Explications et aide à la restauration de la cohérence dans les CSP interactifs : application à la configuration

Jérôme Amilhastre
LIRMM,
161 rue Ada
34392 Montpellier Cedex 5
email: `amilhast@lirmm.fr`

Hélène Fargier
IRIT,
118 route de Narbonne
31062 Toulouse Cedex
email: `fargier@irit.fr`

Pierre Marquis
CRIL/Université d'Artois,
rue de l'Université, S.P. 16
62307 Lens Cedex
email: `marquis@cril.univ-artois.fr`

Abstract

Un produit configurable peut être représenté par un CSP dont les solutions sont les produits réalisables. Dans le cadre d'un processus de configuration interactif, la génération purement automatique de solution n'est pas pertinente : c'est en effet l'utilisateur qui fixe lui-même les valeurs des variables, au terme d'une séquence d'ajouts et de retraits de contraintes unaires. Dans ce contexte, le rôle d'un système d'aide à la décision est d'assurer que les choix de l'utilisateur sont cohérents avec le système de contraintes et si possible d'ôter des domaines les valeurs qui deviennent incompatibles. Si une incohérence apparaît, le système doit aider l'utilisateur à trouver une relaxation cohérente de ses choix.

Cet article présente une approche à base de contraintes de la configuration qui permet de traiter ces différents tâches. Elle s'appuie sur une généralisation des ATMS aux domaines non booléens. D'un point de vue théorique, les problèmes de complexité attachés à cette extension des CSP sont étudiés dans le cas général et dans le cas particulier de la configuration. D'un point de vue pratique, nous proposons de compiler le CSP initial sous la forme d'un automate, cette structure de données étant exploitée en ligne pour assurer le maintien de la cohérence globale et le calcul de restaurations.

1 Introduction

L'approche CSP (Constraint Satisfaction Problem) offre un cadre de représentation général pour de nombreux problèmes d'aide à la décision. La majorité des travaux dans ce domaine sont consacrés à la résolution automatique de CSP. Néanmoins, le traitement pratique de nombreux problèmes d'aide à la décision présente un caractère fortement interactif pour lequel la résolution automatique convient mal. Pour ces problèmes, c'est typiquement l'utilisateur qui choisit lui-même les valeurs des variables. Il s'agit alors pour la machine, non plus de résoudre le CSP, mais d'aider l'utilisateur dans la recherche d'une solution.

C'est en particulier le cas pour les problèmes de configuration de produit. Un *produit générique* peut être défini par un CSP. Les variables codent les différents composants, options et attributs du produit et les contraintes sont des contraintes de faisabilité, de compatibilité entre composants, etc. Nous supposons ici que les variables sont à domaine fini et que le produit générique est représenté par une conjonction de contraintes. L'ensemble des solutions du CSP forme le *catalogue*, c'est-à-dire constitue l'ensemble des variantes du produit que propose l'entreprise. En situation de configuration, l'utilisateur définit le produit qu'il désire en choisissant interactivement des valeurs pour les variables ou plus généralement en introduisant des contraintes unaires. Après chaque choix, les domaines doivent être filtrés de manière à garantir la cohérence (idéalement, globale) avec les choix courants. Enfin, si l'ensemble des choix courants est incohérent avec les contraintes du CSP ou si les valeurs restantes ne lui conviennent pas, l'utilisateur doit pouvoir revenir sur ses choix précédents et les relâcher. Pour l'aider dans sa démarche, plusieurs types d'outils sont souhaitables :

- **Maintien de cohérence :** il s'agit de garantir que l'ensemble de choix est cohérent à tout instant avec le CSP modélisant le produit générique, ou du moins détecter l'incohérence le plus tôt possible. Il s'agit également de répercuter les conséquences des actions de l'utilisateur en interdisant (ou respectivement en restaurant) idéalement toutes les valeurs incompatibles (ou respectivement compatibles) avec ses choix.
- **Aide à la restauration de la cohérence :** l'objectif est ici de guider l'utilisateur dans ses retours arrières en répondant aux questions du type *"comment rendre mon choix courant cohérent ?"* ou *"que relâcher pour que telle valeur soit autorisée pour telle variable ?"*. Il faut donc identifier des ensembles (maximaux) cohérents de restrictions utilisateur, minimisant éventuellement une fonction de coût sur les choix.
- **Explication de conflits :** il s'agit de répondre aux questions du type *"quels sont les choix qui ont conduit à l'incohérence ?"* ou *"pourquoi telle valeur n'est-elle plus possible pour telle variable ?"*. Il faut donc identifier des ensembles de choix utilisateur (minimaux) incohérents, ou incohérents avec telle valeur pour telle variable.

Les outils de propagation de contraintes associés aux CSP dynamiques (voir [Deb95] pour un *survey*) peuvent tenir compte du caractère évolutif du problème. Cependant, ces algorithmes ne garantissent pas l'idéal de cohérence globale et ne seraient d'aucune aide pour la génération de restaurations. Nous proposons ici une méthode permettant de répondre à ces besoins. Elle repose sur un prétraitement : le calcul d'une représentation par un automate de l'ensemble des solutions du CSP [Vem92] [Ami99, ABJJ97]. Ce prétraitement peut être considéré comme une compilation du CSP permettant de rendre traitables certains problèmes centraux dans la configuration interactive.

Le reste de l'article est organisé comme suit. La prochaine section introduit plus formellement les problèmes traités ainsi que quelques éléments de leur complexité. La section 3 expose le principe de codage du produit générique par un automate. Les algorithmes associés au maintien de cohérence et à l'aide à la restauration sont présentés en section 4.

2 CSP à hypothèses

Dans cette section, nous étendons le modèle CSP de manière à permettre la représentation de restaurations et d'explications dans les systèmes à base de contraintes. Cette extension s'appuie sur la distinction d'un ensemble de contraintes dynamiques - c'est en fait une généralisation aux domaines non booléens des ATMS ("Assumption-based Truth Maintenance Systems" [dK86]), notre ensemble de contraintes dynamiques jouant le rôle de l'ensemble d'hypothèses. Afin de rester généraux, nous ne restreignons pas ce formalisme aux problèmes de configuration et donnons des définitions plus générales que strictement nécessaire ; par exemple, nous ne supposons pas a priori que les contraintes dynamiques sont unaires.

2.1 Définitions préliminaires et notations

Un CSP Π est classiquement défini par un triplet $<\mathcal{X},\mathcal{D},\mathcal{C}>$ où $\mathcal{X}$ est un ensemble fini de variables $\mathcal{X}=\{X_1,X_2,\ldots,X_n\}$, chaque variable X_i prenant ses valeurs dans un ensemble fini D_{X_i}, $\mathcal{D}=D_{X_1}\cup D_{X_2}\cup\ldots\cup D_{X_n}$ et un ensemble fini de contraintes $\mathcal{C}$. Une contrainte C de $\mathcal{C}$, définie sur $V(C)\subseteq\mathcal{X}$, restreint les combinaisons de valeurs que peuvent prendre les variables $V(C)$ qui occurrent dans C. On associe à chaque contrainte C une relation $R(C)$ incluse dans le produit cartésien des domaines des variables de $V(C)$: c'est l'ensemble des tuples qui la satisfont. Une affectation s est un tuple de $D_{X_1}\times D_{X_2}\times\ldots\times D_{X_n}$; elle est solution du CSP si elle satisfait toutes les contraintes, c'est-à-dire si pour toute contrainte C, la projection de s sur $V(C)$ est dans $R(C)$ (sinon on dit qu'elle falsifie C). $\mathcal{S}(\Pi)$ désigne l'ensemble des solutions de Π. On note $\neg C$ la contrainte sur $V(C)$ satisfaite par toute affectation qui falsifie C et falsifiée par toute affection qui satisfait C. Π est dit cohérent s'il possède une solution, sinon il est dit incohérent.

Définition 1 *Un* H-CSP Π *est un quadruplet* $<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$ *où* $<\mathcal{X},\mathcal{D},\mathcal{C}>$ *est un CSP et* $\mathcal{H}$ *un ensemble fini de contraintes tel que* $V(\mathcal{H})\subseteq\mathcal{X}$.

En configuration $<\mathcal{X},\mathcal{D},\mathcal{C}>$ représente le produit générique et $\mathcal{H}$ les choix courants de l'utilisateur.

Une affectation s est solution de Π si s est solution de $\Pi'=<\mathcal{X},\mathcal{D},\mathcal{C}\cup\mathcal{H}>$, le CSP associé à Π; Π est dit cohérent (*resp.* incohérent) si Π' est cohérent (*resp.* incohérent). A tout instant l'ensemble des solutions de Π correspond à l'ensemble de tous les produits respectant les restrictions posés par l'utilisateur. Pour toute variable X_i on notera P_{X_i} la projection de $\mathcal{S}(\Pi)$ sur X_i . C'est l'ensemble des valeurs de X_i permettant de définir un produit réalisable respectant les contraintes $\mathcal{H}$.

2.2 Environnements cohérents et conflits

Dans la suite, les ensembles $E \subseteq \mathcal{H}$ d'hypothèses sont appelés des *environnements*, qu'ils soient cohérents ou pas.

Définition 2 *Soit* $\Pi = <\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H}>$ *un H-CSP.*
Un environnement est cohérent *(resp.* incohérent*) avec* Π *ssi* $<\mathcal{X}, \mathcal{D}, \mathcal{C} \cup E>$ *est un CSP cohérent (resp. incohérent).Un environnement incohérent est appelé un* conflit *de* Π *(ou un conflit de* $\mathcal{H}$ *pour* $\mathcal{C}$*).*

Quand $<\mathcal{X}, \mathcal{D}, \mathcal{C}>$ est cohérent, un conflit peut être compris comme une cause de l'incohérence du H-CSP $<\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H}>$ (e.g., de l'incohérence des choix utilisateur avec les contraintes de configuration) et tout environnement cohérent définit un sous-ensemble de contraintes de $\mathcal{H}$ dont la relaxation restaure la cohérence : si E est un environnement cohérent, il suffit à l'utilisateur de relaxer $\mathcal{H} \setminus E$.

Par exemple, si $<\mathcal{X}, \mathcal{D}, \mathcal{C}>$ est cohérent et $<\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H}>$ est incohérent, $\mathcal{H}$ est un conflit trivial, $\emptyset$ est un environnement cohérent et une solution naive est la relaxation de toutes les contraintes de $\mathcal{H}$. Cet exemple montre aussi que tous les environnements ne sont pas également intéressants : on préfére typiquement les environnements cohérents (resp. incohérents) qui sont maximaux (resp. minimaux) pour l'inclusion ensembliste :

Définition 3 *Un nogood pour* $\mathcal{C}$ *sur* $\mathcal{H}$ *(ou nogood de* Π*) est un conflit* E *de* Π *tel qu'il n'existe pas de conflit* E' *de* Π *tel que* $E' \subset E$*. Un interprétation de* Π *est un environnement cohérent* E *de* Π *tel qu'il n'existe pas d'environnement cohérent* E' *de* Π *tel que* $E \subset E'$*.*

2.3 Explications et restaurations

Même si l'ensemble des choix courants n'est pas incohérent, les valeurs restant dans les domaines des variables peuvent ne plus convenir à l'utilisateur. Il faut alors pouvoir lui expliquer pourquoi telle ou telle valeur n'est plus autorisée, c'est-à-dire lui présenter les configurations minimales qui l'excluent. Ce qui fait appel à une notion d'explication, que nous définissons en toute généralité (i.e., sans faire l'hypothèse que le H-CSP considéré est cohérent).

Définition 4 *Soient* $\Pi = <\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H}>$ *un H-CSP et* L *une contrainte sur des variables de* $\mathcal{X}$*. Une explication de* L *dans* Π *est un environnement* E *tel que* $<\mathcal{X}, \mathcal{D}, \mathcal{C} \cup E>$ *est cohérent et* $<\mathcal{X}, \mathcal{D}, \mathcal{C} \cup E \cup \{\neg L\}>$ *est incohérent.* E *est une explication minimale de* L *dans le H-CSP si elle est minimale pour l'inclusion ensembliste.*

Déterminer dans quelles circonstances un choix possible L est interdit, c'est déterminer avec quel ensemble de choix courants L est incohérent, c'est-à-dire calculer des explications de $\neg L$.

Si l'utilisateur a besoin d'explications de l'interdiction de L, il a surtout besoin de de savoir quels choix précédents relâcher pour autoriser L. La notion duale de l'explication d'une contrainte est la notion de restauration :

Définition 5 *Soient* $\Pi = <\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H}>$ *un H-CSP et* L *une contrainte sur des variables de* $\mathcal{X}$*. Une* restauration *de* L *est un environnement* E *tel que* $<\mathcal{X}, \mathcal{D}, \mathcal{C} \cup E \cup \{L\}>$ *est un CSP cohérent.* E *est maximale si elle est maximale pour l'inclusion ensembliste.*

Remarquons que les restaurations peuvent être considérées comme les environnements cohérents du H-CSP obtenu ajoutant L à $\mathcal{C}$.

2.4 Complexité

La complexité des problèmes de recherche de nogoods, interprétations, explications et restaurations, dans un H-CSP n'a, à notre connaissance, pas été clairement identifiée jusqu'ici. Plusieurs sources contribuent à cette complexité : d'une part, le *nombre* d'objets à générer qui, dans le pire des cas, est exponentiel dans la taille de l'entrée et d'autre part, la difficulté même de reconnaître les objets à générer ou de tester s'il en existe. Nous présentons dans la table suivante quelques résultats de complexité qui permettent d'identifier précisément la difficulté de cette seconde source dans le cas général (i.e., on ne fait aucune hypothèse particulière propre à la configuration). Les notions que nous avons introduites étant des translations directes de notions connues en logique propositionnelle, les démonstrations de difficulté se basent généralement sur le cas particulier suivant : $\mathcal{X}$ est un ensemble de variables booléennes ; $\mathcal{C}$ est la représentation d'une formule de la logique propositionnelle.

Entrée	**Question**	**Complexité**
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E	E est-il un conflit ?	coNP-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E	E est-il un nogood ?	BH_2-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$	Existe-t-il un conflit (resp. un nogood) ?	coNP-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E	E est-il un environnement cohérent ?	NP-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E	E est-il une interprétation ?	BH_2-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$	Existe-t-il un env. cohérent (resp. une interprétation) ?	NP-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E, L	E est-il une explication de L ?	BH_2-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E, L	E est-il une explication minimale de L ?	BH_2-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, L	Existe-t-il une explication (resp. une exp. min.) de $\neg L$?	Σ_2^p-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E, L	E est-il une restauration de L ?	NP-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, E, L	E est-il une restauration maximale de L ?	BH_2-complet
$<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$, L	Existe-t-il une restauration (resp. une rest. max.) de L ?	NP-complet

Plaçons nous maintenant dans le cadre de configuration; les hypothèses suivantes peuvent être posées: $\mathcal{C}$ est cohérent, $\mathcal{H}$ est cohérent, $\mathcal{H}$ est un ensemble fini de contraintes unaires et les différents L considérés sont également des contraintes *unaires*. Par ailleurs, on ne lance une recherche de nogoods que lorsque l'on sait que le H-CSP considéré est incohérent (i.e., $\mathcal{C} \cup \mathcal{H}$ est incohérent); on ne lance une recherche d'explications de $\neg L$ que lorsque l'on sait que le H-CSP considéré est cohérent (i.e., $\mathcal{C} \cup \mathcal{H}$ est cohérent) et L n'est pas possible compte tenu des choix courants $\mathcal{H}$ (i.e., $\mathcal{C} \cup \mathcal{H} \cup L$ est incohérent); enfin, on ne cherche à restaurer un L que lorsque l'on sait qu'il est possible au départ (i.e., $\mathcal{C} \cup L$ est cohérent). Ces restrictions supplémentaires *ne changent rien* aux résultats de complexité donnés dans la table précédente, sauf en ce qui concerne :

- le problème de tester si E donné est une explication de L donné dans le H-CSP $<\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{H}>$ donné qui devient "seulement" coNP-complet (en effet, lorsque l'on sait que $\mathcal{C} \cup \mathcal{H}$ est cohérent, on a forcément aussi $\mathcal{C} \cup E$ cohérent pour tout $E \subseteq \mathcal{H}$).
- les problèmes d'existence qui deviennent tous triviaux. Ainsi, quand $\mathcal{C}$ est supposé cohérent, un environnement cohérent $\emptyset$ existe toujours. Quand $\mathcal{C} \cup \mathcal{H}$ est

incohérent, un environnement incohérent $\mathcal{H}$ existe toujours. Quand $\mathcal{C} \cup \mathcal{H}$ est cohérent et $\mathcal{C} \cup \mathcal{H} \cup L$ ne l'est pas, une explication $\mathcal{H}$ de $\neg L$ existe toujours. Enfin, quand $\mathcal{C} \cup L$ est cohérent, une restauration $\emptyset$ de L existe toujours.

Pour pallier la complexité des problèmes de recherche d'interprétations et de restaurations, nous présentons dans la suite une méthode de compilation des solutions du H-CSP par un automate. La représentation choisie est une forme compactée de ce que l'on peut voir logiquement comme une forme normale disjonctive du H-CSP. L'intérêt est qu'à partir d'une telle forme compilée, la génération des interprétations et des restaurations peut être réalisée en temps linéaire dans la taille de la forme compilée et surtout dans la taille du résultat. Bien évidemment dans le pire des cas la taille de l'automate que l'on va construire est exponentielle dans la taille du H-CSP de départ. Toutefois, cet automate est construit une bonne fois pour toutes. Lorsque sa taille reste raisonnable, ce qui peut tout à fait être le cas en pratique et est un prérequis indispensable à toute utilisation d'une méthode de type compilation, le temps global requis pour aboutir à une configuration en répondant successivement à toutes les requêtes de type recherche d'interprétations ou de restaurations posées par l'utilisateur se trouve grandement raccourci, comparé au temps cumulé nécessaire si aucune compilation préalable n'avait été réalisée.

2.5 Valuation des hypothèses et requêtes associées

L'utilisateur n'est en général pas capable d'appréhender un grand nombre de conflits, d'environnement cohérents, d'explications ou de restaurations. Ces notions ont été définies à partir de la relation d'inclusion entre environnements. Or, dans de nombreuses applications, on peut se référer à d'autres informations pour définir un ordre sur les environnements, et dans un même temps, restreindre le nombre des minimaux et/ou maximaux. L'idée est d'attacher à toute contrainte de $\mathcal{H}$ un poids ϕ dans $\mathbb{N}^+$ (un poids étant d'autant plus fort que la contrainte est importante pour l'utilisateur) et d'évaluer le poids de tout sous ensemble E de $\mathcal{H}$ par: $\phi(E) = \sum_{C \in E} \phi(C)$[1]. On obtient alors un préordre complet sur $2^{\mathcal{H}}$, e.g. : $E \succeq E'$ ssi $\phi(E) \geq \phi(E')$. Par exemple, si ϕ est la fonction constante 1, le poids d'un ensemble de contraintes E est tout simplement sa cardinalité : les conflits et explications optimaux seront ceux mettant en jeu le moins de contraintes, les environnement optimaux cohérents et restaurations optimaux ceux conservant le plus de contraintes.

Définition 6 *Soient* $\Pi = < \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H} >$ *un H-CSP et* ϕ *une application positive de* $\mathcal{H}$ *dans* $\mathbb{N}^+$ *; pour tout* $E \in \mathcal{H}$*, soit* $\phi(E) = \sum_{H \in E} \phi(H)$*.*

Un conflit V-optimal *pour* $\mathcal{C}$ *sur* $\mathcal{H}$ *(ou V-nogood de* Π*) est un conflit* E *de* Π *tel qu'il n'existe pas de conflit* E' *de* Π *avec* $\phi(E') < \phi(E)$*.*

Une explication V-optimale *de* L *sur* $\mathcal{H}$ *est une explication* E *de* L *sur* $\mathcal{H}$ *tel qu'il n'existe pas d'explication* E' *de* L *sur* $\mathcal{H}$ *avec* $\phi(E') < \phi(E)$*.*

[1] Nous pourrions utiliser une autre structure de valuation, comme suggéré dans [TS95], en fait toute structure de valuation $< \mathcal{L}, \succ, \oplus >$ où $\mathcal{L}$ est un ensemble totalement ordonné par $\succ$, muni d'un élément minimal $\perp$ et d'un élément maximal $\top$, et $\oplus$ est une co norme triangulaire, i.e., une loi interne sur $\mathcal{L}$ associative, commutative, monotone, d'élément neutre $\perp$, d'élément absorbant $\top$, et croissante. Nous nous restreindrons ici à la structure $< \mathbb{N} \cup \{+\infty\}, >, + >$ avec un ϕ accordant des valuations différentes de $\top = +\infty$ and $\perp = 0$. Ce choix présente l'avantage de raffiner l'ordre d'inclusion et d'être suffisamment général pour inclure la notion d'optimalité en fonction de la cardinalité. De plus, les calculs de $\oplus$ et $\succ$ ont une complexité qui reste polynomiale dans la taille de leurs arguments.

Un environnement V-maximal cohérent *(ou V-interprétation) de Π est un environnement cohérent avec Π tel qu'il n'existe pas d'environnement E' cohérent avec Π avec $\phi(E') > \phi(E)$.*

Une restauration V-optimale *de L est une restauration E de L telle qu'il n'existe pas de restauration E' de L avec $\phi(E') > \phi(E)$.*

Comme ϕ n'assigne que des poids différents de 0 et de ∞, l'ordre induit est cohérent avec (et raffine) l'ordre d'inclusion. En d'autres termes tout conflit (resp. interprétation, explication, restauration) V-optimal est un nogood (resp. une interprétation, une explication minimale, une restauration maximale).

Il est clair que l'exploitation de l'information préférentielle, si elle permet de limiter en pratique le nombre d'objets à engendrer, ne change rien à la situation de la complexité dans le pire des cas. Ainsi, le nombre d'objets optimaux reste exponentiel dans la taille du H-CSP dans le pire cas. De même, la complexité des problèmes de décision considérés plus haut ne diminue pas dans le pire cas.

3 Représentation d'un H-CSP valué par un automate

En configuration, le CSP initial représentant le produit générique est souvent sous-contraint et possède de nombreuses solutions. Par ailleurs, on a toute latitude pour effectuer des prétraitements à partir de ce problème. Le principe de notre approche est de déplacer la complexité des requêtes que l'utilisateur va poser en ligne (et qui nécessitent donc une réponse très rapide), vers un traitement hors ligne : le calcul et la représentation de l'ensemble des solutions de $< \mathcal{X}, \mathcal{D}, \mathcal{C} >$ par un automate. Cette représentation peut être logiquement comprise comme une forme normale disjonctive du CSP. Le principal intérêt de cette compilation est que les interprétations et restaurations peuvent être générées en temps linéaire (dans la taille du résultat). De plus la compilation n'est réalisée qu'une fois pour toutes - pour toutes les requêtes d'une séquence de configuration, et pour toutes les utilisations de toutes les copies du configurateur.

Il faut bien avoir à l'esprit que cette approche s'appuie sur certaines hypothèses, satisfaites en pratique dans les problèmes de configuration : la stabilité de $< \mathcal{X}, \mathcal{D}, \mathcal{C} >$ et la possibilité de réaliser un traitement hors ligne, avant l'introduction des contraintes dynamiques; la structure de son ensemble de solutions qui permet sa représentation par un automate dont la taille, exponentielle dans le pire des cas, doit rester raisonnable en pratique; la restriction de la partie dynamique du problème, $\mathcal{H}$, à des contraintes unaires.

3.1 Représentation d'un CSP par automates : rappels

Nous rappelons ci-dessous les points clefs de la méthode de représentation de CSP par automates introduite par Vempaty [Vem92]; le lecteur pourra consulter [Ami99, ABJJ97] pour plus de détails.

Automates Nous considérons ici les automates du point de vue de leur *diagramme d'états* : un automate d'états finis (FA) $\mathcal{A}$ d'alphabet Σ est un multi-graphe orienté dont les arcs sont étiquetés par un élément de Σ (les *transitions* du FA), d'ensemble fini de sommets Q (les *états* du FA), possédant une source (*état initial*) et des puits (*etats finaux*) et tel que tout arc et tout état appartient à au moins un chemin de l'état initial vers un état final. Un mot $m = a_1a_2a_3...a_n$ est reconnu par l'automate si il existe un chemin de l'état initial à un état final d'etiquette m dans $\mathcal{A}$: $I \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} ... \xrightarrow{a_n} F$. Le langage reconnu par $\mathcal{A}$ est l'ensemble des mots reconnus par $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$. Un FA est déterministe (DFA) si il n'existe pas deux arcs issus d'un même état et portant la même étiquette.

Automate associé à un CSP Considérons un CSP $\Pi =< \mathcal{X}, \mathcal{D}, \mathcal{C} >$. Etant donné une permutation $O = [X_1, X_2, \ldots, X_n]$ de l'ensemble $\mathcal{X}$ des variables, toute solution de Π définit un mot de longueur n sur l'alphabet $\mathcal{D}$. L'ensemble des solutions de Π définit donc un langage sur $\mathcal{D}$ composé de mots de longueur $n = |X|$ (si le CSP est cohérent, c'est le langage vide sinon). Ce langage, appelé langage solution de Π selon O et noté $\mathcal{S}_{\mathcal{O}}(\Pi)$, est un langage rationnel qu'il est donc possible de représenter par un FA d'alphabet $\mathcal{D}$: FA $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{S}_{\mathcal{O}}(\Pi)$. Ces FA associés aux CSP ont des propriétés particulières : ils n'ont qu'un seul état final (noté F) et tout chemin de I à F est de longueur $n + 1$.

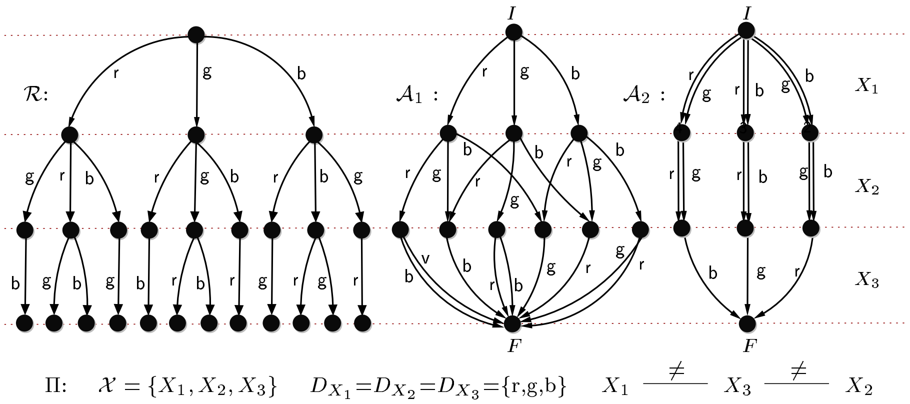

Figure 1: $\mathcal{A}_1$ et $\mathcal{A}_2$ sont respectivement un MDFA et un FA associé au CSP Π pour $O = [X_1, X_2, X_3]$: une représentation compacte de $\mathcal{R}$, l'arbre de recherche des solutions.

Pour tout chemin $c = (I = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} ... \xrightarrow{a_n} F = q_{n+1})$ de I à F dans l'automate on pose : pour tout état q_i de c, $var(q_i) = i$ (si $q_i \neq F$, $var(q_i)$ est l'indice d'une variable de $\mathcal{X}$) ; pour tout arc $a = (q_i, q_{i+1})$ de c, $val(a) = a_i$ (l'étiquette de a dans c) et $var(a) = X_{var(q_i)}$. $(val(q_1, q_2), val(q_2, q_3), \ldots, val(q_n, q_{n+1}))$ est donc une affectation de $(var(q_1, q_2), var(q_2, q_3), \ldots, var(q_n, q_{n+1}))$ solution de Π. Dans la suite, $c_{q,q'}$ pourra dénoter un chemin de l'état q à l'état q' et $val(c)$ l'affectation partielle des variables de $\mathcal{X}$ qu'il représente.

Un arc a est dit *supporter* la valeur d pour X_i (i.e., l'affectation $X_i := d$) ssi $var(a) = X_i$ et $val(a) = d$. a est dit supporter la contrainte unaire L sur X_i ssi il supporte au moins l'une des valeurs de $R(L)$. Enfin, un chemin supporte une valeur pour une variable (resp. une contrainte unaire) ssi il contient un arc supportant cette valeur (resp. cette contrainte unaire).

Construction de l'automate L'automate associé à un CSP est une représentation concise de l'arbre de recherche qui permet de traiter par des parcours de l'automate un nombre important de problèmes relatifs aux CSP. Sa construction peut être effectuée à l'aide d'opérations sur les automates par composition des automates associés à chacune des contraintes ou à partir de tout algorithme de calcul des solutions, en ajoutant successivement ces solutions.

Le coût de construction de l'automate ainsi que celui de son utilisation dépend de sa taille. Ils sont déterminés par deux facteurs essentiels : le choix de l'ordre sur les variables du CSP et l'algorithme utilisé pour minimiser la taille de l'automate. Si Vempaty a proposé une représentation par DFA de nombre minimum d'états (MDFA), il est possible d'utiliser des algorithmes de minimisation moins classiques, mais pouvant permettre de construire des automates non déterministes de taille inférieure (voir [Vem92] [Ami99, ABJJ97] pour plus de détails).

3.2 H-CSP et automates valués

$\mathcal{H}$ représente les choix de l'utilisateur. Chacun de ces choix est une contrainte unaire H_{X_i} munie d'un poids $\phi(H_{X_i})$ - on suppose qu'au plus une contrainte unaire porte sur chaque variable - et invalide un certain nombre d'arcs de l'automate : les arcs a tels que $var(a) = X_i$ et $val(a) \notin R(H_{X_i})$. L'idée est d'associer un poids à chaque arc a portant sur X_i : $\phi(a) = \phi(H_{X_i})$ si l'affectation qu'il représente est interdite par H_{X_i}, $\phi(a) = 0$ sinon. On définit :

- $p(c_{q,q'}) = \sum_{a \in c_{s,s'}} \phi(a)$ le poids du chemin $c_{q,q'}$.
- $p(a)$ le poids du meilleur chemin de I à F passant par l'arc a.
- $p(q)$ le poids du meilleur chemin de I à F passant par l'état q.

4 Résolution des requêtes sur l'automate valué

Puisque toute contrainte de $\mathcal{H}$ a une valuation positive, tout chemin de I à F de poids nul définit un produit faisable qui satisfait toutes les contraintes utilisateur et tout chemin de poids non nul un produit qui viole au moins une restriction de $\mathcal{H}$. Connaître l'ensemble des chemins de poids nul est donc équivalent à connaître l'ensemble des solutions de $< \mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} >$.

Proposition 7 *Soient* $\Pi = < \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H} >$ *un H-CSP,* ϕ *une fonction de valuation positive de ses hypothèses et* $\mathcal{A}$ *un automate valué représentant* Π. *Soit* L *une contrainte unaire :*

a) Tout chemin c *de* I *à* F *définit un environnement cohérent* E *tel que* $\phi(E) = \phi(\mathcal{H}) - p(c)$.

b) Si E *est un environnement cohérent alors il existe un chemin* c *de* I *à* F *tel que* $p(c) \leq \phi(\mathcal{H}) - \phi(E)$.

c) Tout chemin c *de* I *à* F *qui supporte* L *définit une restauration* E *de* L *telle que* $\phi(E) = \phi(\mathcal{H}) - p(c)$.

d) Si E *est une restauration de* L *alors il existe un chemin* c *de* I *à* F *qui supporte* L *et tel que* $p(c) \leq \phi(\mathcal{H}) - \phi(E)$.

Cette proposition permet la reformulation en termes de chemins minimaux de la plupart des requêtes identifiées en introduction. Elle permet en effet de prouver que:

Proposition 8

a) *Tout chemin minimal I de F représente une interprétation V-optimale de* Π *et réciproquement, à toute interprétation V-optimale correspond au moins un chemin minimal.*

b) $\mathcal{H}$ *est un conflit pour* Π *ssi il n'existe pas de chemin nul de* I *à* F.

c) $d \in P_{X_i}$ *ssi il existe un arc qui supporte cette valeur pour* X_i *et qui appartient à un chemin nul de* I *à* F.

d) *Soient* L *une contrainte unaire,* $X_i \in \mathcal{X}$, A_L *l'ensemble des arcs qui la supportent et* A^*_L *ceux de ces arcs qui ont le plus faible poids (*$A^*_L = \{a \in A_L \ / \ p(a) = min_{a \in A_L} p(a)\}$*) : tout arc de* A^*_L *définit une restauration* V*-optimale de* L *et à toute restauration* V*-optimale de* L *corresponds au moins un arc de* A^*_L.

L'ensemble des requêtes qui nous intéressent se ramène donc à des calculs de chemins de poids minimal sur l'automate. Pour calculer efficacement ces chemins de poids minimaux, nous maintenons pour tout état q de l'automate :

- $p_g(q) = Min_{q' \in pred(q)}(p_g(q) + \phi((q,q'))$ le poids d'un chemin de poids minimum de I à q.
- $p_d(q) = Min_{q' \in pred(q)}(p_d(q') + \phi((q,q'))$ le poids d'un chemin de poids minimum de q à F.
- $p_g(I) = p_d(F) = 0$.

$p(q)$, le poids du meilleur des chemins de I à F passant par q, et $p((q,q'))$ le poids du meilleur des chemins de I à F passant par l'arc (q,q') se déduisent automatiquement de ces quantités :

Proposition 9 $p(q) = p_g(q) + p_d(q)$, $p((q,q')) = p_g(q) + \phi((q,q')) + p_d(q')$

Par ailleurs, pour calculer efficacement les ensembles P_{X_i}, nous maintenons pour tout couple variable-valeur (X_i, d) un compteur $cnt(X_i, d)$: c'est le nombre d'arcs de l'automate qui supportent cette affectation et qui appartiennent à un chemin de I à F de valuation nulle.

4.1 Ajout ou retrait d'un choix

L'utilisateur vient d'ajouter à $\mathcal{H}$ une contrainte unaire H_{X_i}, portant sur la variable X_i et de valuation $\phi(H_i)$. De nouveaux arcs deviennent donc interdits : tous les arcs a issus d'un noeud étiqueté X_i et tels que $val(a) \notin R(H_{X_i})$. Il faut mettre à jour leurs valuations ϕ , qui passent de 0 à $\phi(H_{X_i})$. Inversement, lorsque l'utilisateur retire H_{X_i} de $\mathcal{H}$, des arcs interdits deviennent autorisés : tous les arcs a issus d'un noeud étiqueté X_i et tels que $val(a) \notin R(H_{X_i})$. Il faut mettre à jour leurs valuations ϕ, qui passent à 0. Dans les deux cas, ces nouvelles valuations d'arcs (q,q') doivent être propagées à gauche de q (mise à jour des $p_d()$) et à droite de q' (mise à jour des $p_g()$). Cette mise à jour des valuations peut être effectuée en trois temps :

1) Repérage des arcs étiquetés par une valeur de $\neg H_{X_i}$ et mise à jour de leur ϕ.

2) Parcours en largeur des chemins de l'automate depuis les états gauches de ces arcs jusqu'à l'état initial, et mise à jour des $p_d()$ des états traversés à partir des poids droits de leurs successeurs.

3) Parcours des chemins de l'automate depuis les états droits de ces arcs jusqu'à l'état final, et mise à jour de $p_g()$ des états traversés à partir des poids gauches de leurs prédécesseurs.

Les compteurs $cnt()$ sont maintenus durant les étapes 2 et 3 : pour tout arc a traversé, $cnt(var(a), val(a))$ est incrémenté si $p(a)$ passe de 0 à un entier positif et décrémenté lorsque $p(a)$ devient nul.

Cette méthode est polynomiale dans la taille de l'automate ; des implantations linéaires peuvent être proposées en choisissant judicieusement la structure de données codant l'automate. Leur efficacité en moyenne peut évidemment être améliorée, par exemple en prenant soin de ne propager que les changements de poids des états (il est souvent inutile de propager jusqu'aux extrémités).

4.2 Détection de l'incohérence

La maintenance de l'automate valué permet de déterminer à tout instant si l'ensemble des choix courants est cohérent (*cf.* proposition 8.a). En effet :

Proposition 10

- *$\mathcal{H}$ est un conflit si et seulement si $p_d(I) > 0$ (ou $p_g(F) > 0$).*
- *$\phi(\mathcal{H}) - p(I)$ est la valuation des interprétations V-optimales.*

4.3 Maintien de la cohérence globale

D'après la proposition 8.b, P_{X_i} peut être obtenu à partir de l'automate. En fait, il peut être calculé en temps linéaire dans la taille de D_{X_i} à partir de $cnt()$:

Proposition 11 $P_{X_i} = \{d \ / \ cnt(X_i, d) \neq 0\}$.

4.4 Calcul d'interprétations V-optimales

Lorsque le problème est incohérent, il faut énumérer les interprétations V-optimales : elles correspondent aux chemins de poids minimum allant de I à F (*cf.* proposition 8.c). Pour trouver l'*une* de ces interprétations, il suffit de parcourir l'automate de I vers F selon un chemin de coût minimal en utilisant la propriété suivante :

Proposition 12 *c est un chemin de I à F de coût minimal si et seulement si pour tout arc (q, q') de ce chemin, $p_g(q) + p_d(q) = p_g(q') + p_d(q')$.*

Le coût de calcul d'une seule interprétation V-optimale est borné par le produit du nombre de variables et du nombre maximum de successeurs d'un état. Si l'on s'intéresse à la recherche de *toutes* les interprétations V-optimales, il s'agit en quelque sorte de parcourir tous ces chemins optimaux. La difficulté dans ce cas est qu'une même interprétation peut être représentée par plusieurs chemins. On peut imaginer deux méthodes différentes : une méthode de type DPI (cf. [Cas96, FB99]) qui développe tout l'arbre des interprétations V-optimales et ce sans jamais effectuer de retour arrière suite à un échec, puisque les valuations portées par les noeuds permettent de savoir si l'on est dans une branche optimale. Une autre méthode possible est de calculer les interprétations $V-$optimales par un parcours en largeur depuis I des chemins optimaux, en marquant les états avec les éléments de $\mathcal{H}$ relâchés : on récupère sur F les complémentaires sur $\mathcal{H}$ des interprétations V-optimales. Un des inconvénients de cette méthode est la mise à jour des ensembles marquant les états : elle nécessite en effet d'effectuer une union. Au final, que l'on choisisse l'une ou l'autre de ces méthodes, notre représentation permet de calculer toutes les interprétations pour un coût qui reste dans le pire des cas polynomial dans la taille du résultat.

4.5 Calcul de restaurations V-optimales

L'automate permet également d'énumérer les restaurations V-optimales d'une contrainte unaire L : elles correspondent aux chemins les moins coûteux parmi ceux allant de I à F et supportant L. Pour effectuer ce calcul, on peut se ramener à la méthode précédente, en travaillant non plus sur tous les chemins optimaux de I à F mais sur les chemins de poids minimum parmi les chemins de I à F supportant L. On peut s'aider pour cela de la propriété suivante :

Proposition 13 *Un chemin c de I à F passant par l'arc (q, q') est de poids minimal parmi les chemins de I à F passant par cet arc si et seulement si:*

- *pour tout arc (e, e') de c tel que $var(e') \leq var(q)$, $p_g(e') = p_g(e) + \phi(e, e')$,*
- *pour tout arc (e, e') de c tel que $var(e) \geq var(q)$, $p_d(e) = p_d(e') + \phi(e, e')$.*

Si on s'intéresse seulement au calcul d'une unique restauration pour L, il faut repérer l'un des arcs de poids p minimal parmi ceux qui supportent cette contrainte, puis parcourir un chemin optimal de q vers I et de q' vers F en utilisant la propriété 13. Le coût de ce calcul est borné par la taille de l'automate ; moyennant là encore un choix judicieux du codage de l'automate, il est possible d'en concevoir une implantation dont la complexité est bornée par le nombre de transitions supportant L plus le produit du nombre de variables et du nombre maximum de successeurs d'un état.

Pour produire toutes les restaurations $V-$optimales possibles de la contrainte L, on peut marquer chacun des chemin de poids minimal parmi ceux qui supportent cette contrainte (en effectuant un parcours en largeur depuis les états gauches des arcs supportant L de moindre poids p vers l'état I et depuis les états droits de ces arcs vers l'état F) puis, en parcourant les arcs marqués avec l'une ou l'autre des méthodes proposées pour produire l'ensemble de toutes les restaurations. Là encore, le coût de la génération de toutes les restaurations reste dans le pire des cas borné polynomialement par la taille du résultat.

5 Conclusion

Plusieurs travaux [SF96, MF90, SG99] ont proposé d'utiliser des extensions des CSP pour résoudre des problèmes de configuration, traitant généralement des difficultés inhérentes à la structuration des ces problèmes. La prise en compte d'un autre aspect, essentiel, de la configuration, à savoir son caractère interactif, nous a conduit à étendre le modèle CSP dans une nouvelle direction, et à définir des "CSP à hypothèses" - ce qui revenait à étendre aux domaines non booléens des concepts classiques en logique propositionnelle, comme les notions de nogood, d'explication et d'interprétation [2].

Les requêtes associées à cette généralisation des ATMS [dK86] posent évidement des problèmes hautement combinatoires dans le pire des cas. D'un point de vue pratique, notre approche s'appuie sur un déplacement de cette combinatoire : on travaille à partir d'une compilation du CSP en un automate (de taille exponentielle dans le pire des cas) mais en proposant des algorithmes efficaces sur cette structure de données (polynomiaux dans la taille du résultat). Ceci est justifié par le fait que les catalogues de produits, s'ils sont énormes décrits en extension, sont décrits de manière compacte par l'automate. Notre approche tire fortement parti du fait que seul un sous-ensemble de contraintes est dynamique, et que ces contraintes sont unaires. Son utilisation d'un automate est à rapprocher de différents travaux sur l'utilisation de variantes des OBDD pour le calcul d'impliquants / impliqués premiers en logique propositionnelle (*cf.* [OC91], pour les travaux précurseurs, et surtout [CLSS98]).

Les principes proposés dans cet article ont été implantés et testés sur un problème réel ; la taille de l'automate est faible (4675 arcs et noeuds pour un problème présentant 936 000 solutions) et les temps de mise à jour comme de recherche d'interprétations sont plus que raisonnables (moins de 2ms). La transcription d'un autre exemple de produit générique réel, de taille plus importante (200 variables), est en cours. Cela dit, le problème de l'établissement d'un protocole d'expérimentation plus éloigné des problèmes réels mais qui mette en évidence de manière statistique la faisabilité ou les limites de l'approche reste entier.

Enfin, nous n'avons dans le présent article qu'effleuré le problème de calcul d'explications ou de nogoods, car il est moins justifié en pratique dans les problèmes de configuration (on les utiliserait essentiellement pour générer des restaurations/interprétations via un calcul d'ensembles intersectants - *hitting sets*-, alors que le calcul direct est souvent plus simple) à moins que le nombre d'explications ne soit faible et le nombre de restaurations grand : l'utilisateur pourrait alors lui même construire sa restauration en sélectionnant un élément par explication. Le calcul de nogoods reste néanmoins intéressant dans d'autres application potentielles des H-CSP ; nous pensons en particulier au diagnostic à base de contraintes. Il est possible d'en proposer une version par un algorithme de type DPI [Cas96, FB99] qui tourne sur l'automate mais dont la complexité au pire cas est exponentielle dans la taille de l'automate et sans garantie de non-retour arrière comparable à celle utilisée pour le calcul des interprétations. Ceci est inévitable si la structure de données de base reste notre forme compilée ; en effet, contrairement au nombre d'interprétations et de restaurations, le nombre d'explications et de nogoods est exponentiel dans la taille de la forme disjonctive qu'est l'automate. Si l'on accepte de relâcher l'idéal de minimalité et de complétude, une autre voie de recherche serait l'utilisation d'algorithmes de filtrage des CSP dynamiques (DN-AC4,6) pour le calcul, via les justifications maintenues, d'ensembles d'explications.

[2] Ces définitions quoique fort proches de celles proposées par les travaux de [VL99] en diffèrent par la référence à *deux* ensembles de contraintes ($\mathcal{C}$ et $\mathcal{H}$). On peut retrouver les définitions de [VL99] en posant $\mathcal{C} = \emptyset$. Cependant l'algorithmique proposée par [VL99] tirerait mal avantage des caractéristiques propres à la configuration (possibilité de prétraitement, recherche d'ensembles de contraintes *unaires*, aspects incrémentaux et répétitifs) voire n'offrirait pas de solution à certains besoins (calcul de restaurations de valeur, maintien de la cohérence globale).

References

[ABJJ97] J. Amilhastre, F. Boutquet, P. Janssen, and P. Jegou. Représentation de l'ensemble des solutions d'un csp et utilisation dans un cadre dynamique. *Revue d'Intelligence Artificielle*, 11(3):313–338, 9 1997.

[Ami99] J. Amilhastre. *Représentation par automate de l'ensemble des solutions de problèmes de satisfaction de contraintes*. PhD thesis, LIRMM (Université Montpellier II / CNRS), 1 1999.

[Cas96] T. Castell. Computation of prime implicates and prime implicants by a variant of the davis and putnam procedure. In *Proceedings ICTAI'96*, pages 428–429, 1996.

[CLSS98] Claudette Cayrol, Marie-Christine Lagasquie-Schiex, and Thomas Schiex. Nonmonotonic reasoning: from complexity to algorithms. *Annals of Mathematics and Artificial Intelligence*, 22(3-4):207–236, may 1998.

[Deb95] Romuald Debruyne. Les algorithmes d'arc-consistance dans les csp dynamiques. *Revue d'intelligence Artificielle*, 9(3):239–267, 1995.

[dK86] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, (28):127–167, 1986.

[FB99] P. Jegou F. Bouquet, L. Henecoque. Enumération et représentation d'impliquants premiers. In *Actes de JNPC'99*, Lyon, 1999.

[MF90] S. Mittal and F. Frayman. Dynamic constraint satisfaction problems. In *Proceedings AAAI'90*, pages 25–32, 8 1990.

[OC91] J.C. Madre O. Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *IJCAI'91*, pages 294–299, 1991.

[SF96] Daniel Sabin and Eugene C. Freuder. Configuration as composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop, AAAI Technical Report FS-96-03*, pages 28–36. AAAI Press, 1996.

[SG99] Timo Soininen and Ester Gelle. Dynamic constraint satisfaction in configuration. In Boi Faltings, Eugene C. Freuder, Gerhard Friedrich, and Alexander Felfernig, editors, *Proceedings of the AAAI'99 Workshop on configuration*, pages 95–100, Orlando, Florida, July 1999.

[TS95] Gérard Verfaillie Thomas Schiex, Hélène Fargier. Valuated constraint satisfaction problems : hard and easy problems. In *IJCAI95*, pages 631–637, 1995.

[Vem92] N. R. Vempaty. Solving constraint satisfaction problems using finite state automata. In *American Association for Artificial Intelligence (AAAI'92)*, pages 453–458, San Jose, 1992.

[VL99] G. Verfaillie and L. Lobjois. Problèmes incohérents: expliquer l'incohérence, restaurer la cohérence. In *Actes de JNPC'99*, pages 111–120, 6 1999.