des critères de terminaison portant, par exemple, sur la taille des domaines des variables, sur le temps de résolution du PSC, ou sur une analyse du traitement des variables afin de déterminer d'éventuels bouclages.

Exemple 4 Considérons la résolution du $PSC_e^{\mathcal{H}}$ de la figure 1. L'ensemble V est constitué des variables x_p^+ et x_q^-. L'ensemble $C(x_p^+)$ est formé de l'unique contrainte : $\{[\![\, q(X) \# \neg p(X)]\!]\}$ satisfaite par l'unificateur $\sigma^{\neg p}(X) = \{X \mapsto a\}$. Le terme $\mu = a$ doit être ajouté au domaine de x_q^+. mais a appartient à $d_{x_q^-}$ puisque a s'unifie avec $X \in d_{x_q^-}$. Le problème est donc inconsistant.

4 Applications

Dans cette section, nous allons illustrer au travers de trois exemples, les possibilités de modélisation du langage de description de problèmes que nous utilisons ainsi que l'apport des techniques de résolution que nous utilisons.

4.1 *Théories des groupes*

Ce problème est tiré de [CHA 73] et est généralement référencé sous l'appellation Chang et Lee $n°$ 2. On veut démontrer que dans un groupe $(E, \circ)$, si chaque élément est son propre inverse alors le groupe est commutatif. Pour exprimer ce problème sous forme CNF on introduit le prédicat $p(X, Y, Z)$ qui s'interprète comme $X \circ Y = Z$. On traduit ensuite les différents axiomes du problème numérotés figure 3 : élément neutre à gauche (1) et à droite (2), produit d'un élément par lui même égal à l'élément neutre (3), associativité (4,5). Le but de ce problème est de vérifier la consistance du système par réfutation. La démarche employée est classique en démonstration automatique de théorèmes et consiste à démontrer l'inconsistance de l'ensemble d'axiomes (1-5) augmenté de la négation de la conclusion $\exists a, b \in E$ tels que $(a \circ b = c) \wedge (a \circ b \neq c)$

2. Unit Resulting resolution

```
Probleme Groupe;

Clauses
 (1)  p(e,X,X).
 (2)  p(X,e,X).
 (3)  p(X,X,e).
 (4)  p(U,Z,W) | -p(X,Y,U) | -p(Y,Z,V) | -p(X,V,W).
 (5)  p(X,V,W) | -p(X,Y,U) | -p(Y,Z,V) | -p(U,Z,W).
 (6)  p(a,b,c).
 (7) -p(b,a,c).

Trouve  consistance;

Liste p-, p+;
```

Figure 3 : *Théorie des groupes*

```
Consistance ? non
Xp+ = { (b,a,c), (c,a,b), (b,c,a), (a,c,b), (c,b,a),
        (a,b,c), (e,X,X), (X,e,X), (X,X,e) }
Xp- = { (b,a,c) }

valeur(s) inconsistante(s) pour Xp = { (b,a,c) }
```

Figure 4 : *Résultat du test de consistance pour le problème* `Groupe`

(6,7). La primitive `Trouve` sert à définir l'objectif de la résolution, i.e. recherche de consistance ou de modèle(s).

La recherche de consistance du problème *Groupe*, présentée figure 4, a permis de déterminer l'inconsistance de l'ensemble de clauses. On a effectivement pu montrer qu'il existait une valeur inconsistante (b, a, c) pour le prédicat p.La primitive `Liste` fournit les domaines des variables x_p^- et x_p^+. On remarquera que les termes de la forme $(e, a, a), (e, b, b), \ldots$ ont été éliminés par subsumbtion.

4.2 *Coloriage de Carte*

On désire colorier en trois couleurs *rouge, vert* ou *bleu* les zones rectangulaires du graphe planaire de la figure 5 de manière à ce que deux zones adjacentes ne soient pas de la même couleur.

Nous introduisons la notation $A_1 \leftarrow A_2, \cdots, A_n$ qui est équivalente à $A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$. Cette notation qui ressemble à l'expression des clauses de Horn en Prolog s'en différencie par l'absence de restriction quant au signe du premier atome A_1. Dans le cas du coloriage de carte, cette notation permet d'introduire une forme de contrôle pour la résolution. En effet, la clause correspondante est traduite par une seule contrainte déductive $[\![A_1 \# \neg A_2, \cdots, \neg A_n]\!]$ au lieu de n contraintes.

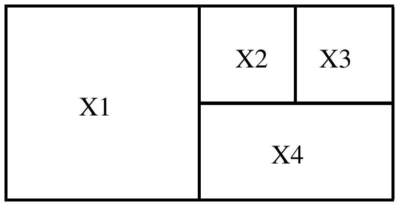

Figure 5 : *Coloriage de Carte*

```
Probleme Coloriage;

Clauses
   coul = {  ( rouge, vert  ), ( rouge, bleu  ),
             ( vert,  rouge ), ( vert,  bleu  ),
             ( bleu,  rouge ), ( bleu,  vert  ) }

  -coul = {  (X,X)  }

   colorie( X1, X2, X3, X4 ) <- coul(X1,X2), coul(X1,X4),
                                coul(X2,X3), coul(X2,X4),
                                coul(X3,X4).

Trouve  Tous_les_modeles;

Liste colorie;
```

Figure 6 : *Problème de coloriage*

Le prédicat *coul* définit les couples de valeurs autorisés par le coloriage. Il est ici exprimé d'une manière tout à fait originale, qui consiste à énumérer les termes le définissant. Cette modélisation sous forme ensembliste ne représente qu'une facilité d'écriture. Le littéral $\neg coul(X, X)$, exprimé par $(-coul = \{(X, X)\})$ peut être ajouté à la spécification pour permettre une vérification plus efficace de l'existence des couples de valeurs générés. Ainsi, l'unification de (X_1, X_2) avec $(rouge, vert)$

puis l'unification de (X_1, X_4) avec $(rouge, vert)$ instancie le littéral $coul(X_2, X_4)$ en $coul(vert, vert)$. Il faut alors vérifier que le couple $(vert, vert)$ appartient au domaine de X^+_{coul}, ce qui n'est pas le cas. Cela est réalisable de deux façons différentes : soit en parcourant d^+_{coul} pour s'aperçevoir que $(vert, vert)$ n'en fait pas partie; soit en montrant,ce qui est plus efficace, que $(vert, vert)$ appartient à d^-_{coul} puisque $(vert, vert)$ s'unifie avec $(X, X) \in d^-_{coul}$.

Le résultat de l'exécution du programme Coloriage est présenté figure 7. On obtient effectivement l'ensemble des quadruplets de couleurs solutions du problème. Il eut été également possible, tout comme en Prolog, de spécifier un but à résoudre, par exemple : $Trouve\ colorie(rouge, X, Y, Z)$, pour obtenir toutes les solutions telles que le rectangle $X1$ soit colorié en rouge.

```
Consistance ? oui
Xcolorie+ = {  (bleu,vert,bleu,rouge),
               (bleu,rouge,bleu,vert),
               (vert,bleu,vert,rouge),
               (vert,rouge,vert,bleu),
               (rouge,bleu,rouge,vert),
               (rouge,vert,rouge,bleu) }
Xcolorie- = { }
```

Figure 7 : *Résultat de la recherche de modèles pour le problème* `coloriage`

4.3 *Schubert's Steamroller*

Le problème appelé Schubert's Steamroller (cf. fig 8) illustre une faiblesse de certains démonstrateurs de théorèmes employant le chaînage avant. Dans [LOV =93], il est fait état de leur difficulté à traiter les problèmes dont le nombre de clauses annexes n'ayant que peu d'intérêt pour le but à résoudre est important. Comme l'a montré Ramsay[RAM 91], une légère modification du problème avec introduction de la clause : $\neg animal(X) \vee \neg animal(Y) \vee quicker(X, Y) \vee smaller(Y, X)$ juste avant la clause marquée par (*) figure 8, conduit souvent à une explosion combinatoire de l'espace de recherche [RAM 91].

5 Résultats

Pour valider notre approche, nous avons développé en C++, un prototype appelé *Sacre* [3] sur une station SUN Sparc Ultra. *Sacre* traite les exemples exposés en

3. pour SAtisfaction de Contraintes et REsolution

```
Probleme Steamroller;

Clauses
  -wolf(X) | -fox(Y) | -likes(X,Y).
  -wolf(X) | -grain(Y) | -likes(X,Y).
  -bird(X) | -snail(Y) | -likes(X,Y).
  -animal(X) | -animal(Y) | -likes(X,Y) | -grain(Z) | -likes(Y,Z).
  -wolf(w).            -animal(X) <- wolf(X).
  -fox(f).             -animal(X) <- fox(X).
  -bird(b).            -animal(X) <- bird(X).
  -snail(s).           -animal(X) <- snail(X).
  -caterpillar(c).     -animal(X) <- caterpillar(X).
  -grain(g).           -plant(X)  <- grain(X).
  -plant(i(X)) <- snail(X).  plant(h(X))<- caterpillar(X).
  smaller(X,Y) <- caterpillar(X), bird(Y).
  smaller(X,Y) <- snail(X),bird(Y).
  likes(X,Y) <- bird(X), caterpillar(Y).
  smaller(X,Y) <- bird(X), fox(Y). likes(X,i(X)) <- snail(X).
  smaller(X,Y) <- fox(X), wolf(Y). likes(X,h(X)) <- caterpillar(X).
  (*) -animal() | -animal(Y) |-smaller(Y,X) | -plant(W) |
       -likes(Y,W) | -plant(Z) | likes(X,Y) | likes(X,Z).

Trouve  Consistance;
```

Figure 8 : *Schubert's steamroller*

quelques millisecondes sans qu'aucune heuristique de choix n'ait encore été implantée. Nous avons fait figurer table 1, les résultats comparatifs entre Otter[4], SATCHMO[5] et *Sacre* pour les problèmes exposés. Otter [WOS 92] et SATCHMO [MAN 88] sont des démonstrateurs automatiques pour la logique du premier ordre.

Otter, écrit en langage C, utilise différentes règles d'inférence articulées autour du principe de résolution (UR-resolution, hypperresolution, binary resolution) , ainsi qu'un traitement de l'égalité par paramodulation. Il est considéré comme l'un des programmes les plus complets et les plus efficaces de sa catégorie. SATCHMO, fonctionnant sous Prolog, utilise la génération de modèles en tant que règle d'infèrence et peut être considéré comme un moteur de système expert.

On vérifiera aisément que les temps de résolution sont du même ordre de grandeur quoique les problèmes traités ne permettent pas de conclure de manière définitive quant à l'efficacité de la méthode. Le problème nommé *Steamroller 1* correspond à l'exemple de la figure 8. *Steamroller 2* est une version augmentée de *Steamroller 1*, comme expliqué section **4.3**. A noter que SATCHMO a été exécuté sous ECLiPSe.

4. Organized Techniques for Theorem-proving and Efficient Research
5. SATisfiability CHecking by MOdel generation

L'introduction d'heuristiques de choix devrait permettre à *Sacre* de se révéler encore plus performant. Il sera alors envisageable de s'intéresser à des problèmes plus difficiles, notamment ceux de la librairie TPTP[6] qui regroupe un ensemble de problèmes type pour les démonstrateurs automatiques de théorèmes. *Sacre* est actuellement en cours de réécriture [BUT 94].

Problème	Otter	Satchmo	Sacre
Coloriage	50	370	100
Chang et Lee 2 (Groupe)	170	390	120
Steamroller 1	240	180	130
Steamroller 2	610	>30min	330

Table 1 : *Résultats comparatifs en ms*

6 Conclusion

Nous avons présenté un procédé novateur pour la résolution des problèmes de la logique des prédicats du premier ordre exprimés sous forme normale conjonctive. Nous avons axé notre approche autour des techniques de satisfaction de contraintes principalement parce que le paradigme des PSC permet la mise en oeuvre de nombreux types de contrôles : filtrage, heuristiques de choix, qui ont pour but d'améliorer les performances des solveurs. D'autres expérimentations, concernant notamment l'implantation d'heuristiques de choix permettront de témoigner de la validité de notre démarche. La méthode qui en résulte permet de rechercher simultanément une réfutation et un modèle d'une formule et réalise un rapprochement coopératif de la programmation en logique et de la déduction automatique. Enfin, signalons que l'architecture du système *Sacre* reste ouverte et est susceptible d'accueillir d'autres outils de résolution pour la logique des prédicats permettant ainsi de constituer un véritable atelier de travail.

7 Remerciements

Nous tenons à adresser nos remerciements aux rapporteurs dont les remarques et commentaires ont permis d'améliorer de manière significative le contenu et la portée de cet article.

6. Thousands of Problems for Theorem Provers

Bibliographie

[BUT 94] BUTLER R. M. et OVERBEEK R. A., « Formula Databases for High-performance Resolution/Paramodulation Systems », *Journal of Automated Reasoning*, vol. 12, n° 2:139–156, 1994.

[CHA 70] CHANG C. L., « The Unit Proof and The Input Proof in Theorem Proving », *Journal of the ACM*, vol. 17, n° 4:698–707, 1970.

[CHA 73] CHANG C. L. et LEE R. C. T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, inc., London, 1973.

[CHA 95] CHABRIER J., JULIARD V., et CHABRIER J.-J., « SCORE(FD/B) : An Efficient Complete Local-Based Search Method for Satisfiability Problems », In *CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France, 1995.

[CHA 96] CHABRIER J., RICHER J.-M., et CHABRIER J.-J., « Resolution of Structured SAT Problems with SCORE(FD/B) », In *Proceegings of the CP'96 Workshop on Constraint Programming Applications*, Cambridge, Massachusetts, U.S.A., 1996.

[CHU 94] CHU H. et PLAISTED D. A., « Semantically Guided First-Order Theorem Proving Using Hyper-Linking », In *Proceedings of CADE-12*, 1994.

[FIT 90] FITTING M., *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, 1990.

[GIL 60] GILMORE P. C., « A Proof Method for Quantification Theory : Its Justification and Realization », *IBM JRD*, vol. 1, :28–35, 1960.

[KOR 85] KORF R. E., « Depth-First Iterative-Deepening : an Optimal Admissible Tree Search », *Artificial Intelligence*, vol. 27, n° 1:97–109, 1985.

[LOV 69] LOVELAND D. W., « A Simplified Format for the Model Elimination Procedure », *Journal of the ACM*, vol. 16, :363–349, 1969.

[LOV 87] LOVELAND D. W., « Near-Horn Prolog », In *Proceedings of ICLP-87*, pp. 456–469, 1987.

[LOV =93] LOVELAND D. W., REED D. W., et WILSON D. S., « SATCHMORE : SATCHMO with RRElevancy », *Journal of Automated Reasoning*, vol. 14, n° 2:325–351, 1995.

[MAN 88] MANTHEY R. et BRY F., « Satchmo : A Theorem Prover Implemented in Prolog », In *Proceedings of CADE-88*, pp. 415–434, Springer-Verlag, 1988.

[MON 74] MONTANARI U., « Networks of Constraints: Fundamental Properties and Applications to Picture Processing », *Information Science*, vol. 7, n° 2:95–132, 1974.

[PLA 82] PLAISTED D. A., « A Simplified Problem Reduction Format », *Artificial Intelligence*, vol. 18, :227–261, 1982.

[RAM 91] RAMSAY A., « Generating Relevant Models », *Journal of Automated Reasoning*, vol. 7, :359–368, 1991.

[SLA 95] SLANEY J., FINDER (Finite Domain Enumerator) : version 3.0 notes and guides, Rapport technique, Centre for Information Science Research, Australian National University, Canberra, 1995.

[STI 88] STICKEL M. E., « A Prolog Technology Theorem Prover : Implementation by an Extended Prolog Compiler », *Journal of Automated Reasoning*, vol. 4, :353–380, 1988.

[WOS 92] WOS L., BOYLE J., LUSK E., et OVERBEEK R., *Automated Reasoning - Introduction and Applications*, McGraw-Hill, Inc., New York, 1992.