N° d'ordre : 3622

UNIVERSITE TOULOUSE III - PAUL SABATIER
U.F.R. M.I.G.

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE TOULOUSE III

Discipline : Informatique

présentée et soutenue

par

Daniel LE BERRE

le 12 janvier 2000

Autour de SAT : le calcul d'impliquants P-restreints, algorithmes et applications

Directeur de thèse : Michel CAYROL

JURY

M. Antoine RAUZY	Chargé de recherche LABRI/CNRS (Bordeaux)	Président
M. Michel CAYROL	Professeur à l'Université Paul Sabatier	Directeur de thèse
M. Mamoun FILALI-AMINE	Chargé de recherche IRIT/CNRS	Examinateur
M. Eric GREGOIRE	Professeur à l'Université d'Artois (Lens)	Rapporteur
M. Philippe JEGOU	Professeur à l'Université d'Aix-Marseille III	Rapporteur
M. Thomas SCHIEX	Chargé de recherche INRA (Toulouse)	Examinateur

Remerciements

La première personne que je souhaite remercier ne lira jamais ces quelques lignes. C'est dans son sillage que j'ai effectué mon DEA et mes deux premières années de thèse. Ses horaires de travail étaient un peu particuliers, c'est pourquoi mes meilleurs souvenirs se situent entre 17h et 22h, devant un tableau plus très blanc. Gonflés au café tiède et à la nicotine, nous étions excités d'avoir trouvé LA solution à notre problème du moment. Nous nous endormions alors, satisfaits. Mais la nuit portant conseil, le matin, nous devions déchanter. Qu'importe, l'essentiel est de continuer d'y croire. Thierry Castell est malheureusement décédé en août 97, quelques mois seulement après avoir soutenu sa thèse. La mienne lui doit beaucoup …

```
switch (personne.id) {
case   CAYROL_CLAUDETTE :
```
Merci d'avoir co-dirigé cette thèse avec Michel, d'avoir su guider et recadrer mon travail, d'avoir corrigé mes articles et ma thèse.
```
break ;
case   CAYROL_MICHEL :
```
Merci d'avoir accepté de diriger cette thèse, d'avoir supporté mes fautes d'orthographe et mon manque de rigueur. De plus, il était important pour moi que mon chef aime ouvrir une machine et la programmer.
```
break ;
case   CORDIER_MARIE_ODILE :
```
Merci de m'avoir encouragé à effectuer mon DEA à Toulouse sous la direction de Michel.
```
break ;
case   CROS_PATRICE :
```
Merci d'avoir relu si soigneusement et rapidement mon manuscrit.
```
break ;
case   FILALI_AMINE_MAMOUN :
```
Merci de m'avoir fait découvrir la méthode de Stålmark et une utilisation du problème SAT que je ne connaissais pas. Merci aussi d'avoir accepté de participer au jury de cette thèse. Dommage que nous ayons découvert nos intérêts communs si tard.
```
break ;
case   GREGOIRE_ERIC :
```
Merci d'avoir accepté de rapporter cette thèse. Merci aussi pour vos mails encourageants et rassurants.
```
break ;
case   JEGOU_PHILIPPE :
```
Merci d'avoir accepté de rapporter cette thèse et d'avoir satisfait les contraintes que cela suscitait.
```
break ;
case   LANG_JEROME :
```
Merci au spécialiste local de la théorie de la décision pour les applications pratiques de ses travaux aux moments les plus inattendus (de préférence lorsqu'il faut choisir un dessert ou une boisson), pour son aide scientifique et sa relecture de ce document.
```
break ;
case   MARQUIS_PIERRE :
```
Merci à mon oracle MP qui a toujours répondu à mes questions, pour les discussions que nous avons eu avec Jack.
```
break ;
case   RAUZY_ANTOINE :
```
Merci d'avoir accepté de participer au jury de cette thèse et pour les discussions que cela a entraîné.
```
break ;
case   SCHIEX_THOMAS :
```
Merci pour ta participation au jury de cette thèse et tes remarques concernant ce manuscrit.
```
break ;
case   VIDAL_VINCENT :
```
Merci d'utiliser ADS, de m'aider à l'améliorer, de me faire découvrir la planification, et tout le reste …
```
break ;
default : if (personne instanceof RCFR95) {
```
Merci à mes camarades de promo Brigitte, Frédo, Laurent, Manu, Philippe, Régis et Seydina pour les pauses-café, les repas de midi, les soirées que nous avons passées ensemble.
```
      } else if (personne instanceof RPDMP) {
```
Merci aux membres de l'équipe pour ces 5 années passées le plus souvent dans la joie et la bonne humeur.
```
      } else if (personne instanceof IASC) {
```
Merci aux personnes du troisième étage que je croise tous les jours, que je retrouve dans la salle machine, ou à la machine à café. On en apprend quelquefois autant en discutant dans un couloir que lors d'un séminaire.
```
      } else if (personne instanceof IRIT) {
```
Merci aux personnes de l'IRIT que j'ai côtoyées durant ces 5 ans.
```
      } else if (personne instanceof CVSI) {
```
Merci aux membres de l'équipe CVSI du CERT qui m'a accueillie pendant mon service militaire, et à qui je dois la découverte du langage JAVA et de la communauté Méthode Formelle.
```
      } else if (personne.nom.startsWith("LE BERRE")) {
```
Merci à Anne et ma famille qui m'ont soutenu pendant toutes ces années.
```
      }
}
```

Table des matières

Table des définitions

Table des algorithmes

Introduction

Le raisonnement en calcul propositionnel[1] est l'objet de nombreux travaux à l'heure actuelle. Leur multiplicité résulte des progrès considérables autour du problème SAT : [Selman, et al. 1997] (traduction personnelle)

> *« Premièrement, de nouveaux algorithmes ont été découverts, comprenant ceux basés sur la recherche locale stochastique mais aussi la recherche systématique, avec de meilleurs comportements d'échelle[2] que la procédure de Davis et Putnam de base. Deuxièmement, l'amélioration des machines a étendu la panoplie des algorithmes. Troisièmement, les chercheurs ont commencé à développer et résoudre d'intéressants problèmes « réels » codés en logique propositionnelle, tels que la planification et le diagnostic, avec à l'horizon le traitement du langage naturel et l'apprentissage. Ces codages n'étaient même pas envisagés il y a quelques années parce qu'ils étaient considérés comme beaucoup trop gros pour être traités par n'importe quelle méthode. Entre 1991 et 1996 la taille des problèmes de satisfaction « difficiles » pouvant être résolus en temps convenable est passé de moins de 100 variables à plus de 10 000 variables.*
> *Les progrès sont dus à l'interaction des chercheurs en IA, recherche opérationnelle, et informatique théorique, qui ont fourni des benchmarks (DIMACS, voir [Johnson/Trick 1996]), […], et partagé algorithmes et codes. »*

On distingue deux catégories de méthodes pour résoudre SAT : les méthodes de recherche systématique et les méthodes de recherche stochastique. Un consensus semble s'établir autour de la procédure de Davis et Putnam [Davis/Putnam 1960, Davis, et al. 1962] dans ses variantes actuelles [Li/Anbulagan 1997] comme le meilleur représentant de la première catégorie. Dans la seconde catégorie, les méthodes les plus connues sont GSAT [Selman, et al. 1992] et ses variantes [Selman, et al. 1994, McAllester, et al. 1997].

Deux axes de développement autour du problème SAT voient le jour :

- Adaptation des méthodes à la résolution d'instances de problèmes réels, parmi lesquels la planification (SATPLAN, [Kautz/Selman 1996] et BlackBox [Kautz/Selman 1999]) et récemment la cryptographie [Massacci 1999]. Il s'agit d'une part de trouver des bibliothèques conséquentes de problèmes « réels » difficiles ayant les mêmes caractéristiques (ce qui est souvent malaisé) et, d'autre part, d'étudier le comportement des algorithmes sur ces instances particulières afin de développer, si nécessaire, des algorithmes spécialisés.

- Utilisation des techniques performantes pour SAT dans le cadre de problèmes connexes : le calcul d'impliquants premiers [Castell/Cayrol 1996] ou de couvertures d'impliquants par exemple [Mazure/Marquis 1996, Schrag 1996], la satisfaisabilité de formules propositionnelles quantifiées (QBF, problème QSAT) [Cadoli, et al. 1998, Gent/Walsh 1999, Rintanen 1999], la prise en compte de quantificateurs aléatoires (Stochastic SAT) [Littman 1999], l'existence d'un modèle proche d'une interprétation (partielle) de référence (DISTANCE-SAT) [Bailleux/Marquis 1999], le raisonnement non monotone [Bessant, et al. 1998, Grégoire, et al. 1998, Grégoire 1999], l'introduction de contraintes métriques dans le langage (LPSAT [Wolfman/Weld 1999]), l'extension du formalisme en conservant les techniques [Castell/Fargier 1998, Rauzy, et al. 1999], etc. Dans ce cas, on considère que les progrès réalisés pour SAT sont tels qu'il est désormais possible d'aller au delà/autour du problème SAT.

Nos travaux se situent dans ce deuxième axe. Cette thèse n'apporte pas de nouvelle stratégie de résolution du problème SAT ni d'amélioration de la procédure de Davis et Putnam. Nous utilisons les travaux existant sur ce domaine dans le cadre de la production d'impliquants P-restreints premiers, une notion que nous définissons. Ce

[1] on le retrouve dans de nombreux domaines en Intelligence Artificielle (raisonnement non monotone, diagnostic, planification, etc.)
[2] *« … that have better scaling properties… »*

concept n'est pas nouveau, et on le retrouve sous diverses formes dans le raisonnement propositionnel : modèles minimaux de Herbrand/Lifschitz, diagnostic consistant, sous-bases maximales consistantes, etc…Nous la formalisons de façon plus générale.
Cadre plus restreint que le calcul d'impliquants/impliqués premiers [Marquis 1999], autre problème très important du raisonnement propositionnel, il s'agit ici de calculer des formules restreintes à un ensemble *consistant* de littéraux. Nous avons abordé ces travaux avec Thierry Castell [Castell 1997], qui présente dans sa thèse des algorithmes de calcul de P-impliquants et P-impliqués premiers avec P un ensemble quelconque de littéraux. Cette différence change tout d'abord la complexité du problème : dans notre cas, notre espace de recherche pour une formule propositionnelle de n variables contient 2^n formules contre 3^n lorsque P est un ensemble de littéraux quelconques. Il en découle que les algorithmes utilisés pour calculer ces formules sont eux aussi différents. Quand P est un ensemble consistant de littéraux, il est possible d'utiliser la procédure de Davis et Putnam pour explorer l'espace de recherche (calcul des impliquants P-restreints d'une base de clauses).

Nous montrons comment modifier cette procédure pour produire uniquement les impliquants P-restreints *premiers* d'une base de clauses. La particularité de notre méthode, MPL (Modèles Préférés par leurs Littéraux), est d'utiliser uniquement des techniques connues pour SAT (ordonnancement des variables et des littéraux de branchement, ajout de clauses dans la base durant la recherche) afin de garantir la primarité des impliquants P-restreints obtenus. Nous montrons ensuite comment calculer des P-impliquants et P-impliqués premiers (avec toujours P ensemble consistant de littéraux) à l'aide de notre seul algorithme MPL. Il s'agit de cas particuliers des problèmes considérés par Thierry Castell.

On retrouve ces P-impliqués premiers particuliers dans un outil qui a connu un large succès lors de son apparition : l'Assumption-based Truth Maintenance System (ATMS) [de-Kleer 1986, de-Kleer 1986b]. Originalement destiné au diagnostic de panne, il a été utilisé dans diverses applications du raisonnement non monotone, car il permet par exemple de détecter les incohérences d'une base de connaissances, d'effectuer du raisonnement abductif, etc... Pour des raisons d'efficacité, il a été redéfini en termes d'impliquants et d'impliqués premiers dans le Clause Management System (CMS) [Reiter/de-Kleer 1987] puis l'Assumption-CMS (ACMS) [Kean/Tsiknis 1990, Kean/Tsiknis 1992]. Nous montrons comment les diverses formules manipulées par l'ACMS peuvent être redéfinies en terme d'impliquants P-restreints ou P-impliqués premiers pour construire un ACMS basé sur MPL.

Nous pensons que la meilleure procédure de résolution de SAT à l'heure actuelle est aussi la plus efficace pour le calcul des impliquants P-restreints premiers.

Nous appliquons ensuite nos travaux dans le cadre du raisonnement non monotone au raisonnement en monde clos (CWR). Nous montrons comment calculer l'ensemble des formules « free for negation » d'une politique donnée à partir du concept d'impliquant P-restreint premier. Ces résultats sont basés sur l'équivalence entre les impliquants P-restreints premiers et les modèles minimaux de Herbrand/Lifschitz [Lifschitz 1985].

Dans le cadre de la théorie de la décision dans l'incertain en calcul propositionnel, en collaboration avec Régis Sabbadin [Sabbadin 1998], nous proposons des algorithmes de calcul de décisions optimales selon deux politiques (optimiste et pessimiste). On retrouve dans ces deux politiques la notion d'impliquant P-restreint premier (politique optimiste) et de P-impliqué premier (politique pessimiste).

Nous étendons ensuite la notion d'impliquant P-restreint premier à une notion de préférence plus générale. Dans le cadre de l'argumentation par exemple, en collaboration avec Leïla Amgoud [Amgoud 1999], nous montrons comment ramener le calcul d'argument préféré à un calcul de P-impliqué préféré. Nous présentons alors deux algorithmes permettant le calcul d'argument préféré selon deux relations de préférence présentées dans [Amgoud, et al. 1996].

Les travaux présentés dans cette thèse ont abouti à l'implantation d'un prototype comportant tous les algorithmes proposés. Il est utilisé dans notre équipe dans diverses expérimentations.

Le plan du manuscrit est le suivant :
Tout d'abord quelques résultats et notations du calcul propositionnel utilisés dans le reste du document suivi par la présentation du problème SAT et de diverses méthodes de résolution de ce problème, en particulier la procédure de Davis et Putnam.

Nous introduisons ensuite la notion d'impliquant P-restreint et P-impliqué premier à l'aide de l'ATMS. Nous formalisons ces notions et présentons des algorithmes de calcul des impliquants P-restreints, P-impliquants et P-

impliqués premiers d'une CNF. Nous donnons ensuite quelques résultats empiriques sur le comportement de nos différents algorithmes appliqués à des formules générées aléatoirement.

Nous confrontons ces notions dans divers domaines parmi lesquels l'ATMS, le raisonnement en monde clos, la théorie de la décision dans l'incertain puis nous introduisons deux variantes : l'utilisation de méthodes stochastiques dans le calcul d'impliquants P-restreints premiers et le calcul du degré de croyance d'une formule à la Dempster/Shafer.

Nous présentons ensuite la notion d'impliquant P-restreint préféré, une généralisation de la notion d'impliquant P-restreint premier à des relations de préférences compatibles avec l'inclusion.

Nous présentons enfin quelques travaux proches des nôtres. Les premiers utilisent une méthode de recherche locale dans le cadre du raisonnement non monotone, les autres proposent d'autres méthodes de calcul de modèles minimaux.

Partie I : SAT & Co

« D'une goutte d'eau, […], un logicien pourrait inférer la possibilité d'un océan Atlantique ou d'un Niagara, sans avoir vu ni l'un ni l'autre, ni même en avoir entendu parler. Ainsi, toute la vie est une longue chaîne dont chaque anneau donne le sens. Comme toutes les autres sciences, la science de la déduction et de l'analyse ne peut s'acquérir qu'au prix de longues et patientes études ; du reste, notre vie est trop brève pour nous permettre d'atteindre à la perfection. Avant de se tourner vers les aspects moraux et intellectuels du sujet, où résident les plus grandes difficultés, le chercheur commencera par triompher des problèmes les plus simples. » Sir Arthur Conan Doyle, Etude en Rouge.

1 Le calcul propositionnel

Les logiques symboliques ont pour but de modéliser le raisonnement de sens commun. Nous allons nous intéresser à la plus simple d'entre elles : la logique propositionnelle. Nous ne cherchons pas à redéfinir la logique propositionnelle mais simplement à fixer certains termes et certaines notations utilisées dans le document. Pour de plus amples renseignements sur la logique symbolique en général et/ou le calcul propositionnel en particulier, nous renvoyons le lecteur à des ouvrages spécialisés ([Chang/Lee 1973] par exemple).

1.1 Le langage

On notera S l'ensemble des *symboles propositionnels* (ou *atomes*, *propositions*). On distinguera deux symboles propositionnels particuliers : $\top$ (top) et $\bot$ (bottom). A partir de ces symboles propositionnels, on construit des formules bien formées.

Définition 1 (formules bien formées)
Les formules sont définies récursivement comme suit :

1. Un atome est une formule (y compris $\top$ et $\bot$).
2. Si α est une formule alors $\neg\alpha$ est une formule.
3. Si α est une formule alors (α) est une formule.
4. Si α et β sont deux formules alors $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \quad \beta)$ et $(\alpha \Leftrightarrow \beta)$ sont des formules.
5. Toute formule est obtenue par application des règles précédentes un nombre fini de fois.

On notera FP_S l'ensemble des formules propositionnelles construites à partir de l'ensemble des symboles propositionnels S.

L'ordre de priorité des connecteurs est (par ordre croissant) : $\Leftrightarrow$, , $\wedge$, $\vee$, $\neg$. Lorsque les parenthèses ne seront pas nécessaires, elles seront omises pour plus de clarté. Par exemple, on notera $\neg\alpha \vee \beta$ la formule $((\neg\alpha) \vee \beta)$.De plus, on notera $\bigotimes_{i\in[1..n]} \alpha_i$ la formule $((((\alpha_1 \otimes \alpha_2) \otimes \alpha_3) \otimes ... \otimes \alpha_n)$ où $\otimes$ est l'un des connecteurs binaires $\{\wedge, \vee\}$.

Définition 2 (conjonction, disjonction, littéraux)
On appelle conjonction**{** XE "conjonction" **}** une formule de la forme $\bigwedge_{i\in[1..n]} \alpha_i$ où les α_i sont des formules quelconques. De même, on appelle disjonction**{** XE "disjonction" **}** une formule de la forme $\bigvee_{i\in[1..m]} \alpha_i$. Enfin, un littéral désigne tout atome ou négation d'atome (par exemple, si s est un atome, s et $\neg s$ sont des littéraux). On notera L_S l'ensemble des littéraux construits à partir de l'ensemble de symboles propositionnels S.

Définition 3 (interprétation)
Une interprétation**{** XE "interprétation" **}** I est une fonction toujours définie de S vers *{vrai, faux}*. On notera $I(s)$, $s \in S$ l'image de l'atome s (sa valeur de vérité) dans l'interprétation I et I_S l'ensemble des interprétations constructibles à partir de S.

> Si S contient n atomes, alors il existe 2^n interprétations.

A partir d'une interprétation (de la valeur de vérité des atomes), on peut définir la valeur de vérité d'une formule.

Définition 4 (valeur de vérité d'une formule)
Soient α et β deux formules de FP_S et $I \in I_S$ une interprétation. Soit $s \in S$. On définit la valeur de vérité d'une formule à partir de la valeur de vérité de ses atomes, de la façon suivante :

1. $I(\top)=vrai$ et $I(\bot)=faux$.
2. $I(\neg\beta)=vrai$ ssi $I(\beta)=faux$. $\neg\beta$ est appelée la négation**{** XE "négation" **}** de β.
3. $I(\alpha \wedge \beta)=vrai$ ssi $I(\alpha)=vrai$ et $I(\beta)=vrai$. $\alpha \wedge \beta$ est appelée la conjonction**{** XE "conjonction" **}** de α et de β.
4. $I(\alpha \vee \beta)=faux$ ssi $I(\alpha)=faux$ et $I(\beta)=faux$. $\alpha \vee \beta$ est appelée la disjonction**{** XE "disjonction" **}** de α et de β.
5. $I(\alpha \quad \beta)=faux$ ssi $I(\alpha)=vrai$ et $I(\beta)=faux$. On dira que « α implique**{** XE "implique" **}** β » ou encore « si α alors β ».
6. $I(\alpha \Leftrightarrow \beta)=vrai$ ssi $I(\alpha)=I(\beta)$.

Définition 5 (satisfaction d'une formule, modèle)
Une formule $\alpha \in FP_S$ est dite satisfaite**{** XE "formule:satisfaite" **}** pour une interprétation $I \in I_S$, ou encore I satisfait α ssi $I(\alpha)=vrai$; sinon, α est dite falsifiée**{** XE "formule:falsifiée" **}** pour I ($I(\alpha)=faux$). Une interprétation $I \in I_S$ qui satisfait une formule $\alpha \in FP_S$ est appelée un *modèle***{** XE "modèle" **}** de α.

Notation : Il est parfois plus simple de représenter une interprétation par l'ensemble des littéraux qu'elle satisfait (dont la valeur de vérité est *vrai*). Nous adopterons cette convention dans la suite du document.

Définition 6 (validité, consistance, inconsistance)
Une formule $\alpha \in FP_S$ est dite valide**{** XE "formule:valide" **}** (on dira aussi que c'est une *tautologie***{** XE "tautologie" **}**) ssi elle est satisfaite par toutes les interprétations de I_S. α est dite *inconsistante***{** XE "formule:inconsistante" **}** (ou *insatisfaisable*, *contradictoire***{** XE "formule:insatisfiable" **}**) ssi elle est falsifiée par toutes les interprétations de I_S. Une formule est dite *consistante***{** XE "formule:consistante" **}** (ou satisfaisable**{** XE "formule:satisfiable" **}**) ssi elle n'est pas inconsistante.

> Une formule valide est consistante. Une formule consistante est une formule qui admet *au moins* un modèle.

1.2 Notion de conséquence et d'équivalence logique

Définition 7 (conséquence logique)
Soient les formules $\alpha_1, \alpha_2, \ldots \alpha_n$ et β de FP_S. β est *conséquence logique* de $\alpha_1, \alpha_2, \ldots \alpha_n$ (noté $\alpha_1, \alpha_2, \ldots \alpha_n \models \beta$) ssi toute interprétation I qui satisfait $\alpha_1 \wedge \alpha_2, \wedge \ldots \wedge \alpha_n$, satisfait aussi β.

Proposition 1 (théorème de la déduction)
Soient les formules α_1, α_2, … α_n et β de FP_S. β est conséquence logique de α_1, α_2, … α_n ssi la formule $(\alpha_1 \wedge \alpha_2, \wedge \ldots \wedge \alpha_n)$ β est valide.

Proposition 2
Soient les formules α_1, α_2, … α_n et β de FP_S. β est conséquence logique de α_1, α_2, … α_n ssi la formule $(\alpha_1 \wedge \alpha_2, \wedge \ldots \wedge \alpha_n) \wedge \neg\beta$ est inconsistante.

> Cette proposition est très importante : elle nous permet de remplacer un problème de déduction par un problème de consistance.

Définition 8 (équivalence)
Deux formules α et β de FP_S sont dites *logiquement* équivalentes**{** XE "formules:équivalentes" **}** (ou encore α est *logiquement* équivalente à β, ou β est *logiquement* équivalente à α), noté $\alpha \equiv \beta$, ssi $\forall I \in I_S\ I(\alpha)=I(\beta)$ (les valeurs de vérité de α et de β sont les mêmes pour toutes les interprétations de I_S).

Proposition 3
Soient deux formules α et β de FP_S. $\alpha \equiv \beta$ ssi $\alpha \models \beta$ et $\beta \models \alpha$.

Preuve : elle découle de la définition de l'équivalence.

1.3 Formes normales

Nous allons maintenant définir les différentes formes syntaxiques des formules que nous allons utiliser. La première correspond à la forme la plus courante de représentation des bases de connaissances.

Définition 9 (cubes, clauses, base de clauses)
Un *cube* est une conjonction de littéraux. Une *clause* est une disjonction de littéraux. Une clause est dite fondamentale**{** XE "formule:fondamentale" **}** ssi elle n'est pas valide. Une *base de clauses* est un ensemble de clauses. Une base de clauses est satisfaite ssi toutes ses clauses sont satisfaites (elle est assimilée à la conjonction de ses clauses).

> Un littéral est à la fois un cube et une clause. Dans la suite du document, nous supposons que nos bases de clauses contiennent uniquement des clauses fondamentales.

Définition 10 (clauses positives, négatives, de Horn)
Une *clause positive* est une clause qui ne contient pas de littéraux négatifs.
Une *clause négative* est une clause qui ne contient pas de littéraux positifs.
La clause vide est positive et négative.
Une *clause de Horn* (resp. *reverse-Horn*) est une clause qui contient au plus un littéral positif (resp. négatif).
Une *clause de Krom* est une clause qui contient au plus deux littéraux [Cadoli 1995].

Dans le reste du document, quand nous parlerons de base de connaissances, ou tout simplement de base, il s'agira implicitement d'une base de clauses. Nous représenterons le plus souvent une clause sous sa forme implicative : $\neg a_1 \vee \neg a_2 \vee \ldots \vee \neg a_n \vee b_1 \vee b_2 \vee \ldots \vee b_m$ sera représenté par $a_1\ a_2 \ldots a_n$ $b_1\ b_2 \ldots b_m$. Une clause positive sera notée $b_1\ b_2 \ldots b_m$ (au lieu de $\top$ $b_1\ b_2 \ldots b_m$) et une clause négative sera notée $a_1\ a_2 \ldots a_n$ (au lieu de $a_1\ a_2 \ldots a_n$ $\bot$). $\neg g \vee v$ sera donc représentée par g v[3]. La partie gauche de la clause est appelée antécédent et sa partie droite conséquent.

[3] Pour nous, g v, $g \neg v$, $\neg v$ $\neg g$ et $\neg g\ v$ sont des notations équivalentes (il n'y a pas de notion de justification et d'effets). Ce n'est pas toujours vrai en logique non monotone, par exemple dans le cadre des logiques à exceptions [Reiter 1980]. Dans ce cas, nous utiliserons la notation pratiquée par cette communauté : $\neg g \rightarrow \neg v$.

Il est en général très difficile de montrer qu'une formule est conséquence logique d'une autre formule (c'est un problème coNP-complet[4]). Cependant, il existe une propriété syntaxique entre certaines formules qui permet de le déterminer immédiatement.

Définition 11 (sous-sommation)
On dira qu'une clause[5] α sous-somme (« *subsume* ») une clause α', noté $\alpha \trianglelefteq \alpha'$, ssi l'ensemble des littéraux de α est inclus dans l'ensemble des littéraux de α'.

Exemple : $\alpha = (g \quad v)$, $\alpha' = (g \quad v\ c)$. α sous-somme α', ou encore α' est sous-sommée par α.

Proposition 4
Soient δ et δ' deux **clauses fondamentales**. δ sous-somme δ' ssi δ' est conséquence logique de δ $((\delta \trianglelefteq \delta') \equiv (\delta \models \delta'))$.

Preuve : triviale.

Définition 12 (CNF)
Une formule $\alpha \in FP_S$ est dite sous forme normale conjonctive**{** XE "forme normale:conjonctive" **}** (« *Conjunctive Normal Form* », CNF) ssi α est de la forme $\bigwedge_{i\in[1..n]} \delta_i$, où les δ_i sont des clauses.

> Une base de clauses est interprétée comme une CNF. Nous verrons plus loin que la majorité des algorithmes de test de consistance utilisent en entrée des formules CNF.

Définition 13 (DNF)
Une formule $\alpha \in FP_S$ est dite sous forme normale disjonctive**{** XE "forme normale:disjonctive" **}** (« *Disjunctive Normal Form* », DNF) ssi α est de la forme $\bigvee_{i\in[1..m]} \gamma_i$ où les γ_i sont des cubes.

Il existe une autre forme normale que l'on trouve dans la littérature : la forme normale négative (Negative Normal Form).

Définition 14 (NNF)
Une formule $\alpha \in FP_S$ est dite sous forme normale négative (« *Negative Normal Form* », NNF) ssi α est soit :

- un littéral
- une conjonction $\wedge_i \alpha_i$
- une disjonction $\vee_i \alpha_i$

où les α_i sont des formules NNF. Autrement dit, une formule est NNF ssi le symbole négation apparaît uniquement devant des symboles propositionnels.

Exemple : $\neg f \wedge (g \vee (\neg e \wedge h))$ est une formule NNF. $\neg f \wedge \neg(g \vee (\neg e \wedge h))$ n'est pas NNF.

> Les formules DNF et CNF sont des cas particuliers de NNF.

On peut noter qu'une clause ou un cube est à la fois sous forme CNF et DNF (donc NNF). De plus, on trouve facilement une forme CNF ou DNF de la négation de ces formules. Cette particularité s'avérera utile lorsque nous utiliserons la Proposition 2 : si α est une CNF et β un cube ou une clause, alors $\alpha \wedge \neg\beta$ est une CNF.

Proposition 5
$\forall \alpha \in FP_S$ une formule propositionnelle, $\exists\ \alpha' \in FP_S$ sous forme normale (conjonctive, disjonctive ou négative) telle que $\alpha \equiv \alpha'$.

---

[4] Nous ne définirons pas dans cette thèse la notion de coNP-complétude. Nous renvoyons le lecteur à [Garey/Johnson 1979] par exemple pour ce qui concerne la théorie de la complexité.

[5] Cette notion ne semble pas définie pour les cubes. Elle provient de la logique booléenne où les disjonctions sont représentées par des sommes ($\neg g + \neg v$ sous-somme $\neg g + \neg v + c$). Elle existe cependant, mais l'inclusion est inversée : $\neg g \wedge \neg v \wedge c$ sous-somme $\neg g \wedge \neg v$ car $\neg g \wedge \neg v \equiv (\neg g \wedge \neg v \wedge \neg c) \vee (\neg g \wedge \neg v \wedge c)$ et $\neg g.\neg v.c$ sous-somme $\neg g.\neg v.c + \neg g.\neg v.\neg c$.

Preuve :

$\alpha' \equiv \bigvee_{I \models \alpha} I$, la disjonction des modèles de α, est une formule *DNF* (donc *NNF*) logiquement équivalente à α (couverture d'impliquants, cf. Définition 16).

$\alpha' \equiv \bigwedge_{I \models \neg\alpha} \neg I$, est une formule *CNF* (donc *NNF*) logiquement équivalente à α (couverture d'impliqués, cf. Définition 18).

> Nous verrons dans la suite que le passage d'une forme CNF à DNF (et inversement) est une opération complexe, qui est souvent utilisée en calcul propositionnel, et nous proposerons un algorithme original pour l'effectuer.

Nous allons maintenant définir la notion d'impliquant/impliqué premier. Nous utilisons les définitions proposées par [Marquis 1999] : nous renvoyons le lecteur à cet ouvrage pour de plus amples informations sur le sujet.

Définition 15 (impliquant, impliquant premier)
Soit α une formule de FP_S. Soit γ une conjonction de littéraux (un cube) de FP_S. γ est appelée *un impliquant* de α ssi $\gamma \models \alpha$. On dit de plus que γ est un *impliquant premier* de α ssi γ est un impliquant de α et $\forall\ \gamma'$ un impliquant de α, si $\gamma \models \gamma'$ alors $\gamma' \models \gamma$.

> Nous avons vu qu'un modèle peut être représenté par un cube particulier (la conjonction des littéraux satisfaits). Un impliquant correspondrait à un « sous-modèle » : une fonction d'une partie des symboles propositionnels vers {*vrai, faux*}. Pour construire un modèle à partir d'un impliquant, il suffit d'assigner une valeur de vérité quelconque aux symboles qui n'en ont pas déjà une. A partir d'un impliquant de *k* littéraux (différents), on peut donc construire $2^{|S| - k}$ modèles. Notons aussi qu'un impliquant est logiquement équivalent à la disjonction des $2^{|S| - k}$ cubes associés. Enfin, nous considérons que le cube associé à un modèle est un impliquant particulier, ce qui explique la notation utilisée dans la suite du document : « une interprétation *I* est un modèle de la formule α » sera noté « $I \models \alpha$ ».

Définition 16 (couverture d'impliquants)
Soit α une formule de FP_S. Soit $\gamma_1, \gamma_2, \ldots, \gamma_n$ des impliquants de α. On dit que $\gamma_1, \gamma_2, \ldots, \gamma_n$ est une couverture d'impliquants de α ssi $\gamma_1 \vee \gamma_2 \vee \ldots \vee \gamma_n \equiv \alpha$.

Définition 17 (impliqués, impliqués premiers)
Soit une formule $\alpha \in FP_S$. Soit δ une disjonction de littéraux (une clause). δ est appelée *un impliqué de* α ssi $\alpha \models \delta$. On dit de plus que δ est un impliqué premier de α ssi δ est un impliqué de α et $\forall\ \delta'$ impliqué de α, *si* $\delta' \models \delta$ alors $\delta \models \delta'$.

Définition 18 (couverture d'impliqués)
Soit α une formule. Soit $\delta_1, \delta_2, \ldots, \delta_n$ des impliqués de α. On dit que $\delta_1, \delta_2, \ldots, \delta_n$ est une couverture d'impliqués de α ssi $\delta_1 \wedge \delta_2 \wedge \ldots \wedge \delta_n \equiv \alpha$.

[Siegel 1987] présente un cadre de sélection des impliqués d'une base de connaissances avec la notion de *champ de production*. Elle est définie dans le cadre de la logique du premier ordre : nous donnons ici sa restriction au calcul propositionnel.

Définition 19 (problème du champ de production)
Soit $BC \subset FP_S$ une base de clauses (les connaissances) et $CP \subset FP_S$ une autre base de clauses appelée *Champ de production*. Déterminer l'ensemble de clauses (impliqués) Produites(BC,CP) = $\{c \mid c \in CP$ et $BC \models c\}$.

Pour des raisons pratiques, on limite l'ensemble des solutions trouvées (une base de clauses) à ses impliqués premiers (qui ne sont pas forcément des impliqués premiers de *BC* !).

Nous présenterons dans cette thèse une méthode de résolution de ce problème quand le champ de production est d'une forme très particulière : CP = {*c* | *c* est une clause dont tous les littéraux sont dans *P*, avec *P* un ensemble consistant de littéraux}.

1.4 Utilisation de la notation ensembliste

Nous présentons dans cette thèse des solutions algorithmiques pour calculer des formules particulières du calcul propositionnel. Nos algorithmes ne travaillent pas sur des formules mais sur des ensembles de formules : une base de clauses est un ensemble de clauses, une clause peut être représentée par l'ensemble de ses littéraux, une interprétation par l'ensemble des littéraux qu'elle satisfait, etc. Nous allons donc « jongler » dans cette thèse entre une notation ensembliste qui est plus proche de l'implantation des algorithmes et une notation logique qui correspond à la sémantique de ces ensembles.

Nous adopterons les conventions suivantes dans le reste du document : les formules logiques seront représentées par des lettres grecques, les ensembles de formules par des lettres majuscules. Nous utiliserons les opérations ensemblistes courantes $\cup$ (union), $\cap$ (intersection) et \ (différence).

Définition 20 (notation ensembliste)
Soient E et F deux ensembles de formules de FP_S.

- $E^\wedge$ dénote la conjonction des éléments de E.
- $E^\vee$ dénote la disjonction des éléments de E.
- $E^\neg$ dénote l'ensemble des formules complémentaires de E[6] ($E^\neg = \{\neg e \mid e \in E\}$).

On en déduit les propriétés suivantes :

Propriété 1

- $(E \cup F)^\wedge \equiv E^\wedge \wedge F^\wedge$
- $(E \cup F)^\vee \equiv E^\vee \vee F^\vee$
- $\{\alpha\}^\wedge \equiv \{\alpha\}^\vee \equiv \alpha$ où $\alpha \in FP_S$.

On retrouve alors les conventions usuelles du calcul propositionnel qui associent à la clause vide $\bot$ et au cube vide la tautologie : $\varnothing^\vee \equiv \bot$ *et* $\varnothing^\wedge \equiv \top$.

Exemples :
Soit BC une base de clauses de FP_S. $BC^\wedge$ dénote la conjonction des clauses de BC. $BC^\wedge$ dénote donc une formule sous forme CNF. Soit $I \in I_S$. $I^\wedge$ dénote la conjonction des littéraux satisfaits par I. Plus généralement, soit $E \subseteq L_S$ un ensemble de littéraux. $E^\vee$ dénote une clause et $E^\wedge$ dénote un cube.

Exemple de passage de la forme ensembliste à la formule correspondante :
$\{\{a,\neg b,c\}^\vee,\{a,\neg e,f\}^\vee,\{\neg b,\neg f\}^\vee\}^\wedge \quad \equiv \quad (a \vee \neg b \vee c) \wedge (a \vee \neg e \vee f) \wedge (\neg b \vee \neg f)$

Nous pouvons résumer nos définitions par le tableau suivant :

Elément logique	Notation logique	Notation ensembliste
Symbole	f	$f, \{f\}^\vee, \{f\}^\wedge, \{\neg f\}^\neg, \ldots$
Littéral	$\neg f$	$\neg f, \{\neg f\}^\vee, \{\neg f\}^\wedge, \{f\}^\neg, \ldots$
Disjonction	$f \vee g$	$\{f, g\}^\vee, \{\{f\}^\wedge, \{g\}^\wedge\}^\vee, \ldots$
Conjonction	$f \wedge g$	$\{f, g\}^\wedge, \{\{f\}^\vee, \{g\}^\vee\}^\wedge, \ldots$
Implication	$f \quad g$	$\{\neg f, g\}^\vee, \ldots$
Equivalence	$f \Leftrightarrow g$	$\{\{f, \neg g\}^\vee, \{\neg f, g\}^\vee\}^\wedge, \ldots$
Tautologie	$\top$	$\varnothing^\wedge$
Clause vide	$\bot$,	$\varnothing^\vee$

Proposition 6
Soient E et E' deux ensembles de littéraux. $E^\vee \trianglelefteq E'^\vee$ ssi $E \subseteq E'$.

Preuve : c'est la définition de la sous-sommation (Définition 11, page 20).

Nous sommes maintenant en mesure de définir la notion de formule construite sur un sous-langage :

[6] Dans le cas particulier où $E=\{l\}$, $\{l\}^\neg$ dénote le littéral complémentaire de l : $\{l\}^\neg = \begin{cases} s \text{ si } l = \neg s \\ \neg s \text{ si } l = s \end{cases}$

Définition 21 (P-impliquant, P-impliqué)
Soit $P \subseteq L_S$. Soit $E \subseteq P$. Soit $\alpha \in FP_S$. $E^\wedge$ est appelé un P-impliquant de α ssi $E^\wedge \models \alpha$. $E^\vee$ est appelé un P-impliqué de α ssi $\alpha \models E^\vee$.

Proposition 7
Soit $P \subseteq L_S$. Soit $\alpha \in FP_S$. Tout P-impliquant (resp.P-impliqué) de α est un impliquant (resp. impliqué) de α.

Preuve : elle découle directement de la définition précédente.

La notion de P-impliqué doit être rapprochée de la notion de champ de production précédemment citée (cf. Définition 19 page 21). Le champ de production est : CP = {l'ensemble des clauses construites à partir de P}.
Cette notion de restriction à un sous-langage P va nous être très utile. Dans cette thèse, nous allons proposer des solutions algorithmiques pour calculer des P-impliquants et P-impliqués, avec certaines conditions sur P. On trouve dans [Castell 1997] un ensemble d'algorithmes pour le calcul de P-impliqués et de P-impliquants plus généraux, c'est à dire sans restrictions sur P.

Définition 22 (réduction d'une formule par un littéral)
Soit $\gamma \in FP_S$. Soit l un littéral de L_S. La réduction de γ par l, notée $\gamma[l]$, consiste à propager la satisfaction de l dans la formule. Elle peut se définir récursivement par :

1. $l[l] = T$
2. $\neg l[l] = \bot$
3. $(\neg\alpha)[l] = \neg(\alpha[l])$
4. $(\alpha \wedge \beta)[l] = (\alpha[l] \wedge \beta[l])$
5. $(\alpha \vee \beta)[l] = (\alpha[l] \vee \beta[l])$

avec α et β deux formules propositionnelles.

Lorsque γ est une base de clauses, cette opération enlève de BC les clauses qui contiennent l et enlève $\neg l$ aux clauses de BC qui le contiennent.

Exemple : $BC = \{\{\neg g, c\}^\vee, \{\neg v, \neg c\}^\vee, \{g,v\}^\vee\}$. $BC[\neg c] = \{\{\neg g\}^\vee, \{g,v\}^\vee\}$.

Dans la suite du document, pour alléger l'écriture, nous noterons $BC[l_1,l_2,\ldots,l_n]$ la base réduite $(((BC[l_1])[l_2])\ldots)[l_n]$.

Proposition 8 (expansion de Shannon)
Soit BC une base de clauses de FP_S. Soit $l \in L_S$. $BC^\wedge \equiv (BC[l]^\wedge \wedge l) \vee (BC[\neg l]^\wedge \wedge \neg l)$.

Pour le démontrer, nous allons utiliser le résultat suivant :

Lemme 1
Soit BC une base de clauses de FP_S. Soit $l \in L_S$. $BC[l]^\wedge \wedge l \equiv BC^\wedge \wedge l$

Preuve :
) Soit M un modèle de BC[l]^∧l. M(l)=vrai. Donc toutes les clauses de BC où apparaît l sont satisfaites par M. Les clauses où apparaît ¬l sont satisfaites par un autre littéral (sinon ces clauses ne seraient pas satisfaite dans BC[l]). Les autres clauses apparaissant à la fois dans BC et BC[l] sont forcément satisfaites. On a donc bien $BC[l]^\wedge \wedge l \models BC^\wedge \wedge l$.
⇐) Par l'absurde. Soit M un modèle de $BC^\wedge \wedge l$. Supposons qu'il existe une clause $cl \in BC[l]$ telle que cl soit falsifiée par M. Or, par construction,

- soit cl est une clause de BC, auquel cas M ne peut pas être un modèle de BC, contradiction.
- soit $cl \vee \neg l$ est une clause de BC. Or $M(l)=vrai$. Donc $cl \vee \neg l$ est aussi falsifiée dans BC. Contradiction.

On a donc $BC^\wedge \wedge l \models BC[l]^\wedge \wedge l$.

Preuve proposition :
$BC^\wedge \equiv BC^\wedge \wedge (l \vee \neg l) \equiv (BC^\wedge \wedge l) \vee (BC^\wedge \wedge \neg l) \equiv (BC[l]^\wedge \wedge l) \vee (BC[\neg l]^\wedge \wedge \neg l)$.

On en déduit les propriétés suivantes :

Propriété 2
Soient B_1 et B_2 deux bases de clauses de FP_S. Soit l un littéral de L_S.
1. Si $B_1[l]$ est consistante alors B_1 est consistante.
2. $B_1^\wedge \wedge l \equiv B_2^\wedge \wedge l$ ssi $B_1[l]^\wedge \equiv B_2[l]^\wedge$
3. Si $B_1^\wedge \models B_2^\wedge$ alors $B_1[l]^\wedge \models B_2[l]^\wedge$

> **Preuves :**
> Découlent de la proposition ou du lemme précédent.

Définition 23 (hitting set, minimal hitting set)
Soit $E = (E_1,E_2,\ldots,E_n)$, où $E_i \subseteq S$. On appelle « *hitting set* » de E un ensemble de littéraux F tq $\forall k, 1 \leq k \leq n$, $E_k \cap F \neq \emptyset$. On appelle « *minimal hitting set* » de E un hitting set F de E tq $\nexists F'$ un hitting set de E tq $F' \subset F$. On notera par $MHS(E)$ l'ensemble des « hitting set » minimaux de E.

Exemple : $E = \{\{c,g\},\{g,v\}\}$. $MHS(E)=\{\{g\},\{c,v\}\}$.

> Nous verrons plus tard que cette opération peut être utilisée pour passer d'une formule CNF en DNF et vice versa. C'est une opération complexe pour laquelle nous proposerons un algorithme original.

2 Le problème SAT

Nous allons nous pencher ici sur l'un des problèmes de base du calcul propositionnel : déterminer la consistance d'une base de clauses (on dira aussi déterminer la *satisfiabilité* de la base de clauses, d'où le nom de *problème SAT*). Le lecteur intéressé par ce problème trouvera dans [Gu, et al. 1997] un résumé des 25 dernières années de recherche sur SAT.

2.1 Présentation

Le problème SAT est un problème de base en logique propositionnelle, mais c'est aussi un problème NP-complet [Cook 1971]. Ce résultat théorique peut décourager, mais les heuristiques et autres résultats empiriques des chercheurs apportent des solutions pratiques à ce problème : en effet, depuis l'organisation *du 2nd DIMACS Challenge on Satisfiability* en 1993 [Johnson/Trick 1996][7], des progrès considérables ont été faits dans le domaine, à tel point qu'il est désormais possible, par exemple, de résoudre des problèmes de planification réels (nécessitant plusieurs centaines de variables propositionnelles) par codage du problème en calcul propositionnel (SATPLAN [Kautz/Selman 1996]). Certains attribuent les bonnes performances de ces algorithmes « généraux » (par rapport aux algorithmes spécialisés) aux heuristiques très élaborées qui sont utilisées. Comme elles ont la particularité d'être indépendantes du domaine d'application (ordonnancement, planification, diagnostic,…), toute amélioration dans ce cadre est immédiatement répercutée sur les applications. Notons par exemple que des prouveurs en logique du premier ordre à domaine fini passent parfois par une transformation du problème en calcul propositionnel (MACE [McCune 1994], ModGen [Kim/Zhang 1994]). De plus, on voit apparaître maintenant des extensions de cette procédure pour résoudre d'autres problèmes du calcul propositionnel comme la satisfiabilité de formules propositionnelles quantifiées (QBF) [Cadoli, et al. 1998, Rintanen 1999].

Le problème de la satisfaction peut être défini théoriquement pour n'importe quelle formule booléenne, et il reste NP-complet. Nous avons vu précédemment qu'il existe pour toute formule une formule logiquement équivalente sous forme CNF. Le problème SAT [Garey/Johnson 1979] suppose donc que les formules sont des CNF (déterminer la consistance d'une formule DNF est dans P). Ce choix est discutable car transformer un problème exprimé sous une forme propositionnelle quelconque en une formule CNF « intelligemment » est un problème en soit. Nous verrons dans la partie 2.5 que ce choix est remis en question. De plus, on ne considérera que des formules fondamentales (non tautologiques). Enfin, on parlera le plus souvent de problèmes k-SAT, k dénotant suivant la littérature le nombre *exact* ou *maximal* de littéraux par clauses. Ainsi, 3-SAT représentera pour nous l'ensemble des bases de clauses ayant *exactement* 3 littéraux distincts par clauses (fondamentales). Si le problème général est NP-complet, il existe des cas où ce problème devient traitable : il a été prouvé que 2-SAT, l'ensemble des problèmes constitués de clauses binaires, peut être résolu en temps polynomial par rapport à la taille de la base de clauses [Even, et al. 1976]. De même si la base est constituée de clauses de Horn (Horn-

[7] C'était un concours pour comparer les performances des différents algorithmes résolvant SAT et/ou d'autres problèmes connexe (coloriage de graphe, recherche de cliques). Pour plus d'informations, http://dimacs.rutgers.edu/challenges/.

SAT), de Krom (Krom-SAT). On dira que 2-SAT, Horn-SAT et Krom-SAT sont des *classes polynomiales* pour SAT.

Il y a une distinction importante à faire entre deux classes de méthodes, systématiques et stochastiques. Dans les premières, chronologiquement les plus anciennes [Davis/Putnam 1960, Robinson 1965], on accepte de parcourir tout l'espace de recherche si cela est nécessaire (preuve de l'inconsistance) ; dans les secondes, beaucoup plus récentes [Gu 1988, Selman, et al. 1992], on limite l'exploration à certaines parties de l'espace de recherche. Dans la littérature, elles sont aussi caractérisées par la réponse qu'elles donnent au problème. Les premières résolvent le problème. Les secondes permettent de prouver la consistance de la formule booléenne (en donnant un modèle), par contre, elles ne peuvent pas faire la preuve de l'inconsistance de cette formule. On dira alors que les premières approches sont « complètes » et les secondes « incomplètes » (notons que n'importe quelle méthode systématique peut être rendue incomplète en limitant son temps d'exécution par exemple). Pourquoi s'intéresser aux méthodes stochastiques alors ? En pratique, elles permettent de *prouver la consistance* de problèmes de très grande taille.

2.2 Approches « systématiques »

Nous pouvons distinguer dans cette approche deux méthodes principales : le principe de résolution [Robinson 1965] et la procédure de Davis et Putnam [Davis/Putnam 1960, Davis, et al. 1962][8].

2.2.1 Le principe de résolution

Il consiste à produire des clauses (appelées résolvantes) conséquences logiques de la base à partir d'autres clauses de la base : si la base est inconsistante, alors la clause vide (⊥,) sera produite. Si on ne peut la produire, alors la base est consistante.

Définition 24 (résolvante)
Soient δ_1 et δ_2 deux clauses de FP_S. δ_1 et δ_2 se résolvent ssi il existe un littéral $l \in L_S$ tel que l apparaît dans δ_1 et $\neg l$ apparaît dans δ_2 ($\delta_1 \equiv \delta_1' \vee l$, $\delta_2 \equiv \delta_2' \vee \neg l$). La clause $\delta_1' \vee \delta_2'$ est la résolvante des clauses δ_1 et δ_2.

> Prenons par exemple les clauses $\delta_1 \equiv c\ b$ et $\delta_2 \equiv b\ d$. La clause $c\ d$ est la résolvante de δ_1 et δ_2. Prenons maintenant $\delta_1 \equiv c\ b\ e$ et $\delta_2 \equiv e\ b\ d$. Ces deux clauses se résolvent (en b et en e) mais la résolvante est une tautologie. Dans la pratique, on se limite aux clauses qui se résolvent sur un seul littéral pour éviter de produire des tautologies.

Proposition 9
Soient δ_1 et δ_2 deux clauses de FP_S. Si δ_1 et δ_2 se résolvent, alors leur résolvante est conséquence logique de $\delta_1 \wedge \delta_2$.

> La résolvante est donc un impliqué de la base.

Définition 25 (preuve par résolution)
Soit *BC* une base finie de clauses et δ une clause fondamentale. Une preuve par résolution de δ dans *BC* est une séquence finie $<\delta_1,\ldots,\delta_n>$ telle que :
- $\delta_n \prec \delta$
- $\forall\ k \in [1..n]$, $\delta_k \in BC$ ou il existe i et j tels que $1 \le i < j < k$ et δ_k est la résolvante de δ_i et δ_j

Lorsque la clause δ est la clause vide ( ), on dit que $<\delta_1,\ldots,\delta_n>$ est une *réfutation* de *BC*.

Proposition 10
Soit *BC* une base de clauses. *BC* est inconsistante ssi il existe une réfutation de *BC*.

Dans le cas général, il existe un très grand nombre de résolutions possibles. Il existe diverses manières de limiter ce nombre de résolutions : par exemple la P-résolution (resp. N-résolution) [Robinson 83]consistant à utiliser au moins une clause positive (resp. négative), la résolution linéaire consistant à toujours utiliser la dernière résol-

[8] Il nous faut noter que le nom de cette procédure fait référence à [Davis/Putnam 1960] alors que cet article décrit une méthode basée sur le principe de résolution ! Cette confusion est due à [Davis, et al. 1962] qui présentent leur méthode comme un changement syntaxique mineur de l'algorithme précédent. [Dechter/Rish 1994] utilisent les termes de *DP-elimination* et *DP-backtracking* pour référencer les deux algorithmes. Nous continuerons néanmoins à appeler ce dernier procédure de Davis et Putnam. La majorité des chercheurs utilise aussi cette convention (certains utilisent parfois le nom de procédure de Davis, Putnam et Loveland (DPL) pour couper la poire en deux).

vante [Kowalsky/Kuehner 1971], la résolution bornée limitant la taille des résolvantes ajoutées à la base [Génisson/Siegel 1994], etc.

L'algorithme initial proposé par [Davis/Putnam 1960] et repris par [Dechter/Rish 1994] est un algorithme de résolution dirigée : il choisit à chaque étape une variable et effectue toutes les résolutions possibles sur cette variable. Ensuite, les clauses contenant la variable sont enlevées de la base. La base obtenue conserve la consistance de la base originale, mais pas son équivalence logique. De plus, les simplifications de type clauses unitaires et littéraux purs (voir plus loin) étaient déjà utilisées dans cet algorithme.

Le principe de résolution, lorsque l'on procède par saturation, est adapté à la preuve de l'inconsistance d'une base de clauses, car on obtient la preuve de l'inconsistance dès la production de la clause vide. La preuve de la consistance est plus difficile, car il faut prouver que l'on ne peut pas produire la clause vide (plus de production de résolvante).

L'un des défauts de cette méthode est que la production de résolvantes peut être énorme, donc son implantation demande de grandes ressources mémoires. D'après [Gu, et al. 1997], la principale raison de l'intérêt porté à cette technique est moins son utilisation dans SAT que la possibilité de l'utiliser dans le cadre de la logique des prédicats du premier ordre.

Une étude récente de [Chatalic/Simon 1999] montre qu'un algorithme basé sur la résolution comme le DP originel n'est pas dénué d'intérêt. Ils obtiennent en effet des résultats comparables aux meilleurs prouveurs actuels sur certaines instances structurées.

2.2.2 La procédure de Davis et Putnam

La procédure de Davis et Putnam est la méthode la plus connue et sans doute la plus utilisée, dans ses diverses variantes, pour résoudre SAT. Celle que l'on utilise aujourd'hui est une variante que l'on doit à [Davis, et al. 1962] pour éviter l'explosion mémoire due à la production des résolvantes : ils remplacent l'étape de résolution sur la variable par une *règle de découpage* (*« splitting rule »*) (qui correspond à l'expansion de Shannon). La base est consistante ssi l'une des deux sous-bases formées après assignation d'une valeur de vérité à la variable est consistante.

C'est un algorithme de type *retour arrière chronologique* (*« bactracking »* en anglais) : il s'agit de parcourir un arbre binaire, chaque nœud de l'arbre représentant une base de clause non triviale[9], chaque branche associant une valeur de vérité à une variable, et chaque feuille correspondant à base triviale.

Voici une version personnelle de cette procédure. Pour plus de détails sur notre implantation, nous renvoyons le lecteur à l'annexe II, page 177, où se trouvent les sources en JAVA des différents algorithmes présentés dans la thèse.

```
DP(BC)
// retourne vrai si BC est consistante, faux sinon
        // la base est trivialement consistante
        Si BC = ∅  alors retourner vrai  Finsi ;
        // la base est trivialement inconsistante
        Si ⊥ ∈ BC alors retourner faux  Finsi ;
        //  on cherche un littéral permettant de simplifier la base
        l ← LittéralPourSimplifier(BC) ;
        Si (l≠null) alors // Si l existe
                Retourner DP(BC[l])
        Finsi ;
        // Choix d'une varible de la base
        l ← Choix_Symbole(BC) ;
        // application de l'expansion de Shannon
        Si DP(BC[l]) = faux alors retourner DP(BC[¬l]) Finsi;
        retourner vrai;
        Finsi
```

Algorithme 1 Davis et Putnam

Le principe de cet algorithme est le suivant :

[9] Une base de clauses est considérée comme triviale ssi soit elle est vide, soit elle contient une clause vide ou deux clauses unitaires complémentaires.

La procédure prend en paramètre une base de clauses *BC*. Si cette base n'est pas assez simple pour en déterminer immédiatement la consistance, elle va être réduite en satisfaisant un de ses littéraux (voir Définition 22, page 23), puis la procédure est relancée pour essayer d'en déterminer la consistance, jusqu'à obtention d'une base « trivialement inconsistante », c'est à dire contenant une clause nulle ou encore deux clauses unitaires complémentaires, ou d'une base « trivialement consistante », c'est à dire ne contenant plus de clauses à satisfaire.

Pour que le résultat soit correct, il faut utiliser la Proposition 8 (expansion de Shannon). Si la base obtenue après satisfaction du littéral est consistante, on répond que la base est consistante, sinon on réitère le processus avec la négation du littéral (c'est ici que si l'algorithme développe ses deux branches : il explose combinatoirement).

Il existe des cas où il n'est pas nécessaire de développer ces deux branches. On peut simplifier la base par certains littéraux tels que la base réduite par ces littéraux préserve la consistance de la base originale (la première est consistante ssi la seconde l'est). La fonction LittéralPourSimplifier(BC) retourne des littéraux qui ont cette propriété (ou null si il n'y en a pas).

Voici quelques exemples de simplifications de l'arbre de recherche : les deux premières sont celles que l'on retrouve dans la procédure originale de Davis et Putnam. Les autres peuvent être considérées comme des généralisations des deux premières.

Propagation des clauses unitaires

Une clause unitaire est une clause de taille 1 (donc un littéral). Dans ce cas, il suffit de rechercher la consistance de la base en satisfaisant ce littéral. Si ce n'est pas possible, c'est que la base est inconsistante. Lorsque l'on travaille avec des bases 3-SAT, une grande partie de l'arbre de recherche consistera à satisfaire ces clauses unitaires (cf. Figure 1, page 36).

Propagation des littéraux purs

Définition 26 (Littéral pur)
Soit $BC \in FP_S$ une base de clauses. Soit $l \in L_S$. On dira que l est *pur* ssi $\neg l$ n'apparaît pas dans *BC*

Proposition 11
Soit $BC \in FP_S$ une base de clauses. Soit $l \in L_S$ un littéral pur de *BC*. *BC[l]* est consistante ssi *BC* est consistante.

Preuve :
) Supposons *BC[l]* consistante. Alors $\exists$ *M* un modèle de *BC[l]* construit sur l'ensemble des variables de BC[l] (qui ne contient ni l ni $\neg$l). Nous pouvons étendre ce modèle *M* en *M'* tel que $\forall s \in L_S, s \neq l$, *M'(s) = M(s)* et *M'(l)=vrai*. *M'* est un modèle de *BC*, donc *BC* est consistante.
$\Leftarrow$) Trivial car $BC[l] \subset BC$.

Ces deux simplifications ont pour but de limiter l'explosion combinatoire résultant du choix de la valeur de vérité des symboles de la base : la valeur de vérité de certains symboles devient fixée, il n'y a donc plus d'alternative pour ces symboles.
Comme la satisfaction d'une clause unitaire raccourcit le plus souvent d'autres clauses de la base, de nouvelles clauses unitaires peuvent apparaître. Leur satisfaction elle aussi peut produire de nouvelles clauses unitaires, et ainsi de suite. C'est pourquoi on parle de *propagation* des clauses unitaires.
Ce même phénomène est observé pour les littéraux purs : comme la satisfaction de littéraux purs enlève uniquement des clauses de la base, les occurrences des littéraux de ces clauses diminue. Lorsqu'un littéral a une occurrence nulle, son complémentaire est pur (s'il apparaît dans la base).

Le pseudo-code suivant correspond à la fonction LittéralPourSimplifier(BC) que l'on retrouve dans la procédure originale de Davis et Putnam.

```
LittéralPourSimplifier(BC)
// retourne un littéral dont la satisfaction permet de simplifier
// la base tout en préservant sa consistance.
   si ∃ l littéral tel que l est une clause unitaire de BC alors retourner l Finsi ;
   si ∃ l littéral tel que l est un littéral pur de BC alors retourner l Finsi ;
   retourner null
```

> Il faut noter que ces deux simplifications sont de natures différentes : *la première préserve l'équivalence logique* (c'est-à-dire que $BC \equiv BC[l] \wedge l$), alors que *la seconde ne préserve que la consistance* (la base est consistante ssi la base simplifiée est consistante). Cette différence nous empêchera d'utiliser la seconde simplification dans le cadre de l'énumération de modèles (nous proposerons un cadre restreint de son utilisation).
> Notons de plus qu'il faut satisfaire toutes les clauses unitaires d'une base pour satisfaire cette base (trivial) alors qu'il n'est pas nécessaire de satisfaire tous les littéraux purs d'une base pour la satisfaire : si $BC = \{ a\ b\ \ \ c\}$ par exemple, il suffit de satisfaire l'un des trois littéraux purs $\{\neg a, \neg b, c\}$ pour satisfaire la base.

On trouve une autre catégorie de littéraux permettant de simplifier la base de clauses : les littéraux impliqués. C'est une extension qui est apparue dans le prouveur C-SAT [Boufkhad 1996, Dubois, et al. 1996] et que l'on retrouve dans la méthode AVAL [Audemard, et al. 1999].

Propagation des littéraux impliqués

Définition 27 (Littéral impliqué)
Soit $BC \in FP_S$ une base de clauses. Soit $l \in L_S$. On dira que l est un *littéral impliqué* de BC ssi $BC^{\wedge} \models l$.

Notons qu'un littéral apparaissant dans une clause unitaire est un littéral impliqué. C'est pourquoi on peut considérer qu'un littéral impliqué est une généralisation des clauses unitaires. Les clauses unitaires permettent donc de trouver des littéraux impliqués triviaux, grâce à une propriété syntaxique. Trouver tous les littéraux impliqués peut être coûteux. On se restreindra le plus souvent à déterminer une partie seulement d'entre eux. Par exemple, [Boufkhad 1996, Dubois, et al. 1996] utilisent les littéraux impliqués générés lors de la propagation de littéraux apparaissant dans des clauses binaires (voir section suivante). [Audemard, et al. 1999] proposent eux aussi une méthode de recherche de littéraux impliqués parmi les littéraux apparaissant dans des clauses binaires. Le plus souvent, ces calculs s'effectuent au milieu de l'arbre de recherche, là où les clauses binaires sont les plus abondantes.

Comme pour les clauses unitaires, la réduction de la base par des littéraux impliqués conserve l'équivalence logique de la base.

Une autre simplification peut être effectuée par une méthode que l'on peut considérer comme une généralisation de la notion de littéraux purs : l'évaluation sémantique [Jeannicot, et al. 1988, Oxusoff/Rauzy 1989].

Evaluation sémantique

Cette méthode est basée sur le théorème de la partition du modèle :

Proposition 12 (théorème de partition du modèle [Oxusoff/Rauzy 1989])
Soit BC une base de clauses et $\{l_1,\ldots,l_j,\ldots,l_k\}$ un ensemble consistant de littéraux. Si $BC[l_1,\ldots,l_k] \subseteq BC[l_1,\ldots,l_j]$ alors $BC[l_1,\ldots,l_j]$ est consistante ssi $BC[l_1,\ldots,l_k]$ est consistante.

Ce théorème est une généralisation de la notion de littéraux purs. En effet, si l est un littéral pur dans BC, alors $BC[l] \subseteq BC$. On retrouve aussi ce théorème sous le nom « *autarky pruning* » dans [Monien/Speckenmeyer 1985]. [Oxusoff/Rauzy 1989] présentent en plus un lemme permettant de détecter les coupures de manière peu coûteuse.

L'évaluation sémantique consiste à appliquer ce théorème dans la procédure de Davis et Putnam. Lorsque l'on a réduit une base de clauses par satisfaction successive des littéraux $\{l_1,l_2,\ldots l_k\}$, on vérifie si $\exists i \in [1..k[$ tel que $BC[l_1,..l_k] \subseteq BC[l_1\ldots,L_i]$. Dans ce cas, on effectue le retour arrière uniquement en $\neg l_i$: les branches l_{i+1} à l_k sont coupées. L'obtention du plus petit i vérifiant cette condition permet d'effectuer la plus grande coupure dans l'arbre selon ce théorème.

[Mazure 1999] a proposé récemment une généralisation de ce théorème : il remplace l'inclusion entre deux bases par la conséquence logique.

Proposition 13 (théorème de partition du modèle généralisé [Mazure 1999])
Soit BC une base de clauses et $\{l_1,\ldots,l_j,\ldots,l_k\}$ un ensemble consistant de littéraux. Si $BC[l_1,\ldots,l_j] \models BC[l_1,\ldots,l_k]$ alors $BC[l_1,\ldots,l_j]$ est consistante ssi $BC[l_1,\ldots,l_k]$ est consistante.

Cette définition a l'avantage de regrouper les 4 autres simplifications sous une même propriété. On retrouve trivialement la notion de littéraux impliqués (car l'équivalence logique est préservée), donc celle de clauses unitaires. De même, comme $BC \subseteq BC' \quad BC' \models BC$, on retrouve le théorème de la partition du modèle, donc la propagation des littéraux purs. Néanmoins, déterminer qu'une base est conséquence logique d'une autre base étant coNP-complet, son utilisation systématique pour effectuer des coupures dans l'arbre de recherche n'est pas une solution envisageable en pratique.

Plus généralement, toute méthode de simplification « sémantique » (littéraux impliqués, théorèmes de partition du modèle) sont souvent plus coûteuses que les simplifications « syntaxiques » proposées par Davis et Putnam. En outre, elles sont trop souvent tributaires d'une implantation particulière pour être réellement efficaces. Les résultats obtenus par [Mazure 1999] sur une version restreinte de son théorème de partition du modèle généralisé à la propagation unitaire le montrent bien : en nombre de nœuds explorés, sa méthode est plus intéressante qu'une méthode de Davis et Putnam standard. Malheureusement, cela ne se traduit pas en termes de temps de résolution.

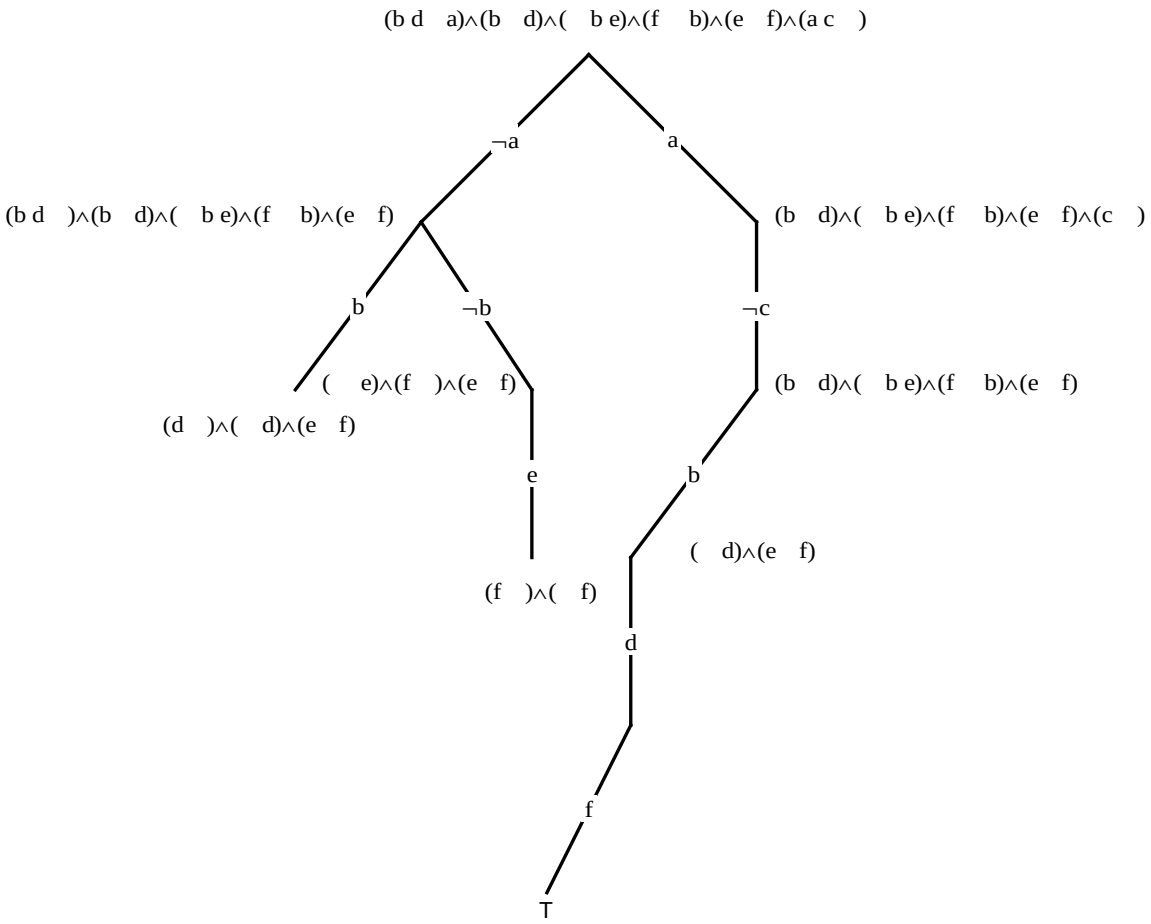
D'un autre coté, leur pouvoir « sémantique » leur permet de résoudre des instances insolubles à l'aide des simplifications classiques. Il semble par exemple que l'utilisation de l'évaluation sémantique généralisée (restreinte à la propagation unitaire) ait de bonnes performances sur les instances structurées.

Il existe encore d'autres simplifications sémantiques parfois utilisées dans la procédure de Davis et Putnam. Nous noterons par exemple la prise en compte de symétries [Benhamou/Saïs 1994, Crawford, et al 1996] qui permet de résoudre certaines instances structurées insolubles sinon (comme le problème de pigeons par exemple). On retrouve dans Rel-SAT [Bayardo/Schrag 1997] l'utilisation de techniques de CSP afin d'effectuer des retours arrières intelligents à partir de l'analyse de conflits (*relevance backjumping*) qui s'avère elle aussi efficace sur des instances structurées (issues de problèmes réels). [Marques-Silva/ Sakallah 1996] proposent aussi un prouveur SAT incorporant différentes techniques de retours arrières intelligents appelé GRASP.

Exemple d'arbre développé par un algorithme de Davis et Putnam

On peut représenter l'exécution d'un algorithme de Davis et Putnam par un arbre ayant au plus 2 branches par nœuds. Chaque branche correspond à la satisfaction d'un littéral de la base, deux branches descendant d'un même nœud correspondant à deux littéraux complémentaires. Dans le pire des cas, pour n symboles propositionnels et 2 branches exactement par nœud (l'arbre est dit *complet*), on obtient 2^n feuilles (ou $2^{n+1} - 1$ nœuds), soit une interprétation par feuille. Dans la réalité, les arbres développés par l'algorithme ne sont jamais complets, d'abord parce que si la base est consistante, on s'arrête dès que l'on a la preuve de la consistance, ensuite l'assignation d'une valeur de vérité à un symbole simplifie la base et permet d'éliminer des interprétations avant d'avoir assigné une valeur de vérité à toutes les variables (on n'éliminera pas *une* mais *plusieurs* interprétations d'un coup). Dans [Crawford/Auton 1993], les auteurs disent développer réellement, pour $20 \leq n \leq 50$, au seuil, entre $2^{n/5}$ et $2^{n/17}$ nœuds.

Voici par exemple l'arbre que pourrait développer la procédure de Davis et Putnam sur la base $(b\ d\ \ a) \wedge (b\ \ d) \wedge (\ \ b\ e) \wedge (f\ \ b) \wedge (e\ \ f) \wedge (a\ c\ \ )$:



La taille et la forme de l'arbre développé vont donc dépendre du choix des littéraux effectué à l'étape 6.

Les performances d'une méthode de type DP dépendent de deux choses : l'implantation - la représentation des clauses et de certains mécanismes comme la propagation de clauses unitaires et des littéraux purs - et surtout l'heuristique de branchement. L'heuristique est primordiale car elle permet de réduire considérablement le nombre de nœuds développés. Une bonne implantation est nécessaire pour résoudre rapidement les problèmes. Mais ces deux caractéristiques sont liées : il sera parfois difficile de trouver une bonne implantation d'une heuristique. La performance se situe donc entre les deux.

Nous allons maintenant nous intéresser aux différentes heuristiques qui ont été proposées. Elles permettent d'implanter la fonction ChoixSymbole(BC).

Heuristiques de choix du littéral

Nous présentons ici notre interprétation du développement des heuristiques pour SAT. Comme il n'existe pas une heuristique meilleure que toutes les autres sur tous les types de problèmes, nous donnons quelques principes généraux de la recherche heuristique pour SAT puis nous présentons les deux heuristiques les plus utilisées à l'heure actuelle.

Deux stratégies de recherche

Nous avons vu qu'une technique d'énumération comme la procédure de Davis et Putnam développe un arbre de recherche dont les feuilles sont soit des feuilles d'échec (une clause vide apparaît) soit une feuille de succès (auquel cas l'algorithme s'arrête). [Wang 1997] propose a posteriori deux stratégies de développement de l'arbre de recherche afin de mettre en évidence le comportement des heuristiques : la première va essayer de favoriser la recherche d'une solution *(« favoring identifying a satisfiable solution »*, FIS) quand la deuxième va essayer de simplifier le problème (*« favoring simplifying the system »*, FSS).
Une stratégie FIS a pour but de simplifier l'instance à traiter jusqu'à l'obtention d'une instance dont on peut trouver facilement (si elle existe) une solution. Comme DP s'arrête dès qu'une solution est trouvée, cette approche convient bien aux instances qui contiennent beaucoup de solutions, car il y aura peu de retour arrière à effec-

tuer. Cette approche est désastreuse sur les instances inconsistantes car elle va provoquer énormément de retours arrières (l'arbre développé est très déséquilibré et très profond).
D'une autre coté, une stratégie FSS va essayer de simplifier la base afin de limiter la profondeur de l'arbre de recherche (arbre équilibré), c'est-à-dire qu'elle va par exemple privilégier la falsification des littéraux dans les clauses les plus courtes afin d'obtenir rapidement l'inconsistance élémentaire (clause vide). En revanche, cette stratégie va s'avérer mauvaise si l'instance du problème est fortement consistante (admet beaucoup de solutions). Trouver une bonne stratégie pour résoudre une instance revient donc à résoudre cette instance ! C'est pourquoi une heuristique « robuste » doit habilement mêler les deux types de stratégies. Nous allons maintenant reprendre un exemple de chaque stratégie présenté par [Wang 1997] afin de mieux comprendre comment elles se caractérisent.

L'idée de la stratégie FIS est de réduire la proportion d'interprétations falsifiant la base après satisfaction d'un littéral (Si BC a 2^n interprétations et en notant IntInc(BC) le nombre d'interprétations inconsistantes avec BC, il faut que $IntInc(BC)/2^n > IntInc(BC[l])/2^{n-1}$). On a ainsi une probabilité plus grande de trouver un modèle de la base. Soit *BC* une base de clauses de FP_S. Soit une clause $C \in BC$. Soit un littéral $l \in L_S$. Soit $N=|S|$ le nombre de symboles propositionnels. Satisfaire le littéral *l* a 3 conséquences possibles :

1. Si $l \in C$ alors *C* peut être enlevée de la base de clauses et donc $2^{N-|C|}$ interprétations inconsistantes avec *BC* sont enlevées.
2. Si $\neg l \in C$ alors $\neg l$ peut être enlevé de *C* et il n'y a pas de changement au niveau des interprétations inconsistantes.
3. Sinon ($l \notin C$ et $l^{\neg} \notin C$) $2^{N-|C|-1}$ interprétations inconsistantes sont enlevées.

Exemple :
S = {a,b,c,d,e}. BC = { a b c, b d e}. BC admet 2^5=32 interprétations. 4 interprétations falsifient *a b c* *({¬a,¬b,¬c,d,e}, {¬a,¬b,¬c,¬d,e}, {¬a,¬b,¬c,d,¬e} et {¬a,¬b,¬c,¬d,¬e})* et 4 interprétations falsifient *b d e ({a,¬b,c,¬d,¬e},{¬a,¬b,c,¬d,¬e},{a,¬b,¬c,¬d,¬e}* et *{¬a,¬b,¬c,¬d,¬e})*. Il y a 7 interprétations qui falsifient *BC* (et non 4+4=8, car une interprétation falsifie les deux clauses). Si l'on satisfait la première clause en satisfaisant *a* par exemple, alors la proportion d'interprétations inconsistantes passe de 7/32 pour BC à 2/16 pour *BC[a]*. Si par contre on satisfait $\neg a$, alors le nombre d'interprétations falsifiant *b c* ne change pas (4). Par contre, le nombre d'interprétations falsifiant *b d e* passe à 2. Ce qui fait 5 interprétations inconsistantes pour 16 interprétations au total pour *BC[¬a]*. Ici, la proportion d'interprétations inconsistantes a augmenté.

En posant $Poids(l) = \sum_{t=1,2,3..} m_t 2^{-t}$ (cette heuristique est communément appelée « Jeroslow-Wang » [Jeroslow/Wang 1990]), où m_t dénote le nombre de clauses de taille *t* contenant *l*, [Wang 1997] montre qu'une estimation par excès du nombre d'interprétations inconsistantes avec *BC* enlevées par la satisfaction d'un littéral *l* est :

$$RF(l) = 2^{N-1}(Poids(BC) + Poids(l) - Poids(l^{\neg})).$$

Comme *Poids(BC)* est une fonction de *BC* indépendante de la variable assignée, il faut donc choisir le meilleur littéral l^* tel que $RF(l^*) = 2^{N-1}(Poids(BC) + \underset{l \in L_S}{Max}(Poids(l) - Poids(l^{\neg})))$.

La seconde stratégie veut au contraire arriver le plus rapidement possible à l'inconsistance élémentaire (apparition de clause nulle, de clauses unitaires complémentaires). On veut donc fixer le plus possible de valeurs de variables afin de raccourcir les clauses. Si l'on fixe la valeur de vérité du littéral *l*, alors on va aussi fixer (par propagation des clauses unitaires) la valeurs des littéraux *l'* tels que $l' \vee l^{\neg}$ est une clause binaire de la base. Si l'on note *VS(l)* le nombre de littéraux dont la satisfaction est décidée *directement*[10] par la satisfaction de *l*, alors une borne supérieure de *VS(l)* est $m_{2,\bar{l}}$, qui dénote le nombre de clauses binaires de la base qui contiennent $\bar{l}$.

[Wang 1997] montre à partir de résultats expérimentaux que la première stratégie est très efficace quand il y a beaucoup de solutions (problèmes sous-contraints) et que la seconde permet, en complément d'une autre heuristique, d'améliorer les preuves de l'inconsistance.

Il en déduit une heuristique générale de choix du littéral :

[10] en prenant par exemple *BC = { a b , b a d, a e }* on obtient pour *VS(¬e)=1* car seule une clause binaire de la base contient *e*. Pourtant, satisfaire ¬*e* va fixer les valeurs de vérité des autres variables. *VS(l)* est donc seulement une approximation de la propagation réelle des clauses unitaires résultant de la satisfaction de ¬*l*..

$$h(l^*) = \underset{l \in L_S}{Max}(Poids(l) - Poids(\bar{l}) + \alpha.m_{2,l}),$$

que l'on pourrait aussi noter $FIS(l)+\alpha FSS(l)$. Il obtient par expérimentation que $0.2 \leq \alpha \leq 0.35$. Cette heuristique n'est pourtant valable que pour les bases consistantes. En comparaison avec une heuristique additive des poids des littéraux complémentaires (« double-sided Jeroslow Wang » [Hooker/Vinay 1995], les résultats sont décevants dans le cadre des instances inconsistantes. De cette étude on peut donc déduire que (pour une heuristique générale) :

- le choix d'un littéral doit se faire de manière équilibrée, c'est à dire qu'il vaut mieux développer deux arbres de taille 2^{N-1} qu'un arbre de taille 2^N puis un arbre de taille 2^{N-2}.
- il faut tenir compte des simplifications de la base résultant de la satisfaction d'un littéral.

Nous pouvons noter que les deux simplifications le plus souvent utilisées dans Davis et Putnam sont de natures différentes en termes de stratégie de recherche : la simplification par littéraux purs est clairement une stratégie FIS car elle enlève un grand nombre d'interprétations inconsistantes. Quant à la propagation des clauses unitaires, elle est par définition de type FSS.

Nous allons maintenant lister un certain nombre d'heuristiques qui ont été développées dans le cadre de SAT. Nous verrons que l'évolution des heuristiques suit ces deux points.

Les heuristiques de choix

On peut distinguer deux types d'heuristiques de choix du prochain littéral de branchement : les plus anciennes sont les MOMS *(« Maximum Occurrences in clauses of Minimum Size »)*, les secondes sont *les* UP*(« unit propagation »)* [Li/Anbulagan 1997].

Maximum Occurrences in clauses of Minimum Size

Les heuristiques de type MOMS sont typiquement des stratégies FIS. On choisit d'éliminer en premier les clauses de petite taille afin d'enlever le maximum d'interprétations inconsistantes.
Parmi les MOMS, une première catégorie d'heuristiques consiste à choisir des littéraux très particuliers [Crawford/Auton 1993]. Les clauses de Horn sont considérées comme des contraintes et les clauses non-Horn comme des variables qui peuvent prendre autant de valeurs qu'elles contiennent de littéraux positifs. Ils en déduisent l'heuristique suivante : ils filtrent successivement les littéraux de la base à l'aide des 4 heuristiques suivantes jusqu'à obtention d'un seul littéral candidat.

1. Choix d'un littéral positif dans les clauses qui ne sont pas de Horn et qui contiennent un nombre minimal de littéraux positifs (ces clauses sont considérées comme des variables qui ne peuvent pas prendre beaucoup de valeurs. Stratégie FIS).
2. Choix de ceux apparaissant dans le maximum de clauses binaires. C'est une manière d'estimer la propagation des clauses unitaires résultant de l'assignation du littéral (stratégie FSS).
3. Choix d'un littéral dont l'un des voisins[11] est un *singleton*, c'est à dire qu'il n'apparaît qu'une fois dans la base. Si la clause le contenant est satisfaite, alors on fait apparaître un littéral pur (stratégie FIS).
4. Choix du littéral ayant le maximum d'occurrences dans la base.

D'autre part, des heuristiques plus numériques permettant de calculer le poids d'un littéral ont vu le jour. Elles peuvent toutes s'exprimer de la façon suivante : $poids(l) = \sum_{C \in BC \text{ tel que } l \in C} PoidsClause(C)$. On peut aussi ajouter des informations concernant la présence du littéral dans des clauses binaires (ajout d'une stratégie FSS).

C'est le poids accordé à chaque clause (PoidsClause(C)) qui va différer selon les heuristiques :

1. PoidsClause(C) = 1 (FIS).
   poids(l) correspond alors au nombre d'occurrences du littéral l dans la base. Cette heuristique est la plus simple à mettre en œuvre.
2. PoidsClause(C) = 1 si $|c| \leq 2$, 0 sinon (FSS).
   On privilégie l'apparition dans les clauses les plus courtes (utilisée dans [Billionnet/Sutter 1992].
3. PoidsClause(C) = 1 si il n'existe pas dans BC de clause de taille inférieure à $|C|$ (FSS).
   Cette heuristique est utilisée par A-SAT [Dubois, et al. 1996].

[11] deux littéraux sont *voisins* ssi ils apparaissent dans une même clause.

4. PoidsClause(C) = $1/2^{|C|}$ (FIS).

   On retrouve notre heuristique J. W. (donnée par [Jeroslow/Wang 1990]. C'est une heuristique plutôt spécialisée dans les instances consistantes.

5. PoidsClause(C) = $1/\gamma^{|C|}$ avec γ constante (FIS).

   C'est une généralisation des deux précédentes [Castell 1997]. On trouve dans [Freeman 1995] par exemple γ=5. Cette fonction de poids semble reprise par plusieurs prouveurs aujourd'hui.

6. PoidsClause(C) = $-\ln\left(1 - 1/(2^{|C|}-1)^2\right)$, $|C|>1$.

   Ce fut l'une des meilleures heuristiques MOMS sur les bases 3-SAT générées aléatoirement : elle est utilisée par C-SAT [Dubois, et al. 1996] pour prouver l'inconsistance d'une base de clauses (FSS).

Ces heuristiques de choix de littéral de branchement sont utilisées depuis la mise en évidence des bonnes performances de « double-sided Jeroslow » [Hooker/Vinay 1995] de manière conjointe sur un littéral et sa négation afin d'obtenir un arbre de recherche équilibré, c'est à dire qu'à chaque nœud de l'arbre, les deux sous-arbres développés sont de taille comparable. D'après [Castell 1997], une partie des heuristiques actuelles peuvent se ramener à la forme :

$$h(l) = \alpha \max(\text{poids}(l), \text{poids}(l^{\neg})) + (1-\alpha)\min(\text{poids}(l), \text{poids}(l^{\neg})) .$$

En ce qui concerne la valeur de α, la valeur de 2/7 est proposée par [Dubois, et al. 1996] : elle est fixée par expérimentation.

On notera aussi l'heuristique des algorithmes comme TABLEAU et POSIT, que l'on retrouve aussi dans SATZ :

$$h(l) = \text{poids}(l) * \text{poids}(l^{\neg}) * 1024 + \text{poids}(l) + \text{poids}(l^{\neg})$$ [12].

De telles heuristiques choisissent donc une variable de branchement, pas un littéral. On choisira alors de commencer par le littéral ayant le poids le plus élevé, ou le nombre d'occurrences le plus élevé, etc.

En plus de ces heuristiques de choix du littéral, des traitements locaux peuvent être effectués. La plupart des prouveurs actuels effectuent un peu de résolution à la racine de l'arbre afin de produire des clauses binaires et/ou ternaires [Dubois, et al. 1996]. De plus, certaines informations sont obtenues avant chaque branchement pour mieux discriminer des littéraux ayant un même poids : en fixant par exemple la valeur des (ou une partie des) variables non instanciées (aléatoirement ou systématiquement), en effectuant un peu de résolution si cela n'augmente pas la taille de la base, etc. Ce sont eux qui font la différence entre ces variantes de DP.

Il faut cependant noter que ce sont souvent les qualités des stratégies FSS qui font la différence. Et cela est encore plus vrai dans la dernière génération de prouveurs.

Unit Propagation

En ce qui concerne les heuristiques de type UP, elles ont été introduites conjointement par [Boufkhad 1996, Crawford/Auton 1993, Dubois, et al. 1996, Freeman 1995]. Il s'agit d'améliorer l'estimation de *VS(l)* dans la stratégie FSS. Le principe des heuristiques UP pour une variable $v \in V$ est simple : elle évalue les bases obtenues après propagation de la clause unitaire v d'une part, et v d'autre part. Si les deux bases sont inconsistantes, alors *BC* est inconsistante. Si une seule des deux bases est inconsistante, alors il suffit de fixer la valeur de v afin d'obtenir une base consistante. Sinon, v et $\neg v$ sont pondérés par le nombre de clauses de tailles minimales apparues dans la base réduite. La difficulté principale est de choisir l'ensemble V des variables : comme ce traitement prend du temps, il faut trouver un juste milieu entre minimiser le nombre de nœuds développés et minimiser le temps d'exécution. Le plus souvent, V contient un sous-ensemble des variables apparaissant dans les clauses binaires. Récemment, [Li/Anbulagan 1997] ont développé une étude sur différentes heuristiques de type UP qui a débouché sur l'un des meilleurs algorithmes « à la DP » : SATZ. Ils garantissent que V contiendra au moins t variables non valuées à chaque nœud de l'arbre en utilisant une politique de sélection des variables de moins en moins restrictive : variables ayant au moins une occurrence positive et négative dans une clause binaire et dont la

[12] Le facteur 1024 est une constante qui doit être largement supérieure au nombre de clauses de la formule mais assez petite pour que la valeur de l'heuristique tienne dans un mot machine [Freeman 1995]. La multiplication des poids privilégie les variables ayant un poids comparable associé à chacun de ses littéraux. La somme permet de résoudre le cas où le poids d'un des littéraux est nul.

somme des occurrences (dans les clauses binaires) est au moins égale à 4, au moins égale à 2, meilleures variables selon l'heuristique, etc.

Voici leur algorithme de calcul des poids connaissant V :

```
Pour toute variable v∈V non valuée dans l'interprétation I
        Soit BC' et BC'' deux copies de BC
        BC' ← UnitPropagation(v) ;
        BC'' ← UnitPropagation(¬v) ;
        Si    ∈ BC' et    ∈ BC'' alors retourner insatisfaisable Finsi ;
        Si    ∈ BC' alors
                BC ← BC'' ;
                I(v)=faux
        Sinon   Si    ∈ BC'' alors
                        BC ← BC' ;
                        I(v)=vrai
                Sinon
                    poids(v)=diff[13](BC',BC) ;
                    poids(¬v)=diff(BC'',BC)
                Finsi
        Finsi
Finpour


Si V ne contient que des variables valuées ou V=∅
    Pour toute variable v de BC non valuée
        poids(v) ←   Σ_{i|¬v∈Ci} 5^{-|Ci|}   poids(¬v) ←   Σ_{i|v∈Ci} 5^{-|Ci|}
```

De plus, on peut récupérer des informations sémantiques à l'aide de ces propagations de clause unitaires : si un littéral l est propagé par $BC \cup \{\ \ v\}$ et $BC \cup \{v\ \ \}$ alors $BC \models l$. On obtient donc un littéral impliqué ! C'est de cette manière que sont déterminés les littéraux impliqués dans C-SAT(voir [Boufkhad 1996] pour une étude complète sur les différentes techniques de production de littéraux impliqués). De plus, si l est propagé par $BC \cup \{\ \ v\}$ et $l^{\neg}$ l'est par $BC \cup \{v\ \ \}$ alors v et l sont logiquement équivalents pour BC. Cette information est aussi utilisée dans C-SAT mais n'est pas détectée « sémantiquement » comme ici : une reconnaissance syntaxique est utilisée sur les clauses binaires et ternaires. Attention : ce sont des méthodes incomplètes pour la recherche de littéraux impliqués et équivalents (seule une partie des littéraux impliqués et équivalents est détectée).

Il faut imaginer que les heuristiques MOMS ont une vision très imprécise des propagations des clauses unitaires lors de la satisfaction d'un littéral. Dans une heuristique de type UP, cette vision est beaucoup plus précise grâce aux « sondages » effectués. Afin d'aller encore plus loin dans la méthode, une estimation de la taille du sous arbre basée sur des sondages [Lobjois/Lemaître 1998] constituera peut être la future génération d'heuristiques. Mais il ne faut pas oublier que le prix à payer pour ce type d'heuristique est le temps. Notre implantation personnelle de l'heuristique UP, utilisant à peu près les mêmes critères de sélection des variables V que SATZ plus la production de littéraux impliqués à la C-SAT, permet une réduction du nombre de nœuds (appels à la procédure) développés : malheureusement, énormément de temps est passé à effectuer ces propagations unitaires et une version « MOMS » de Davis et Putnam est souvent plus rapide. Il vaut mieux quelquefois développer quelques milliers de nœuds de plus pour gagner du temps ! Encore une fois, sur certaines instances, la propagation unitaire permet de résoudre des instances en quelques secondes ou minutes là où une méthode purement MOMS met des heures (souvent des instances structurées). Seule une implantation très efficace de ce mécanisme permet des temps de résolutions raisonnables sur tout type d'instances. On notera par exemple l'algorithme de propagation unitaire proposé par [Zhang/Stickel 1996] permettant d'effectuer cette opération en temps linéaire amorti.

Notons enfin que le calcul de littéraux impliqués peut être effectué par un algorithme particulier en plus de l'heuristique UP. Dans l'algorithme AVAL [Audemard, et al. 1999], il permet de réduire significativement le nombre de nœuds de l'arbre de recherche développé par SATZ dans le cas d'instances aléatoires, voire de résoudre des instances que SATZ ne résout pas (classe ii32 DIMACS). Les temps de résolution sont malheureusement en moyenne plus lents.

Pour résumer, un algorithme « à la DP » actuel doit posséder un certain nombre de caractéristiques :

[13] diff(BC',BC) retourne le nombre de clauses de taille minimale qui se trouvent dans BC' et pas dans BC. Notons que selon la fonction d'évaluation utilisée (nombre de clauses minimales, nombre de propagations effectuées, nombre de clauses satisfaites) on retombe sur une stratégie FSS ou FIS.

- pré-traitement (production de résolvantes binaires par exemple),
- mécanisme de propagation des littéraux purs et clauses unitaires (voire impliqués),
- filtrage des littéraux (apparaissant dans des clauses binaires, non-Horn, etc.),
- sélection du meilleur littéral par une heuristique MOMS et/ou UP.

Nous avons implanté un algorithme de Davis et Putnam ayant ces caractéristiques (en Java) et pourtant ces temps de résolution sont médiocres par rapport à des algorithmes comme SATZ, POSIT, TABLEAU. Outre le langage utilisé (le plus souvent ils sont écrits en C, C++ pour rel-SAT) , les bonnes performances de ces prouveurs sont souvent liées aux astuces d'implantation des différents mécanismes (voir [Freeman 1995] pour quelques règles de programmation pour prouveurs SAT). Rappelons enfin qu'aucun prouveur n'est meilleur que tous les autres sur l'ensemble des instances disponibles.

Quelques remarques sur le comportement empirique (expérimental) des algorithmes « à la DP »

Une bonne partie des améliorations apportées dans le domaine de SAT provient de l'expérience des chercheurs dans la résolution de problèmes réels ou générés aléatoirement. En effet, deux écoles existent ; la première considère que « les problèmes réels sont faciles », car ils contiennent une structure propre au problème (par exemple, une description logique d'un circuit électronique utilisera vraisemblablement une décomposition en composants), donc un algorithme utilisant cette structure doit être capable de résoudre le problème : les problèmes classiques comme le zèbre, les pigeons, Ramsey … entrent dans cette catégorie. La seconde utilise des problèmes générés aléatoirement, arguant que de tels problèmes sans structure permettent d'améliorer globalement les performances des algorithmes, indépendamment des problèmes à résoudre. Nous n'entrerons pas dans ce débat. Notons cependant que la génération de problèmes aléatoires permet d'obtenir facilement des cas de tests pour les algorithmes, voire même au niveau de l'implantation de ces algorithmes d'en vérifier leur comportement. De plus, le succès des prouveurs SAT et CSP « généraux » face à des prouveurs « spécialisés » dans le cadre de la résolution de problèmes tels que la planification (SATPLAN, [Kautz/Selman 1996]) semble conforter l'étude de ces problèmes aléatoires. Néanmoins, nous verrons dans la section 2.5 page 43, que des chercheurs travaillent sur des prouveurs adaptés à la résolution d'instances réelles. Leur particularité : un fort pouvoir sémantique.

Les problèmes générés aléatoirement les plus couramment utilisés sont les problèmes 3-SAT aléatoires. Etant donné un nombre de variables *NBV* et un nombre de clauses *NBC*, on génère aléatoirement *NBC* clauses de longueur *3*, dont les variables apparaissent positivement avec la probabilité PPOS. Celle ci est généralement de 0,5. On peut néanmoins faire varier ce ratio pour obtenir des instances difficiles, voir par exemple [Castell 1997].

Dans la Figure 1, nous pouvons observer le comportement de l'algorithme de type Davis et Putnam que nous avons implanté avec l'heuristique de TABLEAU, POSIT et SATZ.
En abscisse nous retrouvons le ratio $\frac{NBC}{NBV}$.
En ordonnée, le nombre de nœuds développés par l'algorithme pour résoudre 100 bases (de 100 variables et 100*ratio clauses), c'est à dire le nombre cumulé d'appels à la procédure de Davis et Putnam.

Nous distinguons 3 cas parmi ces appels :
- les appels résultant de la simplification par un littéral pur (PURS),
- ceux provenant de la propagation des clauses unitaires (UNITAIRES),
- et enfin ceux (« normaux ») résultant d'un choix sur la valeur de vérité d'un littéral (AUTRES). Il faut noter que la plupart des résultats présentés dans le domaine ne considèrent que ces derniers comme nœuds développés.

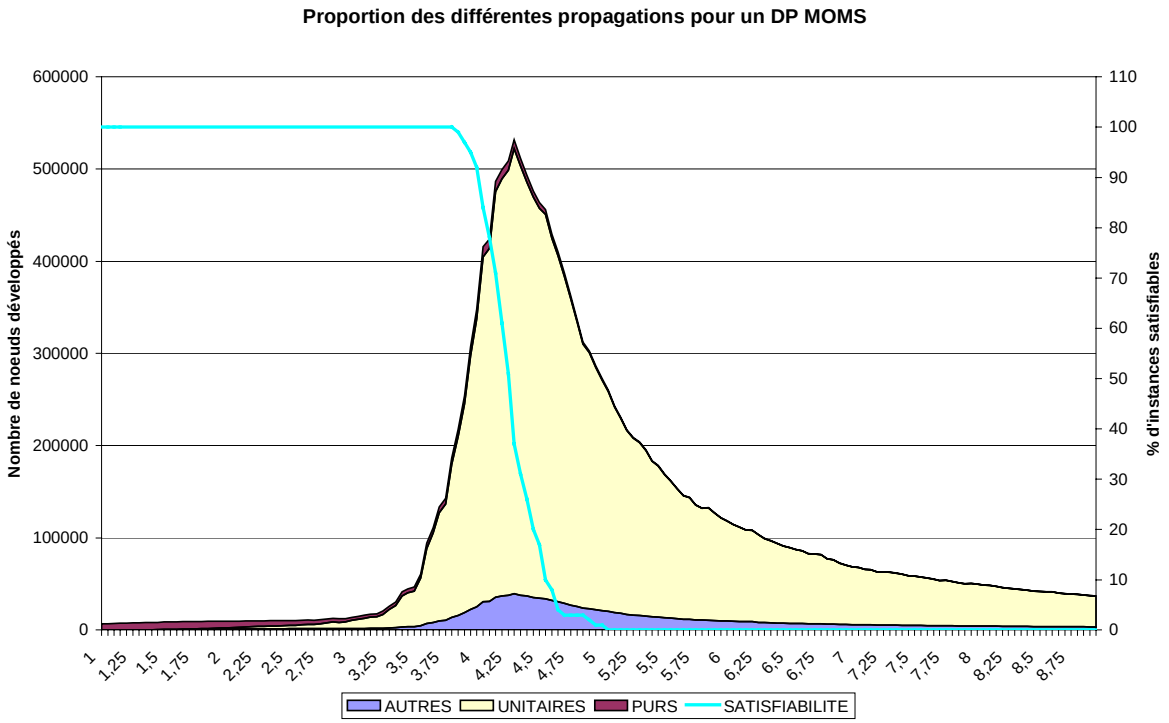


Figure 1 Satisfaction de Bases 3-SAT aléatoires - heuristique MOMS

On comprend facilement à la lecture de ce graphique que la propagation des clauses unitaires est très importante dans la procédure de Davis et Putnam.

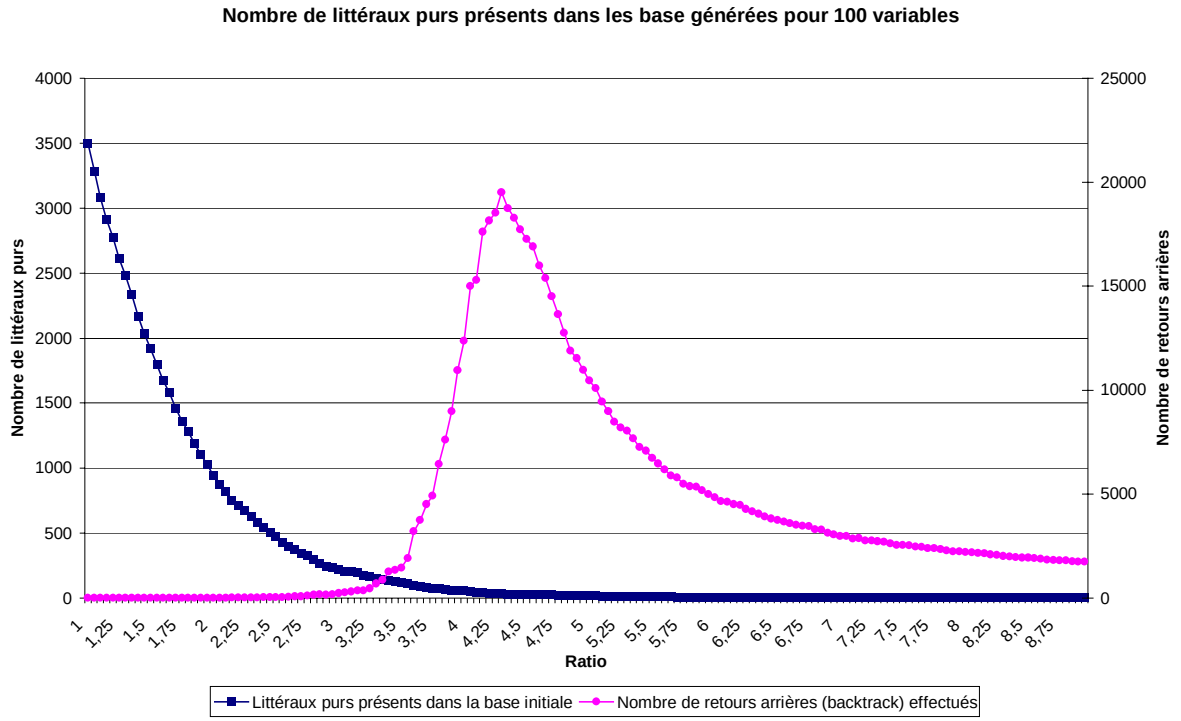
Nous pouvons observer tout d'abord que pour un ratio de 1 à 3,8 quasiment toutes les bases générées sont satisfaisables. A contrario, pour un ratio de 5,2 à 9 les bases générées sont quasiment toutes inconsistantes. Entre 3,8 et 5,2 on génère de moins en moins de bases satisfaisables. Les problèmes générés sont donc, en suivant la progression du ratio, sous-contraints, moyennement contraints et enfin sur-contraints. Ce comportement apparaît dans tous les problèmes générés aléatoirement. Par contre, les valeurs limites du ratio entre ces différentes phases dépendent du nombre de variables, de la taille des clauses, etc …

On peut maintenant remarquer que le nombre de nœuds développés par la procédure de Davis et Putnam est maximal lorsque les problèmes générés sont moyennement contraints (expérimentalement le pic de difficulté apparaît à 50% de satisfiabilité). En deçà, il y a tellement de solutions que l'algorithme en trouve une très vite. Au delà l'algorithme donne une preuve assez rapide de l'inconsistance. Ce type de courbe caractérise des problèmes FACILES-DIFFICILES-FACILES pour les prouveurs actuels. Rien ne prouve que les problèmes moyennement contraints sont intrinsèquement difficiles. Seulement, les problèmes moyennement contraints « trompent les heuristiques », ce qui les rend plus difficiles que les autres. Il faut d'ailleurs remarquer que les mêmes problèmes peuvent devenir faciles ou difficiles suivant l'heuristique et/ou la méthode utilisée pour les résoudre.

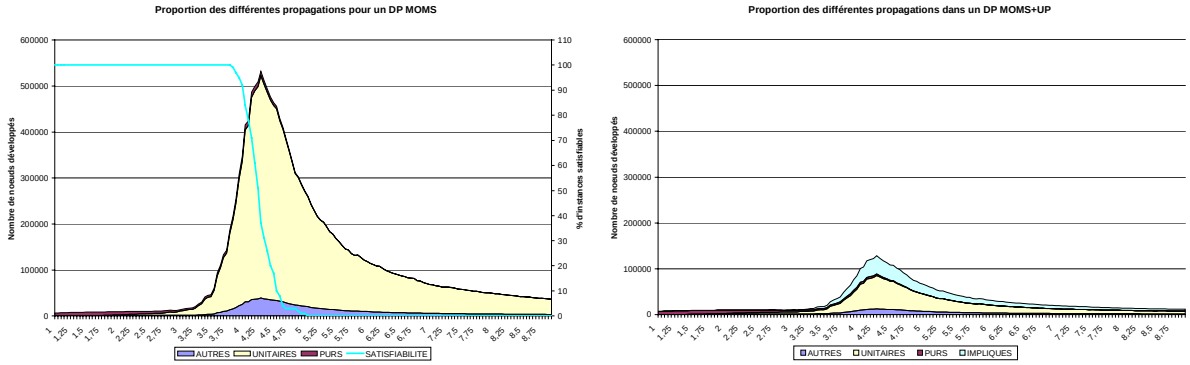
Etudions maintenant comment s'effectuent les simplifications par rapport au ratio :

- Sur les bases sous-contraintes, la simplification par littéraux purs est très active. C'est compréhensible : en générant *n* clauses de 3 littéraux en utilisant *n* variables, il y a généralement des littéraux purs qui apparaissent dans la base (une variable sur trois apparaît soit positivement soit négativement au ratio 1, cf. graphique ci-dessous). Comme la simplification par littéraux purs enlève des clauses de la base sans en raccourcir, il n'y a pas de production de clause unitaire. On peut noter que jusqu'au ratio 1,7, aucun retour arrière n'est nécessaire à la preuve de la consistance.

- Dans les autres cas, la propagation des clauses unitaires représente la majeure partie des nœuds développés. Là encore, la justification est facile : travaillant en 3-SAT, il suffit d'affecter 2 littéraux bien choisis pour faire apparaître une clause unitaire, dont l'affectation va raccourcir des clauses et faire apparaître d'autres clauses unitaires, etc.



Nous allons maintenant comparer les deux approches heuristiques MOMS et MOMS+UP pour 100 variables. 100 bases ont été générées pour chaque valeur du ratio. L'heuristique utilisée est toujours celle de TABLEAU, POSIT et SATZ (elle semble adaptée à la propagation des clauses unitaires). Les deux graphiques suivants correspondent à la Figure 1 et la Figure 2 à la même échelle.



La première chose à remarquer est que la propagation unitaire permet de développer beaucoup moins de nœuds qu'une seule heuristique MOMS (4 fois moins au seuil environ). Malheureusement, dans notre implantation actuelle, ceci ne se traduit pas en termes de temps de résolution. La propagation unitaire prend du temps, et est difficile à mesurer en termes de nœuds explorés (seul un traitement minimal est effectué). On note l'apparition de la propagation de littéraux impliqués, déterminés comme C-SAT à partir des résultats de la propagation unitaire.

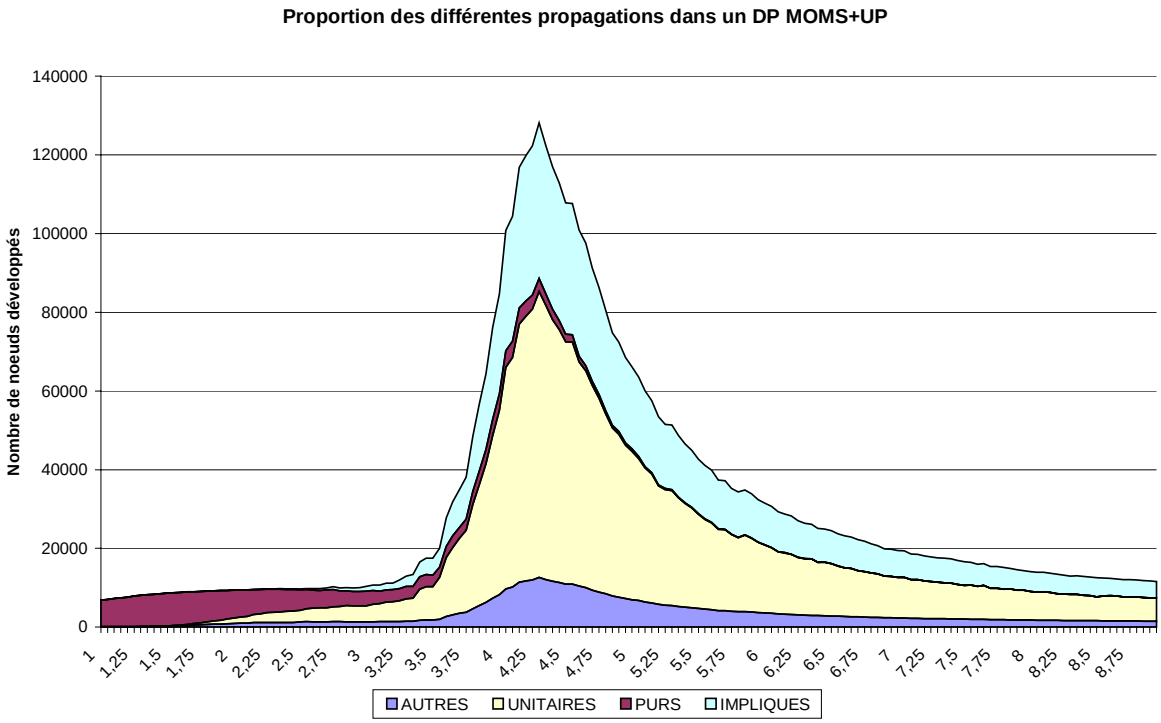
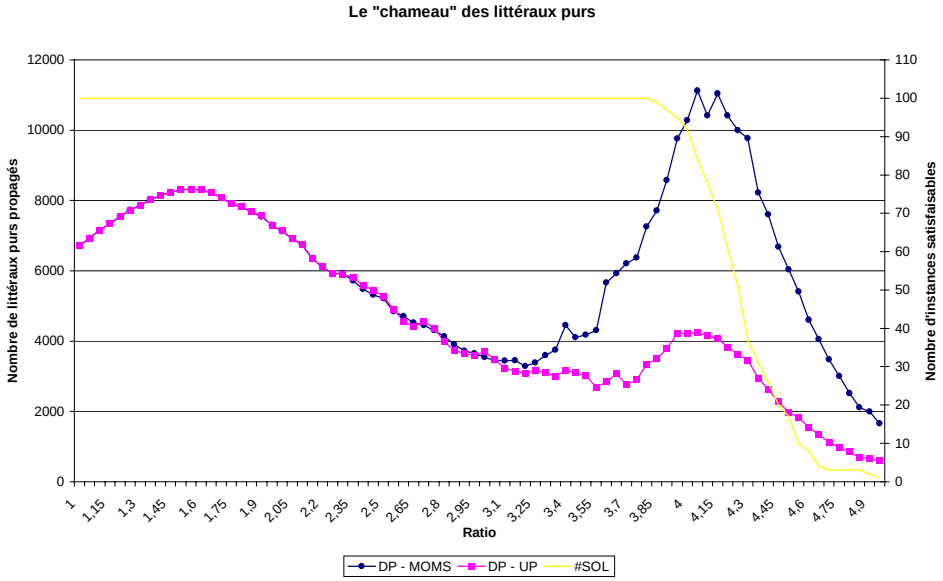


Figure 2 Satisfaction de Bases 3-SAT aléatoires - heuristique UP

Il n'y a pas plus de propagations de littéraux purs, au contraire (voir ci-dessous) : c'est le changement d'échelle qui donne cette impression. La production de littéraux impliqués apparaît vers un ratio de 3, et devient maximale au seuil. Elle est aussi utile pour prouver l'inconsistance.

Nous voulons revenir sur une remarque faite par [Mazure 1999] sur le nombre de littéraux purs propagés par un algorithme de type Davis et Putnam. Pour des instances de 3-SAT avec100 variables, on voit apparaître deux pics du nombre de littéraux purs propagés : un vers un ratio de 1,5 et un autre au seuil. Ce comportement se produit quelle que soit la proportion de littéraux positifs (instances k-SAT déséquilibrées). Il donne à ces courbes le nom de « *courbe du chameau* », à cause de ces deux pics de difficultés, mais ne s'explique pas ce comportement.



Nous nous risquons à fournir une explication, simple voire simpliste : nous avons déjà donné nos remarques sur la propagation de littéraux purs pour un ratio inférieur à 2. Le premier pic correspond sans doute à l'augmentation du nombre de clauses, donc du nombre de littéraux purs à satisfaire pour satisfaire la base.

Comme il n'y a pas de retour arrière pour ce pic, les littéraux purs comptabilisés étaient soit purs dans la base initiale, soit sont devenus purs dans une base réduite. Comme le nombre de littéraux purs initiaux baisse avec l'augmentation du ratio, il est normal que le nombre de littéraux purs diminue dans la recherche. Le second pic de propagations au seuil nous semble intuitivement provenir de l'importance de l'espace de recherche exploré au seuil. Au vu du nombre de clauses unitaires propagées on peut penser qu'il apparaît de temps en temps des littéraux purs lors de la recherche. Si l'on compare le nombre de littéraux purs propagés au nombre de clauses unitaires propagées sur les graphiques précédents, on peut noter que la propagation des littéraux purs est toujours très minoritaire. On peut noter par exemple que ce pic peut varier en fonction de l'heuristique : l'utilisation de la propagation de clauses unitaires, en permettant de diminuer globalement le nombre de nœuds explorés, fait aussi baisser ce pic de propagation. Ce qui semble conforter notre hypothèse.

Mais ce n'est qu'une hypothèse …

2.2.3 Enumération de modèles/Couverture d'impliquants

Nous avons vu que la procédure de Davis et Putnam peut être considérée comme un algorithme « d'énumération par lots », car l'arbre développé est rarement complet, ce qui implique que les interprétations rejetées le sont par lots, tout comme les modèles de la base : on trouve souvent un ensemble de solutions, un *impliquant* de la base (on rappelle qu'une méthode basée sur la résolution produit des *impliqués* de la base).
On peut facilement transformer la procédure de Davis et Putnam pour qu'elle produise une couverture d'impliquants de la base à la place de sa consistance, c'est à dire un ensemble d'impliquants tel que la disjonction de ces impliquants est logiquement équivalente à la base (cf. Définition 16, page 21). Il suffit de :

- garder en mémoire l'affectation des littéraux, et retourner l'affectation courante lorsque l'on arrive à une base trivialement consistante, ici la base vide.
- ne pas utiliser la propagation des littéraux purs car elle ne conserve pas l'équivalence logique de la base,
- ne pas s'arrêter au premier impliquant trouvé.

Cette approche est utilisée par exemple dans [Schrag 1996] et [Mazure/Marquis 1996] dans le cadre de la compilation de bases de connaissances. Nous verrons plus loin qu'on peut aussi utiliser la procédure de Davis et Putnam pour énumérer des interprétations selon certaines relations de « préférence syntaxique ».

Il faut noter cependant que les heuristiques ne sont pas forcément les mêmes, étant donné qu'il faut développer tout l'arbre de recherche.

On peut enfin noter que l'utilisation des « feuilles d'échecs », c'est à dire des feuilles étiquetées par une formule trivialement inconsistante, permet d'obtenir de manière duale une couverture d'impliquants de la négation de la base (c'est-à-dire une couverture d'impliqués de la base). Il faut dans ce cas faire attention à la propagation des clauses unitaires : une clause unitaire permet d'obtenir un impliquant de la négation de la base en falsifiant son littéral.

2.3 Approches « stochastiques »

Il faut voir la recherche d'une solution d'une manière complètement différente dans ce cas. On raisonne maintenant sur l'espace de recherche composé des 2^n interprétations de *BC*. Sachant qu'il est impossible, à partir d'une certaine taille de problème, d'effectuer une recherche exhaustive, les méthodes de recherche locale vont seulement explorer quelques régions dans cet espace. Le problème SAT est alors un peu transformé : on cherche une interprétation qui minimise le nombre de clauses falsifiées (ou maximise le nombre de clauses satisfaites : MAXSAT). Si la solution optimale satisfait toutes les clauses de la base, on a trouvé un modèle de la base, et on peut donc répondre que la base est satisfaisable, sinon on ne peut rien dire. On se trouve maintenant dans le cadre des problèmes d'*optimisation*, et non plus de *satisfaction*. Le véritable pendant de la recherche locale dans ce cas est l'approche de type « *branch and bound* » [Lawler/Wood 1966], qui correspond a une recherche en profondeur d'abord, en ayant connaissance d'une limite supérieure de la solution optimale : quand la solution courante est évaluée au delà de cette limite supérieure, l'algorithme fait un retour arrière.

Les méthodes de recherche locale sont utilisées depuis les années 70 dans le domaine de la recherche opérationnelle, pour résoudre des problèmes comme le voyageur de commerce par exemple. Elles disposent d'un ensemble R de candidats, d'une fonction d'évaluation f des éléments de R, plus une relation de voisinage V entre les éléments de R. Une méthode de recherche locale prend au hasard $r \in R$, puis va chercher dans le voisinage de r s'il existe un autre candidat r' *meilleur* que la solution courante r. Si r' existe, alors ce candidat devient la solution courante, sinon, c'est que l'on se trouve dans un minimum local. Il existe alors plusieurs manières de sortir de ce minimum local : on peut essayer de choisir soit un autre candidat aussi bon que la solution courante parmi

le voisinage (on explore alors un plateau) ou en dehors du voisinage (on creuse alors un tunnel) soit aléatoirement un autre élément du voisinage, même si ce candidat est moins bon que la solution courante. Certains algorithmes réitèrent ce processus à partir de différents $r \in R$.

Dans le cadre de SAT, $R = I_S$ l'ensemble des interprétations constructibles à partir de S. f retourne par exemple le nombre de clauses falsifiées par l'interprétation. Le voisinage d'une interprétation I est le plus souvent défini comme l'ensemble des $|S|$ interprétations obtenues par changement de la valeur de vérité d'une variable de I[14].
Il existe alors différentes heuristiques pour choisir la ou les interprétations suivantes parmi le voisinage :

- Choix du meilleur voisin : dans le cadre général, le choix des interprétations est restreint à l'ensemble des interprétations du voisinage dont l'évaluation est meilleure que l'évaluation de l'interprétation courante, donc dont le nombre de clauses falsifiées est inférieur au nombre de clauses falsifiées actuel. Dans le cadre d'un algorithme glouton (« *greedy local search* »), seul le meilleur de ces voisins est sélectionné (il n'y a pas de retour arrière).
- Choix aléatoire : il suffit de changer la valeur de vérité d'une variable au hasard (cela revient à choisir une interprétation du voisinage au hasard). Une amélioration consiste à forcer le changement de la valeur de vérité d'une variable d'une clause falsifiée.
- Choix d'échappement :
  - Changer la valeur de vérité d'*une* (plateau) ou de *toutes* (tunnel) les variables qui ne changent pas la valeur de la fonction d'évaluation.
  - Eviter de toujours changer la valeur de vérité des mêmes variables (boucle) : une priorité peut être donnée aux différentes variables (très faible si on vient de changer sa valeur de vérité, plus forte sinon), ou alors on peut interdire de changer la valeur de vérité de certaines variables.

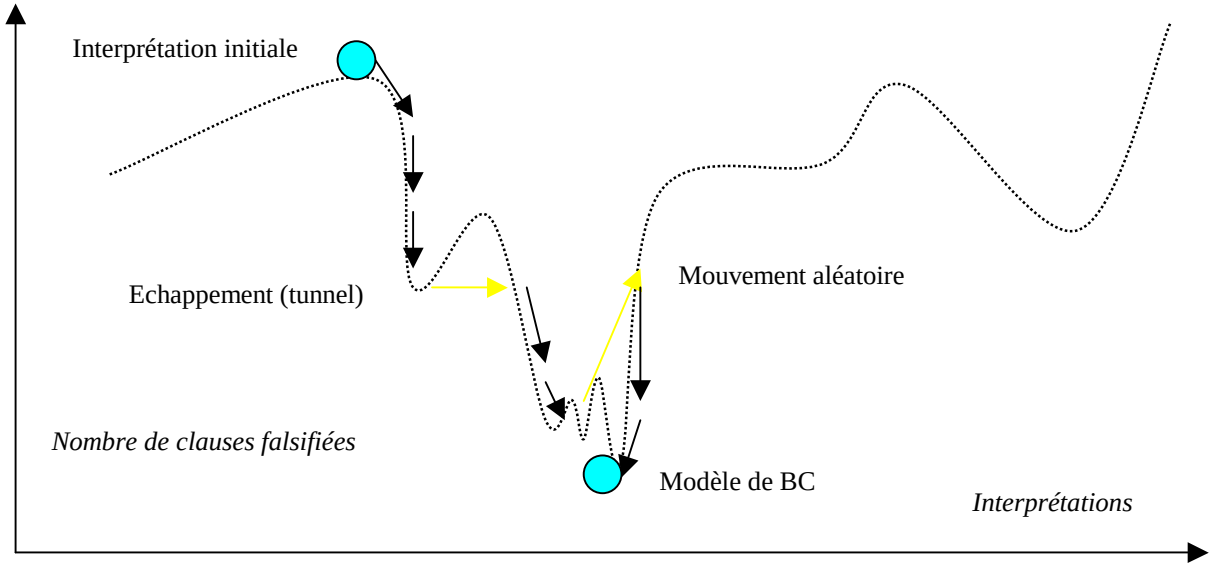


Figure 3 Exemple de parcours d'un algorithme de recherche locale

La Figure 3 nous montre le comportement possible d'un algorithme de type recherche locale. La courbe représente l'évaluation de chaque interprétation par la fonction f (le nombre de clauses falsifiées par l'interprétation). On peut distinguer les différentes phases de la recherche locale dans ce schéma :

- La descente : on choisit le meilleur voisin.
- L'échappement de type tunnel : quand on se trouve dans un minimum local, on « saute » sur une interprétation qui n'est pas dans le voisinage et qui ne change pas la valeur de la fonction d'évaluation.
- Les sauts aléatoires : on effectue un choix aléatoire. Cela permet d'échapper à certains minima locaux.

Voici un algorithme qui met en évidence ces différents comportements :

[14] On appelle *distance de Hamming* entre deux interprétations I et I', notée *Hamming(I,I')*, le nombre de variables pour lesquelles elles diffèrent. Le voisinage d'une interprétation I peut alors être défini comme l'ensemble des interprétations situées à une distance de Hamming de 1 ($\{I' \mid Hamming(I,I')=1\}$).

```
RL(BC)
// algorithme de recherche locale
        I ← une interprétation de BC ;
        Répéter
          Si f(I) = 0 alors retourner I  Finsi ;
          (
                Si ∃I'∈V(I) tq f(I')<f(I) alors I←I' // descente
                Sinon
                        Si ∃I'∈V(I) tq f(I')=f(I)
                                alors I←I' ; // échappement ;
                                sinon I ← I'∈ V(I) tq f(I')=min{f(J)|J∈V(I)}
                        Finsi
                Finsi
           | // ou aléatoire (saut aléatoire)
            I ← Choix_Aléatoire(I)
           )
        Jusqu'à (nombre d'itération | temps écoulé | …)
        Retourner « Je ne sais pas »
```

Algorithme 2 Recherche Locale

Les premiers algorithmes de type recherche locale pour SAT ont été développés vers la fin des années 80 pour la conception et le test des circuits VLSI (« *Very Large Scale Integration* ») [Gu 1988]. L'algorithme de recherche locale qui a bouleversé la communauté SAT est un algorithme de recherche locale glouton, GSAT *(« Greedy SAT »*) [Selman, et al. 1992].

Son principe de fonctionnement est le suivant :

```
GSAT (BC)
        Pour i ← 1 à MAX-TRIES faire
                I ← une interprétation générée aléatoirement ;
                Pour j ← 1 à MAX-FLIPS faire
                        Si I satisfait BC alors retourner I Finsi ;
                        Soit I'∈ V(I) tq g(I')=max{g(J)|J∈V(I)}
                        I ← I'
                Finpour
        Finpour
        Retourner « pas de solution trouvée »
```

Algorithme 3 GSAT

Les paramètres de cette procédure sont MAX-TRIES et MAX-FLIPS : le premier correspond au nombre de régions que l'on veut explorer dans l'espace de recherche, le second au nombre d'itérations (au temps) à consacrer à l'exploration de chaque région. g(J) est l'heuristique *« min-conflicts »* : nombre de clauses falsifiées qui deviendraient satisfaites moins le nombre de clauses satisfaites qui deviendraient falsifiées. Une interprétation parmi les voisins de I maximisant cette fonction est retournée aléatoirement. Comme cette fonction peut retourner un résultat négatif, les notions de remontée et d'échappement sont équivalentes à une valeur maximale de la fonction respectivement négative ou égale à zéro.

Son principe étant des plus simples, ce type d'algorithme teste un nombre très élevé d'interprétations à la seconde (plusieurs dizaines de milliers, selon la machine utilisée et la taille du problème). On ne peut pas espérer explorer entièrement l'espace de recherche car aucun mécanisme n'empêche de retomber sur une interprétation déjà testée. Son principal inconvénient est d'ailleurs qu'elle peut rester coincée dans des minima locaux.

C'est pourquoi elle a été améliorée [Selman, et al. 1994] tout d'abord en introduisant, avec une probabilité *p*, des sauts aléatoires (« *random walk* ») lors de la recherche , puis en choisissant les variables à flipper parmi celles d'une clause falsifiée (*WSAT, walksat*). Le voisinage est un peu plus difficile à définir car il dépend d'une clause falsifiée, c : V(I,c) = { J | Hamming(I,J)=1 et J satisfait c}. Afin d'échapper à certains minima locaux, on trouve une variante de WSAT utilisant en plus de l'heuristique *min-conflicts* un critère d'ancienneté sur les variables (l'itération de son dernier flip) appelée Novelty [McAllester, et al. 1997].

[Mazure, et al. 1997] proposent une version modifiée de WSAT, TSAT, qui maintient lors de la recherche une liste de *k* variables que l'on ne peut plus flipper (liste *tabou* [Glover/Laguna 1993]). Cette liste permet de sortir d'un plateau ou d'un minimum local.

Les méthodes de recherche locale permettent de résoudre de très gros problèmes (jusqu'à 2000 variables au seuil contre 500 variables pour les algorithmes complets). Soulignons que cela est vrai uniquement pour les instances

satisfaisables. Mais leurs performances dépendent non seulement des heuristiques de choix de l'interprétation suivante, mais aussi et surtout des paramètres des différentes procédures (MAX-TRIES, MAX-FLIPS, p, k) pour TSAT par exemple. L'une des difficultés dans l'utilisation de ces méthodes est de savoir comment les paramétrer. On retrouve dans la littérature des valeurs indicatives de ces paramètres pour résoudre des instances aléatoires : *MAXTRIES=5*, *MAXFLIPS=*n^2 [Parkes/Walser 1996], *p=0,5* [Selman, et al. 1994] et *k=0,01875*n+2,8125* [Mazure, et al. 1997] où *n* est le nombre de variables de l'instance du problème à traiter.

Notons que l'implantation de GSAT peut être uniquement basée sur des compteurs et un générateur de nombre aléatoire. C'est pourquoi [Hamadi/Merceron 1997] proposent une version matérielle de cette méthode sur une architecture reconfigurable de type *Field Programmable Gate Arrays* (FPGA)[15]. L'intérêt de cette implantation matérielle est d'effectuer le test de satisfaction des clauses en parallèle : le temps de résolution d'une instance est donc indépendant du nombre de clauses de l'instance ! Par contre, la taille du problème encodé est limité par le nombre de composants disponibles sur le circuit. Malheureusement, seuls des résultats issus de simulations sont disponibles. Ils sont néanmoins encourageants : sur les instances présentées, les accélérations varient de x70 à x300 par rapport aux résultats présentés par [Selman/Kautz 1993].

2.4 Coopération méthodes systématiques/stochastiques

Il existe des cas où la coopération de deux méthodes permet de rendre plus robustes des algorithmes face aux différentes instances à résoudre : on notera tout d'abord la coopération Davis et Putnam/Résolution que l'on trouve dans certains traitements locaux qui effectuent des résolutions si celles-ci n'augmentent pas la taille de la base [Franco 1991].[Castell 1997] propose avec DP_RC une coopération DP/Résolution dédiée à la production de preuves par résolution minimales.

[Mazure, et al. 1996] présentent une méthode intéressante pour prouver l'inconsistance d'une base de clauses à partir d'une méthode de recherche locale. Lorsque la recherche locale ne réussit pas à trouver un modèle de la base, les auteurs utilisent ce qu'ils appellent la *trace* de l'algorithme : à chaque clause est associé le nombre d'interprétations testées qui l'ont falsifiée. De même pour les littéraux. Ainsi, si la base est localement inconsistante[16], les clauses contenues dans la (ou les) sous bases inconsistantes correspondent aux clauses ayant le score le plus élevé. Pour prouver l'inconsistance de la base, il suffit de lancer un algorithme complet sur la base formée de ces clauses. D'après [Mazure, et al. 1996], cette méthode donne de bons résultats lorsque la base est localement inconsistante. Dans le cas contraire, comme toutes les clauses sont impliquées dans l'inconsistance, il faut lancer une méthode complète sur la base entière.

Certains utilisent aussi un prétraitement en recherche locale afin de pondérer les clauses par rapport à l'heuristique globale. On préférera satisfaire d'abord les clauses souvent falsifiées. On trouve ce genre de prétraitement dans C-SAT par exemple.

Une autre coopération semble s'installer entre les deux approches. Nous avons vu que des traitements locaux sont effectués à chaque nœud de l'arbre de recherche. Ces traitements peuvent être considérés comme des recherches locales un peu simplifiées (comme l'heuristique UP par exemple). Mais ils deviennent de plus en plus complexes. On se rapproche donc d'une méthode hybride qui parcourt exhaustivement des régions traitées par recherche locale. [Mazure, et al. 1998] proposent par exemple d'utiliser TSAT à la place de la propagation unitaire afin d'effectuer le branchement sur le littéral ayant le plus fort score (cf. paragraphe précédent). On trouve dans [Castell 1997] et [Mazure 1999] d'autres exemples de coopération.

Notons enfin une approche stochastique des algorithmes systématiques qui semble donner de bons résultats : l'introduction du choix aléatoire de la variable de branchement associée à un redémarrage de l'algorithme systématique [Gomes, et al. 1998]. L'idée est que l'instance n'est pas intrinsèquement difficile, mais que l'heuristique particulière d'un algorithme systématique peut développer un arbre de taille exponentielle par rapport à la taille moyenne des arbres développés par d'autres heuristiques pour cette instance. Il faut alors voir une heuristique comme un seul essai d'un algorithme stochastique : les performances contrastées d'une heuristique sur différents problèmes s'expliquent alors par la variance d'une méthode stochastique lorsqu'on effectue un seul essai. Pour obtenir de meilleurs résultats (en moyenne), il faut donc multiplier ces essais. En pratique, cela revient à ajouter une limite de retours arrière (équivalente au MAX-FLIPS de GSAT), après laquelle la recherche recommence à la racine, plus un choix aléatoire de la variable de branchement parmi les meilleures proposées par l'heuristique. L'algorithme obtenu est toujours complet (seul l'ordonnancement de l'affectation des variables change pour

[15] Une version FPGA de la procédure de Davis et Putnam est décrite dans [Yokoo, et al. 1996]

[16] On dira qu'une base BC est globalement consistante ssi $BC \models \perp$ et $\forall BC' \subset BC, BC' \not\models$. Une base de clause est localement inconsistante ssi $BC \models \perp$ et $\exists BC' \subset BC$ telle que $BC' \models \perp$. La notion de « localement inconsistante » peut être affinée par la taille de BC' par exemple. Dans la pratique, on parle de base localement inconsistante quand BC' est beaucoup plus petite que BC.

chaque essai). Les résultats sont surprenants : en modifiant SATZ de la sorte, un problème (logistics.d) résolu (par SATZ) de manière déterministe en 108 mn est résolu (en moyenne) en 95 s ! De même, des problèmes non résolus de manière déterministe en 24 heures sont résolus (en moyenne) en 165 s et 17 mn.

Nous allons maintenant présenter une méthode de résolution de SAT peu décrite dans la communauté « Intelligence Artificielle » mais très utilisée dans la communauté « méthodes formelles » pour la vérification et la spécification de systèmes : l'algorithme de Stålmark [Stålmarck 1989, Sheeran/Stålmark 1998].

2.5 Algorithme de Stålmark

Cette méthode a été développée dans le cadre de la validation de spécifications, quand celles-ci sont exprimées (ou transformées) en calcul propositionnel. Etant donné un système décrit par la formule α et une propriété exprimée par la formule β, on cherche à savoir si la propriété est satisfaite dans le système : on cherche à prouver la validité de la formule $\alpha \quad \beta$. On se trouve non plus devant un problème de NP (comme SAT), mais de co-NP. On retrouve aussi ce type de problèmes dans l'aide à la conception de circuits. Historiquement, ce problème est traité dans la communauté « vérification matérielle » par l'utilisation de Diagrammes de Décision Binaires[17] (BDDs [Bryant 1986]). L'intérêt de cette méthode est de s'appliquer à une formule quelconque et d'être incrémentale. Néanmoins, elle est très sensible à l'augmentation du nombre de connecteurs et de variables du problème, rendant son utilisation impossible dans le cadre de la résolution de certains problèmes réels [Groote/Warners 1999]. La méthode de Stålmarck est une méthode brevetée [Stålmarck 1989] utilisée dans un produit commercial capable de résoudre de grandes instances de problèmes issus d'applications industrielles (dans cette communauté, la taille des problèmes est leur taille en Ko !). Ceux ci sont intraitables à l'aide de BDDs, ce qui explique l'engouement de la communauté « méthodes formelles » pour cette méthode. Stålmarck explique les bons résultats de sa méthode sur ces problèmes en définissant une nouvelle classification des problèmes dans laquelle bon nombre de problèmes réels sont des problèmes « faciles ».
Divers travaux reprenant certains principes de cette méthode existent dans le cadre de la satisfaction [Groote/Warners 1999] ou de la validation de formules booléennes [Harrison 1996]. Ils constituent un premier lien entre cette méthode et les méthodes développées par la communauté IA (Davis et Putnam, recherche locale).

Nous allons d'abord présenter la méthode de Stålmarck telle qu'elle est définie dans [Sheeran/Stålmark 1998], puis nous nous intéresserons à deux variantes de cette méthode.

2.5.1 La méthode de Stålmarck vue par … Stålmarck

Nous décrivons ici la version « simplifiée » de la méthode Stålmarck présentée dans [Sheeran/Stålmark 1998]. Nous renvoyons le lecteur à cette même référence pour une explication plus complète et plus rigoureuse.

Contrairement aux approches habituelles, cet algorithme travaille sur des formules quelconques, exprimées à l'aide des cinq connecteurs $\wedge,\vee,\neg,$ et $\Leftrightarrow$, et pas seulement des CNF. Cette formule est tout d'abord transformée de manière purement syntaxique en une formule α *ne contenant que le connecteur* . L'idée est ensuite de remplacer chaque sous-formule $P \quad Q$ de α, par une nouvelle variable propositionnelle v. La formule α est donc transformée *en temps linéaire* en un ensemble de triplets (v,p,q) de la forme $v \Leftrightarrow (p \quad q)$, avec v nouvelle variable propositionnelle et p et q deux variables (nouvelles ou non), grâce à l'ajout de nouvelles variables. $\top$ et $\bot$ sont remplacés respectivement par 1 et 0 dans les triplets.

Exemple :
La formule $\alpha \equiv p \quad (q \quad p)$ peut s'écrire sous forme de triplets $\{(b1,q,p),(b2,p,b1)\}$ où $b2$ représente la formule α.

Stålmarck définit alors des règles de déduction sur ces triplets. Par exemple, $p \quad q$ est fausse ssi p est vraie et q est fausse donne la règle $(R1)\dfrac{(0,p,q)}{p/1 \quad q/0}$: si l'on a le triplet $(0,p,q)$ alors on peut remplacer p par 1 et q par 0 dans les autres triplets. On peut rapprocher ces règles à celles de la méthode des tableaux analytiques [Smullyan 1968].

Le but de ces règles est d'obtenir un triplet terminal (contradictoire) : $(1,1,0)$ est un triplet terminal par exemple (les autres triplets terminaux sont $(0,x,1)$ et $(0,0,x)$).

[17] Une formule est valide ssi son BDD se réduit à un seul nœud terminal 1.

Dans notre exemple, la seule règle R1 suffit à prouver que α est valide (c'est à dire $\neg\alpha$ inconsistante) :
Posons b2 est fausse (ce qui correspond à $\neg\alpha$). Cela se traduit par le remplacement de b2 par 0 dans les triplets. On peut donc appliquer la règle R1 sur le triplet *(0,p,b1)*, ce qui donne :

$$\{(b1,q,p)\}[p/1,b1/0] = \{(0,q,1)\}\ [p/1,b1/0].$$

Or *(0,q,1)* est terminal. La formule $\neg\alpha$ est donc inconsistante. α est donc valide.

D'autres règles de déduction existent :

$$(R2)\frac{(v,p,1)}{v/1} \qquad (R4)\frac{(v,1,q)}{v/q} \qquad (R6)\frac{(v,v,q)}{x/1 \quad q/1}$$

$$(R3)\frac{(v,0,q)}{v/1} \qquad (R5)\frac{(v,p,0)}{v/\neg p} \qquad (R7)\frac{(v,p,p)}{v/1}$$

Certaines de ces règles n'assignent pas de variables, mais expriment l'équivalence de deux littéraux (règles R4 et R5).

Soit E un ensemble de triplets. Soit R un ensemble de règles applicables sur E. On notera E R E'[S] l'application de règles R sur E avec E' ensemble de triplets résultant de l'applications de ces règles et S l'ensemble des substitutions produites.

L'application des seules règles R1 à R7 ne garantit pas de trouver un triplet terminal. C'est pourquoi une règle de branchement est introduite : Stålmarck l'appelle *dilemma rule*. Il s'agit de l'expansion de Shannon appliquée à la déduction.

$$\frac{\begin{array}{c}E\\ \hline E[v/1] \quad E[v/0]\\ R \qquad R'\\ E1[S1] \quad E2[S2]\end{array}}{E[S]}$$

Etant donné un ensemble de triplets E, pour une variable propositionnelle v, on applique les règles R1 à R7 d'une part sur E[v/1], d'autre part sur E[v/0], ce qui donne respectivement les ensembles de triplets E1[S1] et E2[S2]. Si l'un des ensembles de triplets contient un triplet terminal, alors S est l'ensemble des substitutions de l'autre. Si les deux ensembles de triplets contiennent un triplet terminal, alors E représente une formule inconsistante (et le problème est donc résolu). Sinon, S = S1 $\cap$ S2 : toute nouvelle information qui se déduit pour v vraie et v fausse est indépendante de v. Nous allons donc obtenir 2 types d'information : la substitution d'une variable par une valeur (0 ou 1) et la substitution d'une variable par un littéral. La première information est exactement la notion de littéral impliqué (cf. page 28) utilisée dans C-SAT. A notre connaissance, la seconde n'a pas d'équivalent dans les prouveurs *classiques*.

La *dilemma rule* est appliquée séquentiellement sur l'ensemble des variables non instanciées tant que des nouvelles informations sont produites. Notons à présent que la *dilemma rule* se définit indépendamment des règles utilisées pour obtenir de nouvelles substitutions.

Stålmarck, définit donc une hiérarchie de méthodes de preuve M_i basée sur l'utilisation de la *dilemma rule* sur des tuples de variables : M_0 se définie comme l'utilisation des seules règles R1 à R7, M_{i+1} permet d'utiliser la *dilemma rule* avec une méthode de preuve M_i affectant un i-uplet de variables pour trouver de nouvelles substitutions sur chaque branche.

Il définit de plus une nouvelle classification des problèmes basée sur cette hiérarchie de méthodes de preuves. Une formule est dite i-facile ssi elle est prouvable dans M_i. Une formule valide est dite i-difficile si elle n'est pas prouvable dans M_j, pour tout $j<i$. Une formule est dite de degré de difficulté *i* ssi elle est i-facile et i-difficile. Il s'avère qu'empiriquement, beaucoup de problèmes industriels ont des degrés de difficulté 0 ou 1. Stålmarck montre que son algorithme est applicable dans ce cadre. Comme cette notion de difficulté ne prend pas en compte le nombre de variables, on peut comprendre pourquoi de grands problèmes industriels peuvent être considérés comme « faciles » pour la méthode de Stålmarck.

La version implantée dans l'outil commercialisé est plus puissant car il permet de travailler avec des règles sur des formules avec les connecteurs $\wedge$ et $\vee$. De plus, il propage la valeur de sous formules et pas seulement de littéraux. Il permet aussi de travailler sur des types énumérés et pas seulement des variables booléennes. Cela est important pour le public visé (validation de spécifications).

2.5.2 La méthode de Stålmarck vue par … Harrison

La présentation de la méthode de Stålmarck par Harrison est citée en référence par Stålmarck lui-même. Elle présente de façon plus intuitive (plus logique) les concepts de la méthode.

Cette formule est tout d'abord transformée de manière purement syntaxique et *en temps linéaire* en une formule *ne contenant que les connecteurs* $\neg$, $\wedge$ *et* $\Leftrightarrow$. La formule obtenue est ensuite transformée en triplets de la forme $v \Leftrightarrow (p \wedge q)$ ou $v \Leftrightarrow (p \Leftrightarrow q)$, avec *v* variable propositionnelle et *p* et *q* littéraux, grâce à l'ajout de nouvelles variables.

Montrons cela sur l'exemple de [Harrison 1996] :
$\alpha \equiv \neg((a \Leftrightarrow (b \wedge c)) \wedge ((b \Leftrightarrow \neg c) \wedge a))$
On construit les 5 triplets :
$v1 \Leftrightarrow (b \wedge c)$, $v2 \Leftrightarrow (a \Leftrightarrow v1)$, $v3 \Leftrightarrow (b \Leftrightarrow \neg c)$, $v4 \Leftrightarrow (v3 \wedge a)$, $v5 \Leftrightarrow (v2 \wedge v4)$.
On peut noter que α et $\neg v5$ sont logiquement équivalentes sur le vocabulaire de α.

Le but est d'obtenir à partir de ces « équations » une contradiction de la forme $v \Leftrightarrow \neg v$ (l'équivalent de la clause nulle pour la résolution). Pour cela, l'algorithme va appliquer des règles de déduction sur ces triplets afin de déterminer la valeur de vérité des différentes variables (l'auteur parle de déduction naturelle [Prawitz 1965]).

$v \Leftrightarrow (p \wedge q)$		$v \Leftrightarrow (p \Leftrightarrow r)$	
Si	Alors	Si	Alors
$v \Leftrightarrow \neg p$	$p \Leftrightarrow \top$ et $q \Leftrightarrow \bot$	$v \Leftrightarrow p$	$q \Leftrightarrow \top$
$v \Leftrightarrow \neg q$	$p \Leftrightarrow \bot$ et $q \Leftrightarrow \top$	$v \Leftrightarrow \neg p$	$q \Leftrightarrow \bot$
$p \Leftrightarrow q$	$v \Leftrightarrow q$	$v \Leftrightarrow q$	$p \Leftrightarrow \top$
$p \Leftrightarrow \neg q$	$v \Leftrightarrow \bot$	$v \Leftrightarrow \neg q$	$p \Leftrightarrow \bot$
$v \Leftrightarrow \top$	$p \Leftrightarrow \top$ et $q \Leftrightarrow \top$	$p \Leftrightarrow q$	$v \Leftrightarrow \top$
$p \Leftrightarrow \top$	$v \Leftrightarrow q$	$p \Leftrightarrow \neg q$	$v \Leftrightarrow \bot$
$p \Leftrightarrow \bot$	$v \Leftrightarrow \bot$	$v \Leftrightarrow \top$	$p \Leftrightarrow q$
$q \Leftrightarrow \top$	$v \Leftrightarrow p$	$v \Leftrightarrow \bot$	$p \Leftrightarrow \neg q$
$q \Leftrightarrow \bot$	$v \Leftrightarrow \bot$	$p \Leftrightarrow \top$	$v \Leftrightarrow q$
		$p \Leftrightarrow \bot$	$v \Leftrightarrow \neg q$
		$q \Leftrightarrow \top$	$v \Leftrightarrow p$
		$q \Leftrightarrow \bot$	$v \Leftrightarrow \neg p$

On peut par exemple chercher à prouver l'inconsistance de la formule α en posant $\alpha \Leftrightarrow \bot$, ou sa validité en posant $\alpha \Leftrightarrow \top$.

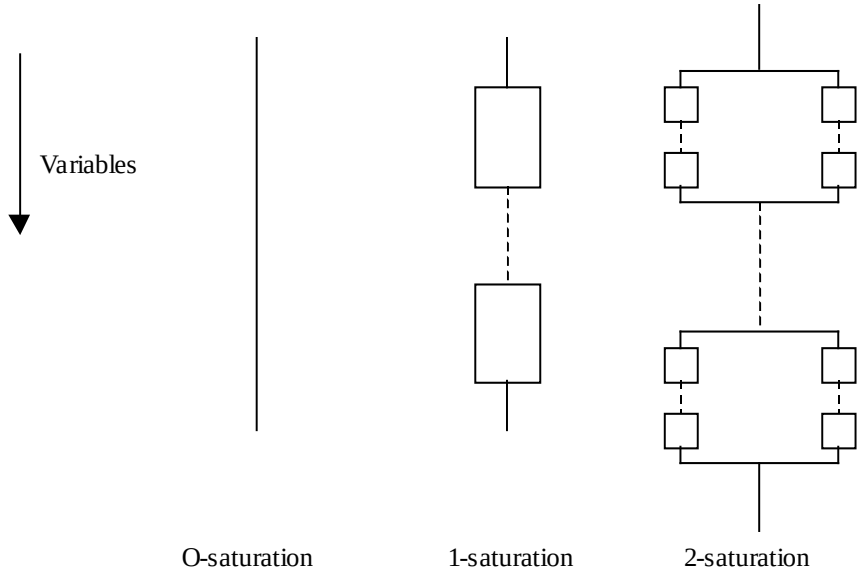
Dans l'exemple précédent, on trouve
$\neg v5 \Leftrightarrow \bot$ ($v5 \Leftrightarrow \top$). A l'aide du 5ème triplet, on trouve $v2 \Leftrightarrow \top$ et $v4 \Leftrightarrow \top$.
De $v4 \Leftrightarrow \top$ et du 4ème triplet, on trouve $v3 \Leftrightarrow \top$ et $a \Leftrightarrow \top$.
De $v3 \Leftrightarrow \top$ et du 3ème triplet on trouve $b \Leftrightarrow \neg c$.
De $b \Leftrightarrow \neg c$ et du 1er triplet, on obtient $v1 \Leftrightarrow \bot$.
De $v2 \Leftrightarrow \top$ et du 2ème triplet, on obtient $a \Leftrightarrow v1$.
De $a \Leftrightarrow \top$ et $a \Leftrightarrow v1$ on obtient $v1 \Leftrightarrow \top$.
De $v1 \Leftrightarrow \top$ et $v1 \Leftrightarrow \bot$ on obtient $v1 \Leftrightarrow \neg v1$. contradiction. α n'est pas inconsistante.

L'équivalence entre deux littéraux permet d'éliminer une variable: si $p \Leftrightarrow q$ alors q est remplacé par p dans toute la formule. De plus, la transitivité et la symétrie de l'équivalence est aussi appliquée : $p \Leftrightarrow q$ et $\neg p \Leftrightarrow r$ donne $q \Leftrightarrow \neg r$. L'application des règles de déduction plus la transitivité et la symétrie de l'équivalence entre littéraux tant que de nouvelles équations sont produites est appelée 0-saturation.

Mais la 0-saturation ne suffit pas la plupart du temps à trouver une contradiction ou une assignation consistante des variables de la formule. Il faut alors utiliser la *dilemma rule*. Cette règle ressemble beaucoup à la notion de

propagation unitaire et à la notion de littéraux impliqués. Il s'agit, pour un ensemble d'équations E et une variable donnée v, d'appliquer une 0-saturation sur $E \cup \{v \Leftrightarrow \top\}$ puis sur $E \cup \{v \Leftrightarrow \bot\}$. Appelons E1 le résultat de la première 0-saturation et E2 le résultat de la seconde. On a $E \subseteq E1$ et $E \subseteq E2$. Si ces inclusions sont strictes, alors de nouvelles informations sont apparues. On s'intéresse aux informations communes aux deux ensembles produits ($E1 \cap E2$). Si l'une des 0-saturations produit une contradiction, alors on garde les informations produites par l'autre 0-saturation. Si les deux 0-saturations produisent une contradiction, alors le problème est inconsistant. On appelle 1-saturation l'application de la *dilemma rule* à toutes les variables dont la valeur de vérité n'est pas fixée tant que cela produit de nouvelles informations.

Si le problème n'est toujours pas résolu après application de la 1-saturation, on applique alors la 2-saturation. On applique non plus la *dilemma rule* sur une variable v mais sur une paire de variables (v,w). On va donc s'interesser à l'intersection des 4 0-saturations E1 ($v \Leftrightarrow \top, w \Leftrightarrow \top$), E2 ($v \Leftrightarrow \top, w \Leftrightarrow \bot$),E3 ($v \Leftrightarrow \bot, w \Leftrightarrow \top$) et E4($v \Leftrightarrow \bot, w \Leftrightarrow \bot$). Plus généralement, on appellera n-saturation l'application de la 0-saturation à un n-uplet de variables.



A partir de la n-saturation, on retrouve la notion de difficulté définie par Stålmarck : une formule est dite n-facile ssi elle peut être prouvée par n-saturation et n-difficile si elle ne peut pas être prouvée par (n-1)-saturation.

Les résultats présentés dans [Harrison 1996] concernant son implantation de l'algorithme sur des benchs DIMACS sont difficilement comparables aux résultats obtenus dans le cadre de la satisfaction :

- D'une part les instances inconsistantes sont niées pour obtenir des instances valides : les problèmes sont donc différents.
- L'implantation de l'algorithme est en CAML-light interprété, ce qui explique certainement les résultats médiocres obtenus.

Un prouveur SAT utilise les mêmes principes que l'algorithme de Stålmarck, sans utiliser la notion de triplet (qui est brevetée) : HeerHugo [Groote/Warners 1999]. Nous comparerons ses performances avec un algorithme classique de type DP.

2.5.3 La méthode de Stålmarck vue par … Groote

Cette variante de la méthode de Stålmarck a pour principale caractéristique de remplacer les triplets (brevetés) par des clauses, ce qui permet à l'auteur d'utiliser des techniques classiques de SAT dans ce cadre. Par exemple, $v \Leftrightarrow (p \Leftrightarrow q)$ sera remplacé par {v p q, v q p, v p q, p q v} et $v \Leftrightarrow p \wedge q$ par {v p, v q, p q v}. On se retrouve donc avec une formule CNF contenant des clauses de taille inférieure ou égale à trois.

Sur cette formule sont appliquées des règles comme la propagation des clauses unitaires ou des littéraux purs et la recherche de littéraux équivalents basée sur des cycles d'implication (si p q, q $\neg r$ et $\neg r$ p alors p, q et $\neg r$ sont équivalents). Cela permet de réduire la taille du problème.

Les règles de déduction sont basées sur des tests de sous-sommation et la production de résolvantes,.

Exemple donné par l'auteur pour l'introduction de la clause C1 : p q r. On cherche une clause C2 ayant une forme particulière. Si une telle clause n'existe pas, la clause C1 est ajoutée simplement dans la base.

C2	
p q r	Ne pas ajouter C1 (Sous-sommation)
r p q	Ne pas ajouter C1, enlever C2, ajouter p q (Résolution)
q p r	Ne pas ajouter C1, enlever C2, ajouter p r (Résolution)
p q r	Ne pas ajouter C1, enlever C2, ajouter q r (Résolution)
p q	Ne pas ajouter C1 (Sous-sommation)
q p	Ne pas ajouter C1, ajouter p r (Résolution)
p q	Ne pas ajouter C1, ajouter q r (Résolution)
p r	Ne pas ajouter C1 (Sous-sommation)
r p	Ne pas ajouter C1, ajouter p q (Résolution)
p r	Ne pas ajouter C1, ajouter q r (Résolution)
q r	Ne pas ajouter C1 (Sous-sommation)
r q	Ne pas ajouter C1, ajouter q r (Résolution)
q r	Ne pas ajouter C1, ajouter p r (Résolution)

On peut noter que seules des clauses de taille deux sont produites par résolution.

Lorsque la clause p q est ajoutée, de nouvelles règles s'appliquent.

C2	
p q	Ne pas ajouter C1 (Sous-sommation)
q p	Ne pas ajouter C1, enlever C2, ajouter p (Résolution)
p q	Ne pas ajouter C1, enlever C2, ajouter q (Résolution)
p q	Ne pas ajouter C1, enlever C2, ajouter p $\Leftrightarrow$ q
p q r	Ajouter C1, enlever C2 (Sous-sommation)
r p q	Ajouter C1, enlever C2 (Sous-sommation)
q p r	Enlever C2, ajouter p r
q r p	Enlever C2, ajouter r p
p q r	Enlever C2, ajouter q r
r p q	Enlever C2, ajouter r q

On note que là encore, les clauses produites sont soit unitaires, soit de taille deux. De plus, des équivalences entre littéraux sont produites. Là aussi, la transitivité de l'équivalence est appliquée.

Groote note bien la similitude entre la notion de littéraux impliqués et la dilemma rule, en soulignant que la première est une version simplifiée de la seconde ne prenant pas en compte l'équivalence entre littéraux. Groote considère que la méthode de Stålmarck est une approche en largeur d'abord qu'il oppose à la recherche en profondeur d'abord de la procédure de Davis et Putnam.

Il propose en plus dans sa méthode des techniques proches de celles utilisées dans les méthodes de type DP : l'ajout de clauses durant la recherche (« nogood recording »), l'introduction d'une méthode de recherche locale type GSAT pour trouver une instanciation consistante des variables. Comme la méthode de Stålmarck est dédiée à la production de contradictions, elle n'est pas adaptée au cadre d'instances satisfaisables. Son utilisation conjointe avec une méthode de recherche locale doit lui permettre d'améliorer ses performances dans ce domaine.

On voit donc que HeerHugo est une implantation de la méthode de Stålmarck très proche des techniques utilisées dans la communauté IA travaillant sur SAT.

Pour comprendre l'intérêt de cet algorithme, nous allons donner ici quelques tests effectués sur HeerHugo dans le cadre de base de clauses 3-SAT générées aléatoirement. Nous avons utilisé pour cela la version de SATZ distribuée avec le système de planification « blackbox » et la version 0.3 de HeerHugo, disponible sur le site de l'auteur. Nous avons compilé les deux prouveurs sous Linux, sur un Pentium 133, 32 Mo. Voici les temps obtenus pour différentes instances de problèmes 3-SAT. Les temps affichés sont ceux obtenus par la commande Unix

`time %commande%`, HeerHugo ne donnant pas une mesure correcte du temps de résolution. Pour SATZ, les temps entre parenthèses sont obtenus en utilisant le paramètre –s du prouveur. Cette option permet d'enlever la production de résolvante à la racine. Dans le cadre des instances 100x800, SATZ effectuait une preuve par résolution de l'inconsistance (production de 126000 à 197000 résolvantes !), ce qui explique ces très mauvais résultats.
Comme les résultats de HeerHugo sont sans comparaison avec ceux de SATZ (ce dernier a résolu tous les problèmes en moins d'une seconde avec l'option -s), nous avons ajouté le temps d'une version personnelle de DP écrite en Java[18]. Elle utilise une heuristique MOMS classique plus une propagation unitaire à la SATZ sélectionnant la variable qui satisfait le plus de clauses. Ses performances sont loin d'atteindre celles de SATZ, mais on peut noter qu'elle est robuste : ses temps d'exécution se situent entre 1,98 et 6,87s. HeerHugo a un comportement beaucoup plus imprévisible : selon les instances, il trouve un résultat en 0,23 ou 85,17s ! De plus, on peut noter que les temps de résolution baissent avec l'augmentation du ratio : c'est normal puisque dans ce cas l'inconsistance devient « facile » à prouver (on le retrouve avec les deux autres algorithmes).

Instance	SATZ (SATZ –s)	HeerHugo	Prototype
100x500x#1	0,14s (0,08s)	47,38s	6,09s
100x500x#2	0,14s (0,07s)	24,92s	4,84s
100x500x#3	0,15s(0,10s)	44,45s	6,21s
100x500x#4	0,14s(0,07s)	51,68s	4,39s
100x500x#5	0,13s(0,08s)	51,24s	5,44s
100x600x#10	0,16s	18,29s	3,68s
100x600x#11	0,27s	5,61s	3,02s
100x600x#12	0,17s	18,43s	3,79s
100x600x#13	0,24s	10,05s	3,74s
100x600x#14	0,17s	28,44s	4,18s
100x700x#20	0,38s	4,97s	2,69s
100x700x#21	0,36s	3,55s	3,02s
100x700x#22	0,27s	7,59s	2,58s
100x700x#23	0,74s	2,57s	3,24s
100x700x#24	0,28s	7,12s	2,96s
100x800x#30	114,18s(0,06)	2,98s	2,80s
100x800x#31	332,70s(0,06)	0,23s	2,69s
100x800x#32	146,32s(0,08)	2,04s	2,47s
100x800x#33	139,31s(0,08)	2,16s	2,20s
100x800x#34	198,73s(0,08)	3,00s	2,36s
100x900x#40	(0,09)s	1,8s	2,36s
100x900x#41	(0,09)s	2,6s	2,47s
100x900x#42	(0,09)s	1,55s	2,30s
100x900x#43	(0,09)s	1,82s	2,53s
100x900x#44	(0,09)s	1,63s	1,98s
100x400x#50	0,12s	0,440s	3,41s
100x410x#50	0,10s	1,93s	3,46s
100x420x#50	0,09s	2,47s	2,31s
100x430x#50	0,09s	84,79s	6,87s
100x440x#50	0,10s	82,79s	6,81s
100x450x#50	0,14s	76,39s	6,16s
100x460x#50	0,13s	85,17s	5,77s
100x470x#50	0,12s	37,99s	5,06s

On peut noter particulièrement sur les instances autour du seuil que les méthodes issues de DP sont plus robustes que HeerHugo. Quand il y a beaucoup de solutions, HeerHugo trouve rapidement une solution. Quand la preuve de l'inconsistance est aisée, il trouve rapidement une preuve de l'inconsistance. Par contre, entre les deux, ses temps d'exécution explosent.

Nos tests sont juste indicatifs : SATZ est pour l'instant l'un des meilleurs représentants de DP, et HeerHugo n'a pas cette prétention pour les méthodes de type Stålmarck. De plus, il faudrait tester les deux algorithmes sur un

[18] Il faut noter que le temps présenté est le temps de traitement de l'instance (on ne compte pas le temps de lecture contrairement aux temps des autres prouveurs) mais aussi le temps nécessaire à la compilation « à la volée » du bytecode Java en code natif. A titre d'indication, une instance traitée une première fois dans ces conditions en 8,41s se trouve résolue en 2,20s après compilation du bytecode. Les tests ont été effectués sur la même machine que précédemment, mais sous Windows.

beaucoup plus grand nombre d'instances pour valider ces résultats. Néanmoins, une version personnelle de DP s'avère souvent plus efficace que HeerHugo. De plus, d'autres tests sur des formules venant du domaine de la validation de spécifications (taille = 100ko) ont été résolus par HeerHugo ainsi que par SATZ (là encore en moins d'1s !). Il semble donc que les instances de ces problèmes soient faciles.

Une petite remarque encore sur HeerHugo : nos tests sur une machine Linux disposant de 32 Mo nous ont montré la forte consommation en mémoire de cette méthode par rapport à un DP classique.

Nous devons tout de même nuancer notre propos : nous utilisons ici des tests sur des bases de clauses générées aléatoirement, sans structure, au format CNF. HeerHugo et l'algorithme de Stålmarck permettent d'exprimer un problème sous une forme plus intuitive et plus expressive. De plus, la prise en compte de l'équivalence entre littéraux qui est effectuée dans la méthode de Stålmarck est de peu d'intérêt sur ces instances aléatoires. On trouve dans [Groote/Warners 1999] des résultats concernant des instances DIMACS. On peut noter par exemple les bonnes performances de HeerHugo sur les instances inconsistantes bf* (instances résolues entre 2 et 35s) et ssa* (chaque instance est résolue en moins de 16s) alors que certaines d'entre elles sont difficiles pour les meilleurs prouveurs classiques. Or ces problèmes viennent d'applications réelles : HeerHugo semble donc véritablement adapté au traitement des instances réelles !

Il semble qu'on se trouve devant deux types de prouveurs complets : ceux permettant de résoudre efficacement certains problèmes (structurés) issus d'applications réelles mais qui ont de grandes difficultés à résoudre des instances aléatoires (particulièrement au seuil), et ceux qui résolvent des problèmes aléatoires et certains problèmes réels, sans doute les moins structurés. Les premières se distinguent par un très fort pouvoir sémantique (détection des littéraux impliqués, équivalents, symétries, retours arrières intelligents, etc.) qui semble adapté aux problèmes réels qui sont par nature structurés. Les deuxièmes utilisent des heuristiques afin de réduire toujours plus l'espace de recherche. Ces deux approches sont complémentaires. Les résultats présentés par [Groote/Warners 1999] laissent penser que d'ici peu de temps, nous retrouverons ce type de prouveur au coté de SATZ et al.

On peut imaginer de fusionner les deux approches en considérant la 0-saturation comme une version de la propagation unitaire à très fort pouvoir déductif. Elle permettrait peut-être d'obtenir plus de littéraux impliqués et des littéraux équivalents. On pourrait considérer qu'il s'agit d'une version « profondeur d'abord » de l'algorithme de Stålmarck. A l'heure ou nous écrivons ces lignes, une nouvelle version de SATZ incorporant la détection de littéraux équivalents à chaque nœud de l'arbre de recherche est annoncée [Li 1999][19]. Elle permet de résoudre le fameux « 32-bit parity problem », le challenge numéro 2 de [Selman, et al. 1997]. Notons que [Warners/Van-Maaren 1998] proposent aussi une méthode capable de résoudre une partie des instances de ce problème (celles compactées), fondée sur … la notion d'équivalence entre littéraux !

Autre symbiose possible : l'utilisation de 0-saturation voire 1-saturation en prétraitement afin de produire des littéraux impliqués et des équivalences entre variables afin de réduire le nombre de variables du problème initial. Dans [Kautz/Selman 1999], les auteurs parlent d'un programme développé par J. Crawford, « compact », qui applique la propagation unitaire (une forme simplifiée de 0-saturation), une heuristique de type UP (une 1-saturation simplifiée), puis une heuristique UP sur deux littéraux (2-saturation simplifiée) à une formule propositionnelle afin de la réduire. Les résultats obtenus sont impressionnant pour certaines instances (le monde des cubes), les formules sont réduites de 79% à 100% (soit la clause vide soit une base vide est produite), et deviennent triviales. Dans d'autres instances, la simplification va de 27% à 49%, mais les formules réduites restent difficiles à résoudre.

Ces deux dernières observations semblent conforter celles de Stålmarck et Groote.

[19] Des notions similaires sont aussi présentées dans [Brisoux 1999].

Partie II : P-impliqués premiers

« Pour résoudre un problème de cette nature, le principal est de savoir raisonner à rebours. C'est un art très utile, qui est peu pratiqué. On le néglige parce que la vie de tous les jours fait appel le plus souvent au raisonnement ordinaire. Pour cinquante personnes capables d'un raisonnement synthétique, à peine en est-il une qui sache faire un raisonnement analytique.[…] Je suppose que vous racontiez une série d'événements à un groupe de personnes, et que vous leur demandiez de vous en dire la suite ; elles le repasseront dans leur esprit et la plupart d'entre elles trouveront ce qui en découle. Maintenant le contraire : vous leur donnez d'abord la fin d'une autre série d'événements ; combien pourront en inférer la série ? Fort peu. C'est cette dernière opération que j'appelle le raisonnement analytique ou le raisonnement à rebours. » Sir Arthur Conan Doyle, Etude en Rouge.

Guide de lecture du chapitre

Ce chapitre est organisé d'une manière un peu particulière, qui correspond dans ses grandes lignes à la chronologie de nos travaux. Nous présentons tout d'abord une application de nos travaux (l'ATMS, ou plus exactement le CMS) pour illustrer de façon intuitive la notion de P-impliqué premier. Puis nous définirons la notion d'impliquant P-restreint qui nous sera nécessaire pour définir logiquement notre méthode de calcul des P-impliqués premiers basée sur l'utilisation d'une variante de la procédure de Davis et Putnam. Nous présenterons ensuite une méthode analogue de calcul de P-impliquants premiers. Nous donnerons enfin quelques résultats expérimentaux à propos de ces algorithmes.

1 Impliqués premiers et ATMS

La notion de P-impliqué premier que nous abordons dans cette thèse (*P* étant un ensemble consistant de littéraux) est un peu différente de la notion de P-impliqué que l'on trouve habituellement dans la littérature. En effet, la plupart du temps, P n'est pas réduit à un ensemble de littéraux consistant. Nous allons montrer dans cette section un exemple d'utilisation de cette notion de P-impliqué premier particulière et comment cette particularité nous permet de présenter un algorithme de calcul de ces P-impliqués premiers basé sur la procédure de Davis et Putnam.

1.1 Présentation informelle

L'histoire se déroule en Bretagne, plus exactement dans le sud Finistère. La ville de Quimper est en émoi depuis quelques jours : son député-maire socialiste, B. P. a disparu. Aucune rançon n'a été demandée. Tout de suite, les soupçons se sont portés sur l'opposition : en effet, lors du dernier meeting électoral, B.P. avait lancé aux bigoudens « coiffes longues, idées courtes » et aux penn-sardins une boutade sur l'origine du kouign-amman. Ceux ci étaient rentrés dans une colère noire, jurant la perte du député-maire. Le préfet venait de donner un ultimatum à l'inspecteur Yohann De Kernec : « je veux une explication avant ce soir ! » avait-il dit. L'inspecteur avait mis ses meilleurs éléments sur le coup et les résultats étaient là : 5 suspects arrêtés. 3 bigoudens (Per, Youen et Alan) et 2 penn sardinns (Denez et Jean). Tous de l'opposition. Ils sont connus des services de police depuis longtemps.

Voici leurs témoignages concernant leur emploi du temps de la soirée de la disparition :

- *Per : « Je ne me souviens plus très bien. J'étais soit au bistrot, au PSG (ndlr : « Plogastel St Germain »), soit chez Youen. En tout cas, j'ai fini la soirée dans un état lamentable … »*

- *Youen : « C'est vrai que Per est venu chez moi il y a quelques jours. Mais ce jour là précisément, il me semble que je suis allé à un festnoz à Douarnenez : c'était gratuit ! La soirée en tout cas a sûrement été arrosée. »*
- *Alan : « Mes compagnons Per et Youen étaient complètements bourrés ce soir là. Comment voulez-vous que l'on aie fait le coup ? Ce sont les penn sardinns qui ont fait le coup ! »*
- *Denez : « Je crois que je suis allé faire un tour au Port Rhu à moins que ce soit le jour du festnoz, auquel cas j'y étais. En tout cas, j'ai fini la soirée à la frégate. »*
- *Jean : « Je suis videur à la frégate. J'ai aperçu Denez vers 23h et j'ai refusé de laisser entrer Alan : il était trop ivre. Vous voyez bien que je n'y suis pour rien. Ce sont les bigoudens qui ont fait le coup ! »*

Vérifications faites, il y avait bien eu un festnoz à Douarnenez le soir de la disparition, et quelqu'un pouvait témoigner avoir vu Per au bistrot.

« Quelque chose cloche dans ces déclarations : jamais les bigoudens et les penn sardinns ne se seraient alliés pour faire le coup ! ». L'inspecteur De Kernec but un verre de lambig cul sec : « l'Alcool fait Travailler les Méninges des Sages ! » soupira-t-il en reposant son verre. Il lui restait une heure avant la rencontre tant redoutée avec le préfet.

Le préfet était de mauvaise humeur. Il écoutait attentivement le compte rendu de l'inspecteur :
« Monsieur le préfet, nous avons 5 suspects. L'un d'eux ment sur son emploi du temps le soir du délit. Trois témoignages corroborent la thèse d'une action bigoudène, et trois autres ceux d'une action penn sardinn. Ce sont les seules informations dont nous disposons pour l'instant. »

« Je vous remercie inspecteur. Mais je ne veux pas d'hypothèses, je veux des faits. Rien que des faits ! »

Comment l'inspecteur De Kernec peut-il avancer cela ?

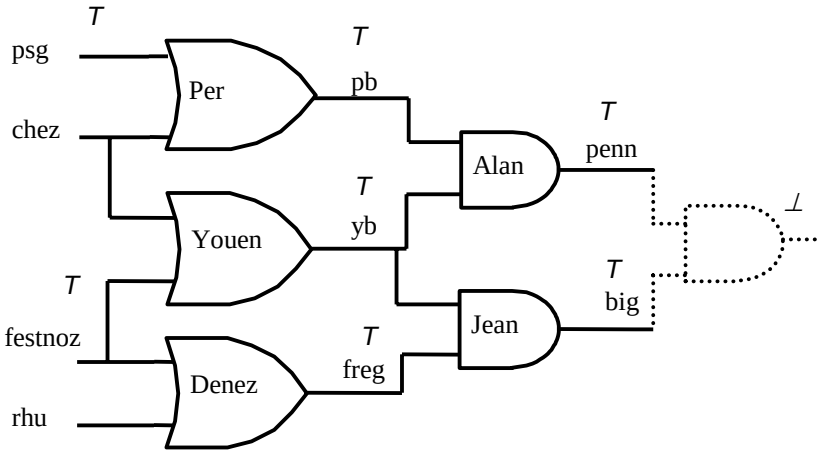
Traduisons les faits sur les emplois du temps des 5 suspects en calcul propositionnel. Les symboles propositionnels sont les suivants :

- psg : Per était au bistrot
- pb : Per était ivre
- chez : Per et Youen étaient ensemble chez Youen
- yb : Youen était ivre
- festnoz : ce soir là il y avait un festnoz à Douarnenez
- penn : ce sont les penn sardinns qui ont fait le coup
- freg : soirée à la frégate
- rhu : soirée au Port Rhu
- big : ce sont les bigoudens qui ont fait le coup

Voici *BZH* la base de clauses correspondant aux cinq dépositions :

{psg pb, chez pb, psg chez ,
chez yb, festnoz yb, chez festnoz ,
pb yb penn,
festnoz freg, rhu freg, festnoz rhu ,
freg yb big}

On peut représenter ces règles sous la forme d'un circuit logique :



Voici *OBS* les informations recueillies par les policiers :

{ festnoz, psg, penn big }

Pour savoir si les dépositions sont crédibles, il suffit de tester la consistance de *BZH∪OBS*. Or ici $(BZH \cup OBS)^{\wedge} \models \bot$ donc au moins une des dépositions est fausse (en supposant que les informations recueillies par les policiers sont exactes). Comment faire pour en savoir plus ?

Dans cette petite histoire, il nous faut retenir une chose : pour raisonner, l'inspecteur doit faire des hypothèses. Nous allons faire de même : nous utiliserons donc 5 symboles propositionnels spéciaux, qui seront nos hypothèses : *P* (Per dit la vérité), *Y* (Youen dit la vérité), *A* (Alan dit la vérité), *D* (Denez dit la vérité) et *J* (Jean dit la vérité). Chaque hypothèse permet d'ignorer une déposition. Par exemple, si l'hypothèse *P* est vraie, alors on tiendra compte de la déposition de per, sinon on l'ignorera, ce qui se traduit en logique par *P ((psg pb) ∧ (chez pb))* soit *(P psg pb) ∧ (P chez pb).*

On obtient alors la nouvelle base de clause *BZH'* :
{P psg pb, P chez pb, psg chez ,
Y chez yb, Y festnoz yb, chez festnoz ,
A pb yb penn,
D festnoz freg, D rhu freg, festnoz rhu ,
J freg yb big}

Désignons l'ensemble des hypothèses *{P,Y,A,D,J}* par *H*. Nous cherchons d'où proviennent les contradictions dans les témoignages, c'est à dire qui sont les personnes qui ne peuvent pas toutes dire la vérité en même temps. En logique, cela revient à chercher $E \subseteq H$ tel que $(BZH' \cup OBS \cup E)^{\wedge} \models \bot$. Pour localiser précisément la (ou les) source(s) de l'incohérence des témoignages, on se restreindra au plus petit *E* (pour l'inclusion) qui satisfait cette propriété. Dans notre exemple, nous trouvons *E = H*, ce qui signifie que c'est l'ensemble des dépositions qui produit l'inconsistance de la base (inconsistance globale). Il suffit donc d'ignorer au moins une déposition pour obtenir un déroulement crédible de la soirée. Malheureusement, rien ne permet de choisir entre les 5 dépositions : il y a donc 5 cas de figure possibles si on ignore le minimum de dépositions nécessaires à un déroulement crédible de la soirée, 5 « interprétations » de ce qui s'est passé ce soir là.

Comme certaines dépositions accusent directement les penn-sardins ou les bigoudens, il apparaît judicieux de chercher les dépositions qui étayent ces accusations. Soit en logique chercher $E \subseteq H$ tel que $(BZH' \cup OBS \cup E)^{\wedge} \models$ *penn* et $E' \subseteq H$ tel que $(BZH' \cup OBS \cup E')^{\wedge} \models$ *big*. On obtient un unique *E={P,Y,A}* et un unique *E'={Y,D,J}*. Il faut noter que ces deux ensembles de témoignages sont cohérents avec les observations faites par la police.

Nous comprenons maintenant que l'inspecteur De Kernec a eu une « simple » démarche logique dans l'analyse de la situation. Mais rien ne lui permet de désigner un coupable ! Nous verrons dans la partie suivante que l'utilisation dans son raisonnement de connaissances exogènes pourrait lui permettre de conclure.

Toutes les notions utilisées par l'inspecteur De Kernec se retrouvent dans un outil, l'ATMS (*« Assumption-based Truth Maintenance System »*, système de maintien de la cohérence basé sur les hypothèses) conçu initialement

pour le diagnostic de panne par De Kleer [de-Kleer 1986, de-Kleer 1986b]. Elles sont la base de tant d'autres domaines, notamment en logique non monotone, que l'ATMS est actuellement utilisé dans de nombreuses applications : restauration de la cohérence, argumentation, révision, etc. Nous allons tout d'abord formaliser les différentes notions que nous venons de voir dans le cadre de l'ATMS (notions de labels, nogoods,…) puis nous les reprendrons en termes plus « logiques » (P-impliqués). Nous présenterons ensuite une méthode de calcul de ces P-impliqués à partir du calcul « d'impliquants P-restreints », une notion que nous proposerons et formaliserons. Nous étudierons ensuite différentes applications de nos méthodes.

1.2 L'ATMS de De Kleer

Le système présenté par de Kleer descend d'une série de systèmes de maintien de la cohérence (TMS) qui ont pour but d'aider un système d'inférence. Ce système d'inférence utilise souvent une logique de type 1er ordre : il contient la connaissance et les règles d'inférence des différents domaines. Chaque inférence est communiquée au système de maintien de la cohérence. Celui-ci doit déterminer si ce qui est inféré est cohérent avec ce qui a déjà été déduit. Le TMS, contrairement au moteur d'inférence, travaille en logique propositionnelle. Le TMS est donc un guide pour le moteur d'inférence.

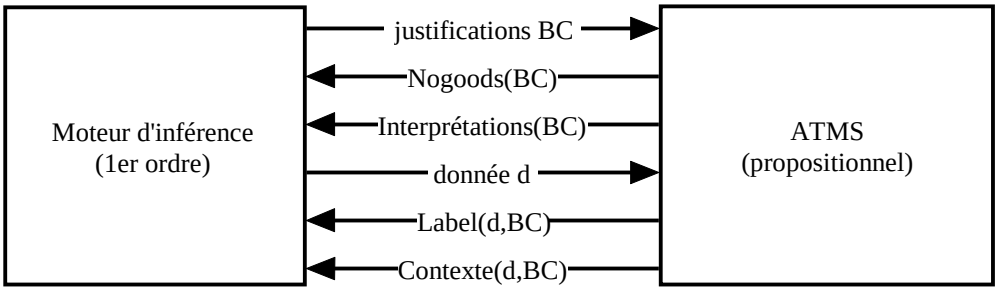


Figure 3 relations moteur d'inférence/ATMS

Nous allons tout d'abord donner les définitions des différents concepts utilisés dans l'ATMS de De Kleer sous leur forme initiale [de-Kleer 1986, de-Kleer 1986b], c'est à dire sous la forme d'ensembles de littéraux particuliers. Puis nous nous intéresserons plus particulièrement à leur sémantique, comme dans [Reiter/de-Kleer 1987]. Nous ne rentrerons pas ici dans le détail de l'implémentation de l'ATMS de De Kleer. Nous renvoyons le lecteur à ses articles pour une description complète du système.

1.2.1 Assumption-based Truth Maintenance System

Soit S l'ensemble des symboles propositionnels appelées aussi *données*. $H \subseteq S$ désigne l'ensemble des *symboles hypothèses* (ou simplement *hypothèses*). $E \subseteq H$ est appelé un *environnement*. Les clauses utilisées dans l'ATMS sont de la forme $d_1\ d_2 \ldots d_n \quad d$ (clauses de Horn) avec $d_i \in S$ et $d \in S \cup \{\bot\}$. De Kleer les appelle *justifications de d*.

Soit BC un ensemble de justifications. On dira qu'un environnement E est *incohérent* avec BC ssi $(BC \cup E)^\wedge \models \bot$. On dira qu'un environnement E est *cohérent* avec BC ssi il n'est pas incohérent avec BC.

Définition 28 (nogoods)
Soit $BC \subset FP_S$ un ensemble de justifications. On appelle *nogood* de BC tout environnement minimal incohérent de BC (E est un nogood de BC ssi $(BC \cup E)^\wedge \models \bot$ et $\nexists\ E' \subset E$ tel que $(BC \cup E)^\wedge \models \bot$). On notera $NG(BC)$ l'ensemble des nogoods de BC.

Exemple : Dans notre exemple, $\{P,Y,A,D,J\}$ était le seul nogood de $BZH' \cup OBS$.

> Le calcul des nogoods d'une base de clauses résout le problème SAT : si $NG(BC)=\{\varnothing\}$ (l'environnement vide) la base est inconsistante sinon elle est consistante.

Définition 29 (interprétations de l'ATMS)
Soit $BC \subset FP_S$ un ensemble de justifications. On appelle *interprétation*[20] de BC tout environnement maximal cohérent de BC (E est une interprétation de BC ssi $(BC \cup E)^\wedge \not\models \bot$ et $\nexists\ E' \supset E$ tel que $(BC \cup E')^\wedge \not\models \bot$). On notera $INTER(BC)$ l'ensemble des interprétations de BC.

[20] A ne pas confondre avec la notion d'interprétation du calcul propositionnel (Définition 3, p. 18).

Exemple : *INTER(BZH'∪OBS)={{Y,A,D,J},{P,A,D,J},{P,Y,D,J},{P,Y,A,J},{P,Y,A,D}}*

> *INTER(BC)* peut être calculé à partir de *NG(BC)* : un nogood représentant le plus petit ensemble d'hypothèses incohérent avec *BC*, il suffit d'enlever de *H* une hypothèse de chaque nogood pour obtenir un environnement cohérent Autrement dit, il suffit d'enlever de *H* un « hitting set » (Définition 23, p.24) de *NG(BC)* pour obtenir un environnement cohérent. (De Kleer appelle ces « hitting set » des *candidats* dans [de-Kleer 1986b]. Si ce « hitting set » est minimal, alors l'environnement obtenu sera maximal. On a donc *INTER(BC)={H \ E | E ∈ MHS(NG(BC))}*.

Proposition 14 relation nogoods/interprétations
Soit $BC \subset FP_S$ un ensemble de justifications. *INTER(BC)={H \ E | E ∈ MHS(NG(BC))}*

Définition 30 (label d'une donnée)
Soit $BC \subset FP_S$ un ensemble de justifications. On appelle *label* d'une donnée *d* dans *BC*, noté *label(d,BC)*, l'unique ensemble d'environnements ayant les propriétés suivantes :
Cohérence : *∀E ∈ label(d,BC), E* cohérent avec *BC*
Correction : *∀E ∈ label(d,BC), (BC∪E)^⊨ d*
Minimalité : *∀E ∈ label(d,BC), ∄ E'⊂E* tel que *(BC∪E')^⊨ d*
Complétude : *∀E'⊆ H* cohérent avec BC tel que *(BC∪E')^⊨ d ∃E∈ label(d,BC)* tel que *E ⊆ E'*.

Exemple : *label(penn,BZH'∪OBS)={{P,Y,A}}* et *label(big,BZH'∪OBS)={{Y,D,J}}*.

> Si l'on relâche la propriété de cohérence, *NG(BC)=label(⊥,BC)*. Dans l'ATMS de de Kleer, les labels des données et l'ensemble des nogoods sont calculés de manière similaire.

Définition 31 (contexte d'un environnement)
Soit $BC \subset FP_S$ un ensemble de justifications. Soit $E \subseteq H$. On appelle contexte de l'environnement cohérent *E* dans *BC*, noté *contexte(E,BC)*, l'ensemble des données déductibles de *BC∪E* (*contexte(E,BC)={d | (BC∪E)^ ⊨ d}*).

Exemple : *contexte({P,Y,A},BZH'∪OBS) = {P,Y,A,pb,yb,penn,festnoz,psg}*
contexte(({D,Y,J},BZH'∪OBS) = {P,Y,A,yb,freg,festnoz,psg}

De Kleer distingue certains contextes particuliers.

Définition 32 (extensions)
Soit $BC \subset FP_S$ un ensemble de justifications. Les contextes des interprétations de *BC* sont appelés les *extensions* de *BC* (*Extensions(BC)={contexte(E,BC) | E ∈ INTER(BC)}*).

L'algorithme que propose de Kleer pour calculer ces différents ensembles est un algorithme de type chaînage avant, ce qui explique la nécessité *d'utiliser des clauses de Horn*. Ce processus a l'avantage d'être incrémental : à tout moment, le label de toutes les données est connu, et l'introduction d'une nouvelle justification dans l'ATMS provoque une mise à jour de ces labels. Néanmoins, cette caractéristique est aussi une contrainte de l'ATMS : il est souvent inutile de connaître le label de toutes les données. Dans le cadre d'un diagnostic de panne par exemple où les justifications représentent la description complète d'une voiture, on se restreindra souvent à un sous ensemble du système : si elle ne démarre pas, inutile de vérifier le système de freinage ou l'ampoule du plafonnier ! C'est pourquoi la notion de « focus » est rapidement apparue [Forbus/De-Kleer 1988, Yacoub 1997]. On fournit à l'ATMS un ensemble de données pertinentes, le *focus*, et seules les justifications dont la partie antécédent est incluse dans le focus sont traitées. On peut ainsi réduire fortement le nombre de labels calculés. Toute la difficulté de cette approche est de constituer cet ensemble de données.

Dans le but non plus de limiter le nombre de labels calculés mais la taille de ces labels, des probabilités [de-Kleer/Williams 1987, de-Kleer 1991], des possibilités [Dubois, et al. 1990], voire plus généralement des coûts [Ngair/Provan 1993], ont été ajoutés aux hypothèses : on calcule alors les environnements les plus probables, les plus certains, ceux minimisant une fonction de coût. Nous renvoyons le lecteur à [Provan 1996] pour une information plus complète sur les fonctions de coût utilisables dans ce cadre.

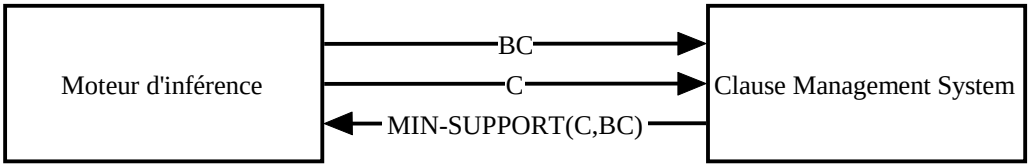
Une autre manière de limiter l'explosion combinatoire due au nombre de labels et à leur taille consiste à ne calculer ces labels que lorsque l'on en a vraiment besoin : c'est l'idée développée par [Tatar 1994, Palmade 1992].

L'un des reproches que l'on peut faire à l'ATMS est son expressivité limitée, due à l'utilisation de justifications (clauses de Horn). C'est pourquoi de Kleer a ajouté la possibilité de rajouter des disjonctions d'hypothèses (« *choose* » [de-Kleer 1986b] et des négations d'hypothèses avec le *Negated assumption ATMS* [de-Kleer 1988]).

Toutes les notions que nous venons de voir ont été définies par de Kleer pour son ATMS. Elles sont très proches d'un problème de base en calcul propositionnel : le calcul d'impliquants/impliqués premiers. En effet, dans [Reiter/de-Kleer 1987], les auteurs remplacent les concepts de l'ATMS dans un cadre plus général, le CMS.

1.2.2 Clause Management System

On peut voir le CMS comme une sorte de généralisation de l'ATMS. Nous avons souligné le fait que les définitions de de Kleer dans son ATMS sont très fortement liées à son implantation du système (justifications, no-goods/label). Le CMS reprend les mêmes concepts que l'ATMS mais indépendamment de toute implantation, il les définit logiquement.



Le CMS est défini à partir de la notion de support :

Définition 33 (support, support minimal)
Soit $BC \subset FP_S$ une base de clauses. Soit δ une clause de FP_S. Soit α une clause de FP_S. α est appelé le *support* de δ dans BC ssi :

1. $BC^\wedge \models \alpha \vee \delta$, ou encore $BC^\wedge \wedge \neg\alpha \models \delta$
2. $BC^\wedge \not\models \alpha$, ou encore $BC^\wedge \wedge \neg\alpha \not\models$ ($BC \wedge \neg\alpha$ consistant).

De plus, α est appelé support minimal de δ ssi $\nexists \alpha'$ un support de δ tel que $\alpha' \angle \alpha$.

Dans cette définition, on retrouve exactement la notion de label. On peut retrouver aisément que $E \in label(d,BC)$ ssi $\overline{E}^\vee$ est un support minimal de d dans BC. On notera toutefois que l'on ne cherche plus ici le label d'une donnée mais plus généralement le label d'une clause.

[Reiter/de-Kleer 1987] montrent que l'ensemble des supports minimaux de α peut être calculé à partir des impliqués premiers de BC (dans cet article, les auteurs inversent la notion d'impliqué et d'impliquant : les théorèmes suivants utilisent les appellations standard).

Proposition 15 (théorème 2, p. 186, [Reiter/de-Kleer 1987])
Soit $BC \subset FP_S$. Soit $F^\vee$ une clause de FP_S. Si $E^\vee$ est un *support minimal* de $F^\vee$ dans BC alors $\exists M^\vee$ un impliqué premier de BC tel que $M \cap F \neq \emptyset$ et $E = M \setminus F$.

La réciproque de ce théorème n'est pas vraie[21] : en voici une version affaiblie.

Proposition 16 (théorème 3, p. 186, [Reiter/de-Kleer 1987])
Soit $BC \subset FP_S$. Soit $F^\vee$ une clause non nulle de FP_S. Si $M^\vee$ est un impliqué premier de BC tel que $F \subseteq M$ alors $(M \setminus F)^\vee$ est un support minimal pour $F^\vee$ dans BC.

On peut donc définir l'ensemble des supports minimaux d'une clause dans une formule à partir des impliqués premiers de cette formule :

Proposition 17 (théorème 5, p. 186, [Reiter/de-Kleer 1987])
Soit $BC \subset FP_S$. Soit $F^\vee$ une clause de FP_S. L'ensemble des supports minimaux de $F^\vee$ dans BC, *noté* $MIN\text{-}SUPPORT(F^\vee,BC) = \{(M \setminus F)^\vee \mid M^\vee$ est un impliqué premier de BC *et* $M \cap F \neq \emptyset\}$.

Cette proposition montre qu'il suffit de connaître les impliqués premiers d'une formule pour calculer les supports de n'importe quelle clause dans cette formule.

[21] Prendre par exemple $BC=\{ a\ b,\ a\ c\ d\}$ et $F=\{b,d\}$. L'impliqué premier $a\ c\ d$ respecte $\{a,c,d\} \cap \{b,d\} \neq \emptyset$, donc $a\ c$ est bien un support de $F^\vee$ dans BC mais il n'est pas minimal car a est aussi un support de $F^\vee$ dans BC.

Deux approches sont alors possibles : garder la formule sous sa forme initiale (une base de clauses) et calculer ses impliqués premiers à chaque demande de supports minimaux (approche interprétée), ou alors remplacer la formule par ses impliqués premiers, car une formule est logiquement équivalente à la conjonction de ses impliqués premiers (approche compilée). La première approche limite la taille de stockage du problème, mais demande beaucoup de calculs (il faut recalculer les impliqués premiers à chaque fois) : elle est envisageable si il y a beaucoup de mises-à-jour de la base de clauses, et peu de requêtes. La seconde limite le calcul des supports à leur recherche parmi les impliqués premiers, mais le nombre de ces impliqués premiers pouvant augmenter de façon exponentielle avec la taille du problème, leur stockage et leur calcul peut devenir très coûteux en espace et en temps. Le calcul des impliqués premiers d'une mise à jour de la base peut bénéficier des impliqués premiers déjà calculés (on trouve dans [de-Kleer 1992] une méthode de calcul d'impliqués premiers dédiée à cette utilisation). Cette seconde approche sera utilisée si il y a peu de mises-à-jour de la base et beaucoup de requêtes.

Le nombre d'impliqués d'une formule pouvant être énorme ($O(3^n/n)$ [Marquis 1999]), différents travaux se situent entre l'ATMS et le CMS : alors que la notion de support n'est pas restreinte à un sous ensemble des variables dans le cadre du CMS, l'idée est de garder la notion d'environnement de l'ATMS. [Kean/Tsiknis 1990, Kean/Tsiknis 1992] par exemple proposent avec l'*Assumption based CMS* (ACMS) une restriction du CMS à un sous ensemble des symboles propositionnels. On cherche donc des supports qui contiennent uniquement des symboles hypothèses, c'est à dire que seuls les impliqués premiers contenant des symboles hypothèses en partie antécédent nous intéressent. Pierre Tayrac [Tayrac 1990] appelle ces clauses des CAT (Clauses à Antécédent Typé). Il propose dans sa thèse une méthode de calcul d'impliqués premiers de type CAT basée sur une méthode de résolution dirigée : la CAT-résolution. Plus récemment, Rachid Yacoub [Yacoub 1997] a intégré la notion de focus dans cette approche.

Les travaux d'Olivier Palmade [Palmade 1992], inspirés de ceux de Pierre Tayrac, nous fournissent la mise en œuvre d'un ATMS s'appuyant sur un prouveur propositionnel à base de « P-clash » (principe de résolution à l'envers). La particularité de cet ATMS est son fonctionnement en « deux passes » (voir section suivante). Les travaux que nous avons menés au cours de cette thèse en ont été très fortement inspirés, en partant de la remarque suivante : « si les meilleurs prouveurs de SAT, à l'heure actuelle et en moyenne, sont basés sur des algorithmes de type Davis et Putnam, pourquoi ne pas essayer de remplacer le prouveur de type P-clash par un Davis et Putnam ? ».

1.3 ATMS « à deux passes »

L'idée de ce type d'ATMS est un peu différente de la notion de CMS. Au lieu de tout calculer à partir de la notion de support, nous allons tout calculer à partir de la notion de nogood.

1.3.1 Remarques préliminaires

En effet, les notions de label et de nogoods sont très proches : si $E \in label(d,BC)$ alors E est un environnement minimal cohérent avec BC tel que $(BC \cup E)^\wedge \models d$. D'après la Proposition 2 page 19, on a $(BC \cup E)^\wedge \models d$ ssi $(BC \cup E)^\wedge \wedge \neg d \models \bot$ ssi $(BC \cup \{\neg d\} \cup E)^\wedge \models \bot$. Donc $E \in NG(BC \cup \{\neg d\})$. Comme E est cohérent, $\nexists E' \in NG(BC)$ tel que $E' \subseteq E$. On obtient donc la propriété suivante :

Proposition 18 relation nogoods/label
Soit $BC \subset FP_S$. Soit d une donnée. $Label(d,BC) = \{E \mid E \in NG(BC \cup \{\neg d\})$ *et* $\nexists E' \in NG(BC)$ *tel que* $E' \subseteq E\}$.

Il suffit donc de savoir calculer les nogoods d'une base de clauses pour calculer le label d'une donnée. On peut noter que la seule condition que nous ayons est, pour des raisons pratiques et non théoriques car les algorithmes de résolution de SAT travaillent sur des bases de clauses, « $BC \cup \{\neg d\}$ est une base de clauses ». Nous pouvons donc généraliser la notion de label :

Définition 34 (généralisation du label)
Soit $BC \subset FP_S$. Soit α un cube ou une clause de FP_S . $Label(\alpha,BC) = \{E \mid E \in NG(BC \cup \{\neg\alpha\})$[22] *et* $\nexists E' \in NG(BC)$ *tel que* $E' \subseteq E\}$.

Par rapport à la notion de support définie dans le CMS, nous permettons le calcul du label de conjonctions de littéraux : nous verrons dans la section Applications (p. 92) que cette propriété peut s'avérer intéressante au niveau de l'expressivité de l'outil.

[22] Si α est une clause, $\neg\alpha$ est un cube. Or un cube est une CNF, donc une base de clauses.

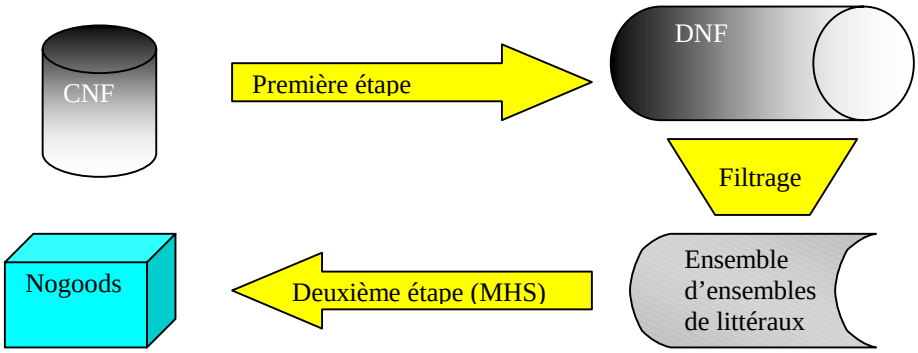
Essayons maintenant de donner au lecteur l'intuition de notre démarche. Chercher les nogoods d'une base de clauses *BC* revient à chercher les ensembles de symboles hypothèses dont la satisfaction rend la base inconsistante. Si *BC* était donnée sous forme *DNF*, il serait aisé de trouver un tel ensemble d'hypothèses s'il existe : il suffirait pour chaque cube de cette formule de choisir un symbole hypothèse qui apparaît négativement. Si l'un des cubes ne contient aucune négation de symbole hypothèse, alors il n'y a pas de nogoods pour cette formule. Sinon, la satisfaction de chacun de ces symboles falsifiant chacun des cubes, la base devient bien inconsistante. Calculer un nogood revient donc à choisir dans chaque cube une des négations hypothèses, en s'assurant que ce choix est minimal, c'est à dire choisir un « minimal hitting set » sur l'ensemble des cubes restreints aux symboles hypothèses apparaissant négativement.

Prenons par exemple *BC = {A b c, B b, c }*. Une DNF correspondante est *(¬B∧¬b∧¬c)∨(¬A∧b∧¬c)*. Si nous filtrons chaque cube sur ses hypothèses apparaissant négativement, nous obtenons les singletons {B} et {A}. On obtient alors comme nogood de la base {A,B} qui est le seul élément de *MHS({{A},{B}})*.

On peut donc différencier deux étapes principales dans cette approche :

1. passer d'une CNF en une DNF,
2. effectuer l'opération MHS.

D'où le nom d'ATMS à deux passes.



Nous allons présenter de manière informelle une approche basée sur la procédure de Davis et Putnam qui nous permet d'effectuer ces deux étapes à l'aide d'un seul algorithme. La formalisation de cette approche fera l'objet de la section suivante.

1.3.2 Quelques bonnes propriétés de DP

Un algorithme de type Davis et Putnam peut être utilisé pour passer d'une formule CNF en une formule DNF : nous avons vu section 2.2.3 (p. 39) que l'on pouvait modifier l'algorithme original afin de calculer une couverture d'impliquants d'une base de clauses, soit passer d'une CNF à une DNF. L'utilisation d'un prouveur basé sur DP à la place du prouveur basé sur la résolution ne pose donc pas de problème à ce niveau. Il nous reste maintenant à effectuer la fameuse opération MHS.

Si nous regardons de plus près cette opération, on peut remarquer que choisir un élément de chaque ensemble (trouver un « hitting set ») ressemble beaucoup à satisfaire un littéral de chaque clause (trouver un impliquant) à la différence que seuls des littéraux purs sont utilisés dans le premier calcul, c'est à dire que l'on ne trouve jamais un littéral et son complémentaire, et que la valeur de vérité donnée au symbole doit être interprétée comme « appartient à l'ensemble » et « n'appartient pas à l'ensemble ». MHS revient donc à effectuer un changement de forme sur une base de clauses constituée uniquement de littéraux purs et de ne tenir compte que de ces littéraux !

Exemple : *MHS({{A,B,C},{D},{E,F}}) = {{A,D,E},{B,D,E},{C,D,E},{A,D,F},{B,D,F},{C,D,F}}*. Un calcul de couverture d'impliquants de la base *{ A B C, D, E F }* à partir d'un algorithme à la DP nous donne *{{**A**,¬B,¬C,**D,E**,¬F }^, {**B**,¬C,**D,E**,¬F}^ {**C,D,E**,¬F}^, {**C,D,E,F**}^, {**B**,¬C,**D,E,F**}^, {**A**,¬B,¬C,**D,E,F**}^, {**B,D**,¬E,**F**}^, {¬B,**C,D**,¬E,**F**}^, {**A**,¬B,¬C,**D**,¬E,**F**}^}*. En ne considérant que les littéraux positifs (en gras), et en minimisant pour l'inclusion, on retrouve bien le même résultat.

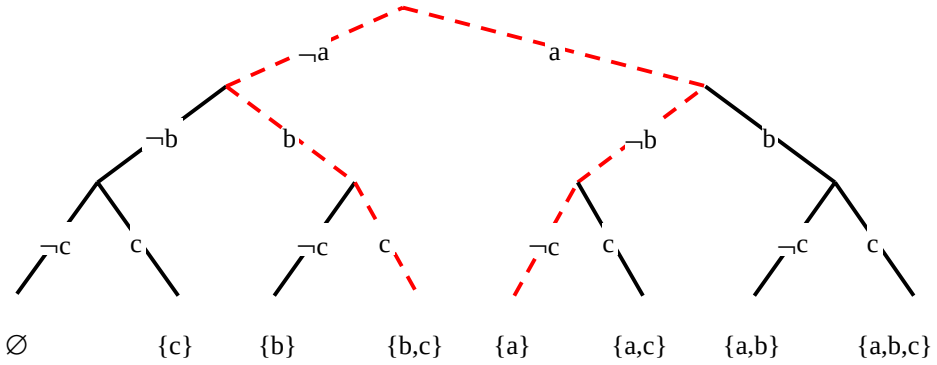
Comme nous avons déjà noté, une variante de DP peut nous permettre d'effectuer le changement de forme. Il nous reste alors à effectuer une opération délicate : le test de sous-sommation nous permettant de produire uniquement les « hitting sets » minimaux ! Comme l'a souligné de Kleer dans [de-Kleer 1992], cette opération est cruciale dans tout calcul d'impliqué premier. De Kleer propose un codage des impliqués trouvés sous forme d'arbre de discrimination. Pour notre part, nous allons utiliser une propriété de la procédure de Davis et Putnam

afin d'ordonner la production des impliquants : en effet, à chaque nœud de l'arbre de recherche, nous devons choisir entre un littéral et son complémentaire. Nous pouvons par exemple choisir de toujours commencer par un littéral négatif, afin d'essayer d'abord les impliquants ayant le plus de littéraux négatifs, et en dernier ceux qui ont le plus de littéraux positifs. La particularité de l'arbre construit est que les impliquants sont produits dans un ordre compatible avec l'inclusion de leur restriction aux littéraux positifs. Ainsi, la partie conséquent du premier impliquant trouvé nous fournira un premier « hitting set » minimal M.

Propriété 3
Soit S un ensemble de symboles propositionnels. Soit <P;P'> une partition de L_S telle que P et P' sont des ensemble consistants de littéraux. Une heuristique de branchement qui, pour tout littéral $l \in P$, explore la branche $l^{\neg}$ avant la branche l, ordonne les interprétations selon un ordre compatible avec l'inclusion restreinte à P, c'est à dire que si B_1 et B_2 sont deux branches de l'arbre et $(B_1 \cap P) \subset (B_2 \cap P)$ alors B_1 est explorée avant B_2.

> **Preuve :**
> Prenons deux branches de l'arbre B_1 et B_2, B_1 étant explorée avant B_2 (on représente une branche par l'ensemble des littéraux qui l'étiquettent). Soit P l'ensemble des littéraux positifs (par renommage, on peut toujours se ramener à ce cas). Il existe une propriété qui indique que toutes les branches d'un arbre se rejoignent en au moins un point. Soit $s \in P$ le symbole de branchement où B_1 et B_2 *se rejoignent* (en parcourant l'arbre de bas en haut, *se séparent* dans l'autre sens). Donc $\neg s \in B_1$ et $s \in B_2$. Donc on ne peut pas avoir $B_2 \cap P \subseteq B_1 \cap P$.

Exemple : on peut le vérifier dans l'arbre binaire développé pour les 3 symboles a,b,c ci-dessous (P={a,b,c}).

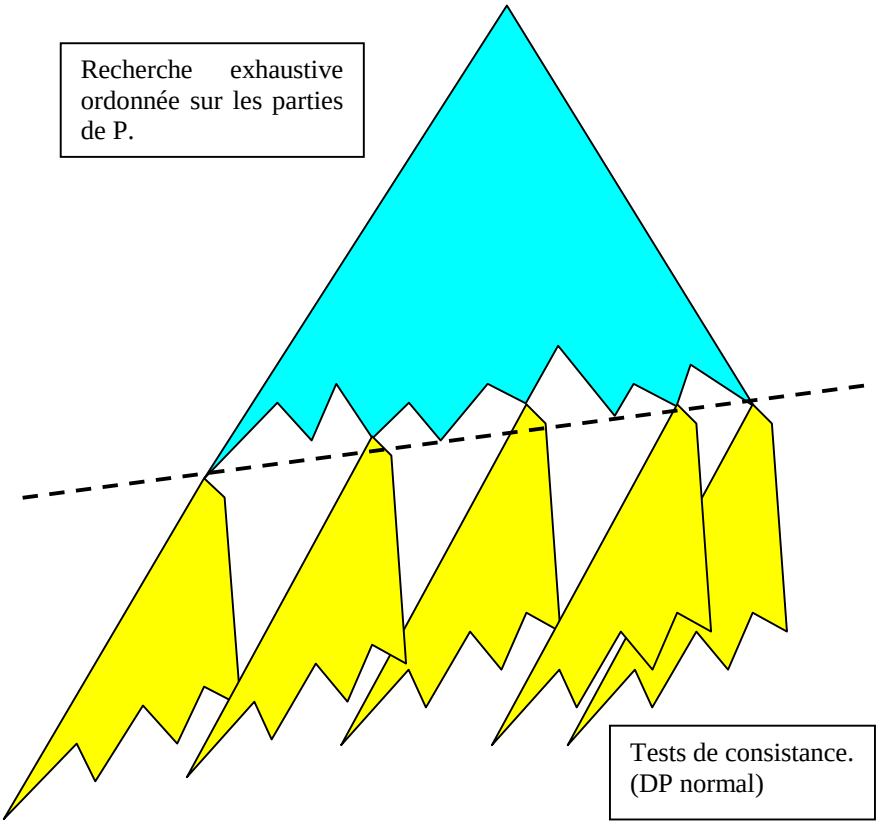


On peut prendre par exemple $B_1 = \{\neg a,b,c\}$ et $B_2=\{a,\neg b,\neg c\}$. Ces deux branches se rejoignent au symbole de branchement a.

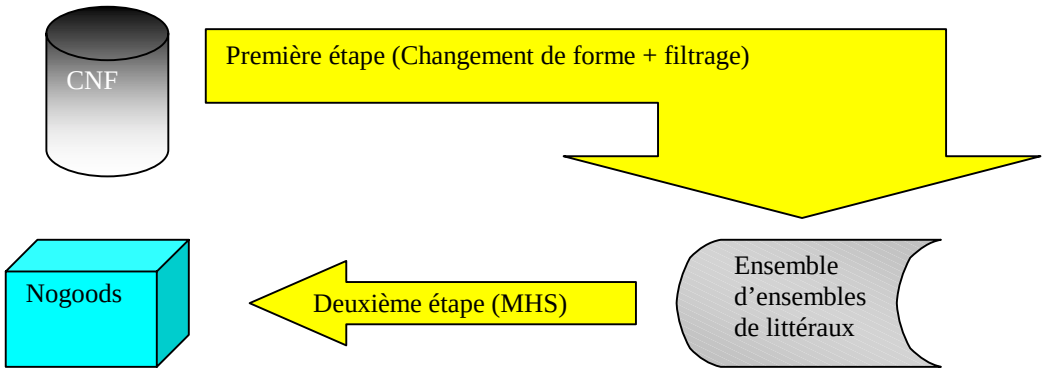
En conséquence, le premier impliquant $M^{\wedge}$ trouvé aura forcément une restriction à P minimale. Il nous faut maintenant garantir que d'autres impliquants M' tels que $(M \cap P) \subset (M' \cap P)$ ne seront pas produits. Or ce sont des impliquants de la base. Nous allons donc transformer la base pour que ces impliquants ne soient plus des impliquants de la nouvelle base. Nous allons rajouter dans la base une clause qui sera falsifiée par tous ces impliquants : $(M \cap P)^{\neg^{\vee}}$. Ainsi, le prochain impliquant trouvé produira lui aussi une restriction à P minimale pour l'inclusion, donc une nouvelle clause sera ajoutée dans la base, et ainsi de suite … A la fin du processus, nous aurons ajouté dans la base exactement une clause par impliquant trouvé. Chacune de ces clauses étant exactement la négation de la restriction à P de l'impliquant qui l'a produite. Dans la pratique, nous récupérerons sous cette forme le résultat de nos calculs (forme CNF au lieu de DNF).

Comme notre base de clauses ne contient pas seulement des variables hypothèses, ce procédé doit être modifié : lorsque tous les littéraux hypothèses d'une base ont une valeur de vérité, il nous importe peu de savoir exactement quelle assignation des littéraux non-hypothèses résulte d'un impliquant de la base (nous voulons uniquement connaître la consistance de la base restante). Nous allons donc développer un algorithme énumératif de type DP qui prend en compte cette remarque. Nous choisirons d'abord les symboles de H pour effectuer les branchements (pour obtenir une énumération selon un ordre compatible avec l'inclusion restreinte à H). Lorsque la base ne contiendra plus de symboles de H non affectés, on cherchera simplement la consistance de la base restante avec un algorithme de SAT habituel (de type DP, bien sur !).

C'est ce que nous représentons sur le schéma suivant :



Par rapport au schéma de principe de la section précédente, nous effectuons donc deux opérations en même temps (le changement de forme et le filtrage), ce qui correspond au schéma suivant :



Voici donc notre démarche expérimentale. Nous avons implanté cet algorithme et avons pu en vérifier le comportement. Mais la signification des ensembles que nous manipulions nous a longtemps échappé. Nous avons tout d'abord exprimé nos résultats en termes de « modèles préférés » [Castell, et al. 1996], car la première application de nos résultats dans le cadre du raisonnement non monotone fut le calcul des modèles <P;Z>-minimaux de Lifschitz [Lifschitz 1985] et son utilisation dans le *raisonnement en monde clos (« Closed World Reasoning »)*, puis en « modèles P-restreints » [Castell, et al. 1998] car cette notion capturait mieux les propriétés logiques des ensembles calculés. La meilleure interprétation nous semble être la notion d'impliquant P-restreint : un modèle étant un impliquant particulier, il n'y a pas d'autre changement dans cette dernière terminologie, outre le nom, qu'une volonté de situer nos travaux dans le cadre général du calcul d'impliquants/impliqués lorsque l'on a certaines propriétés sur les impliquants/impliqués recherchés.

2 Impliquants P-restreints premiers

Une bonne partie des résultats que nous présentons ici sont issues de [Castell, et al. 1998].

2.1 Présentation

Définition 35 (impliquant P-restreint, impliquant P-restreint premier)
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble de littéraux. Soit $E \subseteq P$. $E^\wedge$ est appelé un impliquant *P*-restreint de α ssi $\exists$ M impliquant de α tel que $M \cap P = E$. On dira de plus que $E^\wedge$ est un impliquant *P*-restreint premier de α ssi $\nexists$ $E'^\wedge$ impliquant *P*-restreint de α tel que $E' \subset E$.

Exemple : *BC = {A a, B b, A B }. P={¬A,B,¬B}.* Les impliquants de *BC* sont *{{¬A,¬B}^,{¬A, b}^,{¬B, a}^}. {¬A,¬B}^, {¬A}^* et *{¬B}^* sont des impliquants P-restreints de *BC*. *{¬A}^* et *{¬B}^* sont des impliquants P-restreints premiers de *BC*.

> Remarque : un impliquant *P*-restreint de α n'est pas un forcément un impliquant de α. Ici, seul *{¬A,¬B}^* est un impliquant. C'est même un *P*-impliquant de *BC*.
>
> Dans la suite du document, nous noterons *IRP(α,P)* l'ensemble des impliquants P-restreints premiers de α et par *IP(α,P)* l'ensemble des P-impliquants premiers de α.

Dans le cadre général, le calcul d'impliquants P-restreints est difficile (voir les travaux de Thierry Castell par exemple [Castell 1997]). Nous sommes motivés par le calcul d'impliqués premiers particuliers (les nogoods), restreints à un ensemble de littéraux consistants (*H* ou $H^\neg$). La restriction de nos travaux à un ensemble *P* consistant nous permet d'avoir la propriété suivante (cf. contre exemple page 71) :

Proposition 19
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. Soit $E \subseteq P$. $E^\wedge$ est un impliquant P-restreint de α ssi $(E \cup (P \setminus E)^\neg)^\wedge$ est consistant avec α.

Preuve :
) $E^\wedge$ est un impliquant P-restreint de α donc $\exists$ $M^\wedge$ un impliquant de α tel que $E = M \cap P$. Montrons que $M^\wedge \wedge (E \cup (P \setminus E)^\neg)^\wedge$ est consistant.
Supposons $M^\wedge \wedge (E \cup (P \setminus E)^\neg)^\wedge$ inconsistant. Donc $\exists\, l \in M \mid \neg l \in E \cup (P \setminus E)^\neg$. Donc $l \in M \cap P = E$ ou $l \in M \cap P^\neg$. Supposons que $l \in E$. Alors $\neg l \notin E$ car sinon E serait inconsistant, donc comme $E \subseteq P$, P serait aussi inconsistant. Donc $\neg l \in (P \setminus E)^\neg$. Donc $l \in (P \setminus E)$. Donc $l \notin E$. Contradiction. Donc $l \in M \cap P^\neg$. Donc $\neg l \notin P^\neg$ (sinon P serait inconsistant) donc $\neg l \in E \subseteq M$. Donc $l \in M$ et $\neg l \in M$. Contradiction. Donc $M^\wedge \wedge (E \cup (P \setminus E)^\neg)^\wedge$ est consistant. Soit $M'^\wedge$ un modèle de $M^\wedge \wedge (E \cup (P \setminus E)^\neg)^\wedge$. $M'^\wedge$ étant un modèle de $M^\wedge$, c'est aussi un modèle de α.
⇐) $(E \cup (P \setminus E)^\neg)^\wedge$ consistant avec α donc $\exists\, M^\wedge$ un modèle de α tel que $(E \cup (P \setminus E)^\neg) \subseteq M$. Soit $R = M \setminus (E \cup (P \setminus E)^\neg)$. $M \cap P = ((E \cup (P \setminus E)^\neg) \cup R) \cap P = ((E \cup (P \setminus E)^\neg) \cap P) \cup (R \cap P) = E \cup (R \cap P)$. Supposons que $R \cap P$ ne soit pas vide. Donc $\exists x \in R \cap P$ donc $x \in P$. Or $x \in R \subseteq M$ donc $x \in M \cap P = E$ et $x \notin (E \cup (P \setminus E)^\neg)$. $x \notin E$. Contradiction. Donc $M \cap P = E \cup (R \cap P) = E$. $E^\wedge$ est donc un impliquant P-restreint de α.

Ce résultat est primordial pour nos travaux. Il signifie que si tous les littéraux de *P* sont satisfaits (ensemble *E*) ou falsifiés (ensemble $(P \setminus E)^\neg$), alors il suffit de tester la consistance de la base restante pour obtenir un impliquant P-restreint. Voilà pourquoi nous effectuons d'abord les branchements sur les variables de *P*, puis quand toutes ces variables ont une valeur de vérité, nous pouvons utiliser un algorithme de satisfaction classique pour tester la consistance de la base.

La proposition suivante découle directement de la précédente, et nous sera très utile dans les prochaines démonstrations :

Proposition 20
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. Soit $E \subseteq P$. Si $E^\wedge$ est un impliquant P-restreint de α alors $\exists\, M^\wedge$ un *modèle* de α tel que $M \cap P = E$.

Preuve :
D'après le résultat précédent si $E^\wedge$ est un impliquant P-restreint de α alors $\exists N^\wedge$ un modèle de $\alpha \wedge (E \cup (P \setminus E)^\neg)^\wedge$. Or $N^\wedge$ est un modèle de $(E \cup (P \setminus E)^\neg)^\wedge$ donc $(E \cup (P \setminus E)^\neg) \subseteq N$. Posons $N = E \cup (P \setminus E)^\neg \cup R$. Donc $N \cap P = (E \cap P) \cup ((P \setminus E)^\neg \cap P) \cup (R \cap P) = E \cup (R \cap P)$. On montre de la même manière que précédemment que $R \cap P$ est vide. Donc il existe bien un modèle $N^\wedge$ de α tel que $N \cap P = E$.

Nous allons maintenant présenter un ensemble de résultats dans le but de justifier notre méthode de calcul de P-impliqués premiers par deux appels successifs au même algorithme. Ils sont valables pour des formules quelconques mais nous utiliserons des bases de clauses pour illustrer nos exemples. Afin de mieux comprendre leur utilité, imaginons que nous soyons capables de calculer les impliquants P-restreints premiers d'une formule α (c'est à dire déterminer l'ensemble $IRP(\alpha,P)$). Notre but est de calculer les $P^{\neg}$-impliqués premiers de cette formule (ses nogoods par exemple). Comment pouvons-nous le faire ?

La proposition suivante montre qu'une formule α peut être « encadrée logiquement » par ses P-impliquants et ses impliquants P-restreints premiers.

Proposition 21 relation P-impliquants/impliquants P-restreints
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. $IP(\alpha,P)^{\vee} \models \alpha \models IRP(\alpha,P)^{\vee}$.

> **Preuve :**
> $IP(\alpha)^{\vee} \models \alpha$ cette première relation est triviale car les éléments de $IP(\alpha,P)$ sont des impliquants de BC.
> $\alpha \models IRP(\alpha,P)^{\vee}$. Tout modèle M de α définit un impliquant P-restreint : $M \cap P$. De deux choses l'une. Soit $M \cap P$ est minimal auquel cas il se trouve dans $IRP(\alpha,P)$, soit il contient un élément de $IRP(\alpha,P)$. Dans les deux cas, ce modèle satisfait $IRP(\alpha)^{\vee}$.

Exemple : Soit $BC = \{A \quad c, B\ c \quad\}$ et $P = \{\neg A, \neg B\}$. Les impliquants de BC sont $\{\neg A \wedge \neg B, \neg A \wedge \neg c, \neg B \wedge c\}$. Donc $IP(BC,P) = \{\neg A \wedge \neg B\}$. $IRP(BC,P) = \{\neg A, \neg B\}$. On a bien $\neg A \wedge \neg B \models BC^{\wedge} \models \neg A \vee \neg B$

Le résultat suivant nous indique que les P-impliqués d'une formule peuvent être obtenus à partir de l'ensemble des impliquants P-restreints premiers de cette formule. Ce résultat est primordial car il indique que la seule connaissance des impliquants P-restreints premiers d'une formule permet de calculer ses P-impliqués. Il est à la base de toutes les extensions de nos travaux dans le cadre du raisonnement non monotone, car nous travaillerons le plus souvent sur l'ensemble de impliquants P-restreints premiers d'une formule, et non directement sur la formule.

Proposition 22
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. $\forall E \subseteq P$, $\alpha \models E^{\vee}$ *ssi* $IRP(\alpha,P)^{\vee} \models E^{\vee}$.

> **Preuve :**
> $\Leftarrow$) D'après la proposition précédente, on a $\alpha \models IRP(\alpha,P)^{\vee}$. Donc on a $\alpha \models IRP(\alpha,P)^{\vee} \models E^{\vee}$. Donc $\alpha \models E^{\vee}$.
>) $\alpha \models E^{\vee}$ par hypothèse. Supposons que $IRP(\alpha,P)^{\vee} \not\models E^{\vee}$. Donc $\exists F^{\wedge} \in IRP(\alpha,P)$ tel que $F \cap E = \varnothing$. D'après la définition d'un impliquant P-restreint, il existe donc $M^{\wedge}$ impliquant de α tel que $F = M \cap P$. D'après notre hypothèse, nous avons donc $M^{\wedge} \models \alpha \models E^{\vee}$. Or $E \cap M = E \cap F$ car $E \subseteq P$. Donc $E \cap M = \varnothing$. Donc $M^{\wedge} \not\models E^{\vee}$. Contradiction.

Si nous reprenons notre exemple, le seul P-impliqué de BC est $\neg A \vee \neg B$, et nous avons bien $IRP(BC,P)^{\vee} \models \neg A \vee \neg B$.

Nous présentons maintenant un résultat plus intuitif. Tout P-impliquant d'une formule est un P-impliquant des P-impliquants premiers de cette formule. Ce résultat est à la base du raisonnement par impliquants/impliqués premiers.

Proposition 23
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. $\forall E \subseteq P$, $E^{\wedge} \models \alpha$ *ssi* $E^{\wedge} \models IP(\alpha,P)^{\vee}$.

> **Preuve :**
> $\Leftarrow$) D'après la Proposition 21, on a $IP(\alpha,P)^{\vee} \models \alpha$. Donc on a $E^{\wedge} \models IP(\alpha,P)^{\vee} \models \alpha$. Donc $E^{\wedge} \models \alpha$.
>) Posons $E^{\wedge} \models \alpha$. Comme $E \subseteq P$, $E^{\wedge}$ est un P-impliquant de α. Donc $\exists F^{\wedge} \in IP(\alpha,P)$ tel que $F \subseteq E$. Donc $E^{\wedge} \models F^{\wedge}$. Donc $E^{\wedge} \models IP(\alpha,P)^{\vee}$.

Dans notre exemple, nous avions un seul P-impliquant $\neg A \wedge \neg B$, qui est donc premier, qui est donc trivialement sont propre P-impliquant.

Nous présentons maintenant deux résultats spécialement adaptés aux formules DNF, c'est à dire adaptés au traitement d'une couverture d'impliquants d'une formule CNF par exemple. Le premier permet d'obtenir par simple

filtrage des cubes de la DNF les impliquants P-restreints premiers de cette DNF. C'est de cette façon que l'on agit « à la main ». Le second souligne une propriété de la disjonction des impliquants P-restreints d'une formule en particulier, des DNF contenant uniquement des littéraux purs plus généralement.

Proposition 24
Soit $\alpha \in FP_S$ une DNF. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. L'ensemble des impliquants P-restreints premiers de $\alpha \equiv E_1^\wedge \vee E_2^\wedge \vee \ldots \vee E_n^\wedge$ est l'ensemble $IRP(\alpha,P)=min_\subset\{(E_1\cap P)^\wedge, (E_2\cap P)^\wedge,\ldots,(E_n\cap P)^\wedge\}$ où $min_\subset E$ dénote l'ensemble des cubes de E minimaux pour l'inclusion.

> **Preuve :**
> Appelons R l'ensemble $min_\subset\{(E_1\cap P)^\wedge, (E_2\cap P)^\wedge,\ldots,(E_n\cap P)^\wedge\}$. Chaque $E_i^\wedge$ étant un impliquant de α, chaque $(E_i\cap P)^\wedge$ est donc un impliquant P-restreint de α. R est donc constitué d'impliquants P-restreints. Supposons qu'il existe $F^\wedge$ impliquant P-restreint premier de α qui ne soit pas contenu dans R. Donc il existe un modèle $M^\wedge$ de α tel que $M\cap P=F$. On a deux possibilités :
>
> - soit $\exists i \mid Ei\cap P \subset F$ mais dans ce cas $F^\wedge$ ne peut pas être premier.
> - soit il existe au moins un littéral li de chaque $Ei\cap P$ qui n'est pas dans F. Ces li ne se trouvent pas non plus dans M (sinon ils seraient dans $M\cap P=F$). Donc $M^\wedge \not\models \alpha$. Contradiction.
>
> Donc R contient des impliquants P-restreints, et parmi eux tous les impliquants P-restreints premiers. Comme on ne garde que ceux minimaux pour la sous-sommation, R contient exactement les impliquants P-restreints premiers de α.

A l'aide de ce résultat, il est donc très facile (modulo les tests de sous sommation) d'obtenir l'ensemble des impliquants P-restreints premiers d'une formule DNF.

Dans notre exemple, nous avons $BC^\wedge \equiv (\neg A\wedge\neg B) \vee (\neg A\wedge\neg c) \vee (\neg B\wedge c) \equiv \{\{\neg A,\neg B\}^\wedge,\{\neg A,\neg c\}^\wedge,\{\neg B,c\}^\wedge\}^\vee$. Donc $IRP(BC,P) = min_\subset\{\{\neg A,\neg B\}^\wedge,\{\neg A\}^\wedge,\{\neg B\}^\wedge\} = \{\neg A,\neg B\}$.

Si maintenant cette DNF est constituée uniquement de littéraux (purs) de P, alors les notions d'impliquant, de P-impliquant et d'impliquant P-restreint sont équivalentes (on dit aussi que la formule est indépendante de $P^\neg$).

Corollaire 1
Soit $\alpha \in FP_S$ une DNF. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. Si tous les impliquants de α sont des P-impliquants, alors $IP(\alpha,P)^\vee \equiv \alpha \equiv IRP(\alpha,P)^\vee$.

> **Preuve :** immédiate car pour tout impliquant $E^\wedge$, $E\subseteq P$ ($E^\wedge$ est un P-impliquant) donc $E\cap P=E$ ($E^\wedge$ est aussi un impliquant P-restreint).

> Ce corollaire est très important : c'est grâce à lui que le calcul de « minimal hitting set » peut être effectué à l'aide de notre algorithme. En effet, lorsque tous les littéraux d'une base de clauses sont purs, si l'on prend P égal à l'ensemble de ces littéraux purs, il est équivalent de calculer des P-impliquants premiers ou des impliquants P-restreints premiers.

Nous allons maintenant introduire la notion de réduction d'une formule pour un sous-langage. Nous devons noter que cette notion correspond à la notion de « forget » dans [Marquis 1999] qui généralise la notion d'élimination de variables en logique propositionnelle (elle même généralisée en logique du premier ordre par [Lin/Reiter 1994]. Elle correspond à l'élimination d'une variable quantifiée existentiellement : $(\exists x)\alpha \equiv forget(\alpha,x)$.

Définition 36 (réduction d'une formule)
Soit $\alpha,\beta \in FP_S$ deux NNF. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. β est appelée une réduction de la formule α relativement à P, notée reduc(α,P) ssi :

- β contient uniquement des littéraux de P, et
- les impliquants de β sont les P-impliquants de α.

> Cette formule est unique à l'équivalence logique près, ce qui justifie la notation fonctionnelle reduc(α,P).

En vertu de la définition de reduc, on peut écrire $IP(\alpha,P)^\vee \equiv IP(reduc(\alpha,P),P)^\vee$ et en vertu du Corollaire 1, on obtient $IP(\alpha,P)^\vee \equiv IP(reduc(\alpha,P),P)^\vee \equiv IRP(reduc(\alpha,P),P)^\vee$. On obtient donc la relation suivante entre les P-

impliquants et les impliquants P-restreints d'une formule : les P-impliquants d'une formule sont les impliquants P-restreints de la réduction de cette formule relativement à *P*.

Proposition 25 relation P-impliquant/impliquant P-restreint
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. $IP(\alpha,P)^\vee \equiv IRP(reduc(\alpha,P),P)^\vee$.

L'intérêt de ce résultat est de nous donner une méthode de calcul de P-impliquants premiers à l'aide d'un algorithme de calcul d'impliqués P-restreints premiers.

Si nous prenons notre base *BC = {A c, B c }*, *reduc(BC,P)={A , B }*. Le seul impliquant P-restreint premier de *reduc(BC,P)*, $\neg A \wedge \neg B$ est aussi son seul (P-)impliquant premier.

Nous avons maintenant les moyens d'exprimer la notion de P-impliqué premier à partir de celle d'impliquant P-restreint.

Proposition 26 relation P-impliqué/impliquant P-restreint
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. Soit $E \subseteq P$. $E^\vee$ est un P-impliqué premier de α ssi $E^{\neg\wedge} \in IRP(IRP(\alpha,P)^{\neg\wedge},P^\neg)$ ssi $E^\vee \in IRP(IRP(\alpha,P)^{\neg\wedge},P^\neg)^\neg$.

> **Preuve :**
> *)* $E^\vee$ est un P-impliqué premier de α donc $\alpha \models E^\vee$. En vertu de la Proposition 22, on a $IRP(\alpha,P)^\vee \models E^\vee$. Ce qui conduit à $E^{\neg\wedge} \models IRP(\alpha,P)^{\neg\wedge}$ en prenant la contraposée. On peut alors appliquer la Proposition 23, ce qui donne $E^{\neg\wedge} \models IP(IRP(\alpha,P)^{\neg\wedge},P)^\vee$ (comme E est une partie de P, $E^\neg$ est une partie de $P^\neg$). Par application de la Proposition 25, on obtient $E^{\neg\wedge} \models IRP(reduc(IRP(\alpha,P)^{\neg\wedge},P^\neg)P^\neg)^\vee$, ce qui se simplifie en $E^{\neg\wedge} \models IRP(IRP(\alpha,P)^{\neg\wedge},P^\neg)^\vee$ car $IRP(\alpha,P)^\neg$ contient uniquement des littéraux de $P^\neg$ donc ses impliquants sont ses $P^\neg$-impliquants. Nous avons donc montré que si $E^\vee$ est un P-impliqué de α, alors il $\exists F^\wedge \in IRP(IRP(\alpha,P)^{\neg\wedge},P^\neg)$ tel que $F \subseteq E^\neg$. Comme $E^\vee$ est un P-impliqué premier, on a forcément $E^\neg = F$. En effet, si $F \subset E^\neg$, alors $F^\neg \subset E$ et $F^\neg$ serait un P-impliqué de α (contredisant la primarité de E).
> *⇐)* Nous avons utilisé des équivalences pour prouver).

Reprenons notre exemple. $IRP(BC,P) = \{\neg A, \neg B\}$. $IRP(BC,P)^{\neg\wedge} = A \wedge B$. $IRP(A \wedge B,P^\neg) = \{A \wedge B\}$. $\{A \wedge B\}^\neg = \{\neg A \vee \neg B\}$. $\neg A \vee \neg B$ est bien un P-impliqué de BC !

Cette démarche peut paraître lourde et fastidieuse sur ce petit exemple pour calculer ce P-impliqué premier. Sur des exemples plus conséquents, on peut noter que souvent l'ensemble des impliquants P-restreints premiers d'une formule est de taille inférieure à la formule, et qu'il est très facile d'en obtenir *un* P-impliqué premier.

Nous avons donc montré comment nous pouvons calculer les P-impliqués d'une formule à partir de calculs d'impliquants P-restreints **quand P est un ensemble consistant de littéraux**. Nous renvoyons le lecteur aux travaux de [Castell 1997] pour le cas général.

2.2 Calcul d'impliquants P-restreints premiers

Nous allons maintenant présenter notre algorithme de calcul de *IRP(BC,P)* dans la cas où *BC* est une base de clauses (et *P* un ensemble consistant de littéraux).

Nous avons annoncé dans la section 1.3.2 que nous ajoutons dans la base une clause afin d'éliminer les impliquants P-restreints minimaux.

Seule la falsification d'une de ces clauses ajoutées nous apporte de l'information : la non minimalité de la solution trouvée. Nous allons donc différencier les clauses ajoutées des clauses originales de la base. Nous utilisons pour cela une base de clauses BA (pour Base des clauses Ajoutées), appelée base de minimisation, qui va contenir l'ensemble des clauses ajoutées.

Chercher des impliquants P-restreints premiers de BC∪BA (ce qui revient à ajouter directement les clauses directement dans la base) ou un impliquant P-restreint premier de BC qui ne falsifie pas BA ne change rien au niveau de la complétude du résultat. Par contre, cela oblige à satisfaire des littéraux de $P^\neg$ superflus. Comme BA contient uniquement des littéraux de $P^\neg$ et que le nombre de clauses ajoutées peut être important (plus important que la base de départ), il arrive la plupart du temps qu'un impliquant $M^\wedge$ de *BC* n'est pas un impliquant de *BA*.

Comme satisfaire *BA* consiste à satisfaire des littéraux de $P^{\neg}$, l'impliquant $M'^{\wedge}$ de BC tel que $M \subseteq M'$ et $M'^{\wedge}$ est aussi un impliquant de *BA* a la même restriction à P que M ($M \cap P = M' \cap P$).

Nous pouvons formaliser l'évolution d'une base de minimisation de la manière suivante :

Définition 37 (bases de minimisation)
Soit *BC* une base de clauses. Soit *P* un ensemble consistant de littéraux. Soit m=|IRP(BC,P)|. Soient {BA_1, BA_2, …,BA_m} un ensemble ordonné de base de clauses. {BA_1,BA_2,…,BA_m} sont des bases de minimisations de BC ssi :

1. $\forall i\ |BA_i| = i$;
2. $BA_1 \subset BA_2 \subset \ldots \subset BA_m$.
3. $\forall\ E^{\wedge} \in IRP(BC,P)\ \exists i$ tel que $E^{\neg\vee} \in BA_i$
4. $BC^{\wedge} \wedge BA_m \models \bot$

Il résulte de cette définition que les BAi contiennent uniquement des littéraux purs (les littéraux de $P^{\neg}$) et que BAm contient exactement $IRP(BC,P)^{\neg\vee}$.

Nous cherchons donc des impliquants *M* de *BC* consistants avec *BAi*. La minimalité du résultat est alors garantie par l'ordre dans lequel les interprétations sont énumérées.

Nous cherchons donc des ensembles $E \subseteq P$ tels que :

- $\exists$ M tel que $M^{\wedge} \models BC^{\wedge}$ et $M \cap P = E$,
- $BA_i^{\wedge} \wedge E^{\wedge} \not\models \bot$.

Il faut noter que lorsque l'algorithme termine, $BA = BA_m$, donc contient la négation des impliquants P-restreints premiers.

```
MPL(BC,BA,P,IP)[23]
// Calcule les impliquants P-restreints premiers de BC.
// IP contient les littéraux satisfaits.
// BA est la base de minimisation.
// le résultat retourné est la nouvelle base de minimisation
// Les impliquants P-restreints sont sous forme clausale (niée) dans cette base.
        Si    ∈ BC alors retourner BA Finsi; // BC est inconsistante
        // test de primarité
        Si BA^∧IP^ inconsistant alors retourner BA Finsi ;[24]
        Si BC = ∅ alors  // BC est consistante
                retourner BA ∪ {¬((IP∩P)^)} ;
        Finsi
        l ← LitteralPourSimplifier(BC,BA,P) ;
        Si (l≠null) alors // on peut simplifier BC par l
                Retourner MPL(BC[l],BA,P,IP∪{l})
        Finsi ;
        l ← Choix_Symbole_de_P(BC,P) ;
        Si (l≠null) alors
                SOL ← MPL(BC[¬l],BA,P,IP∪{¬l}) ;
                retourner MPL(BC[l],SOL,P,IP∪{l}}
        Sinon
                Si DP(BC)=vrai alors // BC est consistante
                        retourner BA ∪ {¬((IP∩P)^)}
                Sinon retourner BA
                Finsi
        Finsi
```

Algorithme 4 MPL - Calcul d'impliquants P-restreints premiers

La seule différence entre MPL et des algorithmes de calcul d'une couverture d'impliquants premiers comme DPPI [Schrag 1996] ou DPIC [Mazure/Marquis 1996] concerne le test d'inconsistance entre *BA* et *IP* (test de minimalité) couplé avec une énumération ordonnée des interprétations. La fonction est appelée avec *BA* et *IP* vides pour le calcul des impliquants P-restreints premiers : MPL(BC,∅,P,∅).

[23] Le nom de la procédure, MPL, signifie Modèles Préférés par leurs Littéraux. Voir explication page 70.
[24] Dans l'implantation de l'algorithme, BA est réduite de la même manière que *BC*. Cela nécessite d'avoir un moyen d'ajouter une clause sous sa forme réduite dans une base réduite (Si *BA* = ∅, et que l'on veut ajouter la clause *a b c* alors que *BA* a été réduite par *a*, *b* et *c* sont satisfaits à cet endroit de l'arbre, il faut ajouter la clause nulle dans *BA[a,b,c]*).

La fonction `LittéralPourSimplifier(BC, BA, P)` va permettre de propager les clauses unitaires apparaissant dans *BC* et *BA*. De plus, nous avons trouvé un sous-ensemble des littéraux purs pouvant être propagés sans perdre d'impliquants P-restreints (cf. section V.2 page 147).

Propriété 4 (propagation des littéraux purs dans MPL)
Soit $BC \in FP_S$ une base de clauses. Soit $P \subset L_S$ un ensemble consistant de littéraux. Si $l \in P^{\neg}$ est pur dans *BC* alors $E^{\wedge}$ est un impliquant P-restreint premier de *BC* ssi $E^{\wedge}$ est un impliquant P-restreint premier de *BC[l]*.

Preuve :
) $\exists M^{\wedge} \models BC^{\wedge}$ tel que $M \cap P = E$. Comme $BC[l] \subseteq BC$, $M^{\wedge} \models BC[l]^{\wedge}$. Donc $E^{\wedge}$ est aussi un impliquant P-restreint de *BC[l]*. Il est premier sinon $\exists E' \subset E$ tel que *E'* est un impliquant P-restreint de *BC[l]*. Comme $l \notin P$, *E'* est aussi un impliquant P-restreint de *BC*. Contradiction.
⇐) Comme $l \notin P$, tout impliquant P-restreint de *BC[l]* est un impliquant P-restreint de *BC*. Comme $E^{\wedge}$ est un impliquant P-restreint premier de *BC[l]*, il est aussi premier pour *BC*.

On en déduit la fonction suivante :

```
LittéralPourSimplifier(BC,BA,P)
// retourne un littéral dont la satisfaction permet de simplifier
// la base BC en garantissant que tout impliquant P-restreint premier de la base est
// un impliquant P-restreint premier de la base simplifiée.
Si ∃ l littéral tel que l est une clause unitaire de BC ou BA alors retourner l Finsi ;
Si ∃ l∈ P¬ littéral tel que l est pur dans BC alors retourner l Finsi ;
retourner null
```

La fonction `Choix_Symbole_De_P(BC,P)` retourne le meilleur littéral de P conformément à une heuristique de type MOMS (dans notre implantation, nous utilisons celle utilisée dans TABLEAU, POSIT et SATZ).

Il nous faut maintenant démontrer que notre algorithme est :

- **sain**, c'est à dire que la base BA calculée ne contient que des négations d'impliquants P-restreints premiers,
- **complet**, c'est à dire qu'il calcule tous les impliquants P-restreints premiers.

Preuve :
Cela découle de la Proposition 24, page 63. En effet, à l'aide de ce théorème, nous pouvons calculer l'ensemble des impliquants P-restreints d'une formule à partir d'une couverture d'impliquants de cette formule. Or nous avons noté dans la section 2.2.3 , page 39, qu'un algorithme de type Davis et Putnam permet de calculer une couverture d'impliquants de la base si l'on respecte quelques règles, comme par exemple ne pas utiliser la propagation des littéraux purs, qui préserve la consistance de la base mais pas son équivalence logique.
Comme seuls les impliquants P-restreints premiers nous intéressent, nous utilisons la Propriété 3, page 59, afin de garantir que l'ordre d'énumération des interprétations est compatible avec l'inclusion de leur restriction à P, donc que le premier impliquant trouvé a pour restriction à *P* un impliquant P-restreint premier.
Nous ajoutons alors dans la base *BA* la négation de l'impliquant P-restreint premier trouvé (disons $E^{\wedge}$). On montre que seuls les impliquants P-restreints sous-sommés par $E^{\wedge}$ falsifient cette clause. En effet, pour falsifier cette clause, il faut falsifier chacun de ses littéraux, donc tout impliquant $E'^{\wedge}$ qui falsifie cette clause contient au moins tous les littéraux de *E* ($E \subseteq E'$). Ce mécanisme empêche donc de produire les impliquants P-restreints non premiers (et uniquement eux).
Notons enfin que l'algorithme s'arrête. En effet, à chaque appel récursif de MPL, nous éliminons une variable de la base. Comme le nombre de variables est fini, alors le nombre d'appels récursifs à MPL l'est aussi.

Nous allons maintenant retourner en Bretagne. Nous allons calculer les impliquants P-restreints premiers de BZH'∪OBS, avec P={¬A,¬Y,¬D,¬J,¬P}. Le schéma suivant correspond à la trace de MPL durant ce calcul. Chaque nœud représente un appel à MPL. Les bases associées à chaque nœuds (identifiables par leur numéro) sont données ci-après. Chaque arc est étiqueté par le littéral réduisant la base. Les bases de minimisations sont représentées sous les branches qui les produisent.

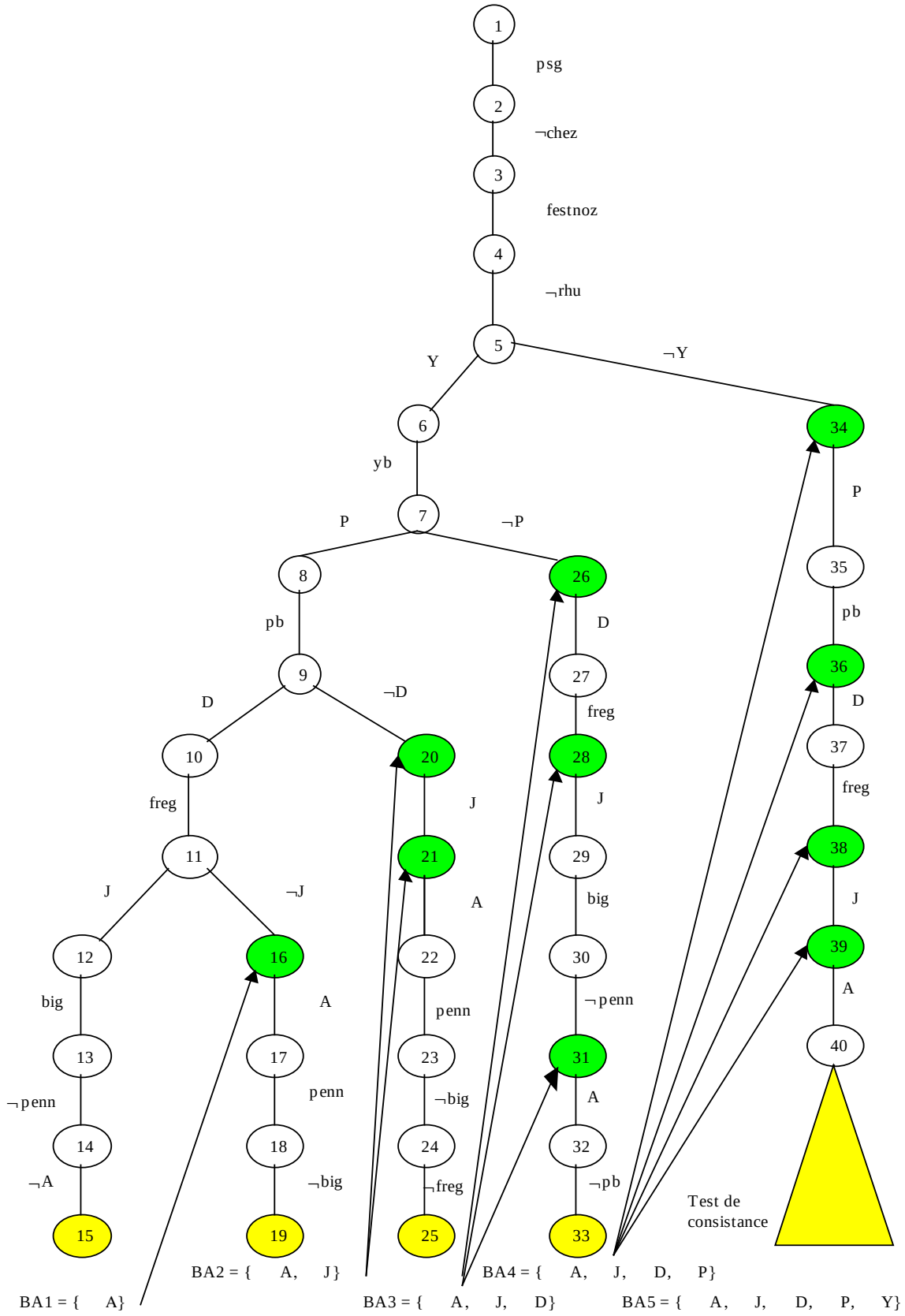


Figure 4 trace de MPL sur BZH'∪OBS

On peut noter qu'il n'y a pas de nœuds aboutissant à des bases inconsistantes car chaque choix produit des clauses unitaires. Remarquons que les branchements se font sur des variables de P, en commençant par des littéraux de $P^{\neg}$. Nous avons noté pour chaque base de minimisation les coupures produites dans l'arbre. Dans cet exemple elles sont nombreuses car les clauses de minimisation sont unitaires (donc on les propage directement). Pour la base n°40, comme aucune variable hypothèse n'est présente dans la base, un test de satisfaction est utilisé. On remarque enfin que BA_5 contient la négation du résultat recherché, ici $IRP(BZH' \cup OBS, H^{\neg}) = \{\neg A, \neg J, \neg D, \neg P, \neg Y\}$.

1. BC = { psg P pb , chez P pb , psg chez , chez Y yb , festnoz Y yb , chez festnoz , pb yb A penn , festnoz D freg , rhu D freg , festnoz rhu , freg yb J big , penn big , festnoz , psg }
2. BC[**psg**] = {P pb , chez P pb , chez , chez Y yb , festnoz Y yb , chez festnoz , pb yb A penn , festnoz D freg , rhu D freg , festnoz rhu , freg yb J big , penn big , festnoz }
3. BC[**psg, ¬chez**] = {P pb , festnoz Y yb , pb yb A penn , festnoz D freg , rhu D freg , festnoz rhu , freg yb J big , penn big , festnoz }
4. BC[**psg, ¬chez, festnoz**] = {P pb , Y yb , pb yb A penn , D freg , rhu D freg , rhu , freg yb J big , penn big }
5. BC[**psg, ¬chez, festnoz, ¬rhu**] = {P pb , Y yb , pb yb A penn , D freg , freg yb J big , penn big }
6. BC[**psg, ¬chez, festnoz, ¬rhu, Y**] = {P pb , yb , pb yb A penn , D freg , freg yb J big , penn big }
7. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb**] = {P pb , pb A penn , D freg , freg J big , penn big }
8. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P**] = { pb , pb A penn , D freg , freg J big , penn big }
9. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb**] = {A penn , D freg , freg J big , penn big }
10. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D**] = {A penn , freg , freg J big , penn big }
11. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg**] = {A penn , J big , penn big }
12. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, J**] = {A penn , big , penn big }
13. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, J, big**] = {A penn , penn }
14. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, J, big, ¬penn**] = {A }
15. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, J, big, ¬penn, ¬A**] = ∅

    BA_1 = { A}
16. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, ¬J**] = {A penn , penn big }
17. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, ¬J, A**] = { penn , penn big }
18. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, ¬J, A, penn**] = {big }
19. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, D, freg, ¬J, A, penn, ¬big**] = ∅

    BA_2 = { A, J}
20. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D**] = {A penn , J freg big , big penn }
21. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D, J**] = {A penn , freg big , big penn }
22. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D, J, A**] = { penn , freg big , big penn }
23. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D, J, A, penn**] = {freg big , big }
24. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D, J, A, penn, ¬big**] = {freg }
25. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, P, pb, ¬D, J, A, penn, ¬big, ¬freg**] = ∅

    BA_3 = { A, J, D}
26. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P**] = {A pb penn , D freg , freg J big , big penn }
27. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D**] = {A pb penn , freg , freg J big , big penn }
28. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg**] = {A pb penn , J big , big penn }
29. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg, J**] = {A pb penn , big , big penn }
30. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg, J, big**] = {A pb penn , penn }
31. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg, J, big, ¬penn**] = {A pb }
32. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg, J, big, ¬penn, A**] = {pb }
33. BC[**psg, ¬chez, festnoz, ¬rhu, Y, yb, ¬P, D, freg, J, big, ¬penn, A, ¬pb**] = ∅

    BA_4 = { A, J, D, P}
34. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y**] = {P pb , pb A yb penn , D freg , J freg yb big , penn big }
35. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P**] = { pb , pb A yb penn , D freg , J freg yb big , penn big }
36. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P, pb**] = {A yb penn , D freg , J freg yb big , penn big }

37. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P, pb, D**] = {A yb penn , freg , J freg yb big , penn big }
38. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P, pb, D, freg**] = {A yb penn , J yb big , penn big }
39. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P, pb, D, freg, J**] = {A yb penn , yb big , penn big }
40. BC[**psg, ¬chez, festnoz, ¬rhu, ¬Y, P, pb, D, freg, J, A**] = {yb penn , yb big , penn big }

BA_5 = { A, J, D, P, Y}

2.3 Calcul de P-impliquants premiers

Nous pouvons maintenant donner un algorithme de calcul de P-impliquants premiers basé sur l'algorithme MPL. En effet, un P-impliquant de *BC* (s'il existe) satisfait chaque clause de *BC* par un littéral de *P*. Pour obtenir un *P*-impliquant premier de *BC*, il suffit donc de calculer un « minimal hitting set » de l'ensemble des clauses de *BC* restreintes à *P*.

Propriété 5
Soit $BC \subset FP_S$ une base de clauses. Soit $P \subset L_S$ un ensemble consistant de littéraux. Soit $BC' = \{C \cap P \mid C^{\vee} \in BC\}$. *BC'* est une réduction de *BC* relativement à *P*.

Preuve :
Nous pouvons noter que *BC'* contient uniquement des littéraux de *P*. Il nous faut maintenant démontrer que tout impliquant de *BC'* est un *P*-impliquant de *BC*. Soit *M* un impliquant de *BC'*. *M* satisfait chaque clause de *BC'*. Comme chaque clause de *BC* est sous-sommée par une clause de *BC'*, *M* satisfait chaque clause de *BC* donc *M* est un impliquant de *BC*. Comme $M \subseteq P$, *M* est un *P*-impliquant de *BC*.
Montrons maintenant que tout P-impliquant de *BC* est un impliquant de *BC'*. Soit $M^{\wedge}$ un P-impliquant de *BC*. $M^{\wedge}$ satisfait toutes les clauses de *BC*. Comme $M \subseteq P$, $M^{\wedge}$ satisfait chaque clause de *BC* sur sa restriction à *P*, donc satisfait toutes les clauses de *BC'*. Donc $M^{\wedge}$ est bien un impliquant de *BC'*.

En vertu de la Proposition 25, nous pouvons donc proposer l'algorithme suivant de calcul de P-impliquants premiers (avec P un ensemble consistant de littéraux) :

```
PIPQNTP(BC,P)[25]
// BC une base de clauses
// P un ensemble consistant de littéraux
// retourne les P-impliquants premiers de BC
1.  BC' ← { (C∩P)^∨ | C^∨ ∈ BC} ;
2.  INTER ← MPL(BC',∅,P,∅) ; // INTER est une base intermédiaire
3.  Retourner INTER^¬ ;
```

Algorithme 5 PIPQNTP - Calcul de P-impliquants premiers

2.4 Calcul de P-impliqués premiers

En vertu de la Proposition 26, nous pouvons présenter de même un algorithme de calcul de P-impliqués premiers (P consistant) basé sur MPL.

```
PIP(BC,P)
// BC est une base de clauses
// P est un ensemble consistant de littéraux
// BA contient les P-impliqués premiers de BC
1.  INTER ← MPL(BC,∅,P,∅) ; // INTER est une base intermédiaire
2.  Retourner MPL(INTER,∅,P^¬ ,∅) ;
```

Algorithme 6 PIP - Calcul de P-impliqués premiers

3 Formalisations alternatives

Nous avons déjà évoqué dans ce chapitre que nous nous sommes longtemps interrogés sur la nature des ensembles que nous manipulions. Nous présentons dans cette section les différentes formalisations que nous avons utilisées pour définir ce que nous appelons maintenant impliquant P-restreint.

[25] Le nom de l'algorithme n'est pas très simple mais il est difficile de différencier en quelques lettres P-impliquants premiers et P-impliqués premiers.

Une application de notre travail nous a conduit au raisonnement en monde clos (CWR, *« Closed World Reasoning »).* La sémantique des différentes politiques de CWR est souvent présentée en termes de modèles minimaux de Herbrand. C'est pourquoi la première formalisation que nous avons présentée est basée sur la notion très proche de « modèle préféré » [Castell, et al. 1996].

3.1 Des modèles préférés …

Définition 38 (D-interprétation)
Soit $D \subseteq S$ un ensemble de symboles propositionnels. Soit $\alpha \in FP_S$. On appelle *D*-interprétation une fonction de *D* vers *{vrai,faux}*. C'est la restriction d'une interprétation du langage aux seuls symboles de *D*. On notera I_D l'ensemble des *D*-interprétations. Une D-interprétation *I D-satisfait* une formule α ssi $\exists M$ un modèle de α tel que $I \subseteq M$. Une D-interprétation qui D-satisfait une formule α est appelée *D-modèle* de α.

Dans la suite du document, nous représenterons une D-interprétation par l'ensemble des littéraux qu'elle satisfait.

Nous allons maintenant ajouter à la logique classique une relation de préférence entre les interprétations, et plus précisément sur les *D*-interprétations. Soit $\propto$ un ordre partiel strict entre les *D*-interprétations. Si $D=S$ alors nous retrouvons la logique préférentielle classique de [Shoham 1987].
Soient $M_1, M_2 \in I_D$. $M_1 \propto M_2$ signifie que la *D*-interprétation M_2 est strictement préférée à la *D*-interprétation M_1. Nous pouvons alors définir une nouvelle sémantique pour notre logique :

- les *D*-interprétations préférées d'un ensemble *F* de *D*-interprétations sont les éléments maximaux de *F* pour $\propto$.
- Une *D*-interprétation *M* D-satisfait préférentiellement la formule α ($M \models^D \alpha$) ssi *M* D-satisfait α et *M* est maximale pour $\propto$ dans l'ensemble des *D*-interprétations qui D-satisfont α. *M* est appelée un *D*-modèle préféré de α.

Définissons par exemple un ordre partiel strict entre nos *D*-interprétations. Cet ordre partiel sera syntaxique : nous allons préférer les modèles contenant le plus de littéraux négatifs (cf. modèles minimaux de Herbrand), ou plus généralement contenant un maximum de littéraux « préférés ».

Définition 39 (D-interprétation préférée)
Soit $D \subseteq S$. Soit *(Pref,Npref)* une partition des littéraux de L_D telle que *Pref* est une *D*-interprétation (donc *Npref* l'est aussi). *Pref* est la *D*-interprétation maximale pour notre relation $\propto$ dans I_D. *Npref* est la *D*-interprétation minimale pour $\propto$ dans I_D. Soient M_1 et M_2 deux *D*-interprétations. M_2 est préférée à M_1 (noté $M_1 \propto M_2$) ssi $M_2 \cap Npref \subset M_1 \cap Npref$ ssi $M_1 \cap Pref \subset M_2 \cap Pref$.

Littéralement, cela signifie que M_2 est préférée à M_1 ssi la partie non préférée de M_2 est incluse dans celle de M_1.

On retrouve dans cette définition la restriction à un ensemble consistant de littéraux. Par contre, nous travaillons ici sur une D-interprétation et pas seulement sur sa partie non préférée.

On peut montrer que $E^\wedge$ est un impliquant P-restreint premier de α ssi $(E \cup (P \backslash E)^\neg)$ est un D-modèle préféré de α pour $D=symboles(P)$ et $Pref=P^\neg$.

Lemme 2 (équivalence entre les impliquants P-restreints et les D-modèles)
Soit $\alpha \in FP_S$. Soit $P \subseteq L_S$ un ensemble consistant de littéraux. Soit *D* l'ensemble des symboles de *P*. Soit $E \subseteq P$. $E^\wedge$ est un impliquant P-restreint de α ssi $(E \cup (P \backslash E)^\neg)$ est un D-modèle de α.

Preuve :
) Soit $E^\wedge$ un impliquant P-restreint de α. D'après la Proposition 20 page 61, $\exists$ M un modèle de α tel que $(E \cup (P \backslash E)^\neg) \subseteq M$. Comme *D* est l'ensemble des symboles de *P*, $(E \cup (P \backslash E)^\neg)$ est une assignation de tous les symboles de *P*, donc c'est un D-modèle.
⇐) Soit *M* un D-modèle de α. Alors $\exists M'$ un modèle de α tel que $M \subseteq M'$. Soit $E = M \cap P$. On montre facilement que $M = (E \cup (P \backslash E)^\neg)$. On a donc $E = M \cap P \subseteq M' \cap P$. Supposons $E \subset M' \cap P$. Alors $\exists p \in P$ tel que $p \in M'$ et $p \notin M$. Comme *M* est une D-interprétation on a donc $\neg p \in M$. Donc $p \in M'$ et $\neg p \in M$. Contradiction. Donc $M' \cap P = E$. Donc *E* est un impliquant P-restreint de α.

Proposition 27 (équivalence entre un impliquant P-restreint premier et un D-modèle préféré)
Soit $\alpha \in FPS$. Soit $P \subseteq L_S$ un ensemble consistant de littéraux. Soit D l'ensemble des symboles de P. Soit $E \subseteq P$. $E^{\wedge}$ est un P-impliquant premier de α ssi $(E \cup (P \backslash E)^{\neg})$ est un D-modèle préféré de α pour $Pref = P^{\neg}$ et $Npref = P$.

Preuve :

) D'après le lemme précédent, $(E \cup (P \backslash E)^{\neg})$ est un D-modèle de α. Nous devons montrer que c'est aussi un D-modèle préféré. Supposons que ce soit faux. Donc $\exists$ M un D-modèle de α tel que $(E \cup (P \backslash E)^{\neg}) \propto M$, c'est à dire $M \cap Npref \subset (E \cup (P \backslash E)^{\neg}) \cap Npref$. Or si nous posons $Npref = P$ et $Pref = P^{\neg}$ (ce qui respecte les conditions sur Pref et Npref), nous obtenons $E' = M \cap P \subset E$. Donc $\exists$ $E' \subset E$ tel que $E'^{\wedge}$ est un impliquant P-restreint de α. Cela contredit le fait que $E^{\wedge}$ soit premier.
$\Leftarrow$) Soit M un D-modèle préféré de α. $M \cap P$ est un impliquant P-restreint de α. Supposons que $M \cap P$ ne soit pas premier. Donc $\exists$ M' un modèle de α tel que $M' \cap P \subset M \cap P$. Or $(M' \cap P) \cup (M' \cap P^{\neg})$ est un D-modèle de α (lemme précédent). Donc $M \propto (M' \cap P) \cup (M' \cap P^{\neg})$. Donc M n'est pas un D-modèle préféré de α. Contradiction.

Le terme « préféré » nous a été reproché lorsque nous avons publié nos travaux dans [Castell, et al. 1996] car nous utilisons une relation de préférence « syntaxique » très spéciale, qui ne correspond pas vraiment à la notion de préférence usuelle en raisonnement non monotone (qui est plutôt sémantique). C'est pourquoi nous avons changé la terminologie dans [Le-Berre 1996].

3.2 … aux P-sous-modèles possibles …

La notion de sous-modèle restreint est exactement notre actuelle notion d'impliquant P-restreint.

Définition 40 (P-sous-modèle possible)
Soit α une formule de FP_S. Soit P un ensemble consistant de littéraux. On dira que $p \subseteq P$ est un P-sous-modèle possible de α ssi il existe un modèle de α dont p est la restriction à P : $\exists M$ un modèle de α tel que $M \cap P = p$.

Nous définissions alors la notion de P-impliquant restreint :

Définition 41 (P-impliquants restreints)
Soit α une formule de FP_S. Soit P un ensemble consistant de littéraux. On dira que $p \subseteq P$ est un P-impliquant restreint de α ssi toute interprétation contenant p est un modèle de α

Ces notions étaient maladroites car elles redéfinissaient une notion déjà existante : la notion de P-impliquants, et ne réussissait pas à donner l'intuition de la sémantique associée à la notion de P-sous-modèles possibles. La prochaine étape sera la bonne.

3.3 … jusqu'aux modèles P-restreints

Nous présentons enfin dans [Castell, et al. 1998] une sémantique proche de celle présentée dans cette thèse : la notion de modèle P-restreint.

Définition 42 (Modèle P-restreint)
Soit α une formule de FP_S. Soit P un ensemble de littéraux. On dira que $p \subseteq P$ est modèle P-restreint de α ssi il existe un modèle de α dont p est la restriction à P : $\exists M$ modèle de α tel que $M \cap P = p$.

On peut montrer que si l'on prend pour P un ensemble consistant de littéraux, alors la notion de modèle P-restreint et celle d'impliquant P-restreint sont équivalentes.

En effet, Supposons que E soit un impliquant P-restreint de α. Alors $\exists$ I un impliquant de α tel que $I \cap P = E$. Nous pouvons maintenant construire un modèle M de α tel que $M \cap P = E$ en complétant I par un ensemble consistant de littéraux n'apparaissant pas dans P (Si P est inconsistant, c'est à dire contient deux littéraux complémentaires, il n'est pas toujours possible de construire un tel modèle). Donc E est aussi un modèle P-restreint de α.

Contre exemple : Soit $BC = \{ \quad A, \quad c \}$ avec $S = \{A,B,c\}$ et $P = \{A,B,\neg B\}$. BC admet deux modèles , $A \wedge \neg B \wedge c$ et $A \wedge B \wedge c$ qui donnent deux modèles P-restreints $A \wedge \neg B$ et $A \wedge B$. $A \wedge c$ est un impliquant de BC. On en déduit l'impliquant P-restreint A. Il est impossible de construire un modèle $M^{\wedge}$ de α tel que $M \cap P = \{A\}$ car M doit

contenir B ou $\neg B$ *(donc* $M \cap P=\{A,\neg B\}$ ou $M \cap P=\{A,B\}$*)*. Si $P=\{A,\neg A,B\}$, alors c'est possible car pour chaque littéral de P restant à satisfaire, son complémentaire n'est pas dans P : on prendra $M = \{A,\neg B, c\}$.
Si maintenant $P=\{A,\neg B\}$, il est toujours possible de construire M.

Nous allons maintenant nous intéresser à la complexité de ces différents algorithmes. Nous étudierons tout d'abord celle-ci dans le pire des cas et ensuite nous essayerons de rapprocher ces résultats des observations faites sur nos implantations des ces algorithmes.

4 Etude des différents algorithmes

4.1 Etude théorique

4.1.1 Complexité des algorithmes

La complexité de MPL approchée par le nombre de nœuds développés au pire des cas est la même que celle de la procédure de Davis et Putnam, c'est à dire $O(2^{|S|})$. En effet, dans les deux cas, la preuve de l'inconsistance d'une formule peut entraîner le développement complet de l'arbre de recherche.
Il faut néanmoins noter que la complexité « réelle » de MPL est bien différente de celle de DP. La complexité de MPL est en moyenne plus élevée que celle de DP car celui-ci s'arrête dès qu'il trouve un modèle de la base. Dans le cadre de MPL, cela n'est pas vrai. On peut donc penser que MPL développe un nombre de nœuds comparable à la preuve de l'inconsistance. C'est en partie vrai. Mais il faut noter que des heuristiques spécialisées (stratégies FSS) permettent d'obtenir rapidement des clauses vides afin de limiter la taille de l'arbre. Elles perdent leur efficacité dans le cadre de MPL car il faut à la fois adopter une stratégie FIS afin de trouver des impliquants et une stratégie FSS afin de limiter la profondeur de l'arbre. De plus l'ordre des littéraux est fixé, contrairement aux heuristiques traditionnelles de DP, mais cela n'est pas très grave car il faut de toute façon explorer les deux branches.

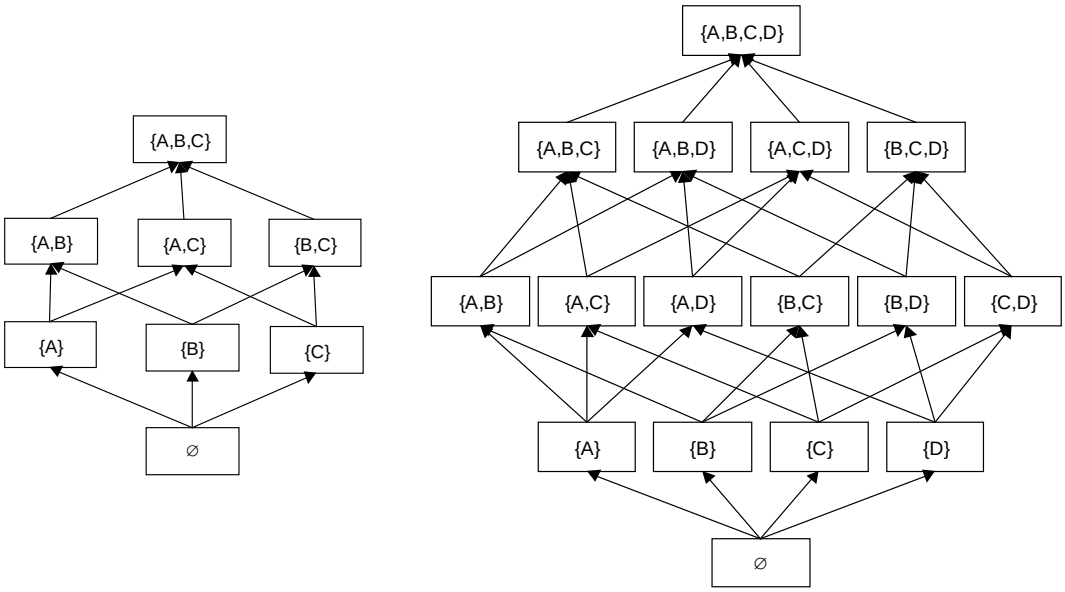
Le filtrage de la base s'effectue en $\sum_{c \in BC} |c| * |P|$ car pour chaque clause $c \in BC$ il faut déterminer si chacun de ses éléments appartient ou non à P. Comme $|P| \leq n$ et $|c| \leq n$, le filtrage s'effectue en $O(|BC| * n^2)$. La complexité du calcul des P-impliquants premiers est donc $O(2^n + n^2)$, c'est à dire - théoriquement - la même que celle de MPL.

La complexité du calcul de P-impliqués premiers est égale à la complexité de la première passe $O(2^{|S|})$ plus la complexité de la deuxième passe $O(2^{|P|})$. La complexité de cette méthode dépend donc aussi de la taille de P : $O(2^{|S|} + 2^{|P|}) = O(2^{|S|})$.

4.1.2 Nombre maximal d'impliquants P-restreints premiers

Une différence entre les algorithmes résolvant des problèmes de décision comme SAT et des algorithmes résolvant des « function problems » comme le calcul d'impliquants/impliqués est que la taille du résultat peut être exponentielle par rapport à celle du problème initial. Or nous ajoutons pour chaque solution trouvée une clause dans la base qui va nous garantir la minimalité de la prochaine solution trouvée. Notre base peut donc devenir exponentiellement plus grande que la base originale.

Etudions le nombre maximal d'impliquants P-restreints premiers en fonction de la taille de P. L'ensemble des parties de P forme pour l'inclusion ensembliste un treillis de $\varnothing$ à P. Prenons par exemple $P = \{A,B,C\}$ *puis* $P = \{A,B,C,D\}$. Voici les treillis correspondant aux parties de P :



On peut noter ici qu'il peut y avoir au maximum 3 impliquants P-restreints premiers si $|P|=3$ et 6 impliquants P-restreints premiers si $|P| = 4$.

Nous allons montrer[26] que ce nombre correspond au nombre maximal d'éléments incomparables, soit $C_{|P|}^{|P| \text{ div } 2}$.

Preuve :

Nous allons montrer que la fonction C_n^p, pour *n* fixé et *p* variable, est une fonction croissante jusque $p = n \text{ div } 2$ (division entière de *n* par 2) puis décroissante jusque $p = n$. Comme $C_n^p = C_n^{n-p}$, il suffit de montrer que C_n^p est croissante de $p= 0$ à $p= n \text{ div } 2$.

Notons tout d'abord une propriété de C_n^p :

$$C_n^p = \frac{n!}{p!(n-p)!} = \frac{n!(n-p+1)}{p*(p-1)!(n-(p-1))!} = \frac{n-p+1}{p} C_n^{p-1}$$

C_n^p est croissante ssi $C_n^p > C_n^{p-1}$ ssi $\frac{n-p+1}{p} > 1$ ssi $n - p + 1 > p$ ssi $p < \frac{n+1}{2}$. Comme *p* et *n* sont des entiers, $p < \frac{n+1}{2}$ ssi $p \leq n \text{ div } 2$. Donc C_n^p est croissante jusque $p = n \text{ div } 2$.

Exemple : pour |P|=100, il peut y avoir dans le pire des cas C_{100}^{50} = 100 891 344 545 564 193 334 812 497 256 impliquants P-restreints premiers !

Ce résultat ne permet pas de connaître le nombre d'impliquants P-restreints maximal d'une formule mais permet de mettre en évidence son caractère exponentiel.

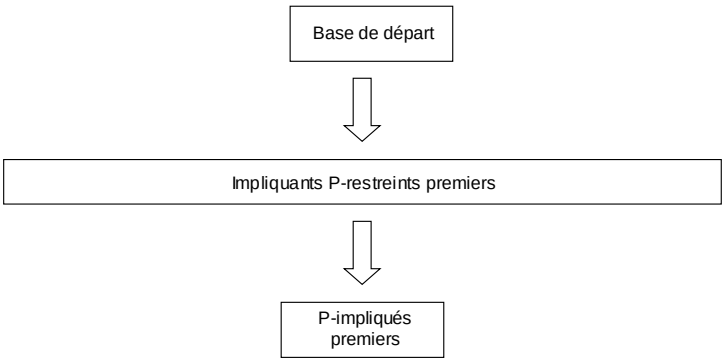
De plus, la taille des solutions (en nombre de littéraux), augmente avec leur nombre (en théorie |P| div 2). Ceci implique que les coupes de minimalité apparaissent plus tard dans l'arbre de recherche. Donc leur pouvoir de coupe est de moins en moins important.

4.1.3 Relation entre le calcul d'impliquants P-restreints et de P-impliqués premiers

Notre méthode de calcul de P-impliqué premier est basée sur un changement de forme : le calcul des impliquants P-restreints premiers. Cette approche connaît une limite que l'on connaît bien en compilation[27] par exemple : il est possible lors d'un changement de forme de trouver une formule de taille exponentielle par rapport à la taille de la base de départ.

[26] D'après des indications de Pierre Marquis, Jérôme Lang et Michel Cayrol.
[27] Ici, ce sont les solutions qui sont sous forme compilée. Pas la base de connaissance.

Ainsi, dans le pire des cas, nous pouvons trouver :



Imaginons pour cela que *BC* contienne uniquement des littéraux purs. Effectuer un calcul d'impliquants P-restreints premiers dans ce cas là (avec P l'ensemble des littéraux de *BC*) revient à calculer les P-impliquants premiers de *BC*, c'est à dire à effectuer un « minimal hitting set » sur *BC*. On comprend aisément que dans ce cas précis, le nombre d'impliquants P-restreints est exponentiel par rapport à la taille de *BC*. De plus, si l'on calcule les P-impliqués premiers de *BC*, on retrouve une solution de la taille de *BC* (voire même *BC* si aucune clause n'était sous-sommée !). Nous calculons donc un résultat trivial en passant par une solution intermédiaire exponentielle.

Notons qu'une méthode basée sur la résolution aurait produit le résultat attendu très facilement, étant donné qu'aucune résolution n'est possible.

Nous allons maintenant illustrer ces résultats sur le comportement de l'algorithme.

4.2 Etude expérimentale

4.2.1 Remarques préliminaires

Les résultats que nous présentons dans cette partie ont été obtenus sur des bases de clauses 3-SAT générées aléatoirement. On retrouve dans [Schrag/Crawford 1996] une étude empirique sur la présence d'impliqués premiers dans les instances 3-SAT. Nous étudions ici surtout les impliquants P-restreints premiers et les P-impliqués premiers.

Nous ne mesurons pas ici la performance de notre implantation de MPL, nous étudions son comportement. Pourquoi ?

- d'une part car nous n'avons pas connaissance de benchmarks disponibles pour ce type de problèmes[28],
- et d'autre part car notre implantation de la procédure de Davis et Putnam ne rivalise pas avec les implantations actuelles comme C-SAT, SATZ, SATO, etc. Cela s'explique d'une part par le langage choisi (Java contre C) et d'autre part car une implantation efficace pour un calcul de couverture d'impliquants (premiers) ne l'est pas forcément pour un calcul de satisfiabilité.

Comme dans le cadre de la satisfaction, nos paramètres seront : le nombre de variables (VAR), le nombre de clauses (CLAUSES) ou le ratio $\frac{\text{nombre de clauses}}{\text{nombre de variables}}$ (RATIO), la probabilité de tirer un littéral positif (PPOS). Pour tenir compte des spécificités de notre problème, nous ajoutons à ces paramètres standard du monde SAT un paramètre SUB correspondant à la proportion de symboles de la base qui appartiennent au sous-langage ou la taille de P selon les graphiques.

Les résultats sont exprimés en nombre de nœuds développés, en temps CPU (en millisecondes) ou en nombre de solutions. Nous différencions les nœuds issus d'une simplification de la base (propagation des clauses unitaires (CU) et des littéraux purs (LP)) des nœuds de branchement (AUTRES).

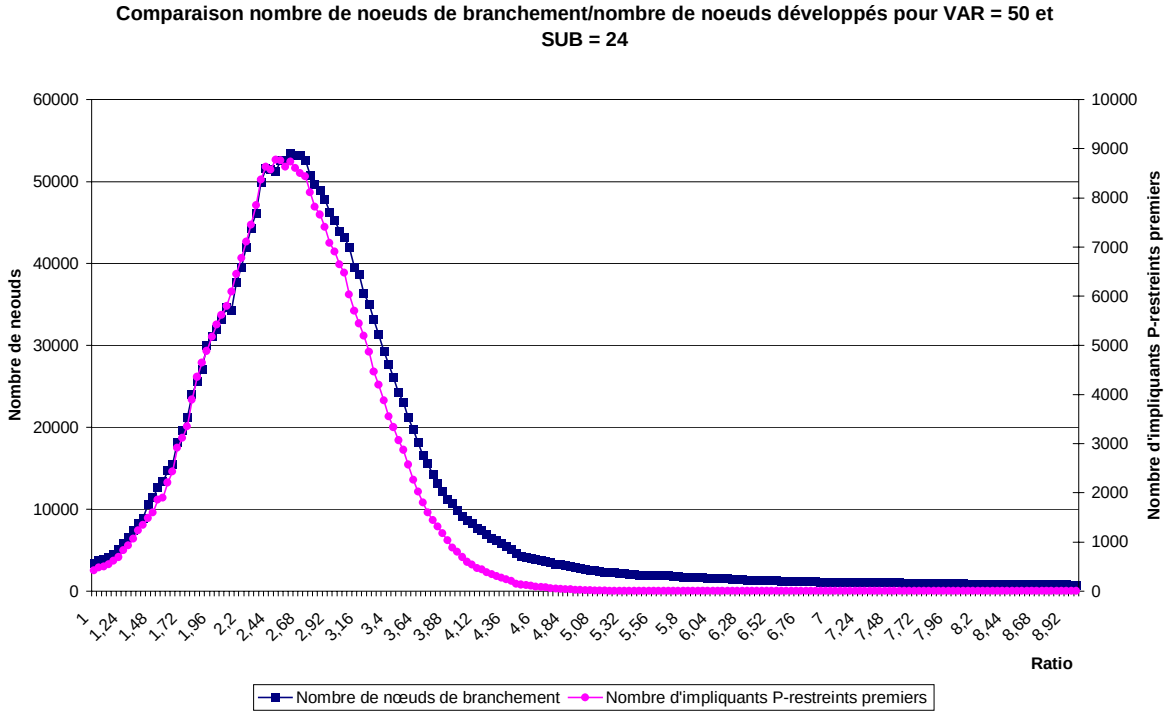
Tous les tests ont été effectués avec 100 bases par paramètre. Les valeurs des résultats sont des valeurs cumulées (pour obtenir le temps moyen de résolution d'un problème ou le nombre moyen de nœuds développés pour ré-

[28] Les benchmarks d'ISCAS'85 (http://wwww.cbl..ncsu.edu/www/CBL_Docs/iscas85.html) sont utilisés dans le cadre du diagnostic de panne (diagnostic consistant), mais sont malheureusement disponibles uniquement sous un format nécessitant une conversion en logique.

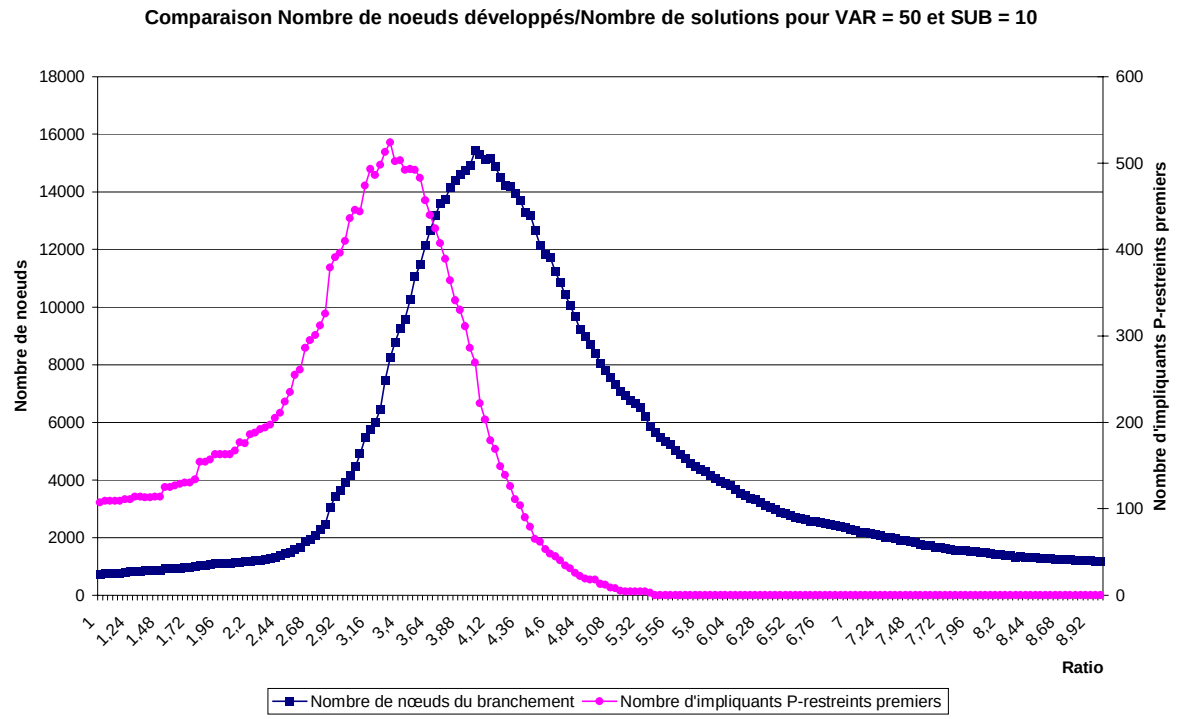
soudre ce problème, il faut donc diviser les résultats par 100). Les temps d'exécution ne sont donnés qu'à titre indicatif car ils dépendent de la machine utilisée, et surtout de la charge de la machine. La plupart des tests ont été effectués sur une Sun Ultra 5, 64 Mo, 300 MHz avec la machine virtuelle Java fournie avec le JDK 1.2 de SUN.

4.2.2 Comportement selon le ratio

La première chose que nous avons voulu vérifier concerne les similitudes de comportement entre MPL et un DP classique. Nous avons donc utilisé un critère classique pour DP : le ratio. Etudions d'abord les résultats obtenus pour VAR=50, PPOS=0,5 et SUB=24.

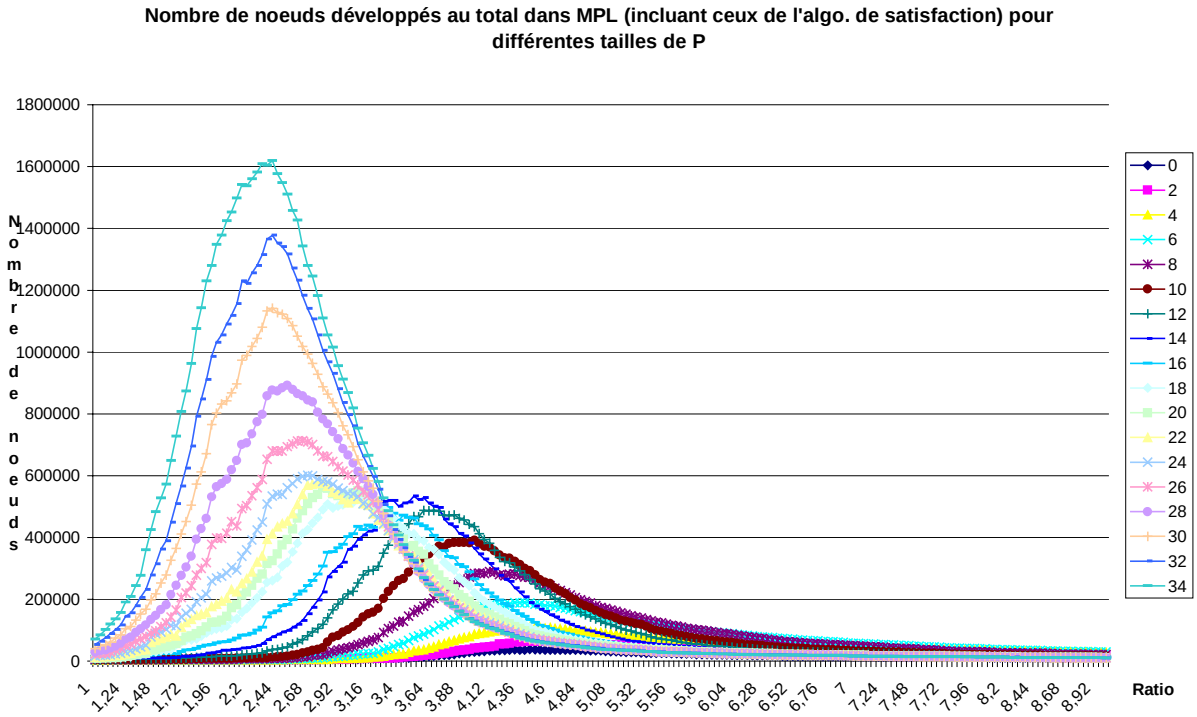


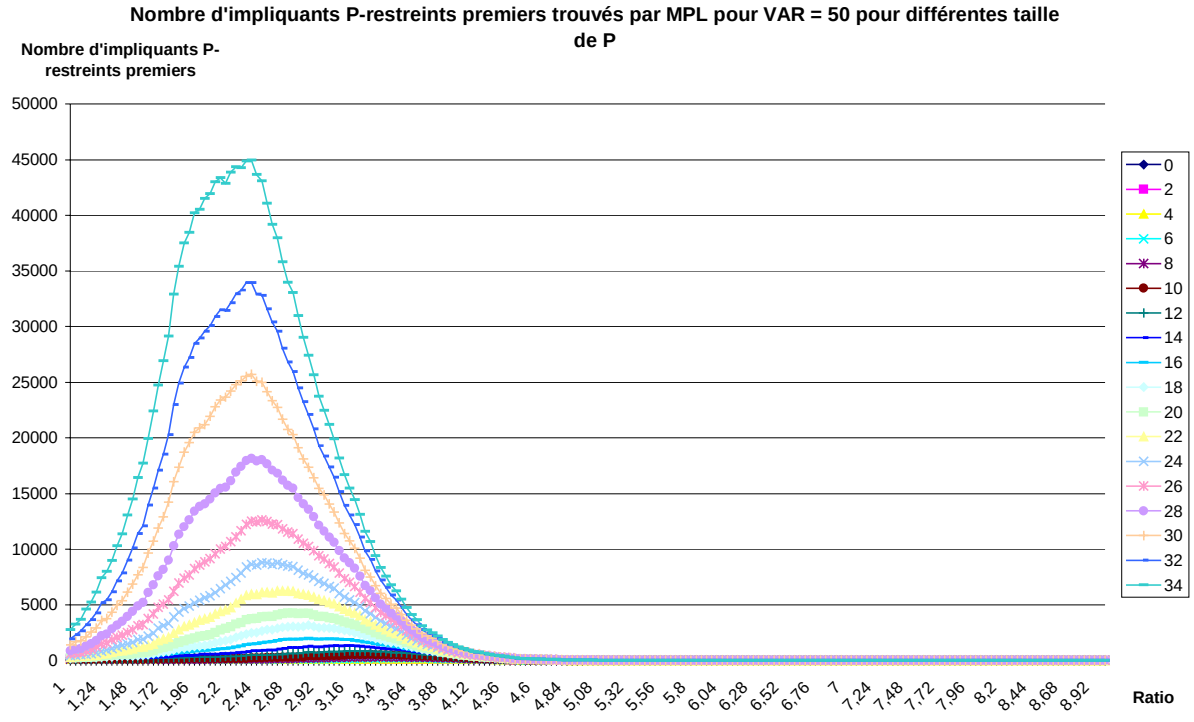
Dans ce cas, on observe bien une distribution des problèmes FACILE- DIFFICILE -FACILE. On peut comprendre ce comportement : à un ratio faible, il y a tellement d'impliquants (de modèles) qu'il n'y a qu'un seul impliquant P-restreint premier, ∅. A un ratio élevé, il n'y a plus d'impliquant, il faut donc prouver l'inconsistance de la base. Entre les deux, on a un pic de difficulté qui semble correspondre au nombre d'impliquants P-restreints premiers, c'est à dire à la taille du résultat.
Un deuxième test avec cette fois-ci SUB=10 nous montre un résultat quelque peu différent.



Dans ce cas, les pics de complexité (du nombre de nœuds) et du nombre de solutions sont clairement décalés. Il semble donc que ces pics se situent à des ratios différents selon la taille de P.

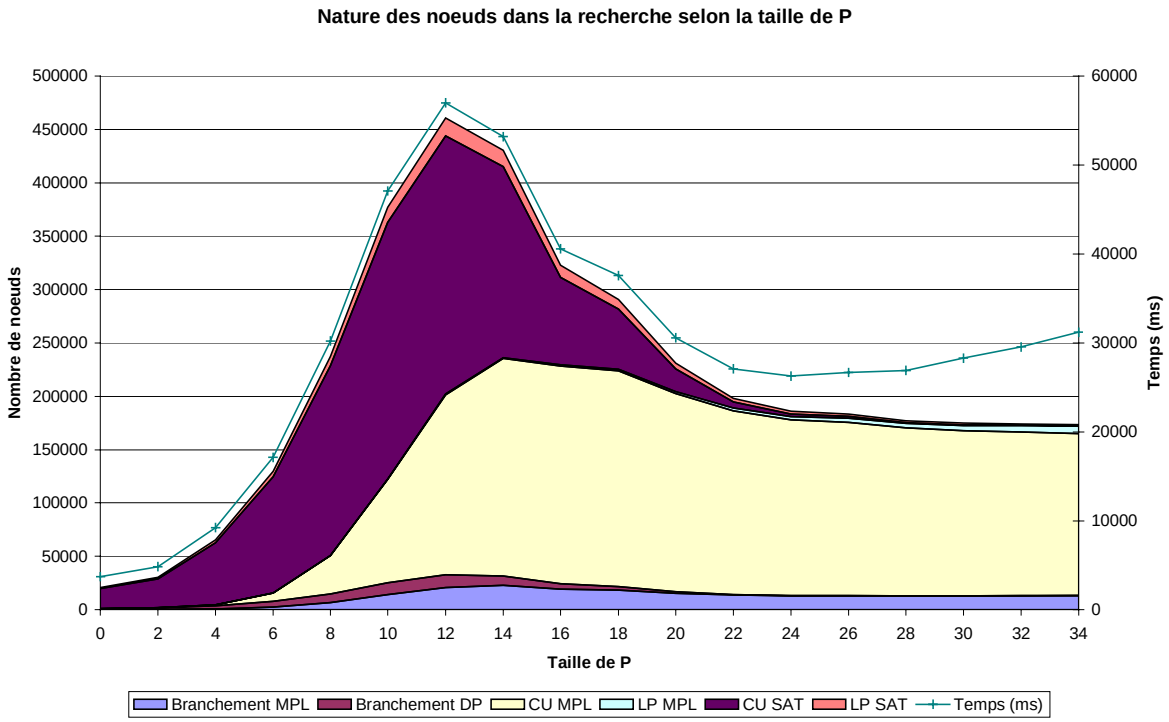
Nous avons donc effectué les mêmes tests en faisant varier SUB de 0 à 34. On s'aperçoit alors que les courbes de complexité et les courbes du nombre d'impliquants P-restreints premiers se rejoignent à partir de SUB=24. L'explication la plus logique est qu'à partir de SUB=50%, une bonne partie de la recherche est consacrée à la recherche d'un grand nombre d'impliquants P-restreints (plus SUB augmente, plus le nombre de solutions augmente). Et plus le nombre de solutions est grand, plus la preuve de l'inconsistance devient négligeable par rapport à la production des solutions.





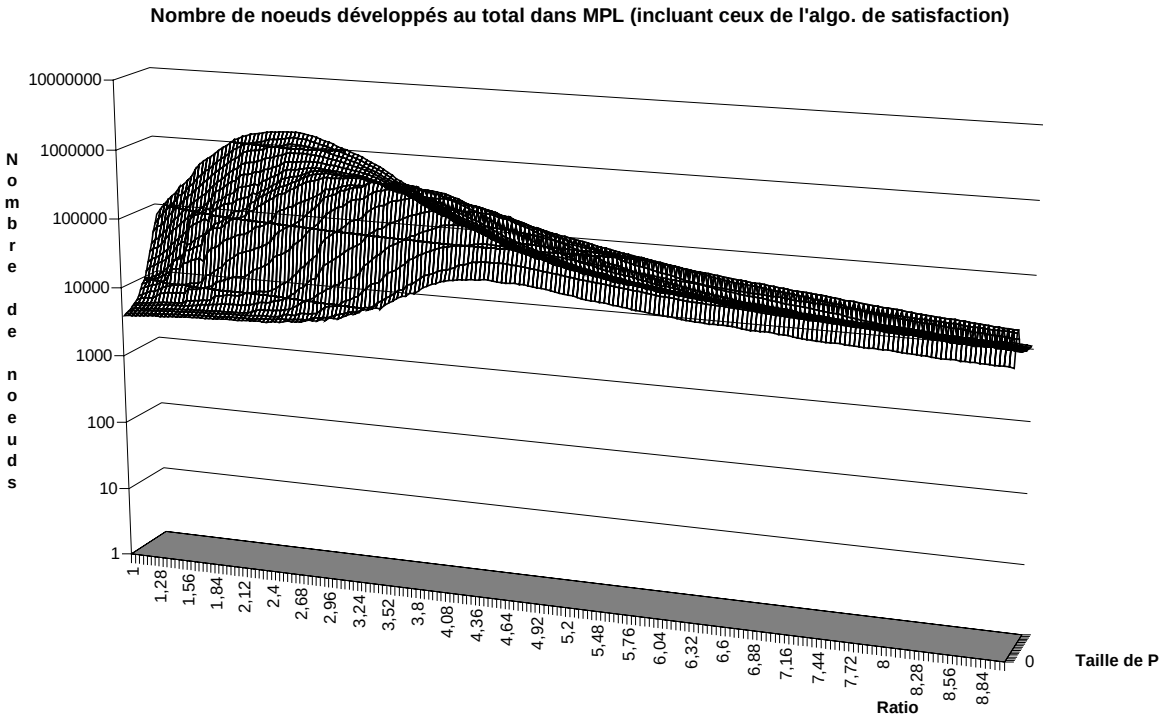
4.2.3 Comportement selon la taille du sous-langage

Un autre paramètre que nous pouvons faire varier est la taille du sous-langage. La taille de P varie de 0 à 34 pour RATIO=3,8.



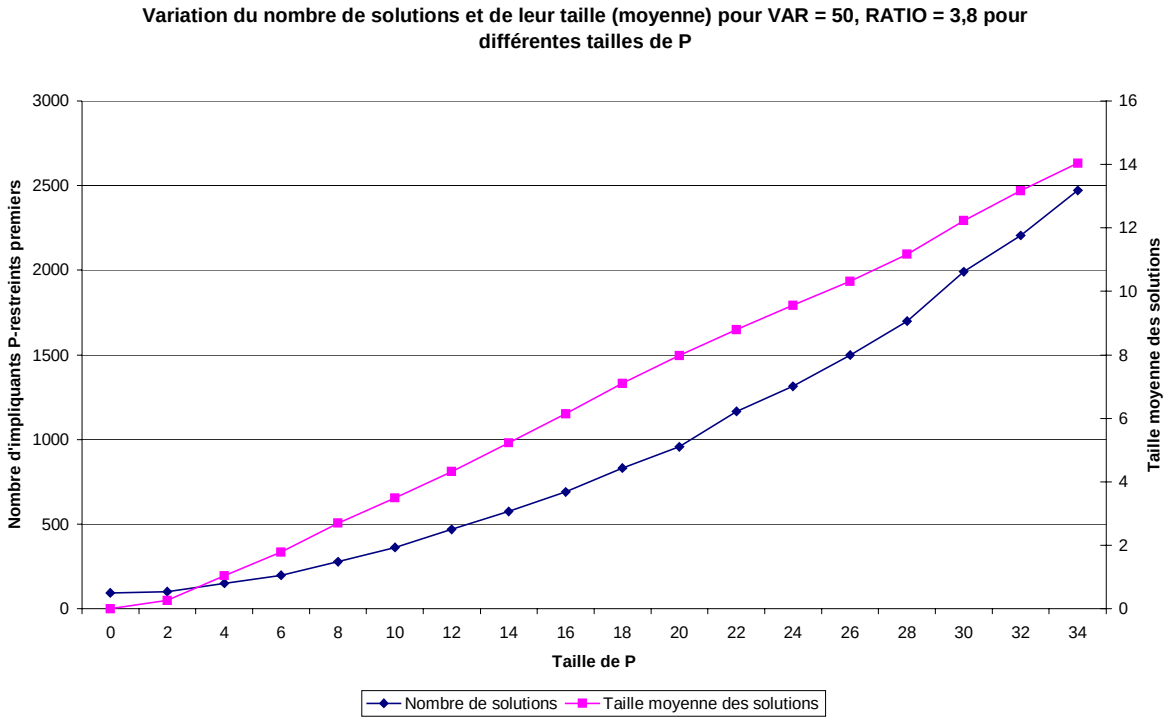
On note l'apparition d'un nouveau pic de difficulté, vers SUB=12. Le nombre de nœuds développés ensuite décroît puis se stabilise, exactement comme pour SAT lorsque l'on fait varier le ratio. On remarque que si le temps de calcul est proportionnel au nombre de nœuds développés jusqu'à SUB=24, il devient plus important ensuite : cela s'explique sans doute par l'ajout des clauses de minimisation. En effet, la propagation devient de

plus en plus lourde avec l'augmentation du nombre de clauses. Le graphique suivant nous montre que ce comportement n'est pas une caractéristique du problème. Nous étudions toujours le nombre de nœuds développés au total par MPL pour différentes tailles de *P* mais cette fois-ci pour différentes valeurs du ratio.

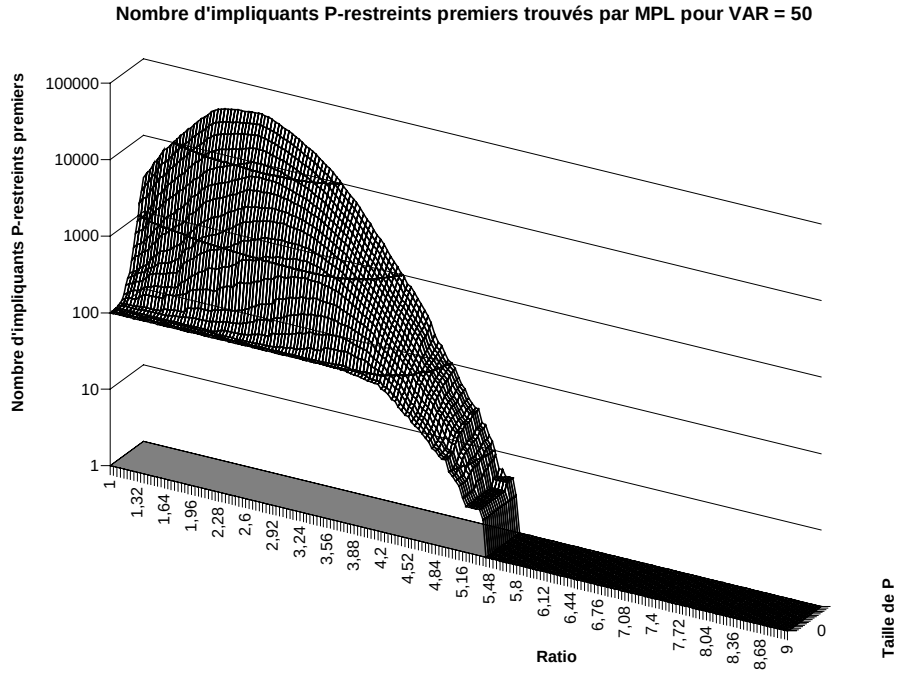


Notons que pour SUB=0 nous retrouvons le pic de difficulté du problème SAT. Le pic de difficulté observé précédemment n'est pas observé pour les valeurs du ratio inférieures à 2. Il apparaît entre les ratios 2 et 4 puis, il s'accentue. Il est intéressant d'observer qu'il existe aussi sur les instances inconsistantes, ce qui indique qu'il ne faut pas chercher du côté du nombre de solutions produites. Nous pensons qu'il s'agit d'un seuil (20% du nombre de variables) pour lequel l'heuristique utilisée pour le choix des variables de P est inadéquate. On remarque de plus que ce pic semble plus large entre les ratio 2 et 4.

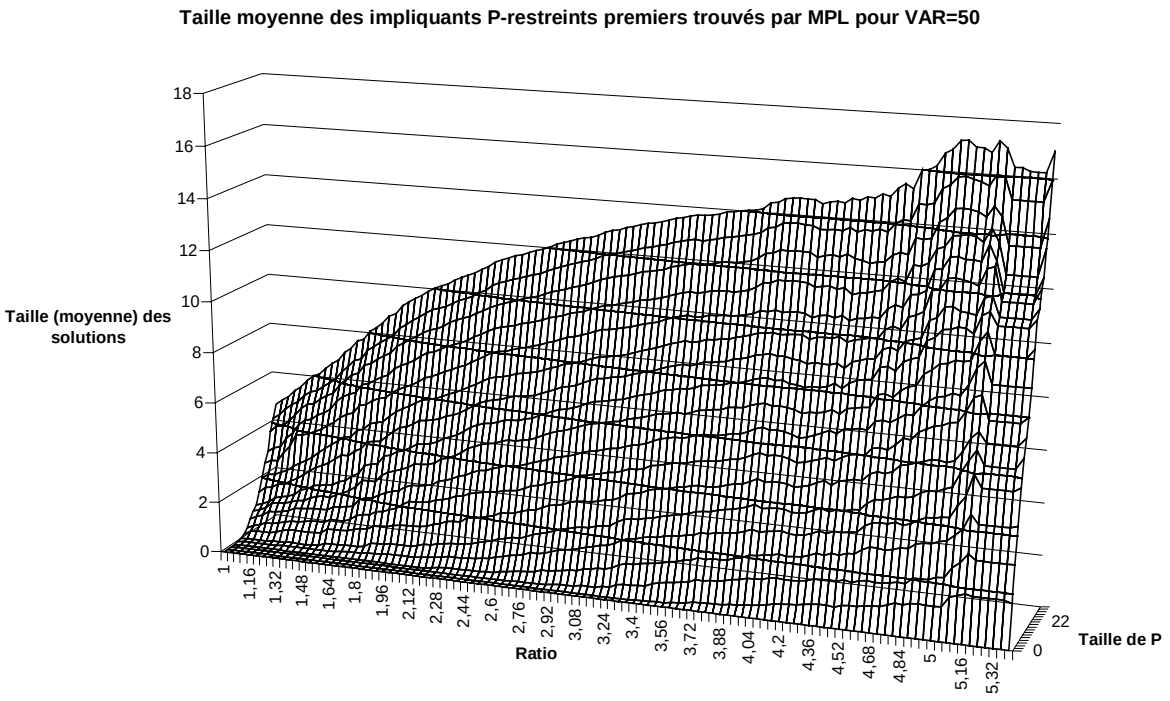
Pour confirmer cette hypothèse, reprenons maintenant le cas particulier de RATIO=3,8. Nous étudions maintenant le nombre de solutions (cumulées) et la taille moyenne de ces solutions. On note que le nombre de solutions augmente avec la taille de P. Or nous avons vu que le nombre de nœuds développés pour une taille de P supérieure à 24 était presque constant. Cela semble confirmer l'idée d'une heuristique plus robuste. On remarque de plus que la taille des impliquants P-restreints premiers trouvés augmente avec la taille de P.



Les graphiques suivant confirment ces résultats pour diverses valeurs du ratio. Le phénomène de seuil que l'on connaît bien pour SAT (que l'on retrouve ici pour SUB=0) est amplifié avec la taille de P.



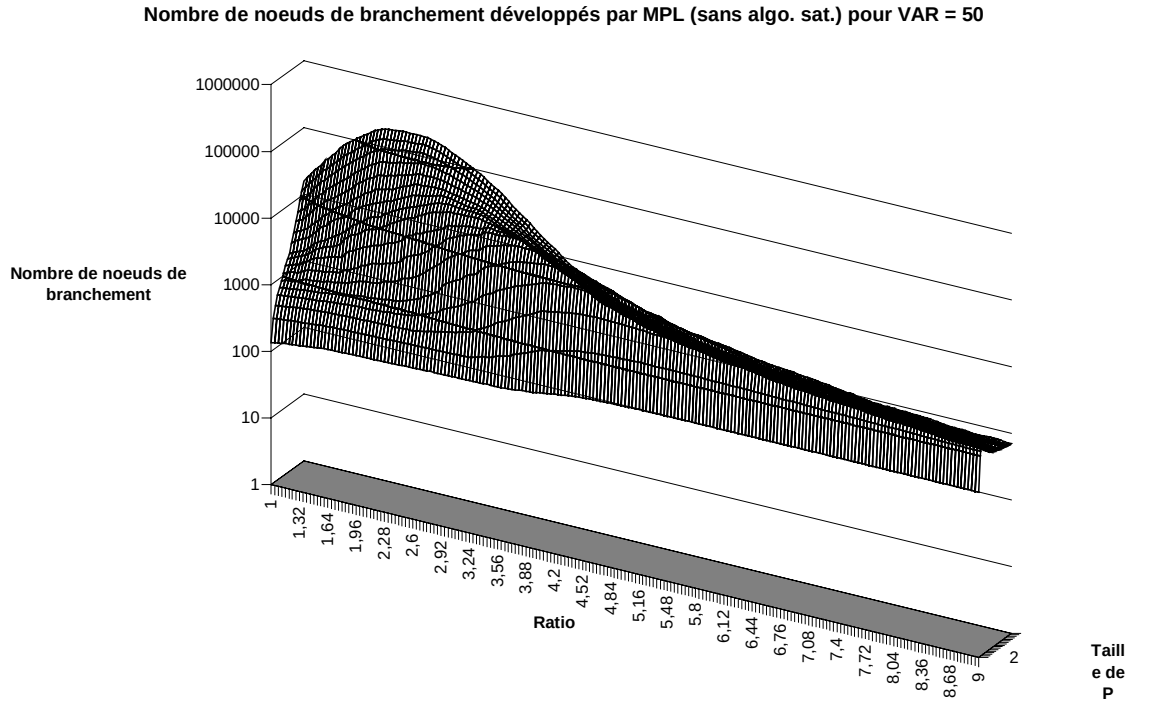
En faisant varier le ratio de 1 à 5,4 (après toutes les instances générées sont inconsistantes), on remarque que la taille (en moyenne) des impliquants P-restreints premiers plafonne à la moitié de celle de *P*.



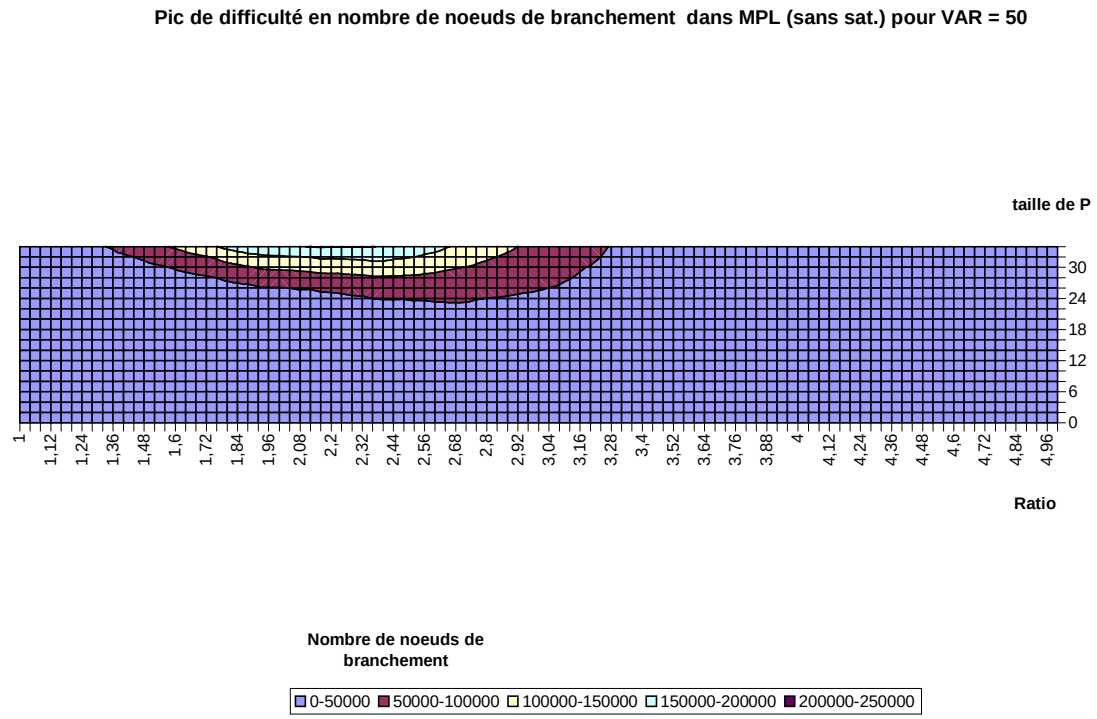
4.2.4 Nature de la recherche

Nous cherchons maintenant à savoir comment se répartit la recherche des solutions entre la partie optimisation de MPL (énumération contrainte des parties de P) et sa partie satisfaction (utilisation d'un algorithme de satisfaction classique, ici DP).

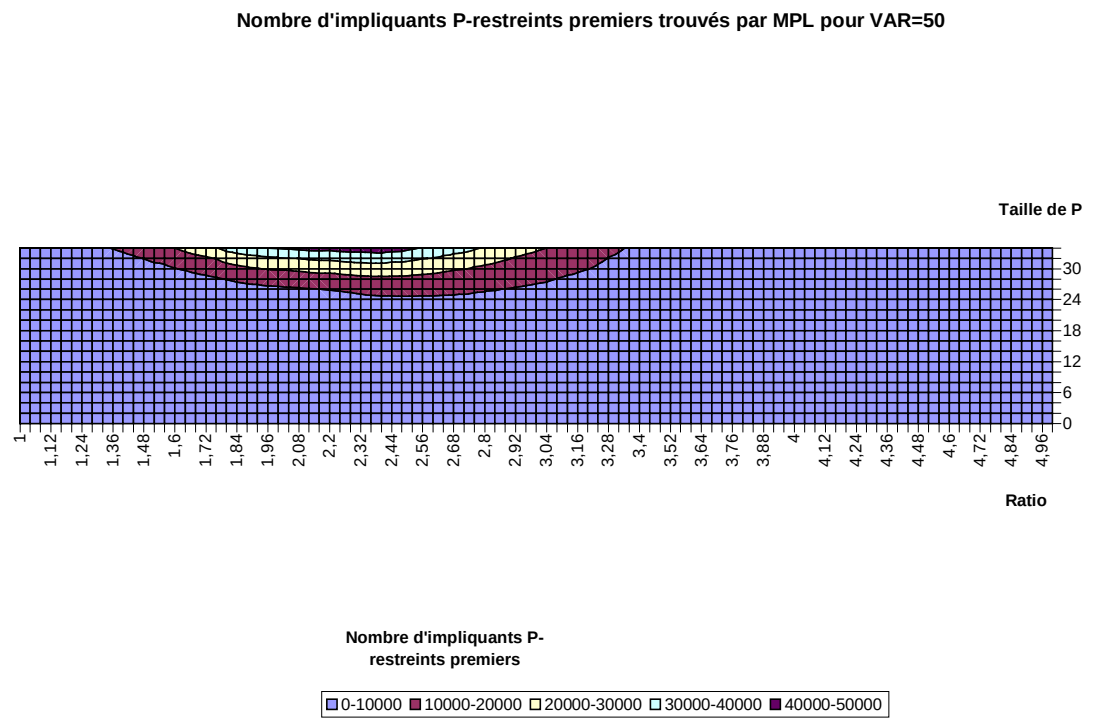
Il faut comparer le graphique ci-dessous, concernant le nombre de nœuds de branchement développés par MPL sans tenir compte des nœuds de branchement de l'algorithme de satisfaction, avec celui comptabilisant tous les nœuds. On remarque une petite différence autour du ratio 3.



Pour mettre en évidence cette différence, nous représentons dans les graphiques suivants les pics de difficulté obtenus pour l'énumération des éléments de P d'une part, et le test de satisfaction d'autre part.

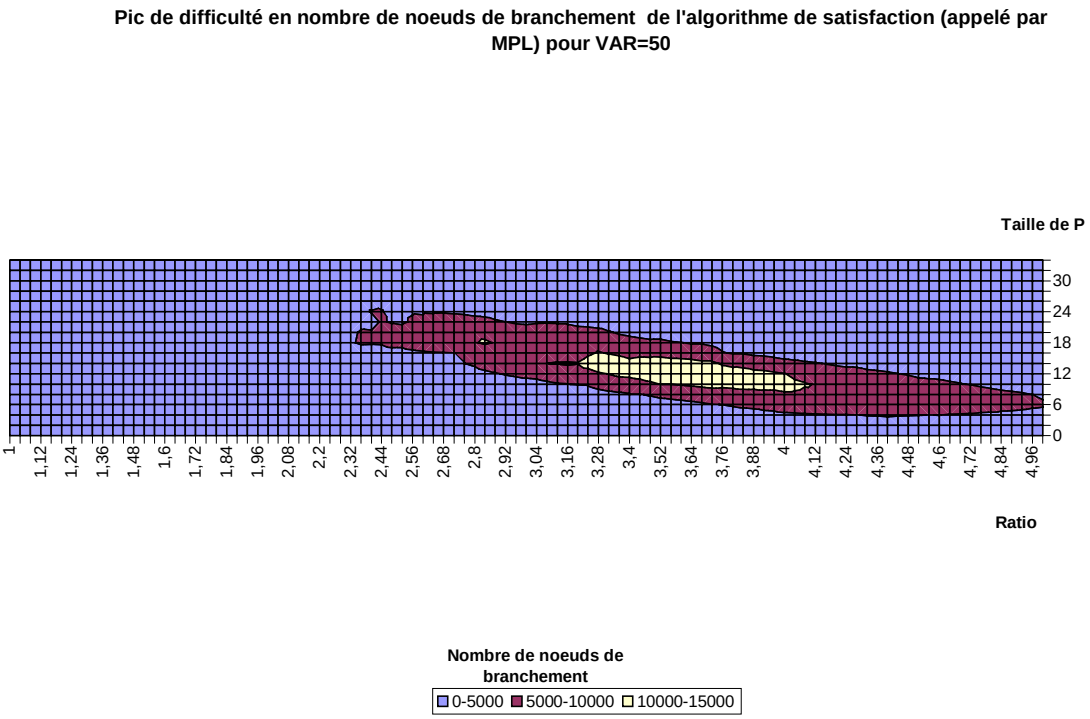


Ici, le pic de difficulté correspond au pic du nombre d'impliquants P-restreints premiers, maximal autour d'un ratio de 2 pour le nombre maximal d'éléments de P (ici 34 sur 50). On le vérifie sur ce graphique.



Cela paraît assez normal. Plus la taille de P augmente, plus il y a de solutions, et étant donné la forme C_n^p du nombre d'impliquants P-restreints premiers (voir section 4.1.2), il paraît normal que son pic se trouve dans une région moyennement sous-contrainte.

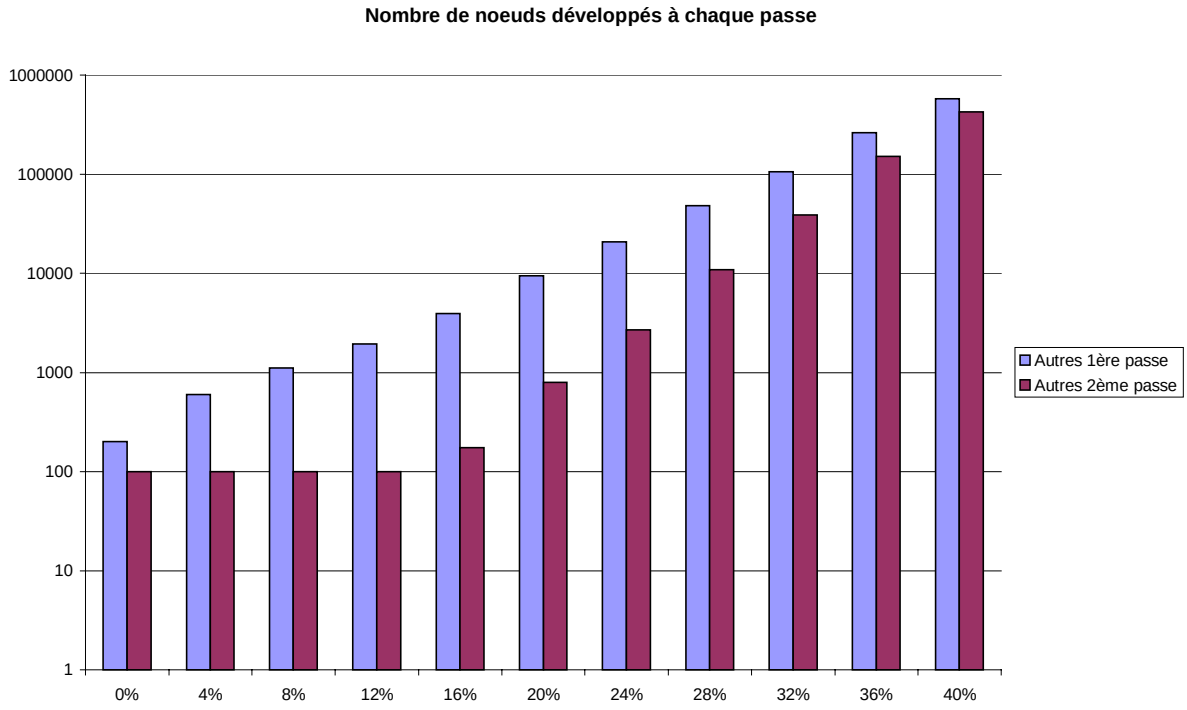
Le pic de difficulté concernant le test de satisfaction apparaît lui plus tard.



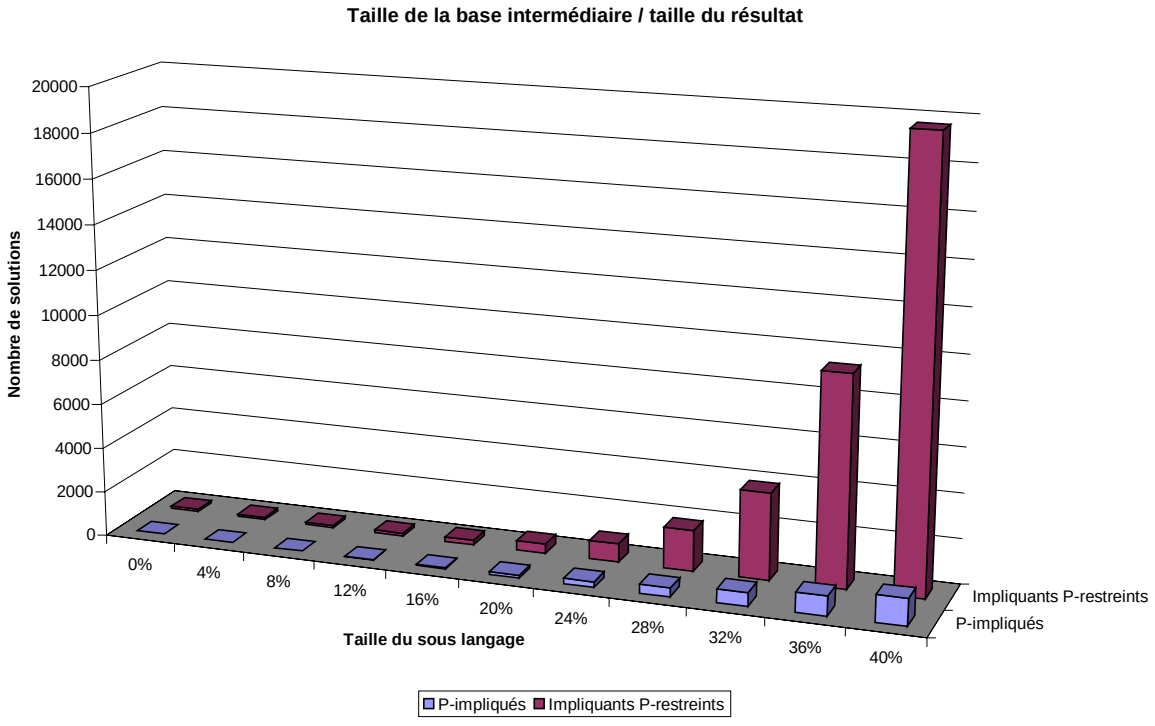
On le retrouve pour un ratio compris entre 3,2 et 4 et une taille du sous-langage de 8 à 14. Nous pensons que ce pic est du au mauvais comportement de l'heuristique. Comme il faut effectuer le branchement sur des variables de P, l'heuristique est très contrainte quand P est assez petit : si P est très petit, il n'est pas très grave de se tromper dans le choix de la variable de branchement, car il y a peu de solutions possibles. Si P devient assez grand, alors l'heuristique sur les variables de P permet de diriger la recherche convenablement. Entre les deux, l'heuristique est perdue. Ce phénomène apparaît à partir d'un ratio plutôt élevé car le nombre de retours arrière augmente dans MPL et donc il faut effectuer de plus en plus de tests de consistance. Ce qui conforte les observations précédentes.

4.2.5 Etude des deux passes

Nous avons vu que la complexité de notre calcul de P-impliqués est en $O(2^{|S|})$. Le test suivant met en évidence l'importance de la taille de P dans la complexité réelle de notre méthode, et illustre la remarque que nous avons faite à propos du pire cas pour notre méthode : calculer les P-impliqués d'une base de littéraux purs. Nous générons des bases de 50 variables et 200 clauses afin d'avoir une base fortement consistante, avec PPOS=0 afin d'avoir une base pure. Nous faisons varier la taille de P de 0 à 40 % du nombre de variables, les temps de calcul au-delà de 40% étant trop élevés. Nous affichons les nœuds de type « autres » développés durant chacune des passes.

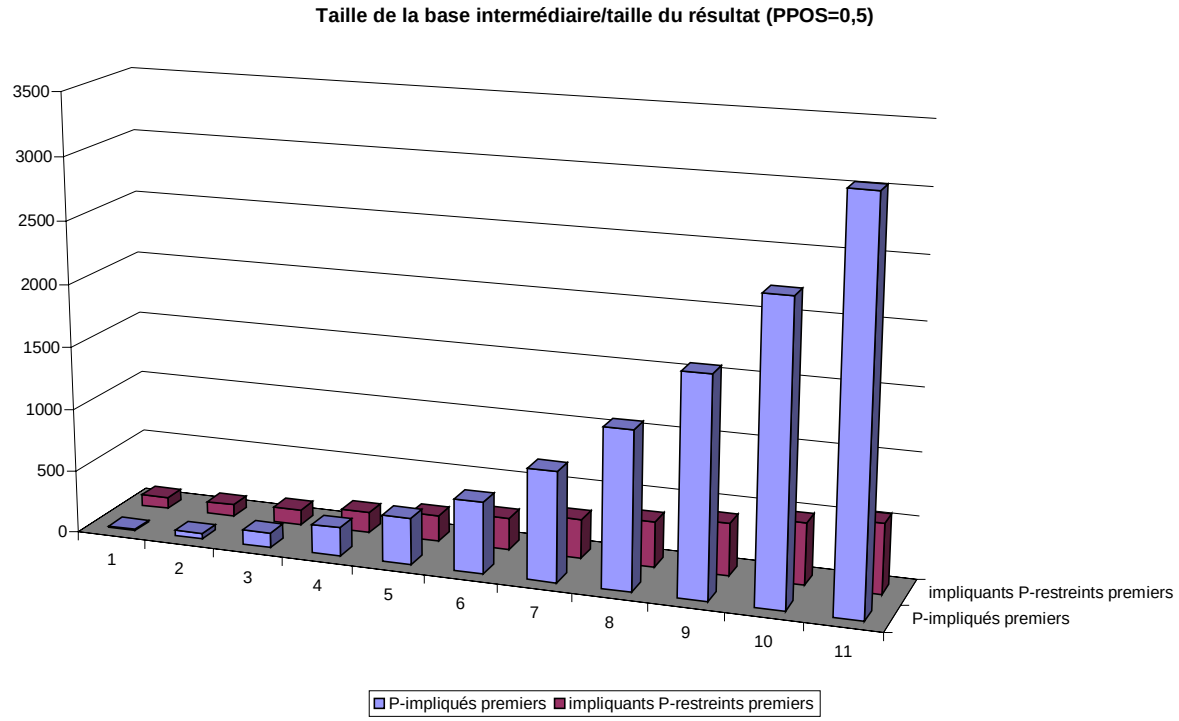


On peut noter que le nombre de nœuds développés lors de la seconde passe tend vers celui de la première passe (attention, ici l'échelle est logarithmique). Le graphique suivant donne une explication à cette observation. Nous comparons maintenant pour les mêmes paramètres le nombre d'impliquants P-restreints premiers trouvés lors de la première passe (taille de la base intermédiaire) au nombre de P-impliqués premiers trouvés lors de la seconde (le résultat).



Il apparaît nettement que la base intermédiaire devient exponentielle par rapport au résultat ! C'est malheureusement le point le plus faible de notre méthode.

Rappelons tout de même qu'il s'agit du pire des cas. Si l'on prend par exemple PPOS=0,5, on trouve des résultats plus encourageants :



On peut néanmoins se poser la question suivante …

4.3 Une passe ou deux passes ?

« Pourquoi passer par un changement de forme et ne pas calculer directement les P-impliqués premiers à partir de la CNF ? ».

Une première réponse, la plus sincère : parce que l'on ne sait pas le faire !

Une seconde réponse est qu'il serait en effet possible d'énumérer les environnements de $\varnothing$ à $P^{\neg}$ en ne gardant que ceux qui rendent la base inconsistante. Mais notre intuition est que cette méthode est beaucoup plus coûteuse que le passage par une DNF (c'est le cas sur certains exemples que nous avons traités). De plus, la rapidité de la réponse dépendrait fortement de la taille des environnements que l'on cherche. Dans notre approche, ce n'est plus si simple. Il existe donc des cas où notre approche sera moins bonne qu'une approche de type « générer et tester », mais nous pensons que de manière globale, elle est plus efficace.

Et pour étayer cette réponse, il faut savoir que 10 ans auparavant, dans un cas similaire, Reiter a adopté la même stratégie dans le cadre du diagnostic [Reiter 1987] (cf. page 100).
Reiter, arguant que l'approche « generate and test » est trop inefficace quand la cardinalité des diagnostics devient importante, à cause du nombre de tests à effectuer[29], utilise une méthode « en deux phases » basée sur la notion de conflit. Dans un premier temps, il calcule tous les ensembles de composants qui ne peuvent pas fonctionner en même temps (notion de *conflit*, identique à la notion de nogood si l'on considère comme hypothèse que le composant fonctionne correctement, un conflit est donc un ¬Ab-impliqué de la base). Ensuite, il calcule un « minimal hitting set » sur ces conflits pour obtenir les diagnostics minimaux. Nos démarches sont donc semblables.

On peut remarquer cependant que :

- ce que Reiter appelle un conflit (le résultat de sa première phase) est pour nous un P-impliqué (notre résultat),
- ce que nous appelons un impliquant P-restreint premier (le résultat de la première phase) est pour Reiter un diagnostic minimal (son résultat) !

[29] *« The obvious problem with this approach is that it is too inefficient for systems with large number of components.* » Section 4, p. 67

La première phase de l'un est le résultat de l'autre. Pourquoi ne pas inverser les méthodes afin de trouver les résultats en une phase ?

Notons que les conflits calculés par Reiter ne sont pas minimaux (ce qui entraîne d'ailleurs un problème dans son calcul du minimal hitting set [Greiner, et al. 1989]). Le calcul des conflits de Reiter est basé sur un prouveur utilisant le principe de résolution. Ceci s'explique sans doute car ce type de prouveur marche bien dans le cadre du *premier ordre*. Or, nous avons déjà fait remarquer que dans le cadre du *calcul propositionnel*, les meilleurs prouveurs actuels étaient basés sur la procédure de Davis et Putnam.

De plus, nous pouvons noter que pour calculer d'une manière énumérative directe les nogoods d'une base *BC* à partir d'un ensemble de $|P|$ hypothèses parmi $|S|$ symboles il faudrait effectuer pour chaque environnement $E \subseteq P$ de taille k un test de consistance de complexité $O(2^{|S|-k})$. Ce qui donne, dans le pire des cas, une complexité $O\left(\sum_{k=0}^{|P|} C_{|P|}^{k} 2^{|S|-k}\right)$.

Comme $(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$ et $\sum_{k=0}^{|P|} C_P^k 2^{|S|-k} = 2^{|S|-|P|} \sum_{k=0}^{|P|} C_{|P|}^{k} 2^{|P|-k}$ on obtient $\sum_{k=0}^{|P|} C_{|P|}^{k} 2^{|S|-k} = 2^{|S|-|P|} 3^{|P|}$.

Ce qui nous donne une complexité en $O(2^{|S|-|P|}3^{|P|})$ qu'il faut comparer avec la complexité théorique du calcul de P-impliqués premiers basé sur MPL *:* $O(2^{|S|} + 2^{|P|})$.

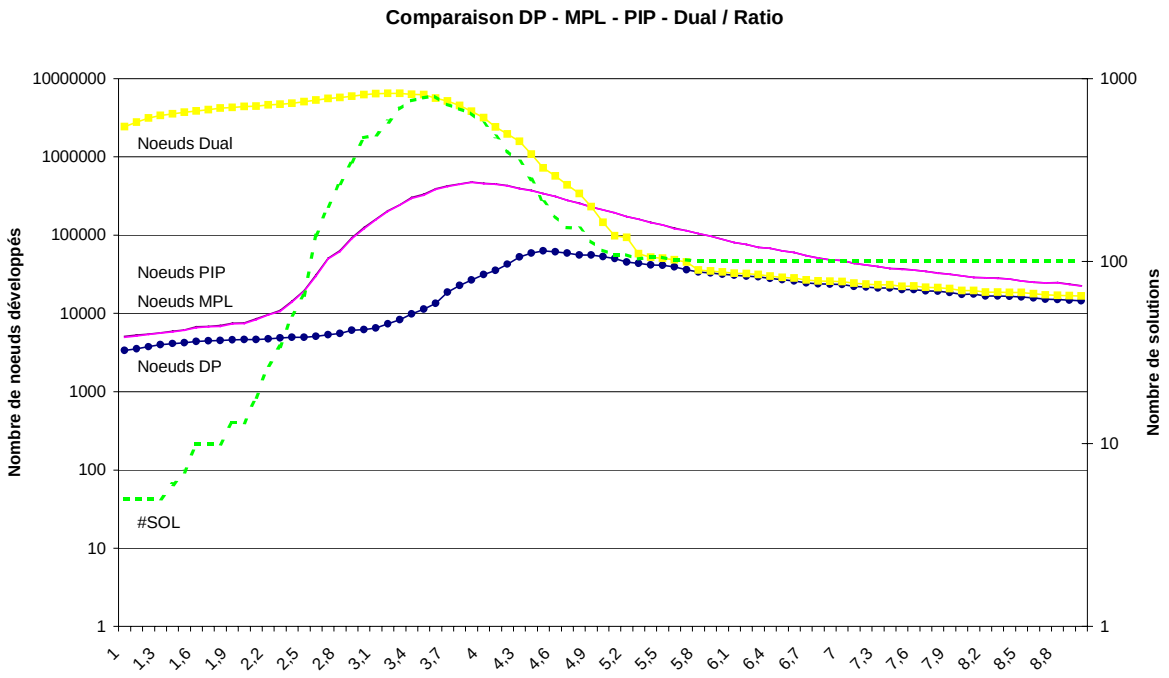
Nous avons implanté une méthode de calcul des P-impliqués premiers d'une formule suivant ce principe. Nous l'avons appelée DualMPL.

```
DualMPL(BC,BA,P,IP)
// Calcule les P-impliqués premiers de BC.
// IP contient les littéraux satisfaits.
// BA est la base de minimisation : le résultat se trouve dans cette base.
        Si BC = ∅ alors retourner ∅ Finsi ;
        Si    ∈ BA^∧IP^ alors retourner ∅ Finsi;
        Si    ∈ BC alors // on a trouvé un P-impliqué premier
                retourner BA ∪ {¬(IP∧)}
        Finsi ;
        l ← Choix_Symbole_de_P(BC,P) ;
        Si (l≠null) alors // il reste des variables de P à assigner
                SOL ← DualMPL(BC,BA,P\{l},IP) ;
                Retourner DualMPL(BC[l],SOL,P\{l},IP∪{l})
        Sinon
                Si DP(BC)=faux alors
                        retourner BA ∪ {¬(IP∧)} ;
                Sinon
                        retourner ∅
                Finsi
        Finsi
```

Algorithme 7 DualMPL - Calcul de P-impliqués premiers

Nous pouvons noter que cet algorithme ne tient pas compte de la propagation des clauses unitaires. En effet, comme on trouve une solution en falsifiant la base, une clause unitaire va maintenant produire une solution et une exploration de l'arbre. L'algorithme est appelé avec BA et IP initialement vide : DualMPL(BC,∅,P,∅).

Nous avons donc comparé cet algorithme avec PIP, l'algorithme de calcul de P-impliqués premier basé sur MPL (« en deux passes »). Nous avons vérifié que chacun des algorithmes trouvait le même nombre de solutions (cumulé) pour chaque paramètre afin de garantir leur implantation. Nous donnons le nombre de nœuds développés par les deux algorithmes pour obtenir les nogoods d'une base de clauses de 50 variables, |P|=10. Nous donnons à titre de comparaison le nombre de nœuds développés par MPL (pour la première phase de PIP) et DP (pour tester la consistance) sur ces mêmes bases. Les échelles des axes des ordonnées sont logarithmiques.



La première remarque que nous pouvons faire est que DP nous donne un minorant du nombre de nœuds à développer pour prouver l'inconsistance de la base. On ne distingue pas la différence entre les courbes MPL et PIP car le nombre de nœuds produits par la seconde passe est négligeable par rapport au nombre de nœuds produit lors de la première passe.

Nous pouvons ensuite noter que le nombre de P-impliqués premiers suit la traditionnelle courbe FACILE-DIFFICILE-FACILE que l'on connaît pour SAT. Seules quelques bases sous-contraintes ont des P-impliqués premiers. Les bases inconsistantes ont pour unique P-impliqué premier la clause vide.

On peut noter que sur les bases inconsistantes, l'algorithme DualMPL est le meilleur. En effet, il prouve tout de suite l'inconsistance de la base à l'aide de DP (on peut noter que les courbes se rejoignent). La preuve de l'inconsistance est plus difficile pour MPL car l'ordre des littéraux est contraint. Par contre, en ce qui concerne les autres bases, les résultats ne sont comparables qu'en prenant une échelle logarithmique (c'est aussi vrai pour la comparaison de ces algorithmes avec DP, ne l'oublions pas).

4.4 Minimisation par ajout de clauses

Nous utilisons (pour garantir la primarité des impliquants P-restreints trouvés) l'ajout de clauses dans une base de minimisation BA. Cette méthode a pour principal avantage de ne pas avoir à gérer explicitement la minimalité par d'autres mécanismes que ceux utilisés pour SAT. Mais elle a aussi un désavantage qui n'est pas mesurable en termes de nœuds développés mais en temps de calcul : lorsqu'il y a beaucoup de solutions, il nous faut gérer une base *BA* (y propager la satisfaction de littéraux) qui peut être de taille exponentielle par rapport à celle de la base de départ. Non seulement il faut de la mémoire pour maintenir cette base, mais au fur et à mesure de la recherche des solutions, le temps nécessaire à la propagation de la satisfaction d'un littéral dans cette base augmente avec le nombre de solutions.

On se retrouve donc en face de deux problèmes :
1) Comment tester la primarité d'un impliquant P-restreint ?
2) Comment représenter l'ensemble des solutions d'une manière compacte ?

4.4.1 Ensemble de littéraux nécessaires

Nous avons déjà noté que De Kleer [de-Kleer 1992] utilisait un arbre de discrimination (ou « *Trie* ») [Rymon 1992] pour effectuer son test de minimisation. Plus récemment, [Castell 1997] présentait une propriété permettant de tester la primarité de P-impliquants (P pouvant être inconsistant) sur la base de compteurs. En effet, un impliquant qui satisfait une base de clauses doit satisfaire chacune des clauses de la base. Il est donc premier si chacun de ses littéraux est le seul à satisfaire au moins une clause (l'auteur appelle ces littéraux des « littéraux

nécessaires »). Il maintient durant la recherche des compteurs associés à chaque littéral satisfait indiquant le nombre de clauses qu'il est le seul à satisfaire. Dès qu'un de ces compteurs est à 0, on est sûr que l'impliquant en cours n'est pas premier.

Définition 43 (ensemble de littéraux nécessaires [Castell 1997])
Soit $BC \in FP_S$ une base de clauses. Soit $X \subseteq L_S$ un ensemble de littéraux. X est *nécessaire* dans BC ssi $\forall x \in X$ $\exists cl^{\vee} \in BC$ telle que $X \cap cl = \{x\}$. Lorsque X n'est pas nécessaire dans BC, on dira que X *déborde* BC.

Le théorème suivant est alors très important :

Proposition 28 (théorème du débordement [Castell 1997])
Soit $BC \in FP_S$ une base de clauses. Soit $D \subseteq L_S$ un ensemble de littéraux. Si $D^{\wedge}$ est un P-impliquant de BC alors $\forall I \subseteq D$ on a I est nécessaire pour BC.

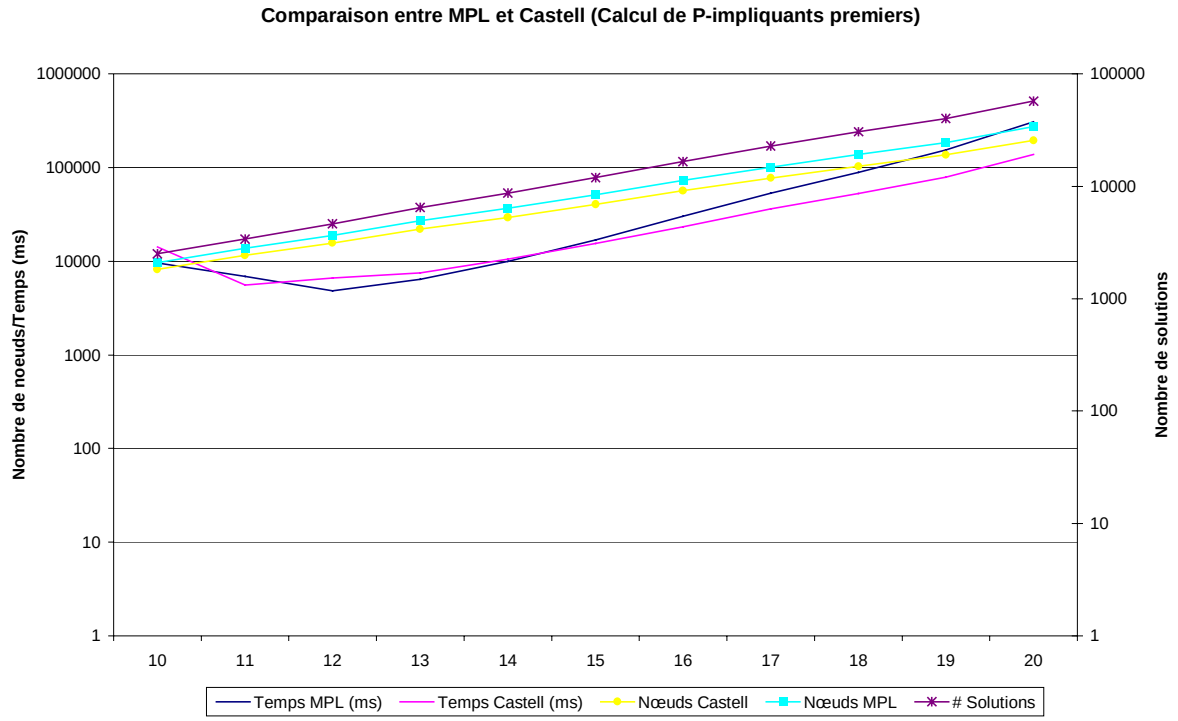
Il permet d'arrêter la recherche d'une solution dès que l'ensemble des littéraux satisfaits n'est plus nécessaire. Nous ne pouvons pas utiliser cette méthode dans MPL car cette propriété n'est plus vraie pour des impliquants P-restreints (premiers). Néanmoins, nous pouvons le faire dans le cas très particulier où la base contient uniquement des littéraux purs (pour la deuxième passe du calcul de P-impliqués par exemple). En effet, dans ce cas, calculer des P-impliquants ou des impliquants P-restreints étant équivalent (cf. Proposition 25 page 64), la propriété devient donc vraie pour ces derniers.

L'utilisation de cette notion pour le calcul de P-impliquants premiers (P consistant), nous donne l'algorithme suivant (pour le cas particulier ou la base ne contient que des littéraux purs) :

```
Castell(BC,IP,P)
// Calcule les P-impliquants premiers de BC
// BC ne doit contenir que des littéraux de P
// IP l'ensemble des littéraux satisfaits
// P un ensemble consistant de littéraux
// retourne les P-impliquants premiers de BC.
       Si    ∈ BC alors retourner ∅ Finsi ;
       Si IP∩P n'est pas nécessaire alors retourner ∅ Finsi ;
       Si BC = ∅ alors // on a trouve un P-impliquant premier
              retourner {(IP∩P)^}
       Finsi ;
       l ← LittéralPourSimplifier(BC) ;
       Si l ≠ null alors // si l existe
              retourner Castell(BC[l],IP∪{l},P)
       Finsi ;
       l ← choisirLiteral(BC) ;
       retourner Castell(BC[¬l],IP∪{¬l},P) ∪ Castell(BC[l],IP∪{l},P)
```

Algorithme 8 Calcul de P-impliquants premiers à la Thierry Castell

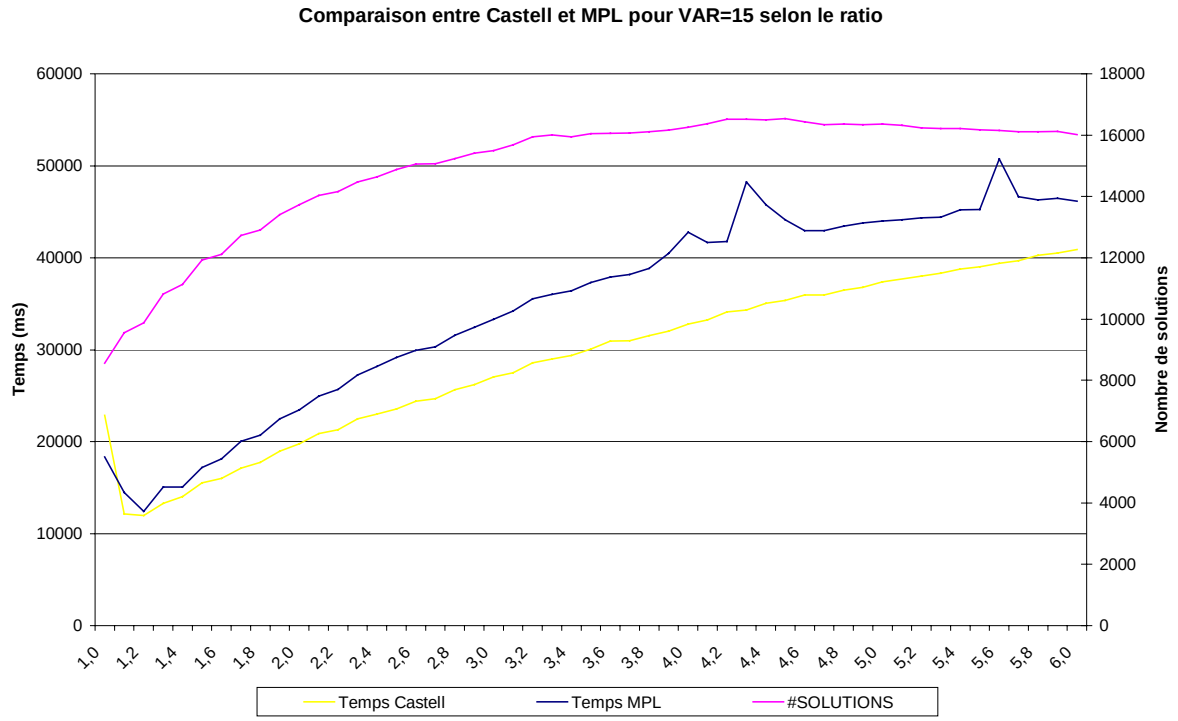
Nous avons testé la méthode de minimisation de Thierry Castell par rapport à la notre dans le cas très particulier du calcul de P-impliquants premiers. Nous avons généré aléatoirement 100 bases au ratio 1,5 contenant uniquement des littéraux purs. Comme ces bases sont sous-contraintes, il y a énormément de P-impliquants premier si P est l'ensemble de ces littéraux purs. C'est pourquoi nous générons uniquement des bases de 10 à 20 variables, avec |P|=|VAR|. A titre d'exemple, une base de 20 variables et 30 clauses donne 520 solutions environ ! Attention : les échelles sont logarithmiques.



Le nombre de solutions est exponentiel par rapport au nombre de variables. Le nombre de nœuds développés est proportionnel au nombre de solutions. Il est plus élevé pour MPL que pour Castell mais ce dernier développe plus de nœuds de branchement. On le ressent un peu sur le temps de résolution[30]. Quand le nombre de clauses ajoutées n'est pas très élevé (ici, moins de 80 par bases) il semble plus intéressant d'utiliser la minimisation par ajout de clauses. Ensuite, l'utilisation des ensembles de littéraux nécessaires est plus adaptée. On le comprend : ajouter 520 clauses à une base qui en contient au départ 30 change la taille du problème !

Nous avons effectué la même comparaison en prenant VAR=15 et en faisant varier le ratio de 1 à 6. Attention, ici PPOS=0 (pour calculer des P-impliquants), on ne retrouve donc pas le phénomène de seuil classique à PPOS=0,5.

[30] La chute du temps de résolution sur les premières instances est due à la compilation à la volée du bytecode Java.



On retrouve le même phénomène : quand il y a peu de solutions (ici encore moins de 80 pour le ratio 1) l'ajout de clauses est comparable à la technique des compteurs.

La minimisation par ajout de clauses semble donc utilisable dans le cadre de la production de P-impliquants si le nombre de solutions est comparable à la taille de la base initiale. Dans le cas contraire, on lui préférera une méthode de calcul de type Castell par exemple.

4.4.2 Diagrammes de Décision Binaires (BDD)

[Bouquet 1999] présente une étude sur l'efficacité du test de primarité de Castell en le comparant avec l'ajout de clauses et l'utilisation de diagrammes de décision binaire (BDD [Bryant 1986]). Pour cela, il présente l'algorithme DPI qui effectue comme MPL une énumération des impliquants compatible avec l'inclusion. Comme il s'agit de calculer des impliquants et non des impliquants P-restreints, l'algorithme développe 3 branches au lieu de 2 : les impliquants qui ne contiennent ni l ni $\neg l$, ceux qui contiennent l et ceux qui contiennent $\neg l$. Ainsi, tout comme MPL, le premier impliquant trouvé est un impliquant premier. Il s'agit ensuite d'éliminer tous les impliquants sous-sommés par un des impliquants premiers déjà trouvés. Cela se fait toujours comme dans MPL par ajout d'une clause (la négation de l'impliquant) dans une base de minimisation. Une variante utilisant un BDD pour stocker les impliquants premiers et faire le test de sous-sommation est appelée DPI-BDD. L'algorithme de Castell est noté PDP. Les trois méthodes ont été implantées par l'auteur et se différencient uniquement par la méthode de minimisation et la manière dont l'arbre est développé : ordre compatible avec l'inclusion pour les deux premières, heuristique pour la troisième. Les tests sont effectués sur des instances aléatoires et des instances du problème des pigeons.
On peut noter que les bons résultats de PDP par rapport à DPI ou DPI-BDD ne peuvent pas mettre en cause uniquement la méthode de minimisation utilisée. Une lecture attentive des résultats permet de mettre en évidence le fort pouvoir de coupe de la notion de littéraux nécessaires par rapport à une énumération compatible avec l'inclusion ensembliste (PDP effectue beaucoup moins de tests de minimisation que DPI). La méthode PDP est donc la plus efficace dans le cadre du calcul d'impliquants premiers. Malheureusement, nous avons déjà noté que cette technique ne peut pas être utilisée dans le cadre du calcul d'impliquants P-restreints premiers. Nous allons donc nous intéresser à la comparaison entre DPI et DPI-BDD.

Dans la plupart des tests, DPI-BDD est beaucoup plus rapide que DPI, ce qui fait dire à l'auteur que cette dernière méthode est totalement inadéquate. Il faut mesurer ces affirmations :

- On peut noter sur bon nombre de problèmes (MK,3-SAT aléatoires, [Bouquet 1999] tables 6.1 6.2 et 6.3 pages 112-117) DPI effectue plus de tests de sous-sommation que DPI-BDD (sauf dans 3 problèmes aléatoires à 100 variables où c'est le contraire) !

- L'auteur utilise dans le cadre des BDD une librairie spécialisée. Il ne fait pas mention d'une librairie concernant l'implémentation de DPI. Or la performance de la minimisation par ajout de clauses est très fortement liée à la performance globale de l'algorithme (propagation de la satisfaction d'un littéral, restauration du contexte, …).

Nous avons contacté l'auteur pour préciser ces points. Ce ne sont pas les nombres d'appels à Sous-sommation qui sont comptabilisés mais les nombres de clauses de minimisations falsifiées dans le cas de DPI et le nombre de coupures par minimisation dans DPI-BDD. La différence entre les deux vient du fait que dans le cas de DPI, plusieurs clauses peuvent être falsifiées en même temps.

Concernant la performance de l'algorithme et de ces mécanismes de propagation, l'auteur la qualifie d'efficace, celui-ci étant basé sur un prouveur SAT déjà existant et modifié pour l'occasion.

On peut donc supposer que l'utilisation de BDD a un coût qui se justifie à partir d'un certain nombre de solutions (disons Nmin). En deçà, on peut utiliser l'ajout de clauses pour effectuer le test de minimisation. Le choix de l'une ou l'autre des méthodes dépendra donc essentiellement de la performance des mécanismes de propagation et de restauration de l'algorithme d'énumération (dont dépend Nmin) et du nombre de solutions des problèmes.

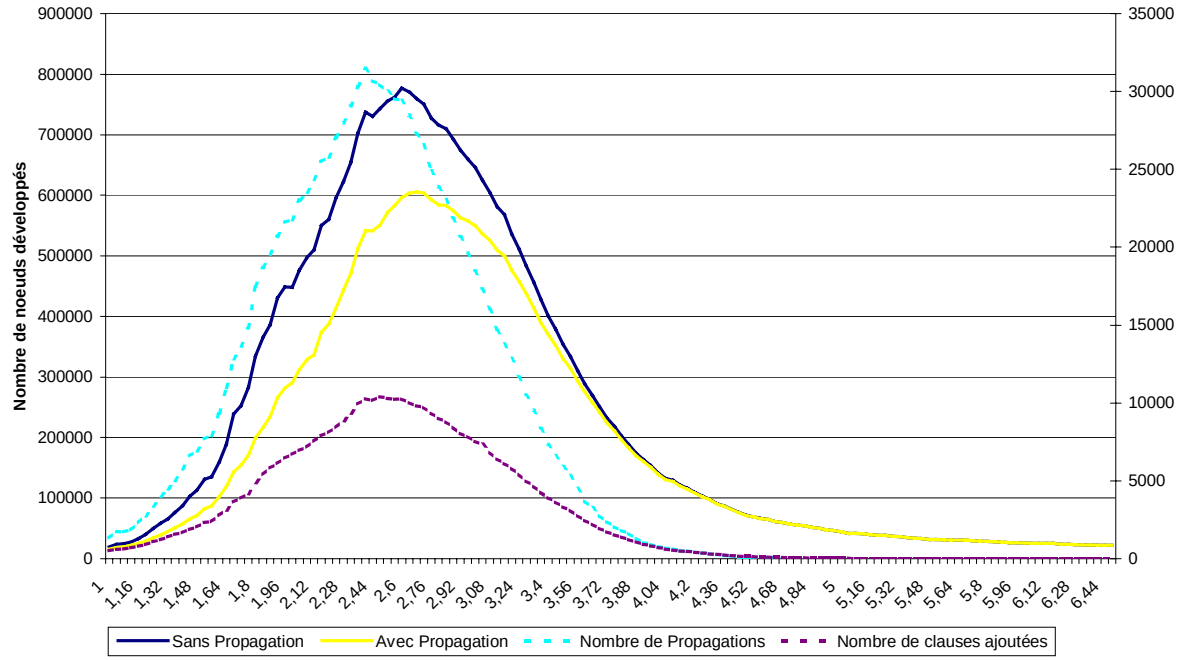
N'oublions pas qu'il faut aussi tenir compte de l'utilisation que l'on doit faire des impliquants (P-restreints) premiers que l'on a calculés. Lorsque nous devons calculer les P-impliqués premiers d'une base, la base des clauses de minimisation est exactement la forme sous laquelle nous devons avoir le résultat du calcul d'impliquants P-restreints : une CNF. Si nous utilisons un BDD pour représenter les impliquants P-restreints premiers, il faudra ajouter au temps de calcul la transformation du BDD en CNF (ou utiliser un algorithme de calcul d'impliquants premiers sur ce BDD).

4.4.3 Propagation des clauses unitaires de la base de minimisation

Un point non mentionné concernant l'ajout de clauses dans la base est la prise en compte des clauses unitaires générées par ces clauses lors de la recherche. Nous avons testé notre algorithme MPL en propageant et sans propager les clauses unitaires produites par la base de minimisation.

Nous avons effectué ces tests sur 100 bases de 50 variables, avec |P|=25. Nous avons fait varier le ratio de 1 à 6,5 afin de faire varier le nombre d'impliquants P-restreints premiers. Nous donnons ici le nombre de nœuds développés au total (incluant la propagation des littéraux purs et des clauses unitaires). Nous avons inclus le nombre de clauses unitaires issues des clauses de minimisations propagées ainsi que le nombre de clauses ajoutées.

Effet de la propagation des clauses unitaires issues des clauses de minimisation



On note que l'intérêt de la propagation des clauses unitaires des clauses de minimisation est bien réel. Au niveau des temps de résolution, on passe dans le meilleur de cas de 198s à 145s par exemple pour résoudre 100 instances.

En conclusion, la minimisation par utilisation de littéraux nécessaires est l'une des meilleures solutions de minimisation dans le cadre du calcul d'impliquants premiers. Nous n'avons malheureusement pas de notion équivalente dans le cadre des impliquants P-restreints premiers. La minimisation par ajout de clauses est une solution qui semble tout à fait envisageable tant que l'on obtient un nombre restreint de solutions ($\leq ?$). Au-delà, on pourra utiliser une représentation plus compacte des impliquants, à l'aide de BDD par exemple. Nous verrons dans la quatrième partie de cette thèse que nous allons utiliser des données exogènes afin de limiter le nombre de solutions produites en ne calculant que les solutions « préférées ».

Peut être qu'une solution intermédiaire serait d'utiliser la notion d'arbre de discrimination (TRIES [Rymon 1992]) pour représenter les clauses (ajoutées). En effet, SATO [Zhang/Stickel 1994] est parmi les meilleurs algorithmes de résolution de SAT à l'heure actuelle. Il doit ses performances à une implantation très fine de la propagation des clauses unitaires et de la représentation des clauses sous forme d'arbre au lieu de simple ensemble. C'est une solution à la lenteur occasionnée par l'ajout de nouvelles clauses dans la base, mais cela ne change rien quant à la représentation compacte des solutions.

Notons que l'utilisation de contraintes pour éviter la productions d'impliquants non premiers est aussi utilisée dans [Palopoli, et al. 1999]. Le calcul des impliquants est basée sur la programmation linéaire en nombre entiers 0-1 : les clauses sont des inégalités à résoudre. Quand un impliquant premier est trouvé, une nouvelle inégalité est ajoutée au problème pour éviter de produire des impliquants conséquence logique de celui-ci.

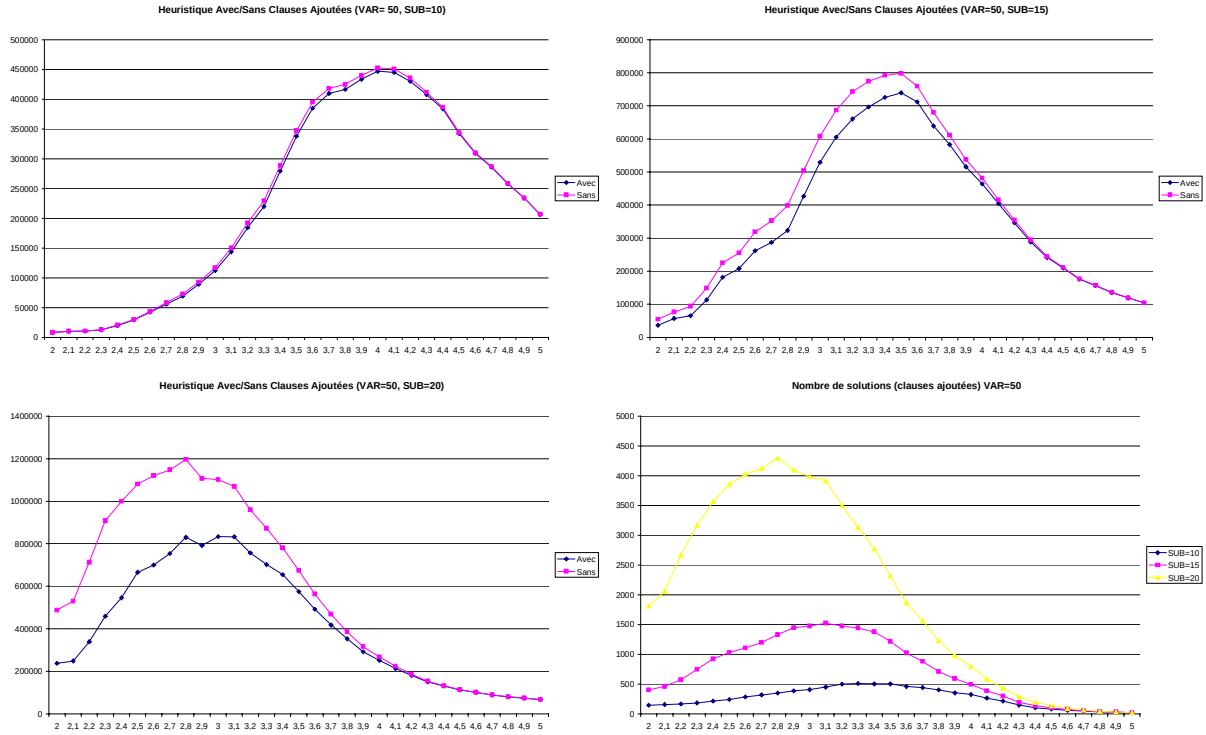
4.5 Et les heuristiques ?

Un dernier point que nous n'avons pas encore abordé et qui se trouve au cœur des préoccupations de la communauté SAT : les heuristiques. Quelles sont les bonnes ou les mauvaises heuristiques dans le cadre du calcul d'impliquant P-restreint ?

Nous n'avons pas cherché une heuristique spécifique à la production d'impliquants P-restreints. Nous pouvons seulement noter quelques observations faites lors de nos tests.

- Comme on pouvait s'y attendre, une heuristique balancée donne globalement de meilleurs résultats qu'une heuristique non balancée. C'est aussi vrai quand la base de clauses contient uniquement des littéraux purs.
- Les heuristiques de type POSIT ou C-SAT (pondérées) n'avaient pas un bon comportement quand nous ne propagions pas les clauses unitaires issues des clauses de minimisation. En prenant en compte ces clauses unitaires, ces heuristiques permettent de limiter quelquefois de manière spectaculaire le nombre de nœuds développés vers un ratio de 3. Quand il y a énormément de solutions (ratio<2), une simple heuristique balancée suffit. On retrouve les comportements observés pour SAT.
- L'heuristique semble perdue lorsque le sous-langage concerne environ 20% des variables propositionnelles.

Une question se pose néanmoins : faut-il comptabiliser dans nos heuristiques les clauses de minimisation ? Il semble que oui. Nous avons effectué des tests sur des bases aléatoires 3-SAT de 50 variables. Nous avons testé pour un ratio compris entre 2 et 5 le nombre de nœuds développés par notre procédure MPL dans 3 configurations : $|P|=10$, $|P|=15$ et $|P|=20$. Augmenter la taille de P permet d'augmenter le nombre et la taille des clauses ajoutées. Nous avons fixé dans un cas le poids des clauses ajoutées à 0 (courbes « Sans ») et dans l'autre le poids des clauses ajoutées est le même que les clauses de la base produite (courbes « Avec »). On observe les résultats suivants :



On note que pour |P|=10, les deux courbes sont comparables. Pour |P|=15, la prise en compte des clauses ajoutées dans l'heuristique permet de réduire un peu le nombre de nœuds développés. Enfin, pour |P|=20, l'intérêt de la prise en compte des clauses de minimisation dans l'heuristique est flagrant. On peut noter sur le dernier graphique que le gain observé dans ces 3 cas semble proportionnel au nombre de clauses de minimisation. Ainsi, l'ajout de clauses de minimisation permet de guider l'heuristique de recherche.

4.6 Conclusion de l'étude

Nous pouvons résumer en quelques points les caractéristiques de notre algorithme :

- La taille de l'arbre développé dépend non seulement du nombre de variables mais aussi de la taille de *P*.
- La minimisation par ajout de clauses semble efficace quand il n'y a pas beaucoup de solutions. Plus qu'une simple méthode de minimisation, elle permet de guider la recherche de solutions : par propagation de clauses unitaires issues de ces clauses, par leur prise en compte dans l'heuristique de recherche. Ceci n'est pas vrai dans le cadre de l'utilisation de BDD pour effectuer la minimisation, par exemple. Néanmoins, dans le cadre d'un calcul de P-impliquant premier, on lui préférera une méthode de minimisation basée sur la notion d'ensembles nécessaires proposée par Thierry Castell.
- **Calcul de P-impliqués premiers : les deux passes ne sont pas identiques :**
  La première sera le plus souvent difficile à calculer. Le plus souvent, il s'agira de calculer un nombre limité de solutions (comparable à la taille de la base). Dans le pire des cas, la taille du résultat obtenu sera exponentiel par rapport à la taille de la base de départ.
  La seconde est très peu coûteuse en temps car elle ne fait qu'énumérer les solutions : on se trouve dans le cas bien particulier d'une base contenant uniquement des littéraux purs. La base est donc totalement sous-contrainte. Néanmoins, lorsqu'un très grand nombre de solutions sont produites, la minimisation par ajout de clauses devient inutilisable. On peut remplacer cette seconde passe par n'importe quel algorithme de calcul d'impliquant premier, celui proposé par Thierry Castell par exemple.

Partie III : Applications & variantes

« Les trois lois fondamentales de la Robotique :
Première Loi : Un robot ne peut nuire à un être humain ni laisser sans assistance un être humain en danger.
Deuxième Loi : Un robot doit obéir aux ordres qui lui sont donnés par les êtres humains, sauf quand ces ordres sont incompatibles avec la Première Loi.
Troisième Loi : Un robot doit protéger sa propre existence tant que cette protection n'est pas incompatible avec la Première ou la Deuxième Loi.

Manuel de la robotique, 58ème édition (2058 après J.-C.) »
Isaac Asimov, Le Grand Livre des Robots

1 Applications

1.1 ATMS/ACMS

Nous avons déjà présenté cet outil dans la partie précédente. Nous indiquons maintenant comment calculer les différents ensembles manipulés par l'ATMS grâce à la procédure MPL.

1.1.1 Calcul des nogoods

Ce calcul ne pose pas de difficultés car un nogood est tout simplement un impliqué particulier de la base (cf. CMS page 56).

Proposition 29 (calcul des nogoods d'une formule)
Soit *BC* une base de clauses. Soit *H* l'ensemble des symboles hypothèses de *BC*. L'ensemble des nogoods de *BC* est exactement l'ensemble des négations des $H^{\neg}$-impliqués premiers de *BC*. Ce qui se traduit en termes de calcul d'impliquants P-restreints en $IRP(IRP(BC,H^{\neg})^{\neg\wedge},H)$.

Preuve : par définition des nogoods en utilisant la Proposition 26 page 64.

Exemple : retournons quelques instants en Bretagne. $BC = BZH' \cup OBS$. $H=\{P,Y,A,D,J\}$. $IRP(BC,H^{\neg}) = \{\neg P,\neg Y,\neg A,\neg D,\neg J\}$. Il faut maintenant calculer les impliquants (H-restreints) de $IRP(BC,H^{\neg})^{\neg\wedge} = \{P, Y, A, D, J\}$. On trouve comme unique impliquant $P \wedge Y \wedge A \wedge D \wedge J$. Ce qui nous donne $NG(BC)=\{\{P,Y,A,D,J\}\}$. Les témoignages ne peuvent pas être tous exacts, i.e. quelqu'un a menti.

1.1.2 Calcul du label d'une formule

Ce calcul est un peu plus compliqué car il s'agit d'enlever lors du calcul du label les nogoods qui pourraient apparaître[31]. Pour ce faire, nous allons ajouter dans la base intermédiaire les nogoods de la base, sous leur forme clausale, c'est à dire niés. Nous allons donc chercher dans la deuxième phase les impliquants P-restreints de la base intermédiaire $BC' \cup NG^{\neg}$. Montrons que ces impliquants P-restreints sont bien des impliquants P-restreints de *BC'*, et que seuls les impliquants P-restreints contenant un nogood sont éliminés.

Lemme 3
Soit *P* un ensemble consistant de littéraux. Soit $BC' \subset FP_S$ une base de clauses contenant uniquement des littéraux de *P*. Soit *BC''* une base de clauses contenant uniquement des littéraux de $P^{\neg}$. Soit $BC = BC' \cup BC''$. $E^{\wedge}$ est un impliquant P-restreint de *BC* ssi $E^{\wedge}$ est un P-impliquant restreint de *BC'* et $BC''^{\wedge} \wedge E^{\wedge} \not\models \bot$.

[31] Etant donné une base de clause *BC*, un environnement *E* et une donnée *d*, vérifier que *E* contient un nogood de *BC* nécessite un test d'inconsistance (classe de complexité co-NP) tandis que vérifier que E contient un élément du label de *d* nécessite un test d'inconsistance et un test de consistance (classe de complexité BH_2). Il est donc normal qu'en pratique, il soit plus difficile de calculer le label d'une donnée que les nogoods.

Preuve :

) $E^\wedge$ est un impliquant P-restreint de BC ssi $\exists$ M tel que $M^\wedge \models BC^\wedge$ et $M \cap P = E$. Donc $M^\wedge \models (BC' \cup BC'')^\wedge \equiv BC'^\wedge \wedge BC''^\wedge$. Donc $M \models BC'$ (1) et $M \models BC''$(2). De (1) on déduit que $E^\wedge$ est un impliquant P-restreint de BC' et de (2) que $BC''^\wedge \wedge E^\wedge \not\models$ (car M est un impliquant de BC'').

$\Leftarrow$) Soit $E \subseteq P$ tel que E est un impliquant P-restreint de BC' et $BC''^\wedge \wedge E^\wedge \not\models$. Donc $\exists M$ tel que $M^\wedge \models BC'^\wedge$ et $M \cap P = E$. Montrons que $M^\wedge \models BC''^\wedge$. Supposons que M falsifie BC''. Comme BC'' contient uniquement des littéraux de $P^\neg$, seuls les littéraux de $M \cap P$ peuvent falsifier BC''. Or $M \cap P = E$ et $BC''^\wedge \wedge E^\wedge \not\models$. Contradiction. Nous avons donc $M^\wedge \models BC'^\wedge$ et $M^\wedge \models BC''^\wedge$, donc $M^\wedge \models (BC' \cup BC'')^\wedge \equiv BC^\wedge$. Comme $E = M \cap P$, $E^\wedge$ est bien un impliquant P-restreint de BC.

Lemme 4

Soit P un ensemble consistant de littéraux. Soit $BC' \in FP_S$ une base de clauses contenant uniquement des littéraux de P. Soit BC'' une base de clauses contenant uniquement des littéraux de $P^\neg$. Soit $BC = BC' \cup BC''$. $E^\wedge$ est un impliquant P-restreint minimal de BC ssi $E^\wedge$ est un P-impliquant restreint minimal de BC' et $BC''^\wedge \wedge E^\wedge \not\models$.

Preuve :

) Supposons que E ne soit pas un impliquant P-restreint minimal pour BC'. Donc $\exists E' \subset E$ un impliquant P-restreint minimal de BC'. $BC''^\wedge \wedge E'^\wedge \not\models \bot$, car sinon $BC''^\wedge \wedge E^\wedge \models \bot$ donc E ne serait pas un impliquant P-restreint minimal de BC. Donc d'après le lemme précédent, $E'^\wedge$ est un impliquant P-restreint de BC. Comme $E' \subset E$, $E^\wedge$ n'est pas un impliquant P-restreint minimal de BC. Contradiction.

$\Leftarrow$) Supposons que E ne soit pas un impliquant P-restreint minimal de BC. Alors $\exists E' \subset E$ un impliquant P-restreint minimal de BC. D'après le lemme précédent, c'est un impliquant P-restreint de BC'. Donc E n'est pas minimal. Contradiction.

Nous sommes maintenant capables, en vertu de la Proposition 18 et du lemme précédent, d'exprimer la notion de label en termes de calcul d'impliquants P-restreints premiers.

Proposition 30 (calcul du label d'une formule)

Soit $BC \subset FP_S$ une base de clauses. Soit H l'ensemble des symboles hypothèses de BC. Soit NG l'ensemble des nogoods de BC (un nogood étant interprété comme un cube). Soit α un cube ou une clause de FP_S. Le label de α est $IRP(IRP(BC^\wedge \wedge \neg\alpha), H^\neg)^{\neg\wedge} \wedge NG^{\neg\wedge}, H)$.

> Nous devons donc calculer les nogoods de BC avant de calculer le label d'une formule.

Exemple (suite) : Si nous cherchons maintenant les témoignages qui corroborent la thèse d'une *action penn sardinn*, pour calculer le label de *penn*, nous allons d'abord calculer les impliquants $H^\neg$-restreints premiers de $BC' = BC \cup \{penn\ \}$. On trouve $IRP(BC', H^\neg) = \{\neg A, \neg P, \neg Y\}$. Nous cherchons maintenant les impliquants P-restreints premiers de $BC'' = \{\ A,\ Y,\ P, A\ Y\ P\ D\ J\ \}$. On trouve $A \wedge Y \wedge P$ (on note qu'il s'agit bien d'un impliquant P-restreint et pas d'un impliquant de BC''). Dans ce cas particulier, nous avons $NG(BC \cup \{penn\ \}) = Label(BC, penn)$.

Si nous prenons maintenant $BC = \{A\ b\ \ c, B\ \ b, c\ \ \}$, $H = \{A,B\}$. $NG = \{\{A,B\}\}$. Si nous cherchons le label de c, nous obtenons $IRP(BC \cup \{c\ \}, H\neg) = \{\neg A, \neg B\}$. Si nous ignorons les nogoods de BC, c'est à dire que nous calculons les nogoods de $BC \cup \{c\ \}$, on obtient $NG(BC \cup \{c\ \}) = \{\{A,B\}\}$. Or $\{A,B\}$ n'est pas un élément du label de c car $\{A,B\}$ est un nogood. C'est pourquoi nous calculons les impliquants de $IRP(BC \cup \{c\ \}, H\neg)^{\neg\wedge} \wedge NG^{\neg\wedge} = \{\ A,\ B,\ A\ B\ \}$. Cette base est inconsistante, elle n'a donc pas d'impliquants. Donc le label de c est vide.

1.1.3 Calcul des interprétations

Une interprétation est un sous-ensemble maximal E de H tel que $BC^\wedge \wedge E^\wedge \not\models \bot$. Donc $BC^\wedge \wedge (E \cup (H \backslash E)^\neg)^\wedge \not\models \bot$. On peut transformer le problème de maximisation en un problème de minimisation en cherchant $E' \subseteq H^\neg$ minimal tel que $BC^\wedge \wedge (E' \cup (H^\neg \backslash E')^\neg)^\wedge \not\models$. C'est à dire E' est un impliquant $H^\neg$-restreint premier de BC.

Nous pouvons donc donner une méthode de calcul des interprétations fondée sur la notion d'impliquant P-restreint premier.

Proposition 31 (calcul des interprétations)
Soit $BC \subset FP_S$ une base de clauses. Soit H l'ensemble des symboles hypothèses de BC. L'ensemble des interprétations de BC est $\{ H \setminus E^{\neg} \mid E^{\wedge} \in IRP(BC,H^{\neg})\}$[32].

> Ce résultat est intéressant car il montre qu'il est plus facile, pour notre méthode, de calculer l'ensemble des interprétations que l'ensemble des nogoods ([Verfaillie/Lobjois 1999] retrouvent ce même résultat dans le cadre de CSP). Dans la pratique, comme les interprétations sont obtenues à l'aide de la première étape du calcul des nogoods, nous calculerons toujours dans notre prototype les interprétations d'une base de clauses (le calcul des nogoods étant nécessaire pour calculer le label d'une donnée).

Exemple (suite) : $IRP(BC,H^{\neg}) = \{\neg P,\neg Y,\neg A,\neg D,\neg J\}$. $H = \{P,Y,A,D,J\}$. Donc $INTER(BC) = \{\{Y,A,D,J\}, \{P,A,D,J\}, \{P,Y,D,J\},\{P,Y,A,J\},\{P,Y,A,D\}\}$.

1.1.4 Calcul du contexte d'un environnement

Il nous reste enfin à montrer comment calculer le contexte d'un environnement à l'aide des impliquants P-restreints.

Nous cherchons les données $d \in S$ telles que $BC^{\wedge} \wedge E^{\wedge} \models d$. Or d'après la Proposition 22 page 62, nous avons $BC^{\wedge} \wedge E^{\wedge} \models d$ ssi $IRP(BC^{\wedge} \wedge E^{\wedge},S) \models d$. Donc d appartient à chaque impliquant S-restreint premier de BC, donc le contexte d'un environnement E dans BC est l'intersection des impliquants S-restreints de $BC^{\wedge} \wedge E^{\wedge}$.

Proposition 32 (calcul du contexte)
Soit $BC \subset FP_S$ une base de clauses. Soit H l'ensemble des symboles hypothèses de BC. Soit $E \subseteq H$. Le contexte de E dans BC est contexte(BC,E) = $\bigcap_{F^{\wedge} \in IRP(BC^{\wedge} \wedge E^{\wedge},S)} F$.

Exemple (suite) : nous cherchons maintenant le contexte de E={P,Y,D,J}, c'est à dire une extension de BC. IRP(BC^∧E^,S) = {big∧yb∧freg∧festnoz∧pb∧psg∧Y∧P∧J∧D}. On ne déduit Contexte(BC,E) = {big, yb, freg, festnoz, pb, psg, Y, P, J, D}.

Nous pouvons récapituler ces résultats dans le tableau suivant :

	Nogoods	*Interprétations*	*Label de d*	*Contexte de E*
Première passe	F = $IRP(BC,H^{\neg})$		F = $IRP(BC^{\wedge} \wedge (\neg d),H^{\neg})$	F = $IRP(BC^{\wedge} \wedge E^{\wedge},S)$
Résultat		$\{H \setminus E \mid E^{\wedge} \in F\}$		$\cap E \mid E^{\wedge} \in F$
Deuxième passe	NG = $IRP(F^{\neg},H)$		LB = $IRP(F^{\neg} \wedge NG^{\neg},H)$	
Résultat	NG		LB	

1.2 CWR

Différentes techniques de raisonnement non monotone issues du domaine des bases de données sont regroupées sous le sigle de CWR (« *Closed World Reasoning* »). En effet, parce qu'il serait trop coûteux (en terme d'espace mémoire) de stocker explicitement toute la connaissance, on se borne à décrire le plus souvent les faits positifs. Puis tout fait positif (« vrai ») qui n'est pas logiquement déductible de la base est considéré comme « faux ». Si nous reprenons l'exemple du début de cette partie, nous pouvons remarquer que l'on ne sait rien concernant la présence de Per au Festnoz ce soir là. On suppose donc qu'il n'y était pas. Nous devons noter que le raisonnement en monde clos est défini dans le cadre de la logique du premier ordre. Nous nous intéressons ici à sa version propositionnelle.

[32] Nous pouvons de plus vérifier ce résultat grâce à la Proposition 14, page 55. Nous avons souligné que les interprétations pouvaient se calculer à partir des nogoods de la base en effectuant un « minimal hitting set » sur l'ensemble des nogoods. Il suffit alors de retrancher chaque ensemble obtenu à H pour obtenir les interprétations de BC. Nous avons aussi montré que le deuxième calcul d'impliquants P-restreints est équivalent à un calcul de « minimal hitting set » (car il est effectué sur une base pure). Si l'on note F l'ensemble obtenu après le premier calcul d'impliquants P-restreints premiers et NG l'ensemble des nogoods, on obtient donc $NG = MHS(F^{\neg})$. Or on peut montrer que lorsque E est un ensemble d'ensembles minimaux pour l'inclusion et non contradictoires, on a $MHS(MHS(E))=E$. En utilisant la Proposition 14, on trouve donc : $INTER = \{H \setminus E \mid E \in MHS(NG)\} = \{H \setminus E \mid E \in MHS(MHS(F^{\neg})\} = \{H \setminus E \mid E \in F^{\neg}\}$.

D'un point de vue purement logique, cela revient à ajouter dans la base de connaissances la négation de formules (des littéraux négatifs ou des disjonctions de littéraux négatifs) non déductibles de la base. On ne considère pas des formules quelconques : seules celles appelées *ffn* (« *free for negation* ») seront utilisées. Les différentes techniques de raisonnement de CWR se différencient par leur calcul des *ffn*. Le raisonnement s'effectue ensuite « classiquement » sur la base augmentée des formules *ffn* niées (notée XXX(B) où XXX dénote la technique suivie).

1.2.1 Définitions

Définition 44 (Raisonnement en monde clos)
Soient α et γ deux formules de FP_S. γ est une conséquence de α sous l'hypothèse XXX ssi $\alpha \wedge \{\neg\beta \mid \beta$ est XXX-ffn dans $\alpha\}^{\wedge} \models \gamma$ (que l'on notera aussi XXX(α) $\models \gamma$) avec XXX$\in$ {CWA,GCWA,EGCWA,CCWA,ECWA}.

La première technique, sans doute la plus connue, a été introduite par [Reiter 1978]. L'hypothèse du monde clos (CWA, « *Closed World Assumption* »), que certains appellent aussi hypothèse du monde clos naïve (NCWA, « *Naive CWA* »), correspond exactement au principe que nous venons d'énoncer. Tout fait positif non déductible classiquement d'une base de connaissances est CWA-ffn.

Définition 45 (Closed World Assumption)
Soient α et β deux formules de FP_S. β est « free for negation » dans l'hypothèse du monde clos (CWA-ffn) dans α ssi β est un littéral positif tel que $\alpha \not\models \beta$.

Reprenons notre base *BZH* avec cette *fois OBS = { psg, festnoz, penn big}*. De *BZH* $\cup$ *OBS*, nous trouvons que seuls les faits *psg* et *festnoz* ne sont pas CWA-ffn dans *BZH*$\cup$*OBS*. On ajoute donc dans la base les négations des autres faits (*chez, rhu, big, freg, pb, yb, Y, P, J, D, A, penn*) dont les clauses *penn* et *big* (c'est à dire que ni les penn sardinns ni les bigoudens ne sont dans le coup). Or OBS contient la clause *penn big* (ce sont soit les penn sardinns soit les bigoudens qui ont fait le coup). *BZH*$\cup$ *OBS* était consistante. *CWA(BZH*$\cup$*OBS)* devient inconsistante. Cela provient de l'utilisation de disjonction de littéraux positifs (des clauses qui ne sont pas de Horn).

> On peut noter que si l'on ajoute le fait *big* dans la base initiale, alors CWA(BZH$\cup$OBS$\cup${ big}) devient consistante. Le raisonnement en monde clos est donc non monotone car l'ajout de connaissances dans la base initiale ne garantit pas que les faits déjà déduits sont déductibles de la nouvelle base (XXX(α) $\models \gamma$ / XXX($\alpha\wedge\beta$) $\models \gamma$).

Afin de résoudre ce problème, [Minker 1982] introduit l'hypothèse du monde clos généralisée (GCWA « *Generalized CWA* »).

Définition 46 (Generalized Closed World Assumption)
Soient α et β deux formules de FP_S. β est « free for negation » dans l'hypothèse du monde clos généralisée (GCWA-ffn) dans α ssi β est un littéral positif tel que pour toute clause γ, si $\alpha \not\models \gamma$ alors $\alpha \not\models \gamma \vee \beta$.

Cette définition est beaucoup plus prudente que la précédente. Si nous reprenons notre exemple, l'ensemble des faits GCWA-ffn devient (*chez, rhu, freg, pb, yb, Y, P, J, D, A*), c'est à dire que nous n'avons plus aucune information sur penn et big, ce qui permet d'éviter l'incohérence de GCWA(BZH$\cup$OBS).

Une extension de cette approche a été développée par [Yahya/Henschen 1985] : l'hypothèse du monde clos généralisée étendue (EGCWA, « *Extended GCWA* »). Alors que les deux approches précédentes ne permettaient que l'utilisation de littéraux positifs comme formules ffn, les auteurs permettent l'utilisation de conjonctions de littéraux positifs.

Définition 47 (Extended Generalized Closed World Assumption)
Soient α et β deux formules de FP_S. β est « free for negation » dans l'hypothèse du monde clos généralisée (EGCWA-ffn) dans α ssi β est une conjonction de littéraux positifs telle que pour toute clause γ, si $\alpha \not\models \gamma$ alors $\alpha \not\models \gamma \vee \beta$.

Dans notre exemple, les formules EGCWA-ffn sont (*chez, rhu, freg, pb, yb, Y, P, J, D, A,penn∧big*). On remarque que l'on retrouve les formules qui étaient GCWA-ffn plus la conjonction de *penn* et *big*. Nous avons maintenant une information sur toutes les variables propositionnelles de la base.

Les deux dernières approches sont des variantes de GCWA et EGCWA dans le cadre d'une application du raisonnement en monde clos à un sous-ensemble des symboles de la formule. On partitionne maintenant les symboles en 3 ensembles : *P* l'ensemble des symboles que l'on préfère falsifier (on dit aussi que l'on veut minimiser la satisfaction de ces atomes), *Q* l'ensemble des symboles dont la valeur de vérité ne peut pas varier lorsque l'on essaye de falsifier un élément de *P* et enfin *Z* l'ensemble des symboles dont la valeur de vérité peut varier lorsque l'on essaye de falsifier un élément de *P*. <P;Q;Z> dénotera dans la suite du document une partition P,Q,Z de S.

L'hypothèse du monde clos prudente (CCWA, « Careful CWA ») voit le jour dans [Gelfond/Przymusinska 1986].

Définition 48 (Careful Closed World Assumption)
Soient α et β deux formules de FP_S. Soit $S = <P;Q;Z>$. β est « free for negation » dans l'hypothèse du monde clos prudente (CCWA-ffn) dans α ssi β est un littéral positif de P tel que pour toute clause $C \subseteq P \cup L_Q$, si $\alpha \not\models C^{\vee}$ alors $\alpha \not\models C^{\vee} \vee \beta$.

Prenons par exemple $P=\{big, penn\}$, $Q=\{psg,festnoz\}$ et $Z=\{pb,yb,chez,freg,rhu\}$. On ne trouve pas de formule CCWA-ffn dans notre exemple.

[Gelfond, et al. 1989] introduisent enfin l'hypothèse du monde clos étendue (ECWA « Extended CWA »), qui est à CCWA ce que EGCWA est à GCWA : la possibilité de considérer des conjonctions de littéraux positifs.

Définition 49 (Extended Closed World Assumption)
Soient α et β deux formules de FP_S. Soit $S = <P;Q;Z>$. $B^{\wedge}$ est « free for negation » dans l'hypothèse du monde clos étendue (ECWA-ffn) dans α ssi $B \subseteq P \cup Q$ tel que pour toute clause $C \subseteq P \cup L_Q$, si $\alpha \not\models C^{\vee}$ alors $\alpha \not\models C^{\vee} \vee B^{\wedge}$.

En gardant notre partition des symboles, on trouve maintenant que *big∧penn* est la seule formule ECWA-ffn de $BZH \cup OBS$.

1.2.2 Calculer les formules ffn

La caractérisation des formules ffn peut sembler ardue au premier abord. Les deux premières approches s'expriment facilement à partir de la notion de modèles minimaux de [Lifschitz 1985].

Définition 50 (modèles minimaux de Lifschitz)
Soit α une formule de FP_S et $S=<P;Q;Z>$. Soient M et N deux modèles de α. On écrira $M \leq_{(P;Q;Z)} N$ ssi M et N assignent la même valeur aux symboles de Q et l'ensemble des symboles de P associés à vrai dans M est inclus dans l'ensemble analogue de N. $\leq_{(P;Q;Z)}$ est un préordre partiel sur les modèles de α et tout modèle de α qui est minimal pour $\leq_{(P;Q;Z)}$ est appelé un modèle $<P;Q;Z>$-minimal de α. Un modèle $<S;\varnothing;\varnothing>$-minimal de α est appelé tout simplement modèle minimal de α. Pour alléger la notation, comme P et Z suffisent pour déterminer la partition de S, nous noterons dans la suite du document « modèle $<P;Z>$-minimal » un modèle $<P;Q;Z>$-minimal.

Remarque : [Raiman/de-Kleer 1992] proposent un ATMS capable de raisonner à partir de modèles minimaux de Lifschitz : le Minimality Maintenance System (MMS). Les environnements ne sont plus minimaux pour l'inclusion mais minimaux pour la cardinalité selon une politique de circonscription <P ;Q ;Z>.

Nous pouvons maintenant caractériser certains ensembles de formules ffn en termes de modèles minimaux [Cadoli 1995, page 39].

β est CWA-ffn dans α ssi β est un littéral positif et il existe un modèle M de α tel que $M \not\models \beta$ (c'est à dire que $\neg\beta \in$ M, que β n'est pas conséquence logique de α).
β est GCWA-ffn dans α ssi β est un littéral positif et pour tout modèle minimal M de α, $M \not\models \beta$ (c'est à dire que $\neg\beta$ appartient à tous les modèles minimaux de α).

β est CCWA-ffn dans α ssi β est un élément de P et pour tout modèle (P;Z)-minimal M de α, $M \not\models \beta$ (c'est à dire que $\neg\beta$ appartient à tous les modèles <P; Z>-minimaux de α).

La notion d'impliquant P-restreint premier est très proche de la notion de modèle <P;Q;Z>-minimal de Lifschitz : il suffit de prendre Z=S\P et Q=∅. Nous n'avons pas de correspondance pour Q. Il existe un cas particulier (autre que Q=∅) pour lequel la correspondance entre ces deux notions existe : il suffit que tous les modèles de la base aient la même valeur de vérité sur les variables de Q, c'est à dire que ∀q∈Q, BC ⊨ q ou BC ⊨ ¬q.

Propriété 6 (Relation entre un impliquant P-restreint premier et un modèle <P ;Q ;Z>-minimal)
Soit α une formule de FP_S. Soit M et N deux modèles de α et $<P;Q;Z>$ une partition des symboles de α. M est un modèle $<P;Z>$-minimal de α ssi $M \cap P$ est un impliquant P-restreint premier de α ∀q∈Q, BC ⊨ q ou BC ⊨ ¬q.

Preuve :
) Soit M un modèle $<P;Z>$-minimal de α. M est un modèle de α. $E=M\cap P$ est donc un impliquant P-restreint de α. Supposons que E ne soit pas minimal. Donc $\exists E' \subset E$ tel que E' est un impliquant P-restreint de α. Donc $\exists N$ un modèle de α tel que $E'=N\cap P$. Donc $N\cap P \subset M\cap P$. Comme ∀q∈Q, BC ⊨ q ou BC ⊨ ¬q, $N <_{<P;Z>} M$. Donc M n'est pas $<P;Z>$-minimal. Contradiction.
⇐) Soit E un impliquant P-restreint premier de α.Donc $\exists M$ un modèle de α tel que $E=M\cap P$. Supposons que M ne soit pas <P;Z>-minimal. Donc $\exists N$ modèle $<P;Z>$-minimal de α tel que $N <_{<P;Z>} M$. Donc $N\cap P \subset M\cap P=E$. Donc E n'est pas un impliquant P-restreint premier. Contradiction.

Il est donc possible de proposer une caractérisation de ces trois ensembles de formules *ffn* en termes d'impliquants P-restreints :

- CWA-ffn(α)= $\bigvee_{E^\wedge \in IRP(\alpha,S)} (S \setminus E)$
- GCWA-ffn(α)= $\bigwedge_{E^\wedge \in IRP(\alpha,S)} (S \setminus E)$
- CCWA-ffn(α)= $\bigwedge_{E^\wedge \in IRP(\alpha,P)} (P \setminus E)$

Les deux autres politiques s'expriment plus facilement en termes d'impliqués [Marquis 1999, proposition 4.6].

- β est EGCWA-ffn dans α ssi β est une conjonction de littéraux positifs et $\neg\beta$ est un $S^\neg$-impliqué premier de α.
- B est ECWA-ffn dans α ssi $B \subseteq P \cup L_Q$ et $B^{\neg\vee}$ est un $P^\neg \cup L_Q$-impliqué premier de α.

Nous avons caractérisé la notion de P-impliqués premiers en termes d'impliquants P-restreints dans la section 2.4 page 69. On peut noter pour le premier cas que $S^\neg$ est bien un ensemble consistant de littéraux. Par contre, dans le cas général, $P^\neg \cup L_Q$ n'est pas consistant. Néanmoins, en posant ∀q∈Q, BC ⊨ q ou BC ⊨ ¬q, on peut réduire L_Q aux seuls littéraux satisfaits de Q, on obtient bien $P^\neg \cup L_Q$ un ensemble consistant de littéraux.

Nous pouvons donc utiliser notre algorithme de calcul d'impliquants P-restreints premiers pour calculer les formules *ffn* d'une base de connaissances α selon les différentes politiques de CWR. Nous rappelons que ces résultats sont valables pour CCWA et ECWA uniquement si ∀q∈Q, BC ⊨ q ou BC ⊨ ¬q (sinon il faut effectuer un calcul d'impliquants P-restreints avec P *inconsistant*).

Nous pouvons utiliser les mêmes principes que ceux utilisés dans MPL pour calculer exactement les modèles <P ;Q ;Z>-minimaux de Lifschitz. En effet, seuls les modèles ayant la même valeur de vérité sur les variables de Q sont comparables. Nous pouvons donc commencer par effectuer les branchements sur les variables de Q (pour fixer une valeur de vérité à ces variables), et quand celles-ci sont assignées, utiliser MPL pour calculer les modèles minimaux associés. Les clauses de la base ne seront plus de la forme $(M\cap P)^\neg$ mais de la forme $(M\cap(P\cup L_Q))^\neg$.

1.2.3 Calculer des formules ffn : pourquoi faire ?[33]

On peut en effet se poser cette question car il n'est pas nécessaire de calculer l'ensemble des formules XXX-ffn d'une formule α pour savoir si une formule β est déductible de XXX(α)(voir [Marquis 1999] par exemple).

Néanmoins, un argument en faveur de cette solution est lié à la « compacité » de la représentation. Imaginons en effet que nous ayons une base de connaissances α et que nous ayons un certain nombre de requêtes à adresser à α (pour savoir si une formule CNF est déductible de α). Supposons que nous pouvons effectuer un prétraitement (hors-ligne) sur α avant de lui adresser des requêtes. Etant donné une formule propositionnelle α et une politique de circonscription <P;Q;Z>, déterminer si une formule CNF γ vérifie ECWA(α) $\models \gamma$ est Π^2_p-complet. Appelons INFERENCE ce problème de décision. Pour limiter la complexité d'INFERENCE, on peut « compiler », i.e. prétraiter les données en calculant :

- les formules β_1, ..., β_n qui sont ECWA-ffn pour α afin de calculer explicitement ECWA(α). Si on suppose que β_1,..., β_n sont données, INFERENCE « devient » coNP-complet (en fait, ce n'est plus le même problème de décision) : il suffit de tester la consistance de ECWA(α)$\wedge\neg\gamma$.
- les modèles m_1, ..., m_p qui sont <*P;Z*>-minimaux pour α puisque ECWA(α) $\equiv m_1 \vee ... \vee m_p$. Dans ce cas, INFERENCE « devient » polynomial.

C'est à dire que l'on peut représenter ECWA(α) en *DNF* ou en *CNF*. Malheureusement, on ne peut pas savoir *a priori* quelle solution permet la représentation la plus compacte de la base, voire même si une telle compilation (selon la première ou la deuxième approche) est intéressante. En effet, dans le pire des cas, le nombre de formules ECWA-ffn ou de modèles <P;Z>-minimaux à considérer peut être exponentiel relativement à la taille de α. En outre, il se peut que le nombre de formules ECWA-ffn que l'on obtient soit exponentiellement plus petit que le nombre de modèles <P;Z>-minimaux de α (et inversement).

On se retrouve donc en face de trois possibilités devant α : compiler en CNF, compiler en DNF, ne rien faire. On peut imaginer une méthode de compilation qui tienne compte de notre utilisation du calcul d'impliquants P-restreints premiers pour calculer les P-impliqués premiers d'une formule :

1. Calculer les impliquants P-restreints premiers de α. Si leur nombre est comparable à la taille de α, utiliser leur disjonction pour effectuer des requêtes.
2. Si le nombre des impliquants P-restreints premiers est exponentiel par rapport à α, calculer les $P\cup L_Q$-impliqués premiers de α, à partir des impliquants P-restreints premiers déjà calculés. Si leur nombre est comparable à la taille de α, utiliser leur conjonction pour effectuer les requêtes.
3. Sinon, utiliser α pour effectuer les requêtes.

Notons que les concepteurs de ECWA utilisent une méthode fondée sur la résolution, MILO (MInimal model Linear Ordered resolution) afin de produire les formules ECWA-ffn de α.

1.2.4 Déduction basée sur les modèles minimaux

Nous allons maintenant proposer une variante de notre algorithme permettant de déterminer si une clause ou un cube se déduit minimalement d'une base de clauses (en utilisant le même principe que [Niemelä 1996] présenté page 147).

Définition 51 (Inférence minimale)
Soit $BC \subseteq FP_S$ une base de clauses. Soit α une formule de FP_S. BC $\vdash_{min,<P,Z>} \alpha$ ssi α est vraie dans tous les modèles <P,Z>-minimaux de *BC*.

Corollaire 2
Soit $BC \subseteq FP_S$ une base de clauses. Soit α une formule de FP_S. BC $\vdash_{min,<P,Z>} \alpha$ ssi $\exists$M un modèle <P ;Z>-minimal de BC qui satisfait $\neg\alpha$.

Nous pouvons en déduire un algorithme permettant de déterminer si une formule est inférée minimalement par BC. L'idée est la suivante : pour chaque modèle <P ;Z>-minimal de BC, vérifier s'il falsifie $\neg\alpha$. Dans l'affirmative, l'algorithme cherche un nouveau modèle <P ;Z>-minimal, sinon il s'arrête car il a trouvé un contre-exemple.

```
EntailsMPL(BC,BA,α,P,IP)
// Calcule les impliquants P-restreints premiers de BC.
// IP contient les littéraux satisfaits.
```

[33] Merci à Pierre Marquis pour ses informations et nos discussions sur le sujet.

```
// BA est la base de minimisation : le résultat se trouve
// sous forme clausale (niée) dans cette base (son contenu peut
// varier après un appel à EntailsMPL.
// α est une formule CNF
// Retourne faux si α est satisfaite par un modèle minimal de BC,
// c'est à dire vrai si ¬α est inférée minimalement par BC
Si    ∈ BC ou    ∈ BA∧IP alors retourner vrai Finsi;
Si BC = ∅ alors
       // on a trouvé un impliquant P-restreint premier de BC
       // on vérifie que le modèle minimal associé ne satisfait pas α
       // il suffit de falsifier les variables de P non assignées
       Si DP(α∧(P¯)\((IP∩P¯) ∪(IP∩P)¯)))=vrai alors
              // ¬α ne peut pas être inférée par BC.
              // on a trouvé un contre exemple. La recherche
              // s'arrête.
              Retourner faux
       Finsi ;
       BA ← BA ∪ {((IP∩P)¯^)} ;
       retourner faux
Finsi ;
l ← LittéralPourSimplifier(BC,BA,P) ;
Si (l≠null) alors  // si l existe
       Retourner EntailsMPL(BC[l],BA,α[l],P,IP∪{l})
Finsi ;
l ← Choix_Symbole_de_P(BC,P) ;
Si (l≠null) alors // il reste des variables de P à assigner
       // on arrête dès que l'on trouve un contre exemple
       Si EntailsMPL(BC[l],BA,α[¬l],P,IP∪{¬l})=vrai alors
              Retourner EntailsMPL(BC[l],BA,α[l],P,IP∪{l}}
       Sinon
              Retourner faux
       Finsi
Sinon  // il ne reste plus de variables de P à assigner
       // on teste uniquement la consistance de BC
       Si DP(BC)=vrai alors
              Si DP(α∧(P¯)\((IP∩P¯)∪(IP∩P)¯))=vrai alors
                     Retourner faux
              Finsi ;
              BA ← BA ∪ {(IP∩P)^¯}
       Finsi ;
       Retourner vrai ;
Finsi
```

Algorithme 9 EntailsMPL : Inférence minimale à l'aide de MPL

1.3 Diagnostic

On retrouve en diagnostic les notions d'impliquants P-restreints et de P-impliquants premiers. Dans ce cadre, on parle de diagnostic minimal consistant et de diagnostic noyau. Généralement on se situe dans le cadre du premier ordre, mais comme le plus souvent il n'y a pas de fonctions et que les domaines sont finis, on peut donc toujours se ramener au cadre propositionnel.

Nous nous situons dans le cadre du diagnostic à base de modèles (model-based diagnosis) où le système à diagnostiquer est représenté logiquement dans une base de connaissances *SD* (System Description). *Comp* dénote un ensemble de composants du système, et un prédicat *Ab(c)* indique qu'un composant *c∈Comp* fonctionne anormalement. *SD* peut comporter une description de bon fonctionnement du composant (fonctionnement normal) ou une description de la panne du composant. On dispose enfin d'une observation *OBS* du système « réel », provenant de capteurs par exemple.

1.3.1 Modèle de bon fonctionnement

Les règles de SD auront la forme de défauts : Ai Bi Ab(c). « Si Ai est vrai alors Bi est vrai ou le composant c fonctionne anormalement ». On va comparer cette observation avec le comportement attendu du système. Bien sur, le cas idéal est d'obtenir une observation conforme au bon fonctionnement du système, c'est à dire SD ∪ OBS ∪ {{¬Ab(c) | c∈Comp}^} consistant. Dans le cas contraire, il faut effectuer un diagnostic.

Définition 52 (diagnostic [Reiter 1987])
Soit SD une description d'un système. Soit OBS une observation du système. Un diagnostic de SD∪OBS est un ensemble de composants. Δ⊆Comp tel que SD ∪ OBS ∪ {({Ab(c) | c∈Δ}∪{¬Ab(c) | c∈Comp\Δ})∧} ⊭⊥.

Exemple : nous pouvons reprendre la base BZH et l'exprimer à l'aide de prédicats d'anormalités. Ils signifieront que le témoin ne peut pas dire la vérité (COMP = {p,y,a,d,j}).
BZHAb *{psg pb Ab(p), chez pb Ab(p), psg chez ,*
chez yb Ab(y), festnoz yb Ab(y), chez festnoz ,
pb yb penn Ab(a),
festnoz freg Ab(d), rhu freg Ab(d), festnoz rhu ,
freg yb big Ab(j)}
Les policiers ont observé OBS = *{ festnoz, psg, penn big }.*
On vérifie que BZHAb ∪ OBS ∪ {{¬Ab(c) | c∈Comp}^} est inconsistant. Δ = { p, y } est un diagnostic de BZHAb ∪ OBS.

Pour réduire le nombre de diagnostics, Reiter utilise le principe de parcimonie[34] : seuls les diagnostics minimaux sont calculés car ils permettent de représenter tous les diagnostics du système si la description du système est un modèle de bon fonctionnement, i.e. Ab(c) n'apparaît que positivement dans SD.

Définition 53 (diagnostic minimal [Reiter 1987])
Soit SD une description d'un système. Soit OBS une observation du système. Un diagnostic Δ de SD∪OBS est appelé un diagnostic minimal ssi ∄ Δ' un diagnostic de SD∪OBS tel que Δ'⊂Δ.

Proposition 33 [de-Kleer, et al. 1992]
Soit (SD,Comp,OBS) une représentation d'un problème de diagnostic tel que Ab(c) apparaît uniquement positivement dans la forme clausale de SD. Pour tout ensemble de composant Δ ⊆ Comp, si Δ' ⊂ Δ pour un diagnostic minimal Δ', alors Δ est lui-même un diagnostic.

Dans BZHAb ∪ OBS, les diagnostics minimaux sont {a}, {p}, {y}, {d} et {j}.

On peut noter que l'on distingue ici le prédicat d'anormalité Ab(c). Si l'on se ramène au cas propositionnel, on distingue donc les littéraux AB = { Ab(c) | c∈Comp}. Si l'on prend BC = SD ∪ OBS, on cherche donc E⊆AB minimal tel que BC∪ {E∧}∪{¬(AB\E)∧} ⊭⊥. A l'aide de la Proposition 19 page 61 on retombe sur la notion d'impliquants AB-restreints premiers avec AB un ensemble consistant de littéraux. On peut donc utiliser MPL pour calculer des diagnostics minimaux (cf. Etude des deux passes page 82).

1.3.2 Modèle de mauvais fonctionnement

Les règles de SD peuvent aussi être de la forme Ab(c) Ai Bi. « Si le composant c fonctionne anormalement et que Ai est vrai alors Bi est vrai ». Le diagnostic ne se fera plus uniquement sur les littéraux d'anormalité positifs.

Définition 54 (diagnostic [de-Kleer, et al. 1992])
Soit SD la description d'un système, Comp un ensemble de composants et OBS une observation du système. Une conjonction C de Ab-littéraux[35] est un diagnostic de (SD,Comp,OBS) ssi SD ∪ OBS ∪ C ⊭⊥.

A la différence de la notion de diagnostic de Reiter, ici l'information indiquant qu'un composant fonctionne normalement est importante.

Exemple : SD = { a b Ab(c2), Ab(c1) a }, OBS = { b} et COMP = {c1,c2}. ¬Ab(c1) et Ab(c2)∧Ab(c1) sont des diagnostics de SD ∪ OBS. On obtient donc à la fois des informations de bon fonctionnement et de mauvais fonctionnement.

Comme le nombre de diagnostics d'un système peut être très important, il faut les représenter de manière compacte. Le critère de minimalité utilisé dans ce cadre diffère légèrement de l'inclusion ensembliste utilisée par Reiter (il s'agit d'une relation d'inclusion entre littéraux et non plus entre variables). [de-Kleer, et al. 1992] disent qu'une conjonction C^ de littéraux *couvre* une autre conjonction D^ de littéraux ssi $C \subseteq D$.

[34] Ce terme s'applique généralement à la minimalité pour la cardinalité et non à l'inclusion ensembliste mais l'idée est la même.
[35] Un Ab-littéral est un élément de $L_{AB} = \{Ab(c) \mid c \in Comp\} \cup \{\neg Ab(c) \mid c \in Comp\}$.

Définition 55 (diagnostic partiel [de-Kleer, et al. 1992])
Un diagnostic partiel est un diagnostic tel que toute conjonction d'Ab-littéraux qu'il couvre est un diagnostic.

Nous pouvons traduire cette notion de diagnostic partiel en termes d'impliquants P-restreints. On cherche $C \subseteq L_{AB}$ tel que BC ∪ OBS ∪ {C^}⊭⊥ et ∀ C' tel que C⊆C', BC ∪ OBS ∪ {C^} ⊭⊥. Donc ∃ un **impliquant** I de BC ∪ OBS tel que $I \cap L_{AB} = C$. Donc C est un impliquant P-restreint de BC ∪ OBS avec $P = L_{AB}$. Notons que P est un ensemble inconsistant de littéraux et que si *C* n'est pas une Ab-interprétation, alors il n'est pas possible de trouver un **modèle** M de BC ∪ OBS tel que $M \cap L_{AB} = C$.

Définition 56 (diagnostic noyau, « kernel diagnosis » [de-Kleer, et al. 1992])
Un diagnostic partiel est appelé diagnostic noyau si aucun autre diagnostic ne le couvre.

Nous retrouvons maintenant la notion d'impliquant P-restreint premier.

La notion d'impliquant P-restreint premier est donc une notion qui n'est pas nouvelle en diagnostic, sous sa forme générale (diagnostic noyau) ou sous sa forme simplifiée (diagnostic consistant à la Reiter).

1.4 Décision dans l'incertain

Nous présentons ici une application de notre calcul d'impliquants P-restreints premiers dans le cadre de la théorie de la décision. Il s'agit pour un agent disposant de ses propres connaissances de prendre une décision, c'est à dire de choisir un ensemble d'actions à effectuer parmi un ensemble d'actions envisageables, pour atteindre un but. Ce but peut être exprimé à l'aide de préférences graduelles (on le mesure alors à l'aide d'utilités ou de coûts). On retrouve ce problème en planification (les décisions sont des plans) en diagnostic (dans ce cas, les décisions sont des réparations), etc. La décision dans l'incertain s'applique lorsque l'agent dispose de croyances (connaissances incertaines). Il faut alors déterminer des critères entre actions pour prendre la meilleure décision. Le critère le plus communément utilisé est celui de l'utilité espérée : on dispose dans ce cadre de distributions de probabilités pour modéliser l'incertitudes des connaissances et d'une fonction d'utilité à valeurs réelles pour modéliser les préférences. Il existe cependant des problèmes qui ne peuvent pas être représentés par ce critère [Allais 53, Ellsberg 1961].
[Sabbadin 1998] présente deux approches particulières de la décision sous incertitude, basées sur la logique possibiliste, dans laquelle les croyances et les préférences sont qualitatives. Nous montrons que ces deux approches peuvent être exprimées en termes d'impliquant P-restreints et de P-impliqués premiers. Nous proposons ensuite deux algorithmes de calcul d'une décision optimale selon ces deux approches qui prennent en compte certaines particularités de notre algorithme MPL. Ces résultats ont été publiés dans [Dubois, et al. 1998] et en version longue dans [Dubois, et al. 1999].

1.4.1 Un peu de théorie

Les variables propositionnelles sont partitionnées en variables de décision (*D*) et en variables d'état (*S**D*). Nous disposons de deux bases de connaissances *K* et *P*. *K* décrit ce que l'agent sait du monde ainsi que les contraintes liant éventuellement les valeurs de vérité des variables de décision. *P* est une base de préférences décrivant les états-buts de l'agent, c'est à dire ceux satisfaisant ses préférences[36]. *K* et *P* sont supposées finies tout comme le langage propositionnel considéré.

Supposons dans un premier temps que *K* et *P* sont des bases de connaissances propositionnelles classiques : les connaissances sont certaines et les préférences sont de type « tout ou rien ».

- Prendre une « bonne » décision, d'un point de vue pessimiste, consiste à satisfaire toutes les formules de *P* en agissant sur les variables de décision qui contrôlent les modèles de *K* et *P*. Cela se traduit logiquement par une conjonction de décisions (elles sont prises en parallèles, il ne s'agit pas d'une séquence) $d^\wedge$ telle que $K^\wedge \wedge d^\wedge \models P^\wedge$. De plus, la formule $K^\wedge \wedge d^\wedge$ doit être consistante (i.e. la décision doit être permise). On retombe ici sur la notion classique de label : *d* contient un environnement du label de *P* dans *K*, en prenant comme variables hypothèses les variables de décision.
- Prendre une bonne décision, d'un point de vue optimiste, consiste à prendre une décision cohérente avec les connaissances et les buts à atteindre. Il s'agit donc logiquement de chercher $K^\wedge \wedge d^\wedge \wedge P^\wedge \not\models \bot$. Ceci est équivaut à chercher un impliquant D-restreint de $K \cup P$.

On retombe alors sur les notions classiques de diagnostic consistant ([Reiter 1987], «décision optimiste) et de diagnostic abductif ([Console/Torasso 1991], « décision pessimiste »).

[36] Les 3 lois de la robotique d'Asimov rappelées au début de cette partie sont des exemples de préférences de l'agent.

Supposons maintenant que K et P sont stratifiées : on ajoute aux connaissances des degrés de certitude et aux préférences des niveaux de priorité. On représente les degrés et les niveaux par des échelles (un ensemble fini totalement ordonné). On suppose que les degrés de certitude et les niveaux de priorité sont commensurables, c'est à dire que l'on peut utiliser une même échelle L simplement ordonnée pour les représenter (cette hypothèse est apparemment forte mais il est toujours possible de se ramener à ce cas). Soit n la fonction de renversement de cette échelle[37]. Nous représenterons les degrés de certitude par des α_i et les niveaux de priorité par des β_j. Le plus grand élément de L sera noté 1 et le plus petit 0. On a donc au moins deux niveaux. Si l'on se ramène au cadre de la logique propositionnelle classique, 1 contient la formule K et 0 est vide. On peut considérer que $K = \bigwedge_{0<\alpha\leq 1} K^{\alpha}$ où K^{α} dénote l'ensemble de connaissances de degré de certitude α. De même, on considère $P = \bigwedge_{0<\beta\leq 1} P^{\beta}$ où P^{β} dénote l'ensemble des préférences de niveau de priorité β. Nous noterons :

- K_{α} (resp. P_{β}) l'ensemble des connaissances de degré de certitude supérieur ou égal à α (resp. de niveau de priorité supérieur ou égal à β).
- $K_{\overline{\alpha}}$ (resp. $P_{\overline{\beta}}$) l'ensemble des connaissances de degré de certitude strictement supérieur à α, $\alpha<1$ (resp. de niveau de priorité strictement supérieur β, $\beta<1$).
- Plus généralement, $\overline{\alpha}$ dénotera la valeur suivant immédiatement α sur l'échelle L.

Nous allons associer une utilité à chaque décision. Cette utilité sera différente suivant la politique choisie (optimiste ou pessimiste). Choisir une décision optimale consiste alors à choisir pour une politique donnée une décision ayant une utilité maximale. Nous choisissons de ne considérer parmi les décisions optimales que celles qui sont minimales par rapport à l'inclusion ensembliste (représentation compacte d'une classe, d'un ensemble de solutions).

Chercher une bonne décision d'un point de vue pessimiste, consiste à trouver $K_{\alpha}^{\wedge} \wedge d^{\wedge} \models P_{\beta}^{\wedge}$ avec α élevé et β faible. C'est à dire α et $n(\beta)$ élevés. Ce que l'on peut traduire par $min(\alpha,n(\beta))$ élevé. Si on pose $\beta=n(\alpha)$, alors le problème devient : maximiser α tel que $K_{\alpha}^{\wedge} \wedge d^{\wedge} \models P_{n(\alpha)}^{\wedge}$ avec $\alpha>0$.

L'utilité qualitative pessimiste d'une décision d est alors [Sabbadin 1998, Définition 3.2.1, page 135] :

$$U_*(d) = max\ \{\ \alpha>0 \mid K_{\alpha}^{\wedge} \wedge d^{\wedge} \models P_{\overline{n(\alpha)}}^{\wedge}\ et\ K_{\alpha}^{\wedge} \wedge d^{\wedge} \not\models \bot\} \text{ avec } max\ \varnothing = 0.$$

En particulier, $U_*(d)=1$ ssi $K_1^{\wedge} \wedge d^{\wedge} \models P_{\overline{0}}^{\wedge}$, c'est à dire qu'une décision pessimiste optimale permettrait de satisfaire tous les buts à partir des connaissances les plus certaines.

De même, on peut définir l'utilité qualitative optimiste d'une décision d comme étant [Sabbadin 1998, Définition 3.2.2, page 136] :

$$U^*(d) = max\ \{\ n(\alpha)\leq 1 \mid K_{\overline{\alpha}}^{\wedge} \wedge d^{\wedge} \wedge P_{\overline{\alpha}}^{\wedge} \not\models \bot\} \text{ avec } max\ \varnothing = 0.$$

En particulier, $U^*(d)=1$ ssi $K_{\overline{0}}^{\wedge} \wedge d^{\wedge} \wedge P_{\overline{0}}^{\wedge} \not\models \bot$, c'est à dire qu'une décision optimiste optimale serait consistante avec l'ensemble des connaissances et l'ensemble des buts.

Nous proposons alors deux algorithmes de calcul de l'utilité d'une décision : le premier pour le point de vue optimiste, le second pour le point de vue pessimiste. Leur principe est le suivant : chercher s'il existe une solution pour une utilité donnée α, en parcourant toute l'échelle de 1 à 0.

1.4.2 Calcul de décisions optimistes optimales

```
Décisions_Optimistes(K,P,D)
// K est une base de clauses représentant les connaissances
// P est une base de clauses représentant les préférences
// D est l'ensemble des variables de décision
// Retourne α* l'utilité des meilleures décisions optimistes
// et ces décisions.
```

[37] Si L est l'échelle $0 = \alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 \ldots < \alpha_n = 1$ alors $n(\alpha_i)=\alpha_{n-i}$.

```
α ← 1 ;
// on cherche une décision consistante avec l'ensemble des connaissances
// et des buts
BA ← MPL( K_{n(α)} ∪ P_{n(α)} ,∅,D) ;
Tant que BA = ∅ et α > 0 faire
       Inc(α) ; // on descend dans l'échelle
       // on cherche une décision consistante avec les connaissances
       // et des buts restants
       BA ← MPL( K_{n(α)} ∪ P_{n(α)} ,BA,D)
Fin tant que
// BA contient la négation du résultat si α > 0, ∅ sinon.
Retourner < α, BA¬ >
```

Algorithme 10 Calcul de décisions optimales (utilité optimiste)

Ce premier algorithme est très simple. On cherche d'abord les impliquants D-restreints minimaux de K∪P. Si aucune solution n'existe, on enlève les connaissances les moins certaines et les buts les moins prioritaires et on recommence jusqu'à obtention d'une solution, auquel cas l'utilité de la (ou des) décision(s) optimiste(s) optimale(s) est α, ou $\alpha=0$ et il n'y a pas de solutions.

Le schéma suivant illustre ce comportement :

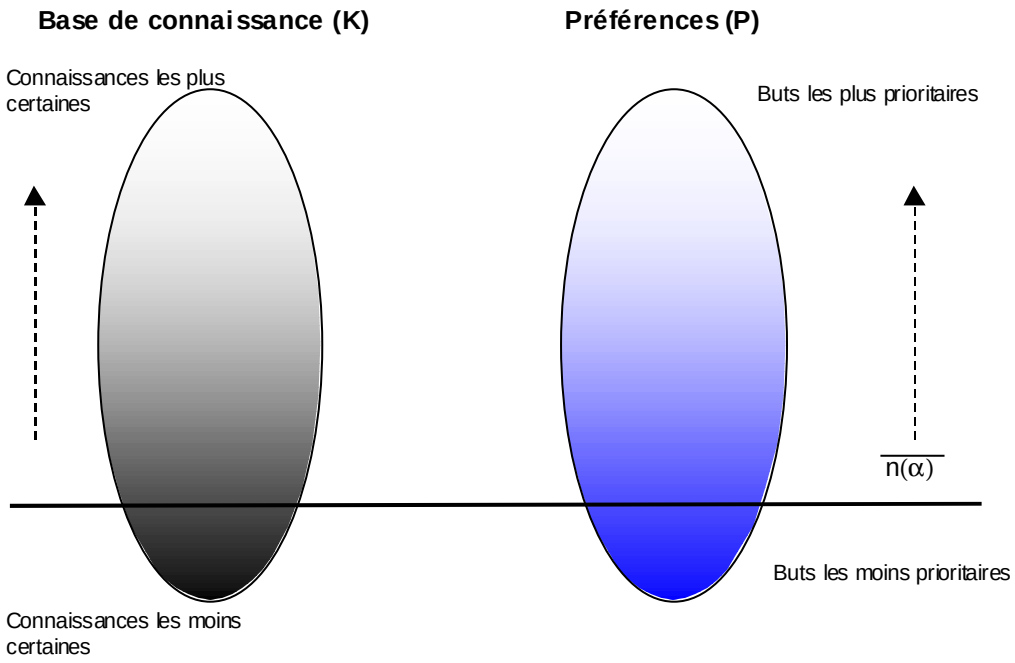


Figure 5 Calcul de l'utilité optimiste d'une décision

Il faut noter que si les formules *K* et *P* sont construites avec *N* symboles propositionnels, alors la complexité de cet algorithme est $O(|L|*2^N)$.

1.4.3 Calcul de décisions pessimistes optimales

```
Décisions_Pessimistes(K,P,D)
// K est une base de clauses représentant les connaissances
// P est une base de clauses représentant les préférences
// D est l'ensemble des variables de décision
// Retourne α* l'utilité des meilleures décisions pessimistes
// et ces décisions.
α ← 1 ;
S ← ∅ ;
Tant que S = ∅ et α > 0 faire
```

```
        // Calcul des nogoods de Kα
        NG' ← MPL(Kα,∅,D) ;          // 1ère passe, NG' est une base intermédiaire
        NG ← MPL(NG',∅, D¬) ;        // 2ème passe : NG contient les nogoods de Kα
                                     // sous forme de CNF
        // Calcul du label de P¹
        S' ← MPL(Kα∪P1,∅,D) ; // 1ère passe, S' est une base intermédiaire
        S ← MPL(S'∪NG,∅, D¬) ; // 2ème passe : S contient le label de P¹
        β ← 1 ;
        Tant  que β > n(α) et S≠ ∅ faire
               // Calcul "incrémental" du label de P_{n(α)}
               // On ajoute le calcul de la première passe à la 1ère passe
               // du calcul précédent
               S' ← S' ∪ MPL(Kα∪¬P^β, ∅,D) ;
               S ← MPL(S'∪NG,∅, D¬) ; // 2ème passe : S contient le label de P_{n(α)}
               Dec(β)
        Fin tant que
        Si S=∅ alors α←max{α̲[38],n(β)} Finsi
Fin tant que
Retourner <α,S¬>
```

Algorithme 11 Calcul de décisions optimales (utilité pessimiste)

Nous utilisons dans cet algorithme une propriété qui nous permet de calculer le label de la conjonction de deux formules α et β dans *BC* à partir des impliquants P-restreints premiers de $BC \wedge \neg\alpha$ et de ceux de $BC \wedge \neg\beta$. Elle est basée sur le résultat suivant :

Lemme 5
Soit $BC \subset FP_S$ une base de clauses. Soit *P* un ensemble consistant de littéraux. Soient α et β deux formules de FP_S. $IRP(BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta), P)^{\wedge} \equiv IRP(BC \wedge \neg\alpha, P)^{\wedge} \vee IRP(BC \wedge \neg\beta, P)^{\wedge}$.

Preuve :

) Soit E un impliquant P-restreint premier de $BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta)$. Donc $\exists$M modèle de $BC^{\wedge}$ tel que $M \models BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta)$ et $M \cap P = E$. Donc $M \models (BC^{\wedge} \wedge \neg\alpha) \vee (BC^{\wedge} \wedge \neg\beta)$.
Soit $M \models (BC^{\wedge} \wedge \neg\alpha)$ auquel cas E est un impliquant P-restreint de $BC \wedge \neg\alpha$, donc $E \models IRP(BC \wedge \neg\alpha, P)$.
Soit $M \models (BC^{\wedge} \wedge \neg\beta)$ auquel cas E est un impliquant P-restreint de $BC \wedge \neg\beta$, donc $E \models IRP(BC \wedge \neg\beta, P)$.
Donc $E \models IRP(BC \wedge \neg\alpha, P) \vee IRP(BC \wedge \neg\beta, P)$.
Nous avons donc $IRP(BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta), P)^{\wedge} \models IRP(BC \wedge \neg\alpha, P)^{\wedge} \vee IRP(BC \wedge \neg\beta, P)^{\wedge}$.

$\Leftarrow$) Soit $E \subseteq P$ tel que $E \models IRP(BC \wedge \neg\alpha, P)^{\wedge} \vee IRP(BC \wedge \neg\beta, P)^{\wedge}$.
Soit $E \models IRP(BC \wedge \neg\alpha, P)$ donc E est un impliquant P-restreint premier de $BC \wedge \neg\alpha$. Donc $\exists$ M tel que $M \cap P = E$ et $M \models BC \wedge \neg\alpha \models BC \wedge (\neg\alpha \vee \neg\beta)$. Donc E est un impliquant P-restreint de $BC \wedge (\neg\alpha \vee \neg\beta)$.
Soit $E \models IRP(BC \wedge \neg\beta, P)$ donc E est un impliquant P-restreint premier $BC \wedge \neg\beta$. Donc $\exists$ M tel que $M \cap P = E$ et $M \models BC \wedge \neg\beta \models BC \wedge (\neg\alpha \vee \neg\beta)$. Donc E est un impliquant P-restreint de $BC \wedge (\neg\alpha \vee \neg\beta)$.
Donc $E \models IRP(BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta), P)^{\wedge}$.
Donc $IRP(BC \wedge \neg\alpha, P)^{\wedge} \vee IRP(BC \wedge \neg\beta, P)^{\wedge} \models IRP(BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta), P)^{\wedge}$.

Nous sommes maintenant capables d'exprimer une méthode de calcul du label d'une conjonction de formules.

Propriété 7 Calcul du label d'une conjonction de deux formules
Soit $BC \subset FP_S$ une base de clauses. Soit H l'ensemble des symboles hypothèses de *BC*. Soient α et β deux formules de FP_S. Le label de $\alpha \wedge \beta$ dans BC est :

$$IRP(IRP((BC^{\wedge} \wedge \neg\alpha), H)^{\neg\wedge} \wedge IRP((BC^{\wedge} \wedge \neg\beta), H)^{\neg\wedge} \wedge NG^{\neg\wedge}, H^{\neg}).$$

Preuve :
Il suffit de remplacer, en vertu du lemme précédent, dans la Proposition 30 $IRP(BC \wedge (\neg\alpha \vee \neg\beta), H)^{\neg\wedge}$ par $IRP((BC^{\wedge} \wedge \neg\alpha), H)^{\neg\wedge} \wedge IRP((BC^{\wedge} \wedge \neg\beta), H)^{\neg\wedge}$.

[38] $\underline{\alpha}$ dénote l'indice de la plus grande strate non vide de *K* moins certaine ou moins prioritaire que α.

Il nous est maintenant possible de calculer « incrémentalement » le label d'une conjonction de formules qui peuvent être soit des clauses, soit des cubes. En effet, il faut que $BC^{\wedge} \wedge (\neg\alpha \vee \neg\beta)$ soit une base de clauses pour calculer le label de la conjonction de $\alpha \wedge \beta$ par la méthode des P-impliqués premiers, ce qui réduit α et β à des littéraux. $BC^{\wedge} \wedge \neg\alpha$ et $BC^{\wedge} \wedge \neg\beta$ sont des bases de clauses.

Nous allons donc calculer le label de $P^1 \wedge P^{\beta_{n-1}} \wedge P^{\beta_{n-2}} \wedge .. \wedge P^{\overline{n(\alpha)}}$ en calculant d'abord le label de P^1, puis le label de $P^1 \wedge P^{\beta_{n-1}}$, puis le label de $P^1 \wedge P^{\beta_{n-1}} \wedge P^{\beta_{n-2}}$ puis … et enfin le label voulu.
Nous pourrions nous contenter de calculer seulement les impliquants P-restreints premiers des $BC \wedge \neg P^{\beta_i}$ puis effectuer un seul calcul effectif de label sur l'ensemble des résultats obtenus. Il faut noter cependant que si le label de $P^1 \wedge P^{\beta_{n-1}} \wedge \ldots \wedge P^{\beta_i}$ est vide, avec $\beta_i < \overline{n(\alpha)}$, alors il n'est pas nécessaire de continuer le calcul car on sait déjà que le label recherché est vide.

De plus, cette information nous permet de déterminer une borne inférieure de la prochaine strate à considérer. En effet, supposons que le label de la formule $P^1 \wedge P^{\beta_{n-1}} \wedge \ldots \wedge P^{\beta_i}$ soit vide. Cela signifie que si d'autres informations ne sont pas ajoutées dans BC, alors on ne peut pas trouver de solutions d'utilité supérieure à $n(\beta_i)$ (car $\beta = \overline{n(\alpha)}$).

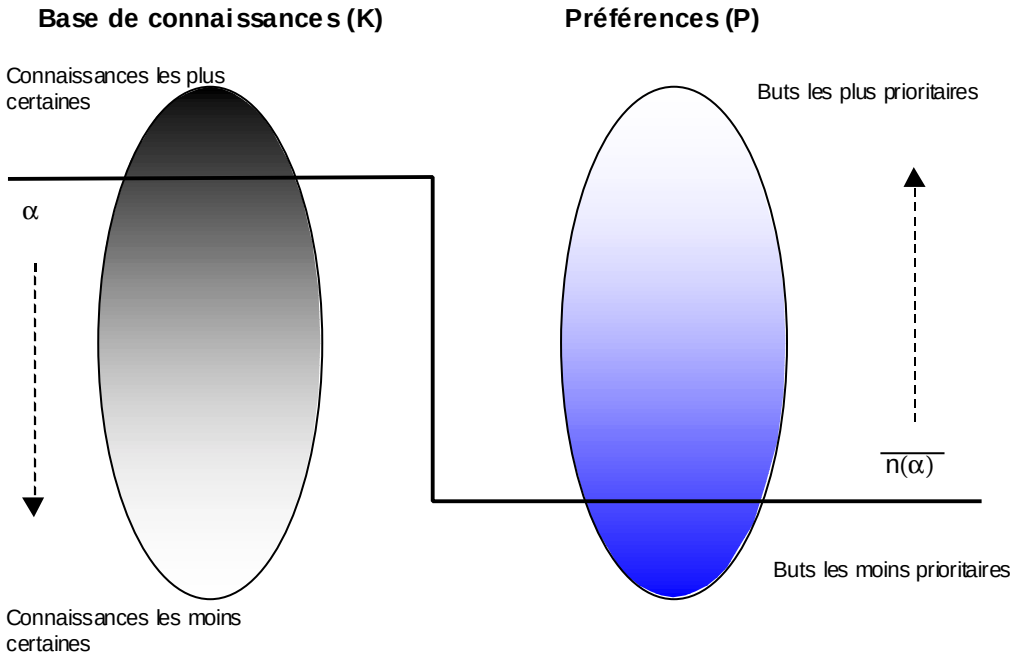


Figure 6 Calcul de l'utilité pessimiste d'une décision

1.4.4 On ne peut pas faire d'omelette …

Nous allons maintenant illustrer ce que nous venons de voir sur un exemple très connu en théorie de la décision : celui de l'omelette de Savage [Savage 1954]. Le problème est le suivant : on dispose dans un récipient d'une omelette contenant 5 œufs. On dispose d'un bol et d'un 6ème œuf. Malheureusement, on ne sait pas si cet œuf est bon. On a alors 3 actions possibles : casser quand même l'œuf dans l'omelette, au risque de perdre celle ci, casser l'œuf dans le bol pour savoir si il est bon et selon le cas le mettre dans l'omelette ou le jeter, ou alors le jeter. Le tableau suivant résume ce problème de décision :

Actions possibles	Etat du monde	
	l'œuf est bon	*l'œuf est mauvais*
Casser l'œuf dans l'omelette	Omelette à 6 œufs	Plus d'omelette
Casser l'œuf dans le bol	Omelette à 6 œufs et un bol à laver	Omelette à 5 œufs et un bol à laver
Jeter l'œuf	Omelette à 5 œufs et un œuf gâché	Omelette à 5 œufs

Voici ce problème codé en logique propositionnelle tel qu'il est compris par notre prototype :

```
// g indique que l'œuf est bon (good),
// r indique que l'œuf est pourri (rotten),
// x-œufs indique que l'on a une omelette à x-œufs
// cw indique qu'il y a un bol a laver (a cup to wash)
// we indique que l'on a gâché un œuf (wasted egg)
// w indique que l'omelette est gâchée (wasted omelette)

// Les actions possibles sont :
// BIO casser l'œuf dans l'omelette (break in the omelette)
// BAC casser l'œuf dans le bol (break apart in a cup)
// TA jeter l'œuf (throw it away)

// on code tout d'abord les actions des décisions

// si l'œuf est bon et qu'on le met dans l'omelette,
// on obtient une omelette à 6 œufs
BIO g -> 6-oeufs ;
// si l'œuf est mauvais et qu'on le met dans l'omelette,
// on perd l'omelette
BIO r -> w ;

// si l'œuf est bon et qu'on le casse dans une tasse,
// on obtient une omelette à 6 œufs
BAC g -> 6-oeufs ;
// si l'œuf est mauvais et qu'on le casse dans une tasse,
// on obtient une omelette a 5 œufs
BAC r -> 5-oeufs ;
// dans les deux cas, on aura un bol à laver
BAC -> cw ;

// Si on jette un œuf qui est bon, on perd un œuf.
TA g -> we ;
// dans tous les cas on obtient un omelette à 5 œufs
TA -> 5-oeufs ;

// On décrit maintenant les contraintes de représentation

// un oeuf est soit bon, soit mauvais.
g r -> ;
-> g r ;

// on a soit une omelette a 5 œufs soit une omelette à 6 œufs
// soit une omelette gâchée
5-oeufs 6-oeufs -> ;
5-oeufs w -> ;
6-oeufs w -> ;
-> 5-oeufs 6-oeufs w ;

// on ne peut pas avoir une omelette a 6-oeufs et un œuf gâché
6-oeufs we -> ;

// il faut prendre une et une seule décision
-> BIO BAC TA ;
BIO BAC -> ;
BAC TA -> ;
BIO TA -> ;
```

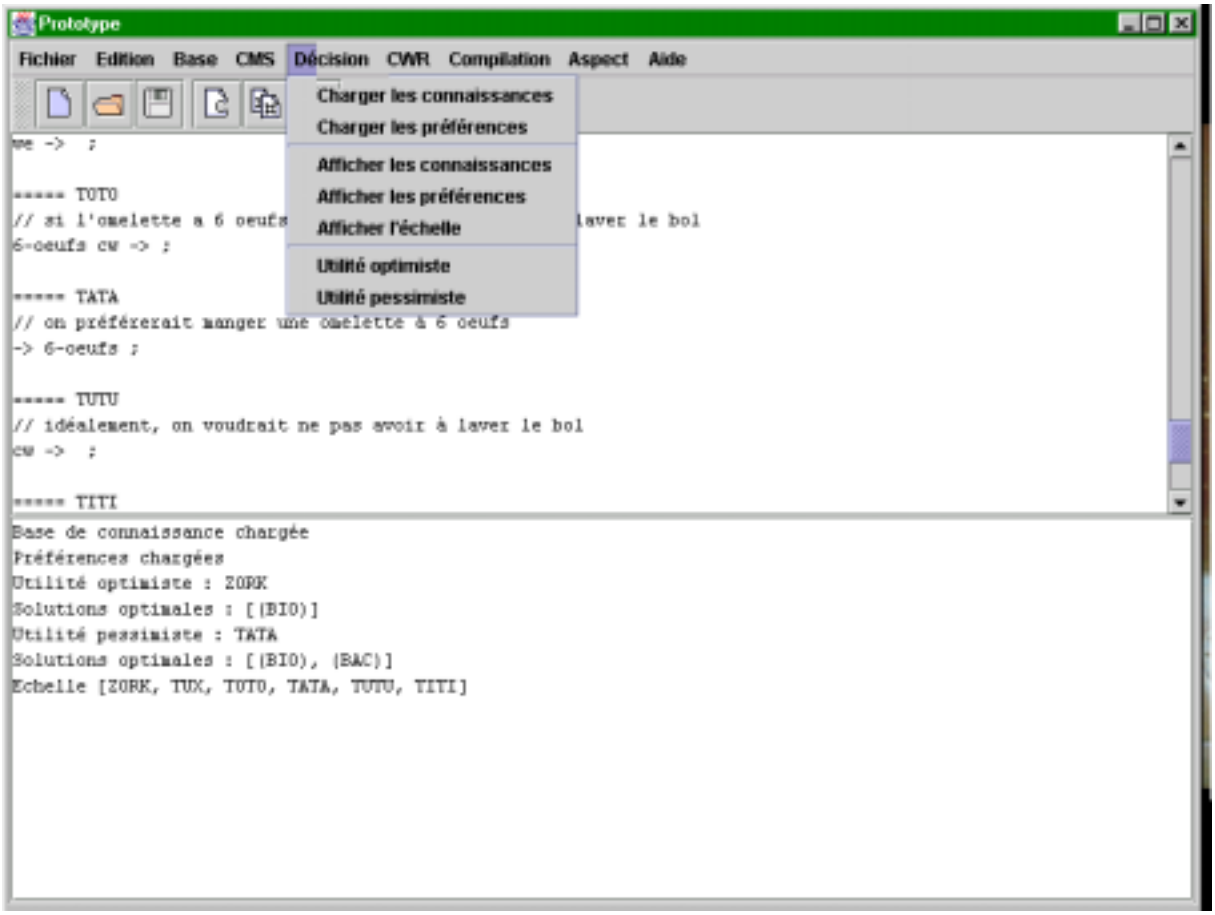
Ces connaissances sont certaines et se trouverons dans K^1. L'incertitude va concerner le fait que l'œuf est bon ou mauvais. Nous allons utiliser une échelle $L = \{0=TITI < TUTU < TATA < TOTO < TUX < ZORK = 1\}$ afin de souligner que seul l'ordre sur L est important. Supposons que nous pensions que l'œuf est bon avec une certitude TATA. Nous allons donc fournir la base de connaissances suivante à notre outil :

```
===== ZORK
// la base précédente
===== TUX
===== TOTO
===== TATA
-> g ;
===== TUTU
===== TITI
```

Il nous faut maintenant expliciter de la même manière les préférences entre les différents états-buts.

```
===== ZORK
// on veut à tout prix manger une omelette
-> 5-oeufs 6-oeufs ;

===== TUX
// et si possible ne pas gâcher d'œuf
we ->  ;

===== TOTO
// si l'omelette a 6 œufs on préfère ne pas avoir à laver le bol
6-oeufs cw -> ;

===== TATA
// on préférerait manger une omelette à 6 oeufs
-> 6-oeufs ;

===== TUTU
// idéalement, on voudrait ne pas avoir à laver le bol
cw ->  ;
===== TITI
```

On peut maintenant appliquer nos deux algorithmes :



En ce qui concerne le point de vue optimiste, on trouve un impliquant D-restreint de K∪P : BIO. Donc la décision optimiste optimale est BIO et son utilité est ZORK.
En ce qui concerne le point de vue pessimiste, le label de P^{TUX} dans K^1 est vide. On cherche donc la prochaine strate non vide de K : K^{TATA}. le label de P^{TUX} dans $K^1 \wedge K^{TATA}$ est cette fois-ci non vide. Tout comme le label de P^{TOTO}. On ne va pas plus loin car n(TATA) = TOTO. On trouve donc deux solutions optimales : BIO et BAC. L'utilité de ces décisions est TATA.

2 Variations autour de MPL

2.1 Introduction de l'incomplétude dans MPL

2.1.1 Principe

Nous avons vu que la particularité de notre algorithme nous permet de garantir qu'un impliquant P-restreint est premier sans pour autant avoir à les produire tous. Nous pouvons utiliser cette propriété pour rendre notre algorithme « incomplet », c'est à dire que seule une partie des impliquants P-restreints premiers sera retournée.

La première méthode pour rendre notre algorithme incomplet est de s'arrêter dès qu'un nombre donné de solutions est atteint : *« chercher 10 impliquants P-restreints premiers de BC »*. Chercher 1 impliquant P-restreint premier revient à résoudre le problème SAT sans la simplification par littéraux purs et avec un ordre sur les littéraux de P. Cette méthode se justifie si toutes les solutions ont même « valeur », c'est à dire que parmi les solutions il n'y a pas de bonne ou de mauvaise solution. Par exemple, on cherchera une manière de restaurer la cohérence dans la base de connaissances, un diagnostic minimal consistant, etc.

La seconde méthode consiste à arrêter le calcul au bout d'un temps donné *: « chercher des impliquants P-restreints premiers pendant 10 secondes »*. On peut alors imaginer une version « anytime » de notre algorithme. Le principe d'un algorithme « anytime » est que la qualité de la réponse augmente si on augmente le temps d'exécution de l'algorithme. Dans notre cas, la qualité sera mesurée en nombre d'impliquants P-restreints premiers trouvés par rapport au nombre total d'impliquants P-restreints premiers.

Nous ne rentrons pas sur le détail des modifications à apporter à MPL pour implanter ces deux notions d'incomplétude. Elles sont triviales : il suffit d'ajouter un compteur de solutions dans le premier cas et une borne temporelle dans le second.

Nous avons tout d'abord pensé introduire les algorithmes incomplets de type GSAT pour effectuer le test de satisfaction dans MPL. Or bon nombre de ces tests sont des preuves d'inconsistance. Il n'est donc pas intéressant d'utiliser une approche incomplète dans ce cadre car elle effectuerait à chaque test inconsistant MAX-TRIES*MAXFLIPS flips.

Son utilisation nous semble plus adaptée à la production d'impliquants P-restreints premiers en utilisant le même principe d'ajout de clauses de minimisation que dans MPL. Nous allons présenter une méthode de production d'impliquants P-restreints premiers basée sur la minimisation de modèles fournis par un algorithme de satisfaction. Nous verrons que cette méthode est parfois plus pratique que MPL pour obtenir un sous-ensemble des impliquants P-restreints premiers d'une base de clauses.

2.1.2 Minimisation d'un impliquant

Une méthode simple pour produire un seul impliquant P-restreint premier est d'utiliser un algorithme de satisfaction normal (un Davis et Putnam par exemple) et de le modifier afin qu'il retourne l'impliquant trouvé au lieu de la simple réponse *vrai* ou *faux*. Une fois cet impliquant trouvé, il suffit d'effectuer un nombre polynomial de tests de satisfaction pour minimiser sa restriction à *P*. On peut aussi dans ce cadre utiliser les algorithmes incomplets les plus performants pour obtenir un impliquant.

Lemme 6
Soit α une formule propositionnelle. Soit *P* un ensemble consistant de littéraux. Soit $E_1 \subseteq E_2 \subseteq P$. $\exists\ E^\wedge$ un impliquant P-restreint de α tel que $E_1 \subseteq E \subseteq E_2$ ssi $\alpha \wedge E_1^\wedge \wedge (P \backslash E2)^{\wedge\neg}$ consistant.

Preuve :
c'est une généralisation de la Proposition 19 page 61.

) Soit $E^\wedge$ un impliquant P-restreint de α tel que $E_1 \subseteq E \subseteq E_2$. D'après la Proposition 19, on a $\alpha \wedge E^\wedge \wedge (P \backslash E)^{\wedge\neg}$ consistant. Or $E1 \subseteq E$ donc $E_1 \cap (P \backslash E) = \emptyset$, donc $\alpha \wedge E_1^\wedge \wedge (P \backslash E)^{\wedge\neg} \not\models \bot$. De même, $E \subseteq E2$, donc $(P \backslash E2) \subseteq (P \backslash E)$ et comme $E1 \subseteq E2$, $(P \backslash E2) \cap E1 = \emptyset$. Donc $\alpha \wedge E_1^\wedge \wedge (P \backslash E_2)^{\wedge\neg} \not\models \bot$.

⇐) Soit $\alpha \wedge E_1^\wedge \wedge (P \backslash E_2)^{\wedge\neg} \not\models \bot$. Donc $\exists$ M modèle de $\alpha \wedge E_1^\wedge \wedge (P \backslash E_2)^{\wedge\neg}$ tel que $M \cap P = E$. $E^\wedge$ est donc un impliquant P-restreint. Comme M satisfait $E_1^\wedge$, on a $E_1 \subseteq E$. De plus, comme M satisfait $(P \backslash E2) \wedge \neg$, $E \cap (P \backslash E2) = \emptyset$, c'est à dire $E \subseteq E_2$. On a donc $E_1 \subseteq E \subseteq E_2$.

On en déduit un algorithme de calcul d'impliquant P-restreint premier à partir d'un impliquant M.
Nous savons qu'il existe un impliquant P-restreint premier E tel que E⊆M∩P. De plus ∅⊆E. Nous avons donc un encadrement E_1=∅ et E_2=M∩P de E. Nous allons ensuite choisir un élément *e* de $E_2 \backslash E_1$ et considérer E'2 = E_2 \ {e}. En appliquant le lemme précédent sur le nouvel encadrement (E_1, E_2'), nous pouvons déterminer si ∃ un impliquant P-restreint entre ces deux valeurs. Si oui, on réitère le processus avec E_2 = E_2'. Sinon on ajoute *e* dans E_1. En effet, comme ∃ un impliquant P-restreint qui contient *e*, et ∄ d'impliquant P-restreint qui ne contient pas *e*, cela signifie que *e* appartient à l'impliquant P-restreint premier E recherché. Donc E est maintenant encadré par $E_1 \cup \{e\} \subseteq E \subseteq E_2$. Et on réitère le processus jusqu'à $E_1 = E_2$. Cela arrive forcément car $E_1 \subseteq E_2$ au départ et M∩P est fini, donc E_2 aussi, et à chaque étape, soit un élément de E_2 est enlevé, soit un élément de E_2 est ajouté à E_1.

Propriété 8
Soit $\alpha \in FP_S$. Soit *M* un impliquant de α. Soit *P* un ensemble consistant de littéraux. $E_2 = M \cap P$.

```
E1← ∅ ;
Tant que E1⊂E2 faire
        e ← un élément de E2\E1 ;
        E2' = E2 \ {e} ;
        Si α∧(E1∪(P\E2')¬)^ ⊨ ⊥ alors
E1 ← E1 ∪{  e}
        Sinon
E2 ← E2'
        Finsi
Fin tant que
```

Après exécution de cet algorithme, $E_1^\wedge$ est un impliquant P-restreint premier de α.

> Cet algorithme permet de trouver UN impliquant P-restreint premier de α. *E* peut en contenir PLUSIEURS.
> Cette méthode est utilisée par [Palopoli, et al. 1999] pour calculer les impliquants premiers d'une formule à l'aide de la programmation linéaire en nombre entiers 0-1. Petite différence : au lieu de calculer un impliquant premier à partir de l'impliquant, ils les produisent tous.

On peut donc, à partir d'une formule α et d'un modèle de α, trouver un impliquant P-restreint premier de α en au plus $|M \cap P|$ tests de consistance. Le modèle peut être donné par un algorithme énumératif classique, ou plus judicieusement un algorithme de recherche locale si la base de connaissances est grande. Le test de consistance par contre doit être effectué par un algorithme de recherche complet.

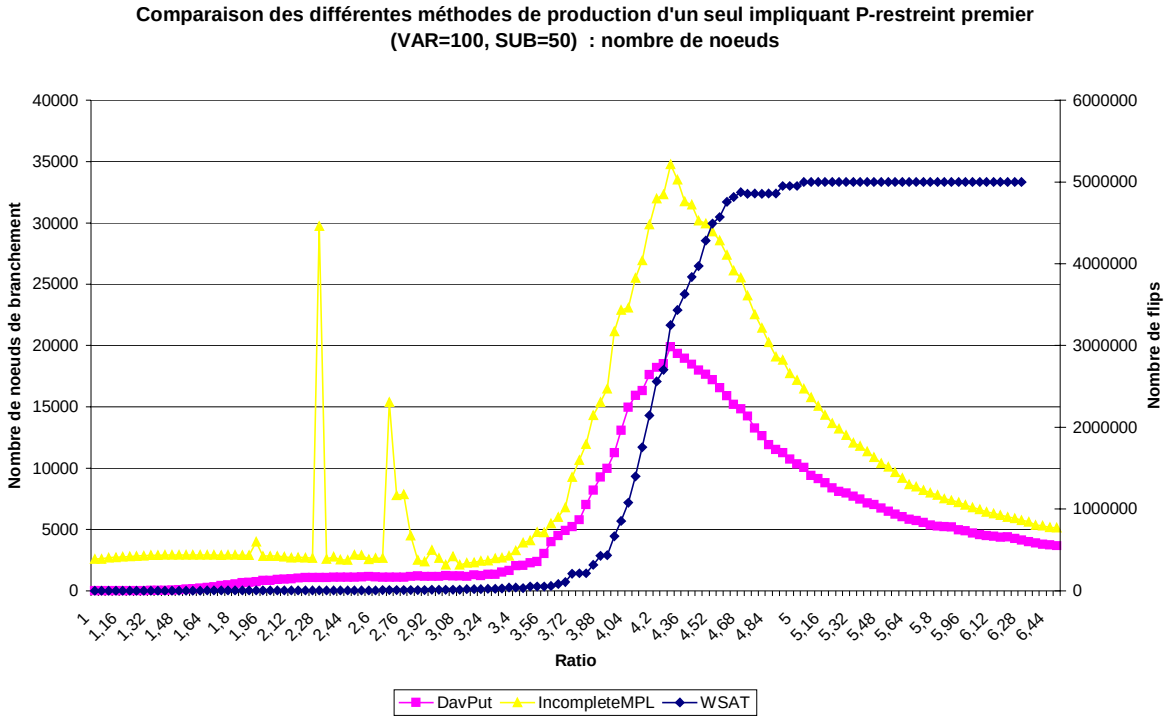
On peut déduire de la propriété précédente un algorithme incrémental de production d'impliquants P-restreints premiers. A la différence de notre procédure MPL, elle utilise uniquement des prouveurs SAT traditionnels.

```
MinimisationDeModèles(BC, MAXSOL)
// Retourne MAXSOL impliquants P-restreints premiers de BC
M ← Modèle_de(BC) ;
PIRP ← ∅ ;
// on cherche MAXSOL impliquants P-restreints premiers
Tant que M ≠ null et |PIRP|<MAXSOL faire
        // initialisation de l'encadrement
        E2 ← M ∩ P ;
        E1 ← ∅ ;
        // tant que E1⊂E2
        Tant que E2\E1 ≠ ∅ faire
                Soit e un élément de E2\E1 ;
                E2' ←  E2 \ {e} ;
                // test de consistance
                Si BC^∧(E1∪(P\E2')¬)^ ⊨ ⊥ alors E1 ← E1 ∪{  e}
                                sinon E2 ← E2'
                Finsi ;
        // La minimisation est terminée, E1=E2 est un impliquant P-restreint premier
        PIRP ← PIRP ∪ {E1^} ;
        // on ajoute sa négation à la base pour ne pas le retrouver.
        BC ← BC ∪ {E1¬∨} ;
        M ← Modèle_de(BC) ;
Fin tant que
Retourner PIRP
```

La fonction Modèle_de(BC) utilise un prouveur SAT traditionnel pour retourner un modèle si la base est consistante, et null sinon. Il faut pour cela modifier un peu la procédure de Davis et Putnam, en gardant la trace des littéraux satisfaits. Dans le cas d'un algorithme incomplet, il suffit de retourner le modèle trouvé, ou null sinon. Le test de consistance doit être effectué par un algorithme complet.

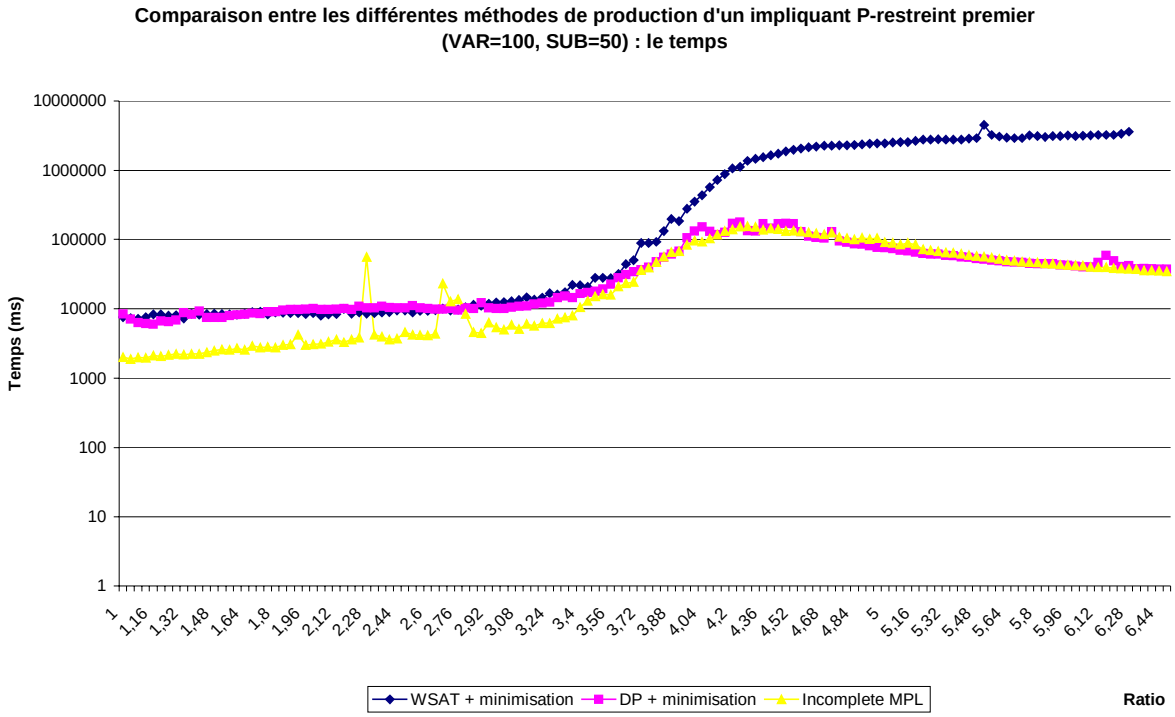
2.1.3 Comparaison

Nous allons essayer cette méthode avec un algorithme de Davis et Putnam classique (heuristique MOMS), notre implantation de la méthode de recherche locale WSAT (MAXTRIES=5, MAXFLIPS=10000, noise=0,5), et la comparer avec notre méthode incomplète s'arrêtant dès l'obtention d'un impliquant P-restreint premier. C'est ce que montre le graphique suivant : nous faisons varier le ratio de 1 à 6.5 afin d'obtenir des formules ayant un nombre de modèles variable. Le pourcentage des symboles appartenant au sous-langage est 50%, pour 100 variables propositionnelles. Comme il est difficile de comparer le nombre de nœuds développés par une méthode de recherche systématique avec le nombre de flips d'une méthode de recherche locale, nous utilisons deux échelles différentes. En ce qui concerne le nombre de nœuds de la procédure de Davis et Putnam et le nombre de flips de WSAT, ils ne comprennent pas le nombre de nœuds nécessaires à la minimisation de l'impliquant P-restreint premier. C'est pourquoi nous présentons pour ces deux méthodes la proportion du temps consacré à la minimisation de l'impliquant trouvé. Nous donnons aussi le temps passé par chacune des méthodes pour trouver un seul impliquant P-restreint sur 100 bases. Nous avons stoppé la méthode de recherche locale quand les instances sont devenues inconsistantes car les temps de résolution devenaient énormes (et constants !).

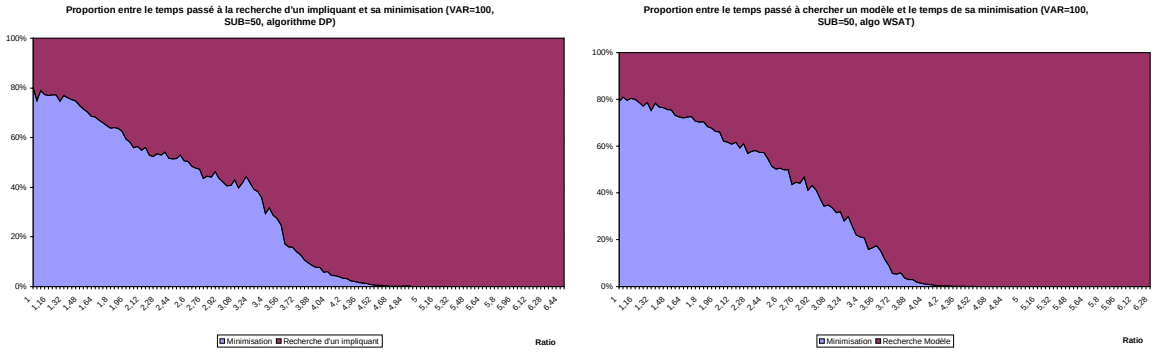


La première chose que l'on remarque est que notre méthode IncompleteMPL et la procédure de Davis et Putnam on un comportement similaire. Le nombre de nœuds plus important pour IncompleteMPL correspond aux contraintes de branchement que nous imposons pour garantir la minimalité de la solution. Pour la méthode incomplète, dès qu'une instance inconsistante apparaît, les temps de résolution explosent. Remarquons que WSAT a trouvé un impliquant pour chaque base jusqu'au ratio 3,8. Quelques instances semblent difficiles pour incompleteMPL parmi les bases sous-contraintes, alors qu'elles ne le sont pas pour les autres méthodes. C'est sans doute à cause de l'emplacement du premier impliquant P-restreint premier dans l'arbre de recherche.

Intéressons nous maintenant aux temps de calcul de chacune des méthodes.

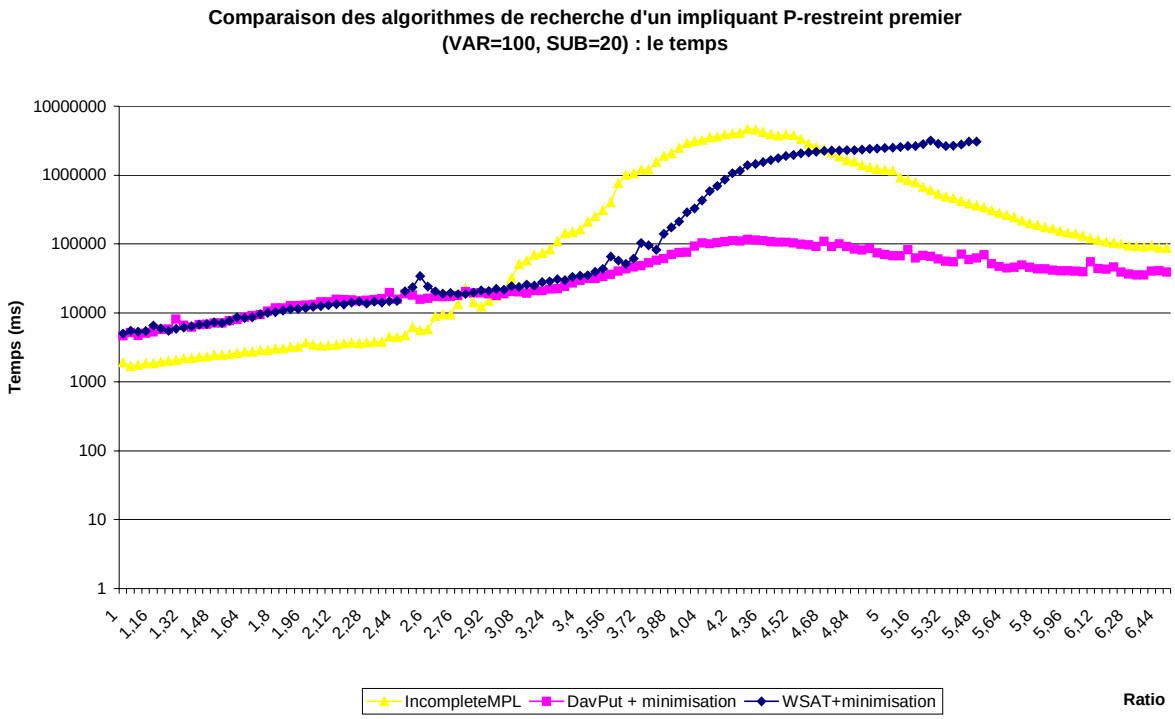


Contrairement à ce que nous pensions intuitivement, il n'y a pas beaucoup de différence entre notre version de MPL avec arrêt au premier impliquant P-restreint premier trouvé et l'utilisation d'un prouveur SAT traditionnel pour rechercher un seul impliquant P-restreint premier. Le premier trouve rapidement une solution malgré les contraintes sur les variables de branchement sur les instances très sous-contraintes (ratio<2). Comme elle est minimale, le temps gagné sur les autres méthodes est important car ces dernières passent jusqu'à 80 % de leur temps à minimiser l'impliquant P-restreint trouvé (cf. graphiques ci-dessous).



Les résultats de WSAT sont à peine meilleurs que ceux de DP dans cet exemple : nous pensons que cela est dû à notre implantation qui n'est pas optimisée pour la recherche locale (entre 2000 et 2500 flips par seconde pour des bases 3-SAT 100 variables 500/600 clauses sur une SUN ultra 5 300 Mhz). On remarque aussi que le temps passé à la minimisation semble plus élevé dans le cadre de WSAT : cela s'explique sans doute par le fait que DP travaille sur des impliquants et WSAT sur des modèles, donc l'impliquant P-restreint fourni par WSAT est sans doute plus grand que celui fourni par DP en moyenne.

Mais ce test correspond à une taille du sous-langage pour laquelle l'heuristique sur l'énumérations des impliquants P-restreints semble bien fonctionner (cf. page 77). Nous avons effectué ce même test pour maintenant un sous-langage comportant 20 % des variables propositionnelles. Les résultats obtenus sont différents :



IncompleteMPL est le plus performant jusqu'au ratio 3 environ, ensuite il devient beaucoup plus mauvais qu'une approche par minimisation d'impliquant. Ce qui semble confirmer que l'heuristique est très mauvaise lorsque le ratio concerne 20 % du sous langage : dès que le nombre de modèles diminue et que leur recherche devient difficile, IncompleteMPL met beaucoup de temps avant de trouver une première solution. On préférera donc une méthode par minimisation d'impliquants à IncompleteMPL pour chercher quelques impliquants P-restreints premiers.

2.2 Calcul du degré de croyance d'une formule

« Votre voiture souffre de pannes intermittentes. Les charbons du démarreur restent bloqués avec une probabilité X. Un court-circuit dans l'éclairage peut se produire avec une probabilité Y. Les essuies glaces marchent une fois sur deux. Pour aller à l'Université, il faut que la voiture démarre, que l'éclairage fonctionne s'il fait nuit et que les essuies glaces fonctionnent s'il pleut. Imaginons que vous deviez faire cours à 7h45 au mois de décembre : il fait nuit et il pleut. Quelle est la probabilité de ne pas pouvoir aller à l'Université ? »

Dans cet exemple, nous disposons de probabilités pour une partie des faits : ceux concernant le fonctionnement de la voiture. A partir de ces probabilités (plus l'hypothèse de leur indépendance), nous pouvons déduire une distribution de probabilités sur l'état de la voiture (par exemple, la probabilité que la voiture démarre, que les feux et les essuies glaces fonctionnement est (1-X)*(1-Y)*0,5). On cherche alors une probabilité pour un fait qui ne concerne pas directement le fonctionnement de la voiture, mais qui en découle. Cette mesure est appelée un degré de croyance [Dempster 1967, Shafer 1976].

D'un point de vue logique, on peut coder notre petit problème par :
BC = {nuit Ab(Eclairage) fac , pluie Ab(EssuiesGlaces) fac , fac Ab(Démarrage) , pluie , nuit }.
Certains auteurs ont montré que le degré de croyance recherché peut être interprété comme la probabilité de prouver *¬fac* dans *BC* [Pearl 1988, Provan 1989, Smets 1993].

Le label de *¬fac* dans *BC*, en considérant comme hypothèses les propositions d'anormalité, est *{{Ab(éclairage)},{Ab(EssuiesGlaces)},{Ab(Démarrage)}}*. Connaissant les probabilités de pannes des différents composants, on souhaite calculer à partir de ce label le degré de croyance de *¬fac* (ce qui est différent de calculer la panne la plus probable). C'est ce que propose [Provan 89] en établissant la relation entre le degré de croyance d'une formule et son label dans l'ATMS. Nous avons utilisé ce résultat dans [Le Berre/Sabbadin 1997] pour la décision dans l'incertain qualitatif dans le cadre du « diagnostic et réparation ».

Nous présentons ici une solution algorithmique permettant de calculer le degré de croyance d'une formule *lors* du calcul de son label et non *après* comme l'a proposé [Provan 1989]. Nous commençons par définir la notion de

fonction de croyance dans le cadre de la théorie des ensembles [Dempster 1967, Shafer 1976], puis nous donnons son équivalent logique, la probabilité de déductibilité [Pearl 1988, Provan 1989, Smets 1993]. Nous décrivons ensuite les liens entre le label d'une donnée et sa croyance en nous basant sur les travaux de [Provan 89]. Nous montrons enfin comment modifier MPL pour calculer le degré de croyance d'une formule.

2.2.1 De la théorie de Dempster-Shafer à l'ATMS

Dans la théorie de Dempster-Shafer (DS dans le reste du document), une fonction de masse associe des poids à des sous-ensembles quelconques.

Définition 57 (fonction de masse)
Soit X un ensemble. $m : 2^X \rightarrow [0,1]$ est une fonction de masse ssi *m* vérifie les propriétés suivantes :

- $\sum_{E \subseteq X} m(E) = 1$
- $m(\varnothing)=0$.

A partir d'une fonction de masse, il est possible de déduire deux *mesures* sur 2^X : la *croyance* et la *plausibilité*.

Définition 58 (fonction de croyance)
bel : $2^X \rightarrow [0,1]$ est une fonction de croyance ssi il existe une fonction de masse *m* telle que $\forall E \subseteq X$, $bel(E) = \sum_{F \subseteq E} m(F)$.

Définition 59 (fonction de plausibilité)
pls : $2^X \rightarrow [0,1]$ est une fonction de plausibilité ssi il existe une fonction de masse *m* telle que $\forall E \subseteq X$,
$Pls(E) = \sum_{E \cap F \neq \varnothing} m(F)$ [39] = *1 - Bel(X\E).*

Certains auteurs ont montré qu'un degré de croyance d'une formule peut s'interpréter comme la probabilité de sa prouvabilité [Pearl 1988, Provan 1989, Smets 1993]. Dans ce cadre, on dispose de distributions de probabilités pour chaque élément d'un ensemble *H* de symboles du langage, dont on calcule une probabilité jointe par l'hypothèse d'indépendance sur les H-interprétations du langage ($X = 2^H$).

Exemple : H = $\{H_1,H_2,H_3\}$. $P(H_1)=0,8$, $P(H_2)=0,4$ et $P(H_3)=0,7$. On en déduit $p(\{H_1,H_2,\neg H_3\}) = P(H_1)*P(H_2)*(1-P(H_3)) = 0,8*0,4*0,3 = 0,096$

Smets appelle *probability of deductibility* d'une formule α dans une base *BC* la somme des probabilités des éléments de *X* consistants avec *BC* qui satisfont α. Il montre que cette mesure est une fonction de croyance sur l'ensemble des interprétations du langage (en associant à chaque formule du langage l'ensemble des H-interprétations qui la satisfont).

Définition 60 (probabilité de déductibilité)
Soit $BC \subseteq FP_S$. Soit $\alpha \in FP_S$ une formule. Soit $H \subseteq S$. Soit $X=2^H$ l'ensemble des H-interprétations. Soit *p* une distribution de probabilités sur les éléments de *X*.
$P_{déd}(\alpha) = \sum_{I \in X, I \models \alpha, I \models BC} p(I)$.

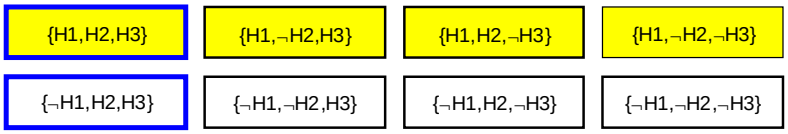
Remarque : Si $S = H$ alors la croyance d'une formule est sa probabilité. Si maintenant *H* est vide, alors toute formule α conséquence logique de *BC* aura pour croyance 1 et les autres formules auront pour croyance 0.

[Provan 1989] montre la relation entre le label d'un ATMS et le degré de croyance de la théorie de DS. On considère ici des environnements généralisés (des environnements contenant des négations d'hypothèses). Ces environnements ont une particularité : ils sont mutuellement exclusifs (si on les interprète comme la conjonction de leurs éléments).

Définition 61 (formules mutuellement exclusives ou indépendantes)
Deux formules α et β sont indépendantes ssi elles ne contiennent pas les mêmes symboles propositionnels et sont exclusives ssi $\alpha \wedge \beta \models \bot$.

[39] $E \cap F \neq \varnothing$ ssi $E \not\subseteq (X \backslash F)$ ssi $F \not\subseteq (X \backslash E)$

La notion de croyance d'une formule β est alors proche de la notion de label de cette formule car ce dernier indique les cubes à satisfaire pour prouver β à partir de α. En termes de H-interprétation, cela revient à donner de manière compacte un ensemble de H-interprétations : Label(β)= $\{\{H_1\}\}$ représente en réalité $\{\{H_1,\neg H_2,H_3\},\{H_1,H_2,H_3\},\{H_1,\neg H_2,\neg H_3\},\{H_1,H_2,\neg H_3\}\}$. Il faut enlever de ces H-interprétations celles qui contiennent un nogood. Si $\{H_2,H_3\}$ est un nogood par exemple, il ne faudra comptabiliser que $\{\{H_1,\neg H_2,H_3\},\{H_1,\neg H_2,\neg H_3\},\{H_1,H_2,\neg H_3\}\}$. C'est ce que nous pouvons schématiser de la façon suivante :



Les cases grisées représentent les H-interprétations qui satisfont le label de β (H_1) et les cases entourées celles qui satisfont les nogoods ($H_2 \wedge H_3$). La croyance de β (normalisée) est donc la somme des probabilités associées aux cases grisées non entourées divisée par la somme des masses de cases non entourées.

Plus formellement, en notant [E] = {F | F∈X et F⊨E} on peut définir la croyance de β dans α par : *bel(β)* = masse([*label(β)*]) - masse([*label(β)*]∩[nogoods]) /(1 - masse([nogoods]).

Provan présente à partir de ces observations une méthode de calcul de la croyance d'une formule à partir d'un ATMS :

1. Utiliser un ATMS pour calculer les nogoods de α et le label de la formule β Label(β)={L_1,…,L_n}
2. Transformer le label de β en une expression booléenne $\alpha = L_1^{\wedge} \vee .. \vee L_m^{\wedge}$ et chercher une formule α'≡ α décomposable en formules mutuellement exclusives ou indépendantes.
3. Calculer la probabilité de déductibilité P de cette expression à partir des probabilités des Hi en utilisant les propriétés suivantes :
   $P_{déd}(\perp)=0$
   $P_{déd}(Hi)=p(Hi)$
   $P_{déd}(\neg Hi)=1-p(Hi)$
   $P_{déd}(F \wedge G)=P_{déd}(F)*P_{déd}(G)$
   $P_{déd}(F \vee G)=P_{déd}(F)+P_{déd}(G)-P_{déd}(F \wedge G)$[40].
4. Répéter les opérations 2 et 3 pour les nogoods et les environnements contenant à la fois un nogood et un élément du label.
5. Utiliser la formule bel(β) = (P([label(β)]) - P([label(β)]∩[nogoods])) /(1 - masse([nogoods]) pour déterminer la croyance de β.

2.2.2 Exemples (issus de [Provan 1989])

Exemple 1 : sans nogood

α = {
x1 ,
x6,
x1 A1 x2,
x2 A2 x3,
x1 A3 x4,
x4 A4 x5,
x2 x4 A5 x5}.

Probabilités
P(A1)=0,5
P(A2)=0,7
P(A3)=0,8
P(A4)=0,6
P(A5)=0,9

Il n'y a pas de nogoods dans α. Prenons par exemple le label de x5 : {{A3,A4},{A1,A3,A5}}. A partir de ce label on peut construire la formule $(A3 \wedge A4) \vee (A1 \wedge A3 \wedge A5)$. Les deux cubes de cette formule ne sont pas mutuellement exclusifs ou indépendants (MEI[41]) car ils contiennent tous les deux A3. On a deux solutions pour remplacer cette formule par une formule logiquement équivalente sous forme MEI :

1. Factoriser A3 :

[40] Cela provient de la relation classique en probabilités p(A∩B)=p(A)+p(B)-p(A∪B) pour A et B deux ensembles quelconques. Notons que si E et F sont deux formules exclusives (E∧F⊨⊥) alors on obtient masse(E∨F)=masse(E)+masse(F).

[41]

A3 ∧(A4 ∨ (A1∧A5))

2. Rendre les 2 cubes mutuellement exclusifs à l'aide de l'équivalence a ∨ b ≡ a ∨ (¬a∧b) :
   (A3∧A4)∨(A1∧A3∧¬A4∧A5)
   Il n'y a pas beaucoup de différence ici. Par contre, si nous avions choisi d'utiliser A1 pour rendre les cubes mutuellement exclusifs, on obtiendrait (¬A1∧A3∧A4)∨(A1∧A3∧¬A4∧A5)∨(A1∧A3∧A4). Il peut donc y avoir plus de cubes qu'au départ. Cette méthode peut donner une formule de taille exponentielle relativement à celle de la formule de départ.

Nous pouvons vérifier qu'en calculant les masses associées à ces deux formules nous obtenons bien le même résultat : P(A3∧(A4∨(A1∧A5))) = P((A3∧A4)∨(A1∧A3∧¬A4∧A5)) = p(A3)p(A4) + p(A1)p(A3)p(A5) - p(A1)p(A3)p(A4)p(A5).

On en déduit la croyance de x5 dans α : 0,8*0,6+0,5*0,8*0,9-0,5*0,8*0,6*0,9 = 0,48+0,36-0,216 = 0,624

L'intérêt d'utiliser une formule DNF mutuellement exclusive est qu'il est alors très simple de calculer la masse de cette formule : il suffit de faire la somme des masses de ses cubes.

Exemple 2 : avec nogoods

Ajoutons maintenant la clause x2 A6 x4 dans α, avec m(A6)=0,4. Nous obtenons maintenant un nogood {A1,A3,A6}.

Le label de x5 ne change pas. Par contre, son degré de croyance va changer : nous allons devoir normaliser la masse du label pour ne prendre en compte que les formules de X consistantes avec α.

Nous représentons sur le schéma suivant l'ensemble des éléments de X, en grisé ceux appartenant au label de x5, ceux contenant un nogood étant entourés.

{A1,A2,A3,A4,A5,A6}	{A1,A2,A3,A4,A5,¬A6}	{A1,A2,A3,A4,¬A5,A6}	{A1,A2,A3,A4,¬A5,¬A6}
{A1,A2,A3,¬A4,A5,A6}	{A1,A2,A3,¬A4,A5,¬A6}	{A1,A2,A3,¬A4,¬A5,A6}	{A1,A2,A3,¬A4,¬A5,¬A6}
{A1,A2,¬A3,A4,A5,A6}	{A1,A2,¬A3,A4,A5,¬A6}	{A1,A2,¬A3,A4,¬A5,A6}	{A1,A2,¬A3,A4,¬A5,¬A6}
{A1,A2,¬A3,¬A4,A5,A6}	{A1,A2,¬A3,¬A4,A5,¬A6}	{A1,A2,¬A3,¬A4,¬A5,A6}	{A1,A2,¬A3,¬A4,¬A5,¬A6}
{A1,¬A2,A3,A4,A5,A6}	{A1,¬A2,A3,A4,A5,¬A6}	{A1,¬A2,A3,A4,¬A5,A6}	{A1,¬A2,A3,A4,¬A5,¬A6}
{A1,¬A2,A3,¬A4,A5,A6}	{A1,¬A2,A3,¬A4,A5,¬A6}	{A1,¬A2,A3,¬A4,¬A5,A6}	{A1,¬A2,A3,¬A4,¬A5,¬A6}
{A1,¬A2,¬A3,A4,A5,A6}	{A1,¬A2,¬A3,A4,A5,¬A6}	{A1,¬A2,¬A3,A4,¬A5,A6}	{A1,¬A2,¬A3,A4,¬A5,¬A6}
{A1,¬A2,¬A3,¬A4,A5,A6}	{A1,¬A2,¬A3,¬A4,A5,¬A6}	{A1,¬A2,¬A3,¬A4,¬A5,A6}	{A1,¬A2,¬A3,¬A4,¬A5,¬A6}
{¬A1,A2,A3,A4,A5,A6}	{¬A1,A2,A3,A4,A5,¬A6}	{¬A1,A2,A3,A4,¬A5,A6}	{¬A1,A2,A3,A4,¬A5,¬A6}
{¬A1,A2,A3,¬A4,A5,A6}	{¬A1,A2,A3,¬A4,A5,¬A6}	{¬A1,A2,A3,¬A4,¬A5,A6}	{¬A1,A2,A3,¬A4,¬A5,¬A6}
{¬A1,A2,¬A3,A4,A5,A6}	{¬A1,A2,¬A3,A4,A5,¬A6}	{¬A1,A2,¬A3,A4,¬A5,A6}	{¬A1,A2,¬A3,A4,¬A5,¬A6}
{¬A1,A2,¬A3,¬A4,A5,A6}	{¬A1,A2,¬A3,¬A4,A5,¬A6}	{¬A1,A2,¬A3,¬A4,¬A5,A6}	{¬A1,A2,¬A3,¬A4,¬A5,¬A6}
{¬A1,¬A2,A3,A4,A5,A6}	{¬A1,¬A2,A3,A4,A5,¬A6}	{¬A1,¬A2,A3,A4,¬A5,A6}	{¬A1,¬A2,A3,A4,¬A5,¬A6}
{¬A1,¬A2,A3,¬A4,A5,A6}	{¬A1,¬A2,A3,¬A4,A5,¬A6}	{¬A1,¬A2,A3,¬A4,¬A5,A6}	{¬A1,¬A2,A3,¬A4,¬A5,¬A6}
{¬A1,¬A2,¬A3,A4,A5,A6}	{¬A1,¬A2,¬A3,A4,A5,¬A6}	{¬A1,¬A2,¬A3,A4,¬A5,A6}	{¬A1,¬A2,¬A3,A4,¬A5,¬A6}
{¬A1,¬A2,¬A3,¬A4,A5,A6}	{¬A1,¬A2,¬A3,¬A4,A5,¬A6}	{¬A1,¬A2,¬A3,¬A4,¬A5,A6}	{¬A1,¬A2,¬A3,¬A4,¬A5,¬A6}

Ensemble d'environnements	Formule ME associée	Masse
Label(x5)= {{A_3,A_4},{A_1,A_3,A_5}} Nogoods={{A_1,A_3,A_6}} [Label(x5)]∩[Nogoods]={{A_1,A_3,A_4,A_6},{A_1,A_3,A_5,A_6}}	(A_3∧A_4)∨(A_1∧A_3∧¬A_4∧A_5) A_1∧A_3∧A_6 (A_1∧A_3∧A_4∧A_6)∨(A_1∧A_3∧¬A_4∧A_5∧A_6)	0,624 0,16 0,1536
[Label(x5)]\[Nogoods]= {{A3,A4,¬A5,¬A6},{¬A1,A3,A4,A6},{A1,A3,A5,¬A6},{¬A1,A3,A4,A5,¬A6}}	(A3∧A4∧¬A5∧¬A6)∨(¬A1∧A3∧A4∧A6)∨(A1∧A3∧A5∧¬A6)∨(¬A1∧A3∧A4∧A5∧¬A6)	0,4704

On en déduit la croyance de x5 : bel(x5)=(0,624-0,1536)/(1-0,16) = 0,56

2.2.3 Aspects calculatoires

Le point délicat de la méthode présentée par Provan est l'obtention de formules mutuellement exclusives ou indépendantes. Il propose donc de transformer ce problème en un problème équivalent largement traité dans la théorie des graphes : le « *network reliability problem*» (trouver la probabilité qu'un chemin existe entre deux nœuds d'un graphe dont les arcs sont valués par des probabilités). Nous n'allons pas rentrer ici dans le détail de ce problème.

Nous proposons de calculer le label (et les nogoods) directement sous la forme de cubes mutuellement exclusifs, grâce à notre second calcul d'impliquants P-restreints, ce qui nous permettra de calculer en même temps la probabilité de ce label (ou ces nogoods). En effet, nous avons vu que la probabilité associée à une formule est la somme des probabilités des H-interprétations qui permettent de la déduire. Lors de notre seconde passe, la base que nous traitons ne contient que des littéraux de H. Notre algorithme de calcul d'impliquants H-restreints premiers effectue donc une énumération complète des H-interprétations. On peut donc utiliser une variante de cet algorithme afin de calculer le degré de croyance d'une formule en notant que :

- Un H-impliquant est une représentation compacte d'un ensemble de H-interprétations, et que la probabilité d'un impliquant est égale à la somme des probabilités de ces H-interprétations.
- Un Davis et Putnam sans arrêt au premier impliquant permet de calculer une couverture d'impliquants d'une formule.
- Ces impliquants sont mutuellement exclusifs.

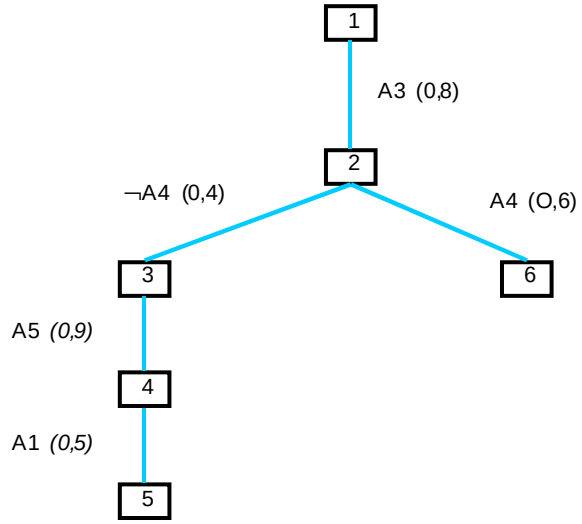
Nous sommes donc en mesure de calculer, pour toute formule CNF *BC* telle que *BC* est constituée uniquement de littéraux purs :

- $\alpha = IRP(BC,P)^{\vee} \equiv BC$ la disjonction des impliquants P-restreints premiers de *BC*,
- $\alpha' \equiv BC$ une couverture d'impliquants mutuellement exclusifs de *BC*.

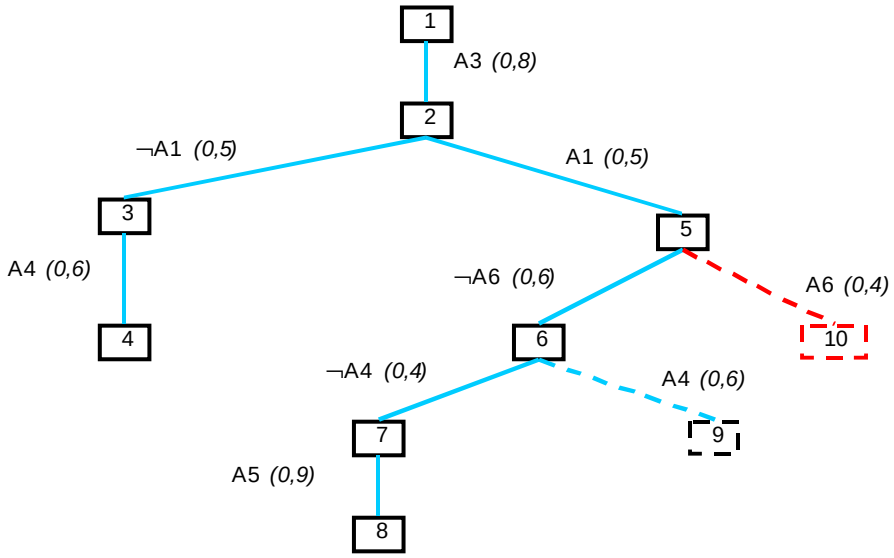
Bien sûr, $\alpha \equiv \alpha'$. Nous pouvons donc relativement facilement obtenir notre label (nos nogoods) sous forme ME.

Nous allons de plus utiliser une autre particularité de notre méthode afin d'éviter certains calculs nécessaires lorsque le calcul du degré de croyance est effectué à partir des données de l'ATMS. On note en effet dans la méthode de Provan la nécessité de calculer le degré de croyance de $[label(\beta)] \cap [Nogoods]$. Cela provient du fait qu'un label représente un ensemble de H-interprétations, dont certaines sont inconsistantes avec la base. Il faut donc enlever la probabilité de ces H-interprétations inconsistantes à la masse du label. Dans notre cas, nous disposons de plus d'informations. En effet, nous ajoutons à la base la négation des nogoods : on peut considérer que nous n'énumérons que les H-interprétations consistantes avec la base. Nous obtenons une formule α' dont les cubes représentent seulement des ensembles de H-interprétations consistantes avec *BC*. Nous n'avons donc pas à calculer l'intersection du label et des nogoods ni a calculer sa probabilité.

En ayant en plus connaissance des probabilités associées à chaque symbole hypothèse, nous sommes alors en mesure de calculer directement la masse d'un label ou des nogoods. Si nous reprenons l'exemple 1 et que nous regardons les branches développées par MPL, on trouve bien deux solutions mutuellement exclusives (nœud 2) logiquement équivalentes à notre label.



Mais cela ne suffit pas. Reprenons maintenant l'exemple 2. Voici l'arbre développé par notre algorithme pour trouver le label de x5 : {{A3,A4},{A1,A3,A5}}.



Les branches en pointillés ne sont pas développées car elles produisent soit un impliquant P-restreint non minimal (nœud 9) soit un nogood (nœud 10). Nous voyons dans l'arbre que les deux solutions se rejoignent au nœud 2. Si l'on ajoute ¬A1 dans le premier environnement, alors on obtient bien deux environnements mutuellement exclusifs. Malheureusement, la formule associée (¬A1∧A3∧A4)∨(A1∧A3∧A5) n'est pas logiquement équivalente à (A3∧A4)∨(A1∧A3∧A5).

Pourquoi ? Tout simplement parce que nous utilisons 2 propriétés des impliquants P-restreints premiers dans notre calcul :

- Une formule construite à partir d'un ensemble consistant de littéraux P est logiquement équivalente à la disjonction de ses impliquants P-restreints premiers.
- Ajouter la négation des nogoods à cette formule nous permet d'en empêcher la production par l'algorithme.

La première propriété n'est plus applicable lorsque l'on cherche une couverture d'impliquants « quelconques ». Nous devons donc enlever le test de minimalité en ce qui concerne le parcours de l'arbre de recherche afin de calculer réellement une couverture d'impliquants (on retombe alors sur une utilisation classique de DP pour calculer une couverture d'impliquants [Mazure/Marquis 1996, Schrag 1996]). Elle sera seulement utilisée pour ne mémoriser que les impliquants P-restreints premiers.

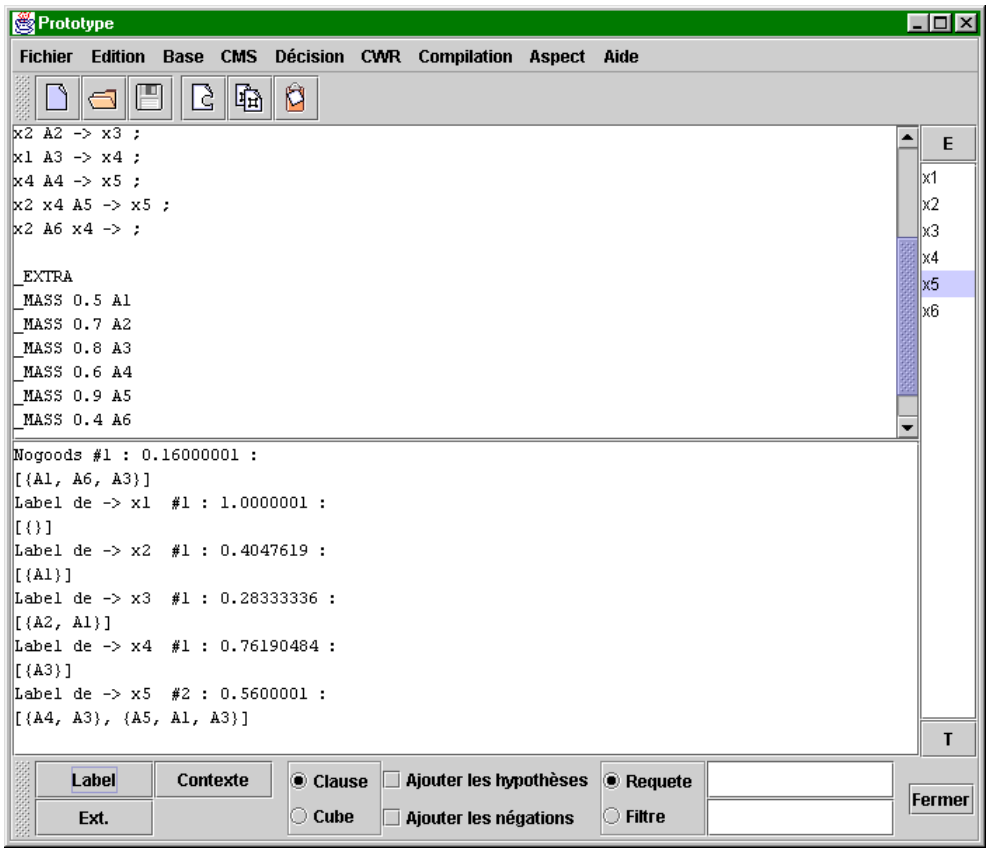
En ce qui concerne les nogoods, on peut considérer que l'on enlève de l'ensemble des éléments de X représentés par le label de x5 ceux inconsistants avec la base. Il est donc normal que nous n'obtenions pas un résultat logi-

quement équivalent à ce label. Nous obtenons une formule logiquement équivalente à [Label(x5)]\[Nogoods] : $(\neg A_1 \wedge A_3 \wedge A_4) \vee (A_1 \wedge A_3 \wedge \neg A_4 \wedge A_5 \wedge \neg A_6) \vee (A_1 \wedge A_3 \wedge A_4 \wedge \neg A_6) \equiv (A_3 \wedge A_4 \wedge \neg A_5 \wedge \neg A_6) \vee (\neg A_1 \wedge A_3 \wedge A_4 \wedge A_6) \vee (A_1 \wedge A_3 \wedge A_5 \wedge \neg A_6) \vee (\neg A_1 \wedge A_3 \wedge A_4 \wedge A_5 \wedge \neg A_6)$.

```
DSMPL(BC,BA,P,IP,p)
// Calcule les impliquants P-restreints premiers de BC.
// IP contient les littéraux satisfaits.
// BA est la base de minimisation : le résultat se trouve
// sous forme clausale (niée) dans cette base.
// p une distribution de probabilité sur les éléments de P.
// retourne la probabilité des impliquants P-restreints de BC.
        Si    ∈ BC alors retourner 0 Finsi ;
        Si BC = ∅ alors
                Si BA∧IP |≠ ⊥ alors
                        // impliquant P-restreint premier : on le
                        // met dans BA
                        BA ← BA ∪ {((IP∩P)^¬)}
                Finsi ;
                retourner 1;
        Finsi
        // Attention : ici on peut seulement simplifier par clauses
        // unitaires : il faut préserver l'équivalence logique
        l ← LittéralPourSimplifier(BC) ;
        Si (l≠null) alors // si l existe
                Res = DSMPL(BC[l],BA,P,IP∪{l},p) ;
                Retourner Res*p(l)
        Finsi
        l ← Choix_Symbole_de_P(BC,P) ;
        Si (l≠null) alors // il reste des variables de P à assigner
                Resg = MPL(BC[¬l],BA,P,IP∪{¬l},p) ;
                Resd = MPL(BC[l],BA,P,IP∪{l},p} ;
                Retourner Resd*p(l)+resg*(1-p(l))
        Sinon // on effectue seulement un test de consistance
                Si DP(BC)=vrai alors
                        Si BA |≠ ⊥ alors
                                // on a un impliquant P-restreint premier
                                BA ← BA ∪ {((IP∩P)^¬)} ;
                        Finsi ;
                        Retourner 1
                Sinon
                        Retourner 0
                Finsi
        Finsi
```

Algorithme 12 DSMPL - degré de croyance d'un ensemble d'impliquants P-restreints premiers

La méthode en soit n'est pas révolutionnaire, car elle correspond dans le domaine du network reliability à une technique appelée dans ce domaine « *Sum of Disjoint Products* (SDP) » présentée la première fois par [Fratta/Montanari 1973] et dont la version améliorée par [Locks 1987] est la meilleure approche pour résoudre ce problème. L'intérêt de notre approche par rapport à celle de Provan est que le calcul du degré de croyance se fait *pendant* le calcul du label et non pas *après*. Il y a certes un surcoût par rapport à MPL lié au pouvoir de coupe de la détection de non minimalité, mais sans doute moins important que celui lié à la transformation du résultat en un graphe et l'utilisation d'un algorithme spécialisé de network reliability. De plus, comme nous l'avons vu précédemment, il ne nous est pas nécessaire de calculer explicitement la probabilité des H-interprétations inconsistantes avec la base présentes dans le label comme cela est nécessaire quand on utilise les seules données de l'ATMS.


Prototype
Fichier Edition Base CMS Décision CWR Compilation Aspect Aide
x2 A2 -> x3 ;
x1 A3 -> x4 ;
x4 A4 -> x5 ;
x2 x4 A5 -> x5 ;
x2 A6 x4 -> ;
_EXTRA
_MASS 0.5 A1
_MASS 0.7 A2
_MASS 0.8 A3
_MASS 0.6 A4
_MASS 0.9 A5
_MASS 0.4 A6
E
x1
x2
x3
x4
x5
x6
Nogoods #1 : 0.16000001 :
[{A1, A6, A3}]
Label de -> x1 #1 : 1.0000001 :
[{}]
Label de -> x2 #1 : 0.4047619 :
[{A1}]
Label de -> x3 #1 : 0.28333336 :
[{A2, A1}]
Label de -> x4 #1 : 0.76190484 :
[{A3}]
Label de -> x5 #2 : 0.5600001 :
[{A4, A3}, {A5, A1, A3}]
T
Label
Contexte
Ext.
Clause
Cube
Ajouter les hypothèses
Ajouter les négations
Requete
Filtre
Fermer

Partie IV : P-impliqués préférés

1 Motivations

Nous avons montré précédemment une méthode de calcul de P-impliqués premiers fondée sur la notion d'impliquant P-restreint premier. Le calcul de ces derniers était effectué par un algorithme dérivé d'un algorithme bien connu dans le domaine SAT : la procédure de Davis et Putnam.
Nous avons souligné que notre méthode de calcul avait comme particularités :

1. D'énumérer les impliquants P-restreints suivant un ordre compatible avec l'inclusion ensembliste.
2. De ne pas effectuer explicitement un test de sous-sommation mais d'utiliser des mécanismes courants dans le domaine SAT pour garantir la primarité des solutions.
3. De bénéficier des travaux effectués autour de SAT au niveau notamment des heuristiques et des algorithmes.

Nous avons aussi souligné que notre approche de calcul de P-impliqué premier avait certaines limites :

1. P doit être un ensemble consistant de littéraux.
2. La minimisation par ajout de clauses semble peu performante quand il y a beaucoup de solutions.
3. Notre méthode de calcul en deux passes nous oblige à nous acquitter obligatoirement de la première passe avant de trouver une solution.

1.1 Objectifs

Notre méthode de calcul de P-impliqués premiers est originale sur divers points. Dans quelle mesure peut-on la « généraliser » ?
Pour l'instant, nous produisons tous les P-impliqués premiers d'une formule. Or il n'est pas toujours nécessaire de les produire tous.

- Dans le cadre du diagnostic de panne par exemple, une propriété syntaxique comme la cardinalité permet souvent de réduire considérablement le nombre de solutions : on préfère se concentrer sur les pannes qui concernent un nombre minimal de composants.
- La connaissance des probabilités de panne de chaque composant permet de limiter l'ensemble des solutions aux plus probables.
- L'expérience d'un expert peut l'amener à préférer les solutions mettant en cause tel composant plutôt que tel autre.

De plus, présenter 10000 solutions à un opérateur ne l'avance pas beaucoup. Il faut lui permettre de filtrer ces solutions selon différents critères. Ceux-ci pourront être globaux (la minimalité pour la cardinalité) ou locaux (préférer tel composant à tel autre), et utiliser des connaissances exogènes (des probabilités, un ordre partiel, etc.).

Nous cherchons maintenant des « P-impliqués préférés ». Les relations de préférence que nous allons utiliser seront diverses. Nous limitons notre étude au cas de relations de préférence compatibles avec l'inclusion ensembliste car ainsi nous pouvons bénéficier des travaux déjà effectués sur les P-impliqués premiers. Ces relations de préférence seront donc des raffinements de la notion de primarité.

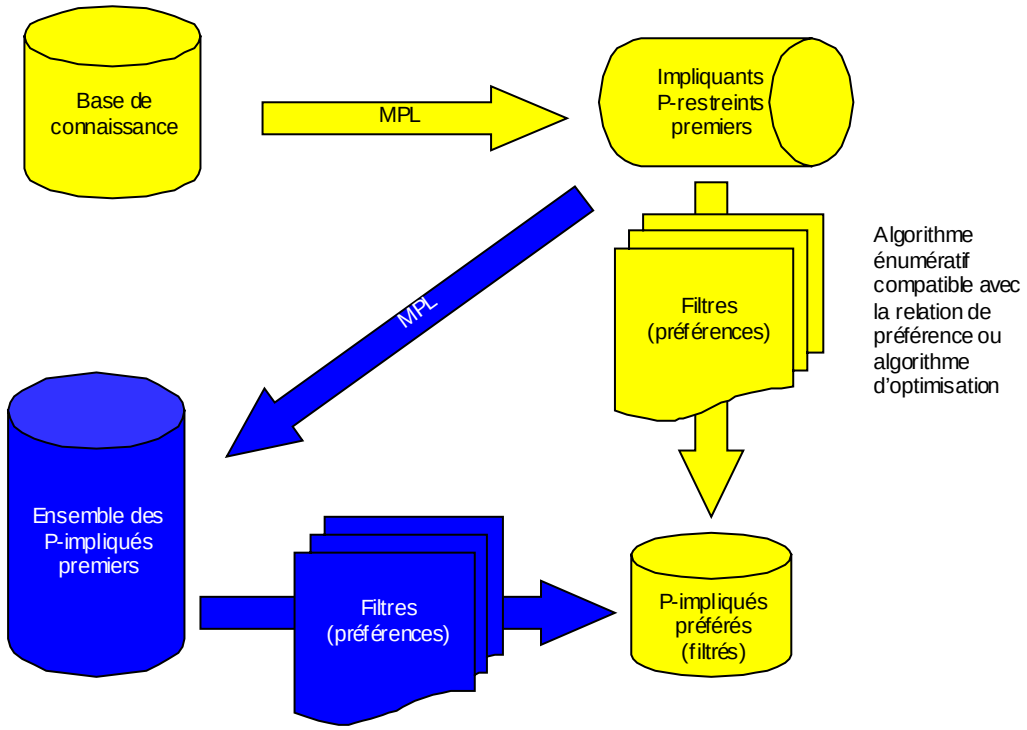
Nous allons présenter dans cette partie différentes solutions algorithmiques. Nous cherchons à garder les mêmes propriétés que MPL dans le cadre d'une relation de préférence autre que l'inclusion ensembliste. Cela ne sera pas toujours possible.

1.2 Applications

Une de nos premières applications de notre calcul de P-impliqués premiers a été l'argumentation. Nous avons présenté dans [Amgoud, et al. 1996] comment utiliser un ATMS pour calculer des arguments, puis comment classer ensuite les différents arguments à l'aide de connaissances exogènes (une stratification de la base de

connaissance) pour ne garder que les préférés. Ce travail n'était pas satisfaisant car il nous obligeait à calculer tous les arguments afin de pouvoir les classer. L'ATMS ne servait qu'à produire les différents arguments. D'où l'idée d'introduire la relation de préférence au niveau du calcul des arguments pour ne produire que les préférés.

Le schéma suivant représente les divers étapes du calcul de P-impliqués préférés : le chemin gris représente l'approche que nous allons développer dans cette partie.



Nous allons pour cela définir la notion d'impliquant P-restreint préféré. Elle généralise la notion d'impliquant P-restreint premier à toute autre relation de préférence compatible avec l'inclusion ensembliste. A cette notion nous ajouterons celle de P-impliquant/P-impliqué préféré.

Le calcul de P-impliqué préféré va nous permettre de définir une sorte d'ATMS capable de calculer uniquement les environnements préférés. Nous présentons dans cette optique le concept de compilation de solution. En effet, notre méthode de calcul nous oblige à calculer d'abord les impliquants P-restreints premiers de la base. On peut considérer que le résultat obtenu est l'ensemble des P-impliqués premiers sous forme compilée. De cette forme compilée, nous montrerons comment filtrer les solutions selon les besoins de l'utilisateur.

1.3 Compilation de solutions

L'idée de base est la suivante : imaginons que les P-impliqués premiers que nous calculons soient les explications possibles d'une panne d'un système. Ils sont présentés à un opérateur qui au vu des solutions doit effectuer la réparation nécessaire. Il dispose de certaines informations acquises par une longue expérience du système. Si nous lui présentons 1000 solutions, il ne pourra rien en faire. Il faut lui donner les moyens de filtrer ces solutions. Supposons qu'il suspecte un composant A d'être en panne. Il doit pouvoir accéder à toutes les explications contenant A. De même, s'il dispose de probabilités de panne, on doit pouvoir lui présenter les pannes les plus probables. Nous devons de même pouvoir calculer les explications les plus probables contenant A.

Le cadre du diagnostic fait depuis longtemps l'objet de travaux en ce sens [de-Kleer/Williams 1987, de-Kleer 1991]. L'ATMS a rapidement été étendu pour calculer les explications les plus probables. Mais peu de travaux se sont tournés à notre connaissance sur la manipulation des solutions. Le plus souvent, ce sont de nouvelles informations issues de capteurs qui arrivent dans la base de connaissances et modifient l'ensemble des solutions (incrémentalement dans le cadre de l'ATMS par exemple). Mais il n'est pas facile de revenir en arrière. Nous proposons une approche consistant à calculer hors-ligne l'ensemble des explications d'une panne du système sous la forme d'une base de clauses. Cet ensemble de solutions sera ensuite filtré par les requêtes d'un opérateur à l'aide de connaissances exogènes.

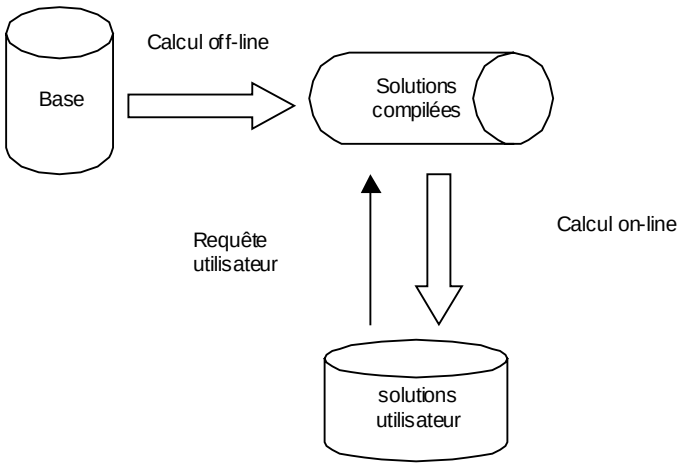


Figure 7 Schéma de principe de la compilation de solutions

Nous allons illustrer cette approche par quelques résultats obtenus dans le domaine de la planification, issus d'une tentative d'application de nos travaux avec Hector Geffner [Bonet/Geffner 1998]. En planification, on dispose d'un ensemble d'actions a effectuer qui sont généralement exprimées sous la forme (préconditions, ajouts, retraits). Exemple :

Action : charger la voiture V1 sur le bateau B1
Préconditions : *la voiture V1 et le bateau Q sont sur le même quai Q*
Ajouts : *la voiture V1 est sur le bateau B1*
Retraits : *la voiture n'est plus sur le quai Q*

On cherche, à partir d'une situation initiale, une *séquence* d'actions à effectuer pour atteindre un but. Ce qui peut se traduire en logique par un raisonnement abductif (recherche du label du but lorsque les hypothèses sont des actions à effectuer) avec un codage approprié du temps. Par exemple

(1) Charger$V1B1_t$ sur$V1Q_t$ sur$B1Q_t$ sur$V1B1_{t+1}$ et

(2) Charger$V1B1_t$ sur$V1Q_t$ sur$B1Q_t$ sur$V1Q_{t+1}$ $\perp$

permettent de décrire l'action « charger une voiture » à l'étape t. Le but et la situation initiale seront exprimées par un ensemble de clauses unitaires. Mais ce codage est très lourd car il faut multiplier le nombre de règles par le nombre d'étapes maximal autorisé. Si l'on cherche maintenant uniquement un *ensemble* d'actions à effectuer pour atteindre le but, on peut coder les ajouts à l'aide de clauses définies (1) sans utilisation explicite du temps. Par contre, les retraits (2) ne sont pas exprimables sans prise en compte explicite du temps. D'où l'idée d'Hector Geffner d'exprimer sous forme logique une version simplifiée du problème de planification en ne tenant pas compte les retraits. Il obtient ainsi une base de clauses définies RG. L'idée est de calculer une heuristique pour le problème réel à partir de la taille du plan minimal (en nombre d'actions) pour le problème simplifié. D'un point de vue logique, il s'agit de calculer E un ensemble d'actions tel que $RG \wedge SI \wedge E^{\wedge} \models but$ et E minimal pour la cardinalité, avec SI la situation initiale, ce qui équivaux à calculer la taille minimale des environnements du label de but dans la base $RG \wedge SI$. Donc à calculer des P-impliqués CARD-préférés de $RG \wedge SI \wedge \neg but$ ($RG \wedge SI$ ne peut pas contenir de nogoods car toutes les clauses sont définies).

Les exemples suivants sont issus de problèmes classique en planification (le monde des cubes, blocks1, et logistics, logX). A titre d'indication, le premier problème contient 130 symboles dont 100 dans P et 180 clauses, le second 362 symboles dont 210 dans P et 252 clauses, le dernier 451 symboles dont 256 dans P et 304 clauses. Tests effectués sur un Pentium 133, 32Mo, JDK1.2.2+hotspot pour Windows.

Problème	1ère passe		2ème passe Inclusion			2ème passe Cardinalité			2ème passe Cardinalité	
	# sol	Temps (s)	# sol	MPL / Castell (s)		# sol	Taille	Temps (s)	# sol	Temps (s)
Blocks1	13	2,19	608	7,47	4,17	38	4	1,1	1	0,4
Log1	212	6144	586754[42]	>16000	>2563	8	47	130	1	79
Log2	431	73379	1631964	?	?	50	37	215	1	119

[42] Le nombre de solutions pour les problèmes Log1 et Log2 a été obtenu par une version de l'algorithme de Thierry Castell calculant les solutions sans les mémoriser (en respectivement 4632s et 33757s sur un Mac Power G3 300 Mhz/64 Mo/MRJ 2.1.4).

On note que généralement le temps de calcul de la première passe est plus petit que celui de la deuxième passe en utilisant comme préférence l'inclusion ensembliste. On remarque dans le premier exemple que les 608 P-impliqués premiers sont calculables à partir de 13 clauses seulement ! Si l'on cherche uniquement les P-impliqués premiers minimaux pour la cardinalité, on n'en trouve que 38 en 1,8s, soit en 4 fois moins de temps. Si l'on cherche uniquement la cardinalité du plus petit P-impliqué premier, on obtient la solution en 0,4 secondes, soit plus de 10 fois moins de temps que l'obtention de toutes les solutions !
Restreindre le nombre de solutions cherchées est encore plus intéressant dans les deux autres problèmes : il ne nous a pas été possible après plus de 3 heures de trouver toutes les solutions de Log1 avec MPL (lorsque nous l'avons arrêté, il en avait trouvé plus de 15000 !). A l'aide de l'algorithme de Thierry Castell, nous avons trouvé plus de 106800 solutions en 2563s, mais nous avons stoppé l'algorithme à cause de problèmes de mémoire. Par contre, en un peu moins de 4 minutes nous obtenons toutes les solutions minimales pour la cardinalité de Log1 et Log2.
Ces exemples nous montrent bien l'idée de compilation de solutions : plus de 500000 solutions regroupées en 212 clauses[43] ! Nous pourrions arrêter MPL(ou Castell) après un certain nombre de solutions, mais nous ne pourrions pas caractériser l'ensemble des impliquants P-restreints premiers obtenus : pourquoi considérer ces solutions et pas les autres ? Dans le cas de la cardinalité, calculer tous les impliquants P-restreints premiers minimaux pour la cardinalité ou seulement l'un d'entre eux a un sens.

2 Enumération d'impliquants P-restreints préférés

Nous allons présenter ici diverses variantes de MPL et d'autres algorithmes de la littérature capables d'énumérer des impliquants P-restreints selon une relation de préférence compatible avec l'inclusion ensembliste.

2.1 Impliquants P-restreints préférés

Nous allons maintenant étendre notre définition d'impliquants P-restreints premiers à des relations de préférence autres que l'inclusion ensembliste : les impliquants P-restreints préférés. Ces relations de préférence sont des préordres partiels entre ensembles de littéraux de P. Nous préfixerons le terme préféré du nom de la relation de préférence pour plus de clarté (nous parlerons par exemple d'impliquants P-restreints CARD-préférés pour des impliquants P-restreints minimaux pour la cardinalité).

Définition 62 (impliquant P-restreint R-préféré)
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble de littéraux. Soit $E \subseteq P$. Soit R une relation de préférence entre ensembles de littéraux de P. On dira que $E^{\wedge}$ est un impliquant P-restreint R-préféré de α ssi $E^{\wedge}$ est un impliquant P-restreint de α et $\nexists\ E'^{\wedge}$ un impliquant P-restreint de α tel que $E' \neq E$ et $E'\ R\ E$.

> On peut vérifier qu'en prenant pour relation de préférence l'inclusion ensembliste, on retombe sur la définition d'impliquants P-restreints premiers.

On notera l'ensemble des impliquants P-restreints R-préférés d'une formule α *IRPREF(α,P,R)* (contre *IRP(α,P)* pour l'ensemble des impliquants P-restreints premiers).

On définit de même les notions de P-impliquants/P-impliqués préférés.

Définition 63 (P-impliquants/P-impliqués R-préférés)
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble de littéraux. Soit $E \subseteq P$. Soit R une relation de préférence entre ensembles de littéraux de P. On dira que $E^{\wedge}$ est un P-impliquant *(*resp. $E^{\vee}$ est un P-impliqué*)* R-préféré de α ssi $E^{\wedge}$ est un P-impliquant (resp. P-impliqué) de α et $\nexists\ E'^{\wedge}$ un P-impliquant *(*resp. $E'^{\vee}$ est un P-impliqué*)* de α tel que $E' \neq E$ et $E'R\ E$.

Nos définitions ne donnent aucune restriction sur la relation de préférence utilisée. Nous allons pourtant réduire nos travaux au cas de relations de préférence compatibles avec l'inclusion. Pourquoi ?

- Car ces relations de préférence raffinent l'inclusion ensembliste, c'est à dire qu'elles permettent de réduire le nombre de formules produites par notre algorithme de calcul d'impliquants P-restreints premiers.
- Car de nombreux algorithmes utilisent implicitement l'inclusion ensembliste (Davis et Putnam par exemple).
- Car les propositions démontrées précédemment se généralisent alors pour la notion d'impliquants P-restreints R-préférés.

[43] On obtient ces résultats car toutes les clauses sont définies donc il n'y a pas de nogoods.

Nous supposons donc par la suite que IRPREF(α,P,R) ⊆ IRP(α,P).

On peut à partir des résultats de la partie II montrer que les P-impliquants/P-impliqués R-préférés se calculent à partir des impliquants P-restreints R-préférés.

Proposition 34
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. $E^\wedge$ est un P-impliquant R-préféré de α ssi $E^\wedge \in IRPREF(reduc(\alpha,P),P,R)$.

> **Preuve :** immédiate à partir de la Proposition 21 page 62. Il suffit de remplacer dans la preuve de cette proposition la preuve de la minimalité pour l'inclusion par la preuve de la préférence pour R (car R est compatible avec l'inclusion).

Pour calculer les P-impliqués R-préférés, nous devons introduire une nouvelle notation.

Définition 64 (relation complémentaire)
$R^\neg$ est la relation de préférence telle que $\forall E,F \subseteq P$, si $E\ R\ F$ alors $E^\neg\ R^\neg\ F^\neg$.

Exemple : dans le cadre de l'inclusion ensembliste, nous avions $\subseteq^\neg = \subseteq$: $\{A,B\} \subseteq \{A,B,C\}$ et $\{\neg A,\neg B\} \subseteq \{\neg A,\neg B,\neg C\}$.

Proposition 35
Soit $\alpha \in FP_S$. Soit $P \subset L_S$ un ensemble **consistant** de littéraux. Soit $E \subseteq P$. $E^\vee$ est un P-impliqué R-préféré de α ssi $E^{\neg\wedge} \in IRPREF(IRP(\alpha,P)^{\wedge\neg},P^\neg,R^\neg)$.

> **Preuve :** découle de la Proposition 26 page 64, en remplaçant la preuve de minimalité pour l'inclusion par la preuve de la préférence pour R.

Certaines relations de préférences permettent en plus de caractériser l'ensemble des solutions préférées. Pour la minimalité pour la cardinalité par exemple, les solutions ont toutes la même taille. Il peut alors être judicieux de calculer une seule des solutions préférées afin de connaître cette caractéristique (on cherchera par exemple la taille d'un plan minimal). C'est pourquoi la caractéristique « la première solution trouvée est préférée » de MPL est intéressante à conserver.

Nous allons montrer dans un premier temps comment nous pouvons modifier la procédure MPL pour calculer des impliquants P-restreints préférés. Comme nous considérons des relations de préférence compatibles avec l'inclusion, nous cherchons à contraindre notre algorithme MPL pour qu'il ne fournisse que les impliquants P-restreints R-préférés : ainsi, nous développerons « au pire des cas » le même nombre de nœuds que MPL. Quand cela ne sera pas possible, nous essayerons de trouver (dans la littérature de la recherche heuristique) des algorithmes permettant de conserver le principe de MPL : une recherche arborescente garantissant que la première solution trouvée est R-préférée.

Pour introduire les différentes relations de préférence, nous devons tout d'abord faire une remarque concernant les problèmes traitant d'ensembles de clauses à la place d'ensembles de littéraux.

2.2 Ensemble de littéraux vs ensemble de clauses

Il existe des problèmes en raisonnement non monotone qui nécessitent non pas la recherche de P-impliquants ou de P-impliqués d'une base de clauses *BC*, mais des sous-bases de *BC*, c'est à dire des ensembles de clauses $BC' \subseteq BC$. On cherchera par exemple une sous-base minimale incohérente, afin de connaître la nature de l'incohérence : incohérence locale si seules quelques clauses de la base entraînent cette incohérence, incohérence globale si toutes les clauses sont la cause de l'incohérence. Dans les deux cas on pourra restaurer la cohérence en éliminant par exemple une clause de chaque sous-base minimale incohérente. On peut de manière duale calculer les sous-bases maximales cohérentes d'une base de clauses.
Ces deux notions sont semblables à la notion de nogood et d'interprétation de l'ATMS. Nous allons montrer ici comment calculer via un simple encodage de la base de connaissances ces sous-bases minimales incohérentes ou maximales cohérentes. Nous étendrons ensuite ce principe pour calculer le support d'une donnée dans le cadre de l'argumentation.

Nous considérons que *BC* contient à la fois des connaissances (elles ne peuvent pas être remises en causes) et des croyances (qui peuvent être remises en causes). Nous noterons $BC = KB \cup CR$ avec *KB* les connaissances et *CR* les croyances $(KB \cap CR = \emptyset)$. *KB* peut être vide et doit être consistante. *CR* peut être inconsistante.

Définition 65 (sous-base minimale incohérente/maximale cohérente)
$CR' \subseteq CR$ est une sous base minimale incohérente ssi $KB \cup CR' \models \bot$ et $\nexists\ CR'' \subset CR'$ telle que $KB \cup CR'' \models \bot$.
$CR' \subseteq CR$ est une sous base maximale cohérente ssi $KB \cup CR' \not\models \bot$ et $\nexists\ CR'' \subseteq CR$ telle que $CR' \subset CR''$ et $KB \cup CR'' \models \bot$.

Voyons tout d'abord comme nous allons coder notre base de clauses *CR* :

Définition 66 (encodage/décodage d'une base de clauses)
Soit $CR \subset FP_S$ une base de clauses. Soit $N = |CR|$. Soit $A = \{A_1, A_2, \ldots, A_N\}$ un ensemble de nouveaux symboles propositionnels n'apparaissant pas dans *CR*. Encode(*CR*,A) = { Ai Ci | Ci∈*CR*, Ai∈A 1≤i≤N}. Soit E⊆A. Décode (*CR*,E) = {C | A_i C ∈ *CR* et A_i ∈ E}.

On peut noter que Encode(*CR*,A) et Décode (*CR*,E) sont bien des bases de clauses.

Exemple : reprenons notre enquête. *KB = (OBS = { penn big , festnoz, psg }) ∪ { psg chez , chez festnoz , festnoz rhu }*. Ces faits sont certains car ils proviennent des vérifications faites par les gendarmes et du fait qu'un homme n'a pas le don d'ubiquité. *CR = BZH \ KB* (les témoignages peuvent être remis en cause). Nous avons déjà noté que *BC = KB ∪ CR* est inconsistante. Voici la base obtenue par *Encode(CR,A)* avec *A={A1,…, A8}*.

Encode(CR,A) = *{* *A1 psg pb,* *A2 chez pb ,*
A3 chez yb, *A4 festnoz yb,*
A5 festnoz freg, *A6 rhu freg,*
A7 freg yb big, *A8 pb yb penn*
}

Montrons maintenant comment calculer une sous base minimale incohérente de *BC*.

Proposition 36
Soit $BC \subset FP_S$ une base de clauses. Soit BC=<KB,CR> une partition des clauses de *BC* en connaissances et en croyances. Soit $N = |CR|$. Soit $A = \{A_1, A_2, \ldots, A_N\}$ un ensemble de nouveaux symboles propositionnels n'apparaissant pas dans *CR*. $CR' \subseteq CR$ est une sous base minimale incohérente de *CR* ssi $\exists\ E^{\neg\vee}$ un $A^{\neg}$-impliqué premier de $KB \cup CR''$ avec *CR''*=Encode(*CR*,A) tel que Décode (*CR*'',E) = *CR*' ssi ∃ E un nogood de $KB \cup CR''$ avec *CR''*=Encode(*CR*,A) tel que Décode (*CR*'',E) = *CR*'.

Preuve :
Montrons tout d'abord que E est un nogood de $KB \cup CR''$ avec *CR*''=Encode(*CR*,A) ssi Décode(*CR*'',E) est une sous-base minimale incohérente de *CR* (on suppose que l'ensemble des hypothèses est A). E est une nogood ssi $CR''^{\wedge} \wedge E \models \bot$. Donc $KB^{\wedge} \wedge CR''^{\wedge} \wedge E^{\wedge} \wedge (A \backslash E)^{\neg\wedge} \models \bot$. Donc toutes les clauses Ai Ci telles que Ai∈A\E sont satisfaites. Donc l'inconsistance provient des clauses Aj Cj telles que Aj∈E. Comme les littéraux de A sont satisfaits, ce sont uniquement les clauses Cj qui sont inconsistantes avec *KB*. Ces Cj forment bien une sous base minimale inconsistante de *CR* avec *KB*. Par construction, il s'agit de Décode(*CR*'',E).
On peut montrer de plus que puisque E est minimal, *Décode(CR'',E)* est une sous-base minimale incohérente.

Reprenons notre exemple : l'ensemble des nogoods de $KB \cup CR'' = KB \cup Encode(CR,A)$ est $\{\{A_1,A_4,A_5,A_7,A_8\}\}$. On retrouve donc la sous-base minimale incohérente Décode(*CR*'', $\{A_1,A_4,A_5,A_7,A_8\}$)= *{psg pb, festnoz yb, festnoz freg, freg yb big, pb yb penn }*.

Proposition 37
Soit $CR \subset FP_S$ une base de clauses. Soit *BC=<KB,CR>* une partition des clauses de *BC* en connaissances et en croyances.Soit $N = |CR|$. Soit $A = \{A_1, A_2, \ldots, A_N\}$ un ensemble de nouveaux symboles propositionnels n'apparaissant pas dans *CR*. $CR' \subseteq CR$ est une sous base maximale cohérente de *BC* ssi $\exists\ E^{\wedge}$ un impliquant A-

restreint premier de $KB \cup CR''$ avec CR''=Encode(CR,A) tel que Décode (CR'',A\E) = CR' ssi ∃ une interprétation I de $KB \cup CR''$ avec CR''=Encode(CR,A) tel que Décode (CR'',I).

Preuve :
Identique à la précédente en remplaçant la notion de nogood par celle d'interprétation de l'ATMS.

Dans notre exemple, l'ensemble des interprétation de $OBS \cup Encode(BZH,A)$ est *{{A8, A7, A6, A5, A4, A3, A2}, {A7, A6, A5, A4, A3, A2, A1}, {A8, A6, A5, A4, A3, A2, A1}, {A8, A7, A6, A5, A3, A2, A1}, {A8, A7, A6, A4, A3, A2, A1}}*
Ce qui donne les sous-bases maximales cohérentes suivantes :

{pb yb penn , freg yb big , rhu freg , festnoz freg , festnoz yb , chez yb , chez pb },
{ freg yb big , rhu freg , festnoz freg , festnoz yb , chez yb , chez pb , psg pb },
{ pb yb penn , rhu freg , festnoz freg , festnoz yb , chez yb , chez pb , psg pb },
{ pb yb penn , freg yb big, rhu freg , festnoz freg , chez yb , chez pb , psg pb },
{ pb yb penn , freg yb big , rhu freg , festnoz yb , chez yb , chez pb , psg pb }

On retrouve enfin la notion de support d'une formule (sous-base minimale satisfaisant une formule) dans le cadre de l'argumentation à partir du label d'une formule dans la base encodée.

Définition 67 (argument , support, conclusion [Elvang-Goransson et al. 1993])
Soit $BC \subset FP_S$ une base de clauses. Soit BC=<KB,CR> une partition des clauses de BC en connaissances et en croyances. Soit $\alpha \in FP_S$ une formule. Un argument de α dans le contexte KB est un couple (CR',α) tel que CR' ⊆ CR satisfaisant :

i) $KB \cup CR' \not\models \bot$ ($KB \cup CR'$ consistant)
ii) $KB \cup CR' \models \alpha$
iii) et ∄ CR'' ⊂ CR' telle que CR'' satisfait i) et ii).

CR' est le support et α la conclusion de l'argument (CR', α).

Exemple : (*{psg pb, pb yb penn, festnoz yb}, penn*) est un argument pour *penn* dans le contexte *KB*.

Proposition 38
Soit $BC \subset FP_S$ une base de clauses. Soit BC=<KB,CR> une partition des clauses de BC en connaissances et en croyances. Soit $N = |CR|$. Soit $A = \{A_1, A_2, \ldots, A_N\}$ un ensemble de nouveaux symboles propositionnels n'apparaissant pas dans CR. Soit α une formule. $CR' \subseteq CR$ est un support minimal de α dans le contexte BC ssi ∃ E un environnement du label de α dans $BC \cup CR''$ avec CR''=Encode(CR,A) tel que Décode (CR'',E) = CR' ssi $E^\wedge \in IRP(IRP(KB^\wedge \wedge CR''^\wedge \wedge \neg\alpha), A^\neg)^{\neg\wedge} \wedge NG^{\neg\wedge}, A)$ où NG est l'ensemble des nogoods de CR''.

Preuve : Par définition du label, on a $KB^\wedge \wedge CR''^\wedge \wedge E^\wedge \models \alpha$. Donc $KB^\wedge \wedge CR''^\wedge \wedge E^\wedge \wedge (A\backslash E)^{\neg\wedge} \models \alpha$. Donc $KB^\wedge \wedge CR''^\wedge \wedge E^\wedge \wedge (A\backslash E)^{\neg\wedge} \wedge \neg\alpha \models \bot$. Or toutes les clauses Ai Ci ∈ CR'' telles que Ai ∈ (A\E) sont satisfaites. Donc l'inconsistance provient des clauses restantes Aj Cj telles que Aj∈E. Comme $E\wedge$ est consistant avec CR'' (définition du label), alors l'inconsistance provient des clauses {Cj | Aj Cj ∈ CR'' et Aj∈ E}=Décode(CR'',E) de KB et ¬α. Donc $KB^\wedge \wedge$Décode(CR'',E)$^\wedge \wedge \neg\alpha \models \bot$. Donc $KB^\wedge \wedge$Décode(CR'',E)$^\wedge \models \alpha$. Décode(CR'',E) est donc un support de α. On montre de plus que puisque E est minimal alors Décode(CR'',E) est un support minimal. La relation avec les impliquants P-restreints premiers découle de la Proposition 30 page 94.

Nous allons maintenant étudier deux relations de préférence sur des sous-bases dans le cadre de l'argumentation. Nous en déduirons deux relations de préférence entre littéraux que nous utiliserons pour calculer les supports préférés d'une formule.

2.3 MPL et les niveaux

La première variante que nous présentons est issue de l'utilisation de notre travail dans le cadre de l'argumentation. En effet, nous présentions dans [Amgoud, et al. 1996] diverses relations de préférences basées sur la notion de niveau. Dans ce cadre, les bases de croyances sont stratifiées. Chaque strate représente des croyances. Les strates sont ordonnées des croyances les plus sûres aux moins sûres. On notera une base de croyance stratifiée CR = <$CR1$; $CR2$; … ; CRn> où chaque CRi dénote une strate de la base. Pour des commodités d'écriture, on pose que si CRi est plus sûre que CRj alors i<j (la strate contenant les croyances les plus

certaines est *CR1*). Comme nous allons utiliser les relations de préférence entre ensembles de clauses et entre ensembles de littéraux, nous les définirons pour des ensembles de formules.

On définit alors la notion de niveau d'une sous-base CR' de CR.

Définition 68 (niveau d'un ensemble de formules)
Soit $F \subseteq FP_S$ un ensemble de formules. Soit $F = <F1 ; F2 ; \ldots ; Fn>$ une stratification de cet ensemble. Soit $E \subseteq F$. $Niveau(E) = min\{i \mid 0 \leq i \leq n \text{ et } (Fi+1 \cup \ldots \cup Fn) \cap E = \varnothing\}$ avec $min\ \varnothing = n$. Le niveau d'un ensemble de formules E est la strate de la formule la moins certaine de E.

Exemple : $F = <\{a, b, c\} ; \{d, e\} ; \{f, g, h\}>$. Niveau({b, d})=2 et Niveau({a, h})=3.

2.3.1 Préférence « Worst In »

Cette première relation de préférence était présentée sous le nom « préférence BDP » dans [Amgoud, et al. 1996]. Son principe est de comparer chaque ensemble de formules uniquement sur ses connaissances les moins sûres.

Définition 69 (préférence « Worst In » (WI))
Soit $F \subset FP_S$ un ensemble de formules. Soit $F = <F1 ; F2 ; \ldots ; Fn>$ une stratification de F. Soient $F' \subseteq F$ et $F'' \subseteq F$. F' est WI-préféré à F'' ssi niveau(F')<niveau(F'').

Exemple : *CR = <{ a, a b c} ; {a d c, d, b } ; {d c}>.*
CR' = { a, a b c, d} est WI-préféré à *CR'' = { d, d c}.*
CR' et *{ a, a d c, d}* ne sont pas comparables pour WI.

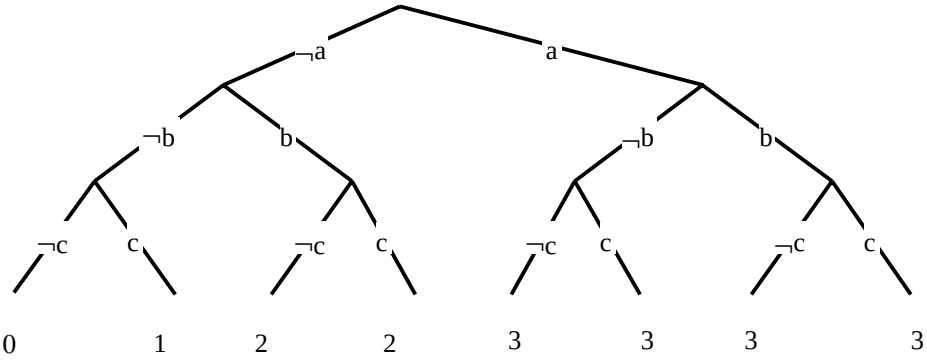
Nous allons modifier la méthode de codage présentée précédemment afin d'obtenir une stratification des variables ajoutées dans la base correspondant exactement à la stratification de la base.

Définition 70 (encodage d'une base de clauses stratifiée)
Soit $CR \subset FP_S$ une base de clauses. Soit $CR = <CR1 ; CR2 ; \ldots ; CRm>$ une stratification de CR. Soit $Nk = |CR_k|$. Soit $A = \{A_1, A_2, \ldots, A_N\}$ un ensemble de nouveaux symboles propositionnels n'apparaissant pas dans *CR*. Soit $A = <A_1 ; A_2 ; \ldots A_m>$ une stratification de A telle que $|Ak| = |CRk|$ $1 \leq k \leq m$. Encode(*CR*,A) = { Ai Ci | Ci∈CRk, Ai∈Ak $1 \leq i \leq Nk$}.

Exemple : prenons comme stratification de *CR = <{ psg pb, chez pb, chez yb, festnoz yb, festnoz freg, rhu freg } ; { freg yb big } ; { pb yb penn }>* qui signifie que l'on préfère les témoignages ne concernant pas directement la culpabilité des penn sardinns ou des bigoudens, puis le témoignage de Jean car il s'appuie sur le témoignage d'un bigouden alors qu'il est penn sardinn, puis enfin le témoignage de Alan.
L'encodage de la base nous donne le même résultat que précédemment avec la stratification de A = <{A1, A2, A3, A4, A5, A6} ;{A7} ;{A8}>.

Nous utilisions une méthode de tri externe à l'ATMS afin de choisir les supports WI-préférés. Nous allons utiliser une propriété de MPL afin de limiter le calcul des impliquants P-restreints premiers aux seuls WI-préférés.

Reprenons par exemple notre arbre binaire de profondeur 3. Supposons maintenant que chaque symbole appartienne à une strate différente : *a* à la strate 3, *b* à la strate 2 et *c* à la strate 1. Nous nous intéressons au niveau des impliquants P-restreints. Les impliquants P-restreints sont produits dans un ordre compatible avec leur niveau, c'est à dire compatible avec la préférence WI. Le niveau 0 correspond au niveau de ∅.



Nous allons maintenant démontrer que pour ordonner les impliquants P-restreints premiers selon la préférence WI, il suffit d'ordonner le choix des littéraux de P par niveau décroissant.

Proposition 39
Soit *CR* une base de clauses de FP_S. Soit P un ensemble consistant de littéraux. Soit $P = <P1 ; P2 ; \ldots ; Pn>$ une stratification de P. MPL(*CR*,∅,P) plus une contrainte de choix des littéraux de P par ordre de niveau décroissant permet d'énumérer les impliquants P-restreints premiers selon un ordre compatible avec la préférence WI.

> **Preuve :**
> Supposons que ce soit faux. Alors ∃ E1 et E2 deux impliquants P-restreints premiers de CR tels que E1 est trouvé avant E2 et E2 WI-préféré à E1. Donc ∃ M1 un modèle de CR tel que M1∩P=E1 et M2 un modèle de CR tel que M2∩P=E2 et niveau(E2)<niveau(E1). Donc ∃l∈E1 tel que ¬l∈M2 et ∀l'∈E2, niveau({l'})<niveau({l}). Comme les littéraux de P sont ordonnés par niveau décroissant, on obtient que E2 est trouvé avant E1. Contradiction.

Il suffit donc de rajouter dans notre algorithme un marqueur indiquant le niveau des impliquants P-restreints premiers trouvés pour ne produire que les WI-préférés.

On peut noter que la propagation des clauses unitaires ne change en rien le niveau des impliquants P-restreints produits car le numéro de la strate associée aux littéraux propagés ne peut être qu'inférieur ou égal un numéro de la strate du prochain littéral de branchement.

Nous proposons donc un algorithme de calcul d'impliquants P-restreints WI-préférés basé sur MPL. La seule différence avec MPL est l'ajout d'un test permettant d'arrêter la recherche dès que le niveau de l'interprétation partielle courante est supérieur au niveau du meilleur impliquant P-restreint trouvé. Le nombre de nœuds développés par cet algorithme est au plus le nombre de nœuds développés par MPL avec les mêmes paramètres.

```
WI(CR,BA, A, niveau, IP, meilleur)
// Calcul des impliquants A-restreints WI-préférés
// CR est une base de clauses contenant chacune une variable unique de A.
// (CR est une base obtenue par la fonction Encode())
// BA est l'ensemble des clauses de minimisation
// A est l'ensemble de variables propositionnelles.
// niveau est une fonction qui retourne le niveau d'un ensemble de littéraux.
// IP est l'interprétation partielle courante.
// retourne les impliquant A-restreints WI-préférés de CR et leur niveau.
       Si    ∈ CR alors retourner <BA,|A| > Finsi; // CR est inconsistante
       // on test le niveau de l'interprétation partielle restreinte courante
       Si niveau(IP∩A)> meilleur alors retourner <BA,meilleur> Finsi ;
       // test de primarité
       Si BA^∧IP^ inconsistant alors retourner <BA,|A|> Finsi ;
       Si CR = ∅ alors  // CR est consistante
              // on a trouvé un impliquant A-restreint WI-préféré
              retourner <BA ∪ {¬((IP∩A)^)},niveau(IP∩A)> ;
       Finsi
       l ← LitteralPourSimplifier(CR,BA,A) ;
       Si (l≠null) alors // on peut simplifier CR par l
              Retourner WI(CR[l],BA,A,niveau,IP∪{l},meilleur)
       Finsi ;
       // on doit retourner les littéraux de P par niveau décroissant
       l ← Choix_Symbole_de_P(CR,P,niveau) ;
       Si (l≠null) alors
```

```
        <SOL,best> ← WI(CR[¬l],BA,A,niveau,IP∪{¬l},meilleur) ;
        // Attention : la valeur de meilleur a peut être changé
        retourner WI(CR[l],SOL,A,niveau,IP∪{l},best}
Sinon
        Si DP(CR)=vrai alors // CR est consistante
                // on a trouvé un impliquant A-restreint WI-préféré
                retourner <BA ∪ {¬((IP∩P)^)},niveau(IP∩A)>
        Sinon
                retourner <BA,|A|>
        Finsi
Finsi
```

Algorithme 13 WI - calcul d'impliquants P-restreints WI-préférés

Cette fonction est appelée par WI(CR,∅,A,niveau,∅,|A|).

Nous présentions de plus un raffinement de WI.

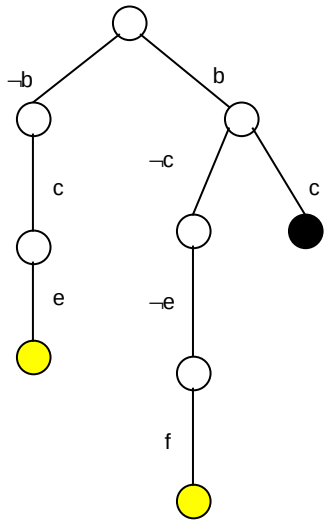
2.3.2 Préférence WI-faible

La préférence WI ne permet pas de comparer deux ensembles de formules atteignant la même strate. Par exemple, $P = <\{a, b, c\} ; \{d, e\} ; \{f, g\} >$. $\{a, c, d, f\}$ et $\{c,g\}$ ne sont pas comparables avec la relation WI. Pourtant, on préfère intuitivement le deuxième ensemble de formules car il ne contient par de formules de la deuxième strate. C'est l'idée proposée par [Amgoud 1999] (appelée BDP-faible).

Définition 71 (préférence WI-faible)
Soit $F \subset FP_S$ un ensemble de formules. Soit $F = <F1 ; F2 ; \ldots ; Fn>$ une stratification de F. Soient $F' \subseteq F$ et $F'' \subseteq F$. F' est WI-faible-préféré à F'' ssi $\exists$ k≤n tel que niveau(F',k)<niveau(F'',k) et $\forall$j, k<j≤n niveau(F', j)=niveau(F'', j).

La seule contrainte sur l'ordre de choix des littéraux de branchement ne permet pas de garantir l'ordonnancement des impliquants P-restreints selon la relation de préférence WI-faible.

Contre exemple :
$CR = \{ \quad c\ b, \quad e\ b, \quad c\ f\}$. $P = <\{f\} ;\{e\} ;\{b,c\}>$. Les impliquants P-restreints premiers de CR sont f∧b, c∧e et b∧c. {b,f} est WI-faible préféré à {c,e}. Exemple d'arbre de recherche développé par MPL sur CR en choisissant pour littéral de branchement celui de strate maximale :



Le premier littéral de branchement choisi est b. On trouve dans le sous arbre gauche l'impliquant P-restreint c∧e et dans le sous arbre droit l'impliquant P-restreint b∧f. Or ce dernier est WI-faible préféré au premier. On peut noter cependant que si chaque littéral appartient à une strate différente, alors ce contre exemple n'est plus possible.

Un autre raffinement de la préférence WI était présenté dans [Amgoud, et al. 1996] : il s'agit de la préférence élitiste.

2.3.3 Préférence élitiste

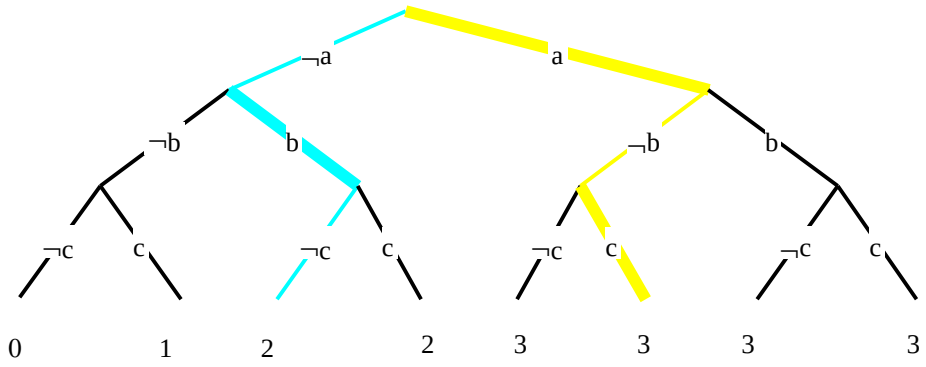
Cette relation a été proposée par [Cayrol, et al. 1993] et redéfinie en termes de niveaux par [Amgoud 1999]. L'idée est que chaque formule retenue doit être meilleure qu'une formule rejetée.

Définition 72 (préférence élitiste (ELI))
Soit $F \subset FP_S$ un ensemble de formules. Soit $P = <P1 ; P2 ; \ldots ; Pn>$ une stratification de P. Soient $E \subseteq P$ et $F \subseteq P$. E est ELI-préféré à F ssi niveau(E\F)<niveau(F\E).

Exemple : CR = <{ a, a b c} ; {a d c, d, b} ; {d c}>.
CR' = { a, a b c, d} est ELI-préféré à CR'' = { a, a d c, d} car niveau(CR'\CR) = niveau ({a b c}) = 1 et niveau(CR\CR') = niveau ({a d c}) = 2.

Nous allons montrer que l'ordre sur le choix des littéraux permet d'énumérer les impliquants P-resteints selon cette nouvelle relation de préférence.

Reprenons notre arbre de recherche et prenons deux impliquants P-restreints : $E=\{b\}$ et $F=\{a,c\}$. Pour la préférence ELI, nous devons étudier le niveau de $E\backslash F$ et de $F\backslash E$. Ces ensembles sont en gras sur chacune des branches de l'arbre. En utilisant un ordre sur les littéraux de branchement, nous garantissons que le niveau d'une solution du sous-arbre gauche de l'arbre ne peut pas être plus important que celui du sous-arbre droit. En effet, si tel est le cas, c'est qu'il existe un littéral de strate supérieure au littéral de branchement, donc que les littéraux ne sont pas utilisés par ordre de strate décroissante. La propagation des clauses unitaires ne change rien à cette propriété car le littéral propagé ne peut être que de niveau inférieur ou égal au niveau du littéral de branchement courant.



Nous voyons dans notre exemple que le niveau de E\F est celui du sous-arbre gauche de la racine et celui de F\E est le niveau du littéral de branchement. D'après la remarque précédente, nous garantissons donc que niveau(E\F) ≤ niveau(F\E) (Dans notre exemple, niveau ({b})=2 et niveau({a,c})=3).

Proposition 40
Soit *CR* une formule de FP_S. Soit *P* un ensemble consistant de littéraux. Soit $P = <P1 ; P2 ; \ldots ; Pn>$ une stratification de *P*. *MPL(CR,∅,P)* plus une contrainte de choix des littéraux de *P* par ordre décroissant de strate permet d'énumérer les impliquants P-restreints premiers selon un ordre compatible avec la préférence élitiste

On en déduit un algorithme de calcul d'impliquants P-restreints ELI-préférés basé sur MPL. La seule différence avec MPL est l'ajout d'un test permettant d'arrêter la recherche dès que le niveau du sous-arbre gauche est inférieur au niveau du littéral de branchement.

```
ELI(CR,BA, A, niveau, IP, meilleur)
// Calcul des impliquants A-restreints WI-préférés
// CR est une base de clauses contenant chacune une variable unique de A.
// (CR est une base obtenue par la fonction Encode())
// BA est l'ensemble des clauses de minimisation
// A est l'ensemble de variables propositionnelles.
// niveau est une fonction qui retourne le niveau d'un ensemble de littéraux.
// IP est l'interprétation partielle courante.
// retourne les impliquant A-restreints ELI-préférés de CR et leur niveau.
       Si    ∈ CR alors retourner <BA,|A|> Finsi; // CR est inconsistante
       // test de primarité
       Si BA^∧IP^ inconsistant alors retourner <BA,meilleur> Finsi ;
       Si CR = ∅ alors  // CR est consistante
```

```
        // on a trouvé un impliquant A-restreint WI-préféré
        // le niveau du sous-arbre est 0
        retourner <BA ∪ {¬((IP∩A)^)},0> ;
Finsi
l ← LitteralPourSimplifier(CR,BA,A) ;
Si (l≠null) alors // on peut simplifier CR par l
        Retourner ELI(CR[l],BA,A,niveau,IP∪{l},meilleur)
Finsi ;
// on doit retourner les littéraux de P par niveau décroissant
l ← Choix_Symbole_de_P(CR,P,niveau) ;
Si (l≠null) alors
        <SOL,best> ← ELI(CR[¬l],BA,A,niveau,IP∪{¬l},meilleur) ;
        // On teste le niveau du sous-arbre gauche.
        // Si celui-ci est inférieur au niveau du littéral courant,
        // inutile d'effectuer le branchement.
        Si best>=niveau({l}) alors
                retourner ELI(CR[l],SOL,A,niveau,IP∪{l},best}
        Sinon
                Retourner <SOL,best>
        Finsi
Sinon
        Si DP(CR)=vrai alors // CR est consistante
                // on a trouvé un impliquant A-restreint WI-préféré
                // le niveau du sous-arbre est 0
                retourner <BA ∪ {¬((IP∩P)^)},0>
        Sinon
                retourner <BA,|A|>
        Finsi
Finsi
```

Algorithme 14 ELI - calcul d'impliquants P-restreints ELI-préférés

L'algorithme est appelé par ELI(CR,∅,A ,niveau,∅,|A|).

En utilisant ces deux algorithmes à la place de MPL (ou Castell) pour l'explicitation des solutions, seuls les P-impliqués WI-préférés ou ELI-préférés seront produits (en vertu de la Proposition 35 page 125). En utilisant l'encodage des bases et ces algorithmes dans notre CMS fondé sur MPL, nous pouvons donc calculer pour une base de clauses *BC = <KB ;CR>* et la formule α les supports WI-préférés et ELI-préférés de α. Mais ce n'est pas suffisant pour décider si α ou $\neg\alpha$ doit être inféré. Il faut aussi pouvoir comparer les supports de α et $\neg\alpha$.

Dans le cadre de la préférence WI, seul le niveau d'un impliquant P-restreint WI-préféré de α est nécessaire pour connaître la classe d'équivalence des impliquants P-restreints WI-préférés. Nous pouvons donc utiliser une variante de notre algorithme WI s'arrêtant dès qu'une solution est trouvée pour obtenir le niveau d'un support WI-préféré de α. On peut l'utiliser pour initialiser `meilleur` dans la recherche du niveau d'un support WI-préféré de $\neg\alpha$. Si une solution est trouvée, elle correspond soit à un support de $\neg\alpha$ WI-préféré au support WI-préféré de α (si le niveau du premier est strictement inférieur au niveau du second), soit à un support de même niveau. Si aucune solution n'est trouvée, c'est qu'il n'existe pas de support de $\neg\alpha$ WI-préféré ou équivalent au support WI-préféré de α. Donc le support WI-préféré de α est WI-préféré à tous les supports de $\neg\alpha$.

Ce raisonnement n'est malheureusement pas applicable avec la relation ELI, car le seul niveau d'un impliquant P-restreint ELI-préféré ne suffit pas à caractériser une classe d'équivalence pour cette relation.

Dans certains cas, le codage permettra de s'en sortir de manière plus satisfaisante.

Si nous reprenons une ultime fois notre enquête bretonne, à partir de notre problème sous sa forme stratifiée, on peut calculer les supports de *penn* et de *big*. On obtient respectivement *{{psg pb, pb yb penn, festnoz yb}}* et *{{festnoz yb, freg yb big, festnoz freg}}*. Si maintenant on cherche les supports de *penn big*, on obtient les deux supports précédents *{{psg pb, pb yb penn, festnoz yb},{festnoz yb, freg yb big, festnoz freg}}*. Si l'on calcule les supports WI-préférés de *penn big* on obtient comme *résultat {{freg yb big, festnoz freg, festnoz yb }}*, c'est à dire que l'on préfère le témoignage dénonçant les bigoudens à celui dénonçant les penn sardinns.

En posant la préférence « les Bigoudens Détestent les Penn-sardins, et vice versa », on peut accorder plus de crédit au témoignage de Jean qu'à celui d'Alan. Ainsi l'inspecteur De Kernec peut supporter devant le préfet la

thèse d'une action bigoudène. Ce dernier, convaincu, lui laisse carte blanche. L'inspecteur lance ses troupes au QG des bigoudens. Le député maire est retrouvé et libéré en quelques minutes …

2.4 Minimalité pour la cardinalité

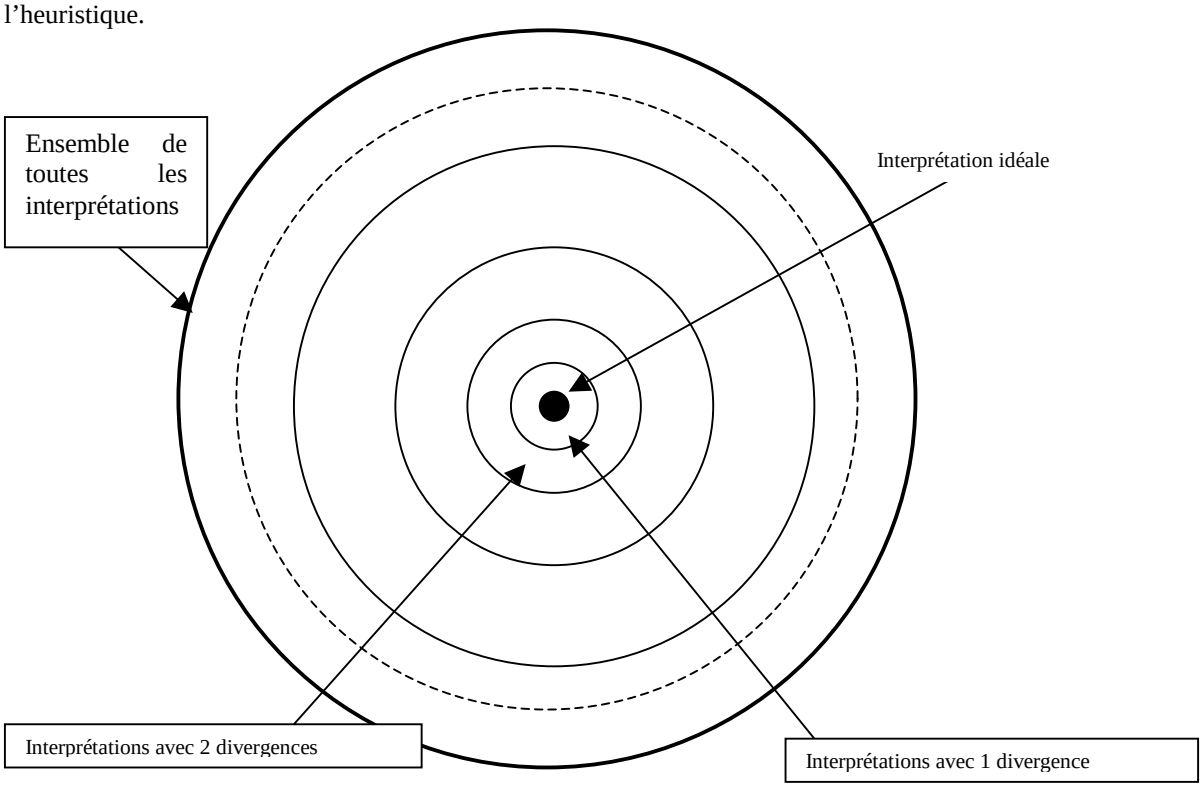
Une relation de préférence classique dans diverses applications logiques est l'utilisation de formules de cardinalité minimale (en nombre de littéraux, en nombre de clauses, etc.). Dans le cadre du calcul d'impliquants P-restreint, on note par exemple le domaine du diagnostic consistant, où l'on cherche les pannes concernant un nombre minimal de composants. Dans celui du calcul de P-impliqués, on retrouve la même notion d'explication de panne mettant en cause un nombre minimal de composants en diagnostic abductif, de plans en nombre minimal d'actions en planification, etc.
Pour cela, nous devons énumérer les impliquants P-restreints dans un ordre compatible avec leur cardinalité (nombre de littéraux). L'utilisation d'un algorithme du type Davis et Putnam ne permet pas cette énumération. Nous allons donc présenter une adaptation d'un algorithme de recherche heuristique pour le faire.

2.4.1 LDS [Harvey/Ginsberg 1995]

Cet algorithme est basé sur l'observation suivante : dans la plupart des cas, lorsque l'on dispose d'une heuristique pour guider une recherche dans un domaine particulier, les solutions trouvées sont «proches » du meilleur candidat proposé par l'heuristique. D'où l'idée d'effectuer une recherche en partant de ce candidat donné par l'heuristique, puis en le dégradant petit à petit.

Une divergence (*discrepancy*) correspond par exemple dans le cadre de SAT à « flipper » une variable à l'interprétation idéale. Supposons que nous ayons *S* variables propositionnelles dans notre problème. Notre heuristique va nous donner 1 interprétation idéale. Il y aura alors *S* interprétations à une divergence, $C_2^S = \frac{S!}{2!(S-2)!}$ interprétations à deux désaccords, et plus généralement, $C_i^S = \frac{S!}{i!(S-i)!}$ interprétations avec i divergences[44].

Schématiquement, cela revient à augmenter circulairement la zone de recherche autour du candidat proposé par l'heuristique.



[44] Le nombre de divergences entre deux interprétations correspond exactement dans le cadre de SAT à la distance de Hamming (p. 40) entre ces deux interprétations.

LDS utilise ce principe, avec toutefois une petite variante : à chaque étape, on ne recherche pas les solutions ayant *exactement* i divergences, mais *au maximum i* divergences, c'est à dire que l'on va repasser par des états déjà explorés.

Voici l'algorithme proposé par [Harvey/Ginsberg 1995] : il s'agit de parcourir un arbre de recherche binaire jusqu'à un nœud but, sachant que pour chaque nœud, le fils gauche correspond au choix de l'heuristique et le fils droit à une divergence de l'heuristique.

```
LDS_Probe(noeud,k)
// retourne un nœud but dont le chemin s'éloigne d'au plus d
// divergences de l'heuristique
        Si le nœud est un nœud but alors retourner nœud Finsi ;
        Si le nœud est une feuille alors retourner ∅ Finsi ;
        // si aucune divergence n'est autorisée, on suit l'heuristique
        Si (k=0) alors retourner LDS_Probe(Gauche(nœud),0) Finsi ;
        Sinon   // on effectue une divergence                              (1)
                Res ← LDS-Prob(Droit(noeud),k-1) ;
                // si un nœud but est trouvé on arrête la recherche
                Si (res ≠ ∅) alors retourner res Finsi ;
                Retourner LDS_Probe(Gauche(nœud),k)
        Finsi

LDS(nœud)
        Pour x ← 0 à profondeur maximale faire
                Res ← LDS_Probe(node,x) ;
                Si res ≠ ∅ alors retourner res
        Fin pour ;
        Retourner ∅
```

Algorithme 15 LDS - Limited Discrepancy Search

Remarque : effectuer une divergence si elle est permise avant de tester le cas où il n'y a pas de divergence produite (1) peut sembler contre-intuitif puisque l'on cherche une solution ayant un minimum de divergences. Il faut se rappeler que cette fonction est appelée pour différentes valeurs de *d*. Si par exemple LDS_Probe est appelée une première fois avec *d = 0*, une seule feuille est explorée (tout à gauche). Si maintenant on l'appelle avec *d=1*, en favorisant le minimum de divergences, la première feuille explorée est … la feuille déjà explorée à l'appel précédent. En favorisant les divergences quand elles sont permises, le problème n'est pas réglé (la feuille est toujours re-explorée), mais repoussé : on retrouve notre feuille après avoir atteint des feuilles non encore explorées.

2.4.2 ILDS [Korf 1996]

C'est une version améliorée de LDS, qui à chaque appel fournit des interprétations ayant *exactement k* divergences. Ainsi, chaque feuille de l'arbre de recherche ne sera produite qu'une seule et unique fois (ce n'est pas vrai pour les autres nœuds de l'arbre). La seule chose à ajouter est la profondeur de l'arbre restant à parcourir. Ainsi, quand la profondeur restante ne permet pas de produire *k* divergences (`prof-1< k`, soit `prof ≤ k`), on n'explore plus le sous arbre gauche.

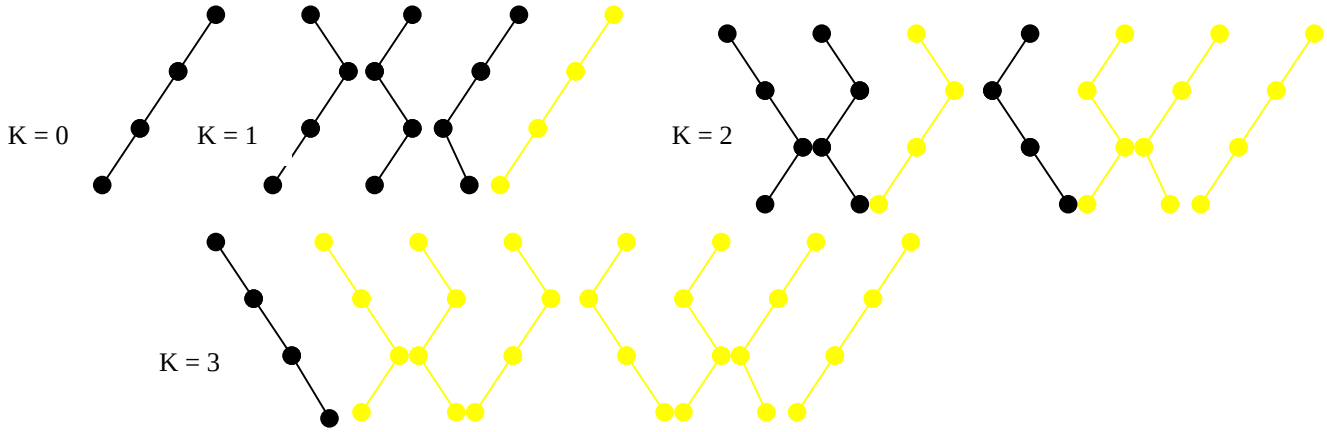
```
ILDS_Probe(noeud,k,prof)
// retourne un nœud but dont le chemin s'éloigne d'exactement d
// divergences de l'heuristique
        Si le nœud est un noeud but alors retourner nœud Finsi ;
        Si le nœud est une feuille alors retourner ∅ Finsi ;
        Si (prof>k) alors
                // il est possible de faire k divergences dans le sous-arbre
                res ← ILDS_Probe(Gauche(nœud),k,prof-1) ;
                Si res ≠ ∅ alors
                        Retourner res
                Finsi
        Finsi ;
        // si au moins une divergence est autorisée, on effectue une divergence
        Si (k>0) alors
                Retourner LDS_Probe(Droit(nœud),k-1,prof-1)
        Finsi
Finsi

ILDS(nœud)
        Pour x ← 0 à profondeur maximale faire
                Res ← ILDS_Probe(node,x,profondeur maximale) ;
```

```
        Si res ≠ ∅ alors retourner res
Fin pour ;
Retourner ∅
```

Algorithme 16 ILDS - Improved LDS

Remarque : LDS et ILDS n'explorent pas les feuilles de l'arbre dans le même ordre, mais en ordre inverse.

Comparons les 2 algorithmes sur un exemple. Le schéma suivant représente l'ordre dans lequel les feuilles d'un arbre binaire complet de profondeur 3 sont atteintes par LDS pour un nombre de divergences K variant de 0 à 3. Les chemins en foncé représentent les feuilles atteintes par ILDS (l'ordre est inversé). Les chemins en clair montrent donc les feuilles déjà atteintes par LDS pour une valeur de K inférieure.



A première vue, le résultat est sans appel : ILDS (8 feuilles explorées) est beaucoup plus performant que LDS (20 feuilles explorées). Il faut cependant noter que dans le cadre de SAT par exemple, les arbres parcourus sont rarement complets, donc en pratique les différences sont moins grandes.

2.4.3 Adaptation au calcul de modèles P-restreints CARD-préférés

Si nous reprenons l'idée de LDS/ILDS avec comme heuristique, comme dans le calcul des impliquants P-restreints, «je préfère un littéral qui n'est pas dans P », nous obtenons comme interprétation (ou plutôt comme P-interprétation) idéale celle composée uniquement de littéraux de $P^{\neg}$. Une divergence va correspondre à l'apparition d'un littéral de P, donc à augmenter la taille de l'impliquant P-restreint. Ainsi, nous allons tester l'existence de l'impliquant P-restreint de taille 0, puis de taille 1, de taille 2, etc. … La recherche s'arrête dès que l'ensemble des impliquants P-restreints d'une taille donnée est non vide.

Nous continuons à utiliser BC qui représente la base de clauses en calcul propositionnel (une CNF), et P un ensemble consistant de littéraux. Quand il ne reste plus de littéraux de P, on fait appel à un algorithme de type DP pour vérifier la consistance de la base. Contrairement aux deux algorithmes proposés précédemment, notre heuristique ($P^{\neg}$) concerne uniquement une partie des variables. Même en forçant le choix d'une variable de P pour le branchement, une clause unitaire ou un littéral pur peuvent fixer la valeur d'une variable n'appartenant pas à P. Il faut donc adapter l'algorithme dans ce cas. De plus, la suppression de l'arrêt de la fonction ILDS_Probe lorsqu'une solution est trouvée nous permet d'obtenir *tous* les impliquants P-restreints CARD-préférés et non *un seul*. Le reste n'est que traduction du parcours de l'arbre dans le cas de SAT.

```
ILDS_Probe(BC,P,ip,d,k)
Si    ∈ BC alors retourner ∅ Finsi ;
Si BC = ∅ alors retourner ip∩P Finsi ;
// on cherche une clause unitaire ou un
// littéral pur qui n'apparaît pas dans P
l ← LittéralPourSimplifier(BC,P) ;
Si l ≠ null alors // l existe
       // il appartient à P : on a une divergence
       Si l∈ P alors retourner LDS_Probe(BC[l],P,ip∪l,d-1,k-1) Finsi ;
       // sa négation appartient à P : il faut mettre à jour la profondeur
       Si ¬l∈ P alors retourner LDS_Probe(BC[l],P,ip∪l,d-1,k) Finsi ;
       // sinon on propage normalement
```

```
        retourner LDS_Probe(BC[l],P,ip∪l,d,k)
Finsi ;
l ← Choisir_Littéral_de(BC,P) ;
Si (l≠ null) alors // il reste des variables de P à assigner
        SOL ← ∅ ;
        // on commence par un élément de P¬ : pas de dégradation
        // on vérifie qu'il reste assez de littéraux de P à assigner
        // pour effectuer k dégradations
        Si (d>k) alors SOL ← LDS_Probe(BC[¬l],P,ip∪{¬l},d-1,k) Finsi ;
        // on continue par un élément de P : une dégradation
        // on vérifie d'abord qu'une dégradation est autorisée
        Si (k>0) alors SOL ← SOL ∪ LDS-Probe(BC[l],P,ip∪{l},d-1,k-1) Finsi ;
        Retourner SOL ;
Sinon   // il n'y a plus de symboles de P à assigner
        // on effectue un simple test de consistance
        Si (DP(base)=vrai) alors
                Retourner ip∩P
        Sinon
                Retourner ∅
        Finsi
Finsi

ILDSMPL(BC,P)
        // on fait varier la taille du nombre exact de divergences
        Pour x ← 0 jusqu'à |P|
                SOL ← ILDS-Probe(BC,P,∅,|P|, x) ;
                Si SOL ≠ ∅ alors retourner SOL Finsi ;
        retourner ∅
```

Algorithme 17 ILDSMPL - impliquants P-restreints minimaux pour la cardinalité

2.4.4 Lien avec Distance-SAT

[Bailleux/Marquis 1999] présentent un nouveau problème de décision proche de SAT. Il s'agit de déterminer s'il existe un modèle d'une formule qui diffère d'une interprétation partielle donnée sur au plus *d* variables. Il s'agit exactement du problème de décision correspondant au problème d'optimisation que nous nous sommes proposés de résoudre.

Définition 73 (différence [Bailleux/Marquis 1999])
Une interprétation partielle PI_1 diffère d'une interprétation partielle PI_2 sur au plus *d* variables ssi le nombre de variables *x* de $Dom(PI_1) \cap Dom(PI_2)$ telles que $PI_1(x) \neq PI_2(x)$ est inférieur ou égal à *d*.

> Il s'agit de la distance de Hamming entre deux interprétations partielles. Une différence correspond bien à une divergence dans LDS/ILDS. En utilisant notre notation habituelle, nous pourrions écrire ssi $|PI_1 \cap PI_2^{\neg}| \leq d$.

Définition 74 (Distance-SAT [Bailleux/Marquis 1999])
Distance-SAT est le problème de décision suivant :

- Données : Une formule CNF *BC*, une interprétation partielle *PI* et un entier naturel *d*.
- Questions : Existe-t-il un modèle *I* de *BC* qui diffère de *PI* sur au plus *d* variables.

Nous noterons ce problème de décision Distance-SAT(BC,PI,d).

Les auteurs proposent pour résoudre ce problème une variante de la procédure de Davis et Putnam appelée $DP_{distance}$. Il s'agit d'une implantation proche de la procédure LDS_Probe décrite ci-dessus. La différence entre cette implantation et la nôtre se situe sur le choix de la variable de branchement. Nous forçons un branchement sur les variables de PI au début de l'arbre de recherche alors qu'aucune contrainte n'existe sur les variables de branchement dans $DP_{distance}$, c'est seulement lorsque la variable est choisie que l'on vérifie si elle appartient ou non à PI. Nous avons fait ce choix en conformité avec l'approche utilisée dans MPL. De plus, nous commençons toujours un branchement par un littéral ne changeant pas le nombre de divergences. Aucune restriction de ce genre n'existe dans $DP_{distance}$, au contraire :
les auteurs arguent que dans le cadre de la résolution de ce problème de décision *« une approche énumérative naïve qui consisterait à engendrer successivement les interprétations I qui ne diffèrent pas avec PI sur au plus d*

variables, et à tester à chaque étape si I constitue ou pas un modèle de BC n'est en général pas réalisable en pratique, même si l'on ne considère que de « petites » valeurs pour n, […], et pour d. »

Nous allons le vérifier sur des bases générées aléatoirement.

Notons au préalable que dans $DP_{distance}$, [Bailleux/Marquis 1999] ne propagent pas de littéraux purs. Or il est possible de le faire, comme nous allons le montrer.

Lemme 7
Soit BC une base de clauses. Soit $BC' \subset BC$. Soit d un entier positif. Si Distance-SAT(BC,PI,d) est vrai alors Distance-SAT(BC',PI,d) est vrai.

> **Preuve :**
> Elle est triviale car tout modèle de *BC* satisfait *BC'*. En particulier celui qui diffère de *PI* avec au plus *d* différences.

Supposons maintenant que l soit un littéral pur de *BC*. Nous avons donc $BC[l] \subset BC$. Donc d'après le lemme précédent : si Distance-SAT(BC,PI,d) alors Distance-SAT(BC[l],PI,d).

Lemme 8
Soit *BC* une base de clauses. Soit d un entier positif. Soit l un littéral pur de *BC*. Si Distance-SAT(BC[l],PI,d) est vrai alors Distance-SAT(BC,PI,d+1) est vrai.

> **Preuve :**
> En effet, si $\exists$ *M* modèle de *BC[l]* qui satisfait Distance-SAT(BC[l],PI,d) alors il suffit de poser M(l)=vrai pour satisfaire BC. Soit l est dans PI, auquel cas la différence entre *M* et *PI* ne change pas, soit $l^{\neg}$ est dans *PI* auquel cas la différence entre *PI* et *M* augmente de *1*.

Si maintenant nous restreignons l à un littéral pur dont la négation n'est pas dans *PI*, on obtient un résultat plus important : Si Distance-SAT(BC[l],PI,d) est vrai alors Distance-SAT(BC,PI,d) est vrai. On a alors équivalence entre les deux problèmes dans ce cas particulier.

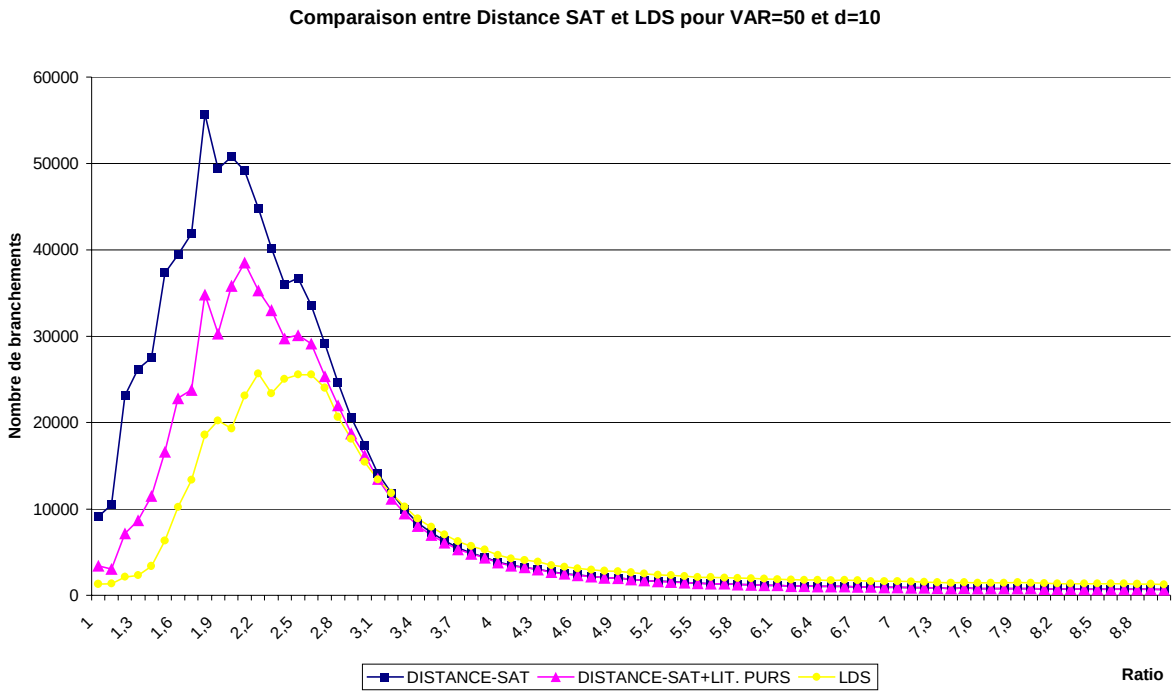
Proposition 41
Soit *BC* une base de clauses. Soit d un entier positif. Soit l un littéral pur de *BC* tel que $PI^{\wedge} \wedge l \not\models \bot$. Distance-SAT(BC[l],PI,d) est vrai ssi Distance-SAT(BC,PI,d) est vrai.

> **Preuve :**
> D'après les deux lemmes précédents.

Nous avons implanté sur la base de notre algorithme de Davis et Putnam l'algorithme présenté par [Bailleux/Marquis 1999] sans l'heuristique appelée LASSO[45] : nous l'appellerons DISTANCE-SAT. Nous avons modifié cette procédure pour prendre en compte la propagation des littéraux purs selon la propriété précédente. Nous la comparons à notre implantation de LDS_Probe avec branchement sur les symboles de PI d'abord. Cette dernière propage elle aussi des littéraux purs.

Voici les résultats obtenus pour PI une interprétation complète (SUB=50) et d=10 pour 50 variables propositionnelles.

[45] Car son calcul demande plusieurs modifications de notre implantation.

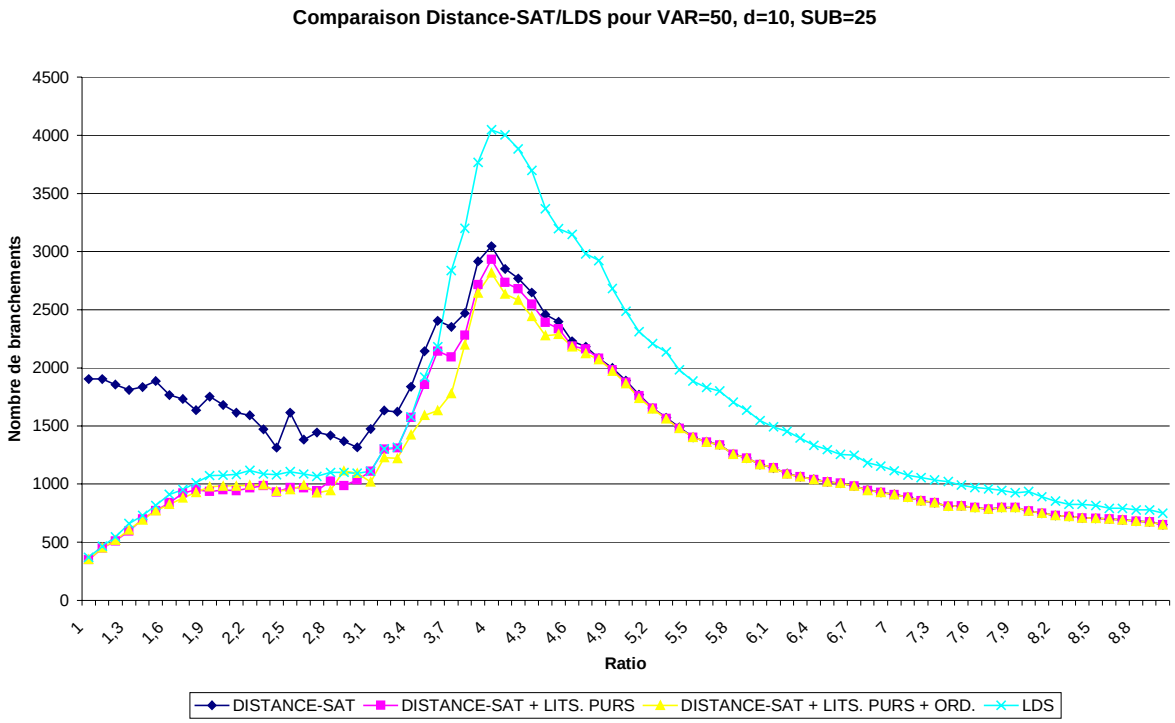


Comme on pouvait s'y attendre, la propagation de certains littéraux purs permet de réduire la taille de l'arbre de recherche. Résultat plus surprenant, l'utilisation d'une heuristique forçant l'ordre des littéraux de branchement (car comme l'interprétation de référence est complète, la priorité des variables n'est pas utilisée ici) permet de réduire le nombre de nœuds développés jusqu'au ratio 3,4 environ !

Pour confirmer les bons résultats de LDS pour résoudre DISTANCE-SAT, nous avons relancé les tests des algorithmes pour 100 variables et une distance de 20, en utilisant toujours une interprétation de référence complète. Nous obtenons des résultats comparables.

Nous avons donc modifié notre implantation DISTANCE-SAT pour forcer un branchement sur un littéral de PI d'abord. Les résultats obtenus sont presque exactement les mêmes que LDS (à quelques nœuds près sur plusieurs milliers, sans doute car nous choisissons la première variable maximisant l'heuristique et que l'ordre dans lequel sont examinées les variables n'est pas forcément le même dans les deux algorithmes).

Nous avons ensuite effectué le même test avec cette fois-ci une interprétation partielle comme référence (contenant la moitié des variables). Les résultats sont différents :



LDS est maintenant plus mauvais que Distance-SAT. Or les seules différences entre ces deux algorithmes sont le choix d'une variable de l'interprétation partielle en priorité lors du branchement, plus l'ordonnancement des littéraux pour ces variables. Or notre variante de Distance-SAT avec ordonnancement des littéraux lorsque la variable de branchement est dans l'interprétation partielle (DISTANCE-SAT+LITS. PURS+ORD) est aussi bonne, voire quelquefois meilleure que DISTANCE-SAT + LITS. PURS, donc l'ordonnancement ne semble pas en cause. Nous pensons donc que le mauvais comportement de LDS provient de la contrainte sur le choix des variables de branchement.

Ce qui confirme les propos de [Bailleux/Marquis 1999] cités précédemment.

On note donc que :

- la propagation des littéraux purs apparaissant dans l'interprétation (partielle) de référence permet de réduire le nombre de nœuds de l'arbre de recherche.
- lorsque le branchement concerne une variable de l'interprétation de référence, commencer par satisfaire le littéral appartenant à l'interprétation permet aussi de réduire le nombre de nœuds de l'arbre de recherche.
- forcer le branchement sur les variables de l'interprétation partielle d'abord est une mauvaise solution car on perturbe l'heuristique de branchement, qui ne prend pas en compte la structure particulière du problème.

Ces résultats ont été obtenus avec une implantation personnelle de $DP_{distance}$ n'utilisant pas l'heuristique LASSO présentée dans [Bailleux/Marquis 1999]. Celle-ci permettrait sans doute de diminuer le nombre de nœuds développés par notre algorithme. Néanmoins, les améliorations que nous proposons pour $DP_{distance}$ peuvent très bien être ajoutées à $DP_{distance+lasso}$: la propagation des littéraux purs par exemple permet de réduire le nombre de branchements. A priori, satisfaire un littéral de l'interprétation avant sa négation ne devrait pas changer les bonnes performances de l'heuristique LASSO puisque celle-ci est une heuristique balancée.

2.5 Filtrage de solutions

Au vu du nombre très important de solutions « compilées » en quelques centaines de clauses (cf. exemples page 123), nous souhaitons trouver une méthode de « filtrage » de solutions vérifiant des contraintes syntaxiques

comme : « quelles sont les solutions contenant le littéral X ? ne contenant pas le littéral Y ? ». Nous appellerons les premières des contraintes de présence et les secondes des contraintes d'absence.

Notre but est double :

- limiter le nombre de solutions (donc l'espace mémoire nécessaire à les stocker),
- limiter l'espace de recherche en fixant la valeur de certaines variables et/ou en rajoutant des contraintes.

Intuitivement, une contrainte d'absence d'un littéral se traduit par la falsification de ce littéral et une contrainte de présence par sa satisfaction. Ce qui permet d'exprimer le filtrage par une base de clauses.

Définition 75 (filtrage d'impliquants P-restreints premiers)
Soit $BC,CTR \sqsubset FP_S$. Soit P un ensemble consistant de littéraux. $F = \{ I \mid I \in IRP(BC,P)$ et $I \cup (P\backslash I)^{\neg} \models CTR \}$ est appelé un filtrage des impliquants P-restreints premiers de BC par CTR.

Nous avons déjà utilisé cette notion de filtrage lors du calcul du label d'une donnée. CTR avait alors une forme très particulière : elle contenait uniquement des littéraux de $P^{\neg}$, c'est à dire uniquement des contraintes d'absence. On en déduit la proposition suivante :

Proposition 42
Soit $BC \sqsubset FP_S$. Soit P un ensemble consistant de littéraux. Soit CTR une base de clauses contenant uniquement des littéraux $P^{\neg}$. $IRP(BC \cup CTR,P)$ est un filtrage des impliquants P-restreints premiers de BC par CTR.

> **Preuve :**
> découle du Lemme 4, page 94. $\forall I \in IRP(BC \cup CTR,P)$, $I \in IRP(BC,P)$ et $CTR^{\wedge} \wedge I^{\wedge} \not\models \bot$. Donc $\exists M$ tel que $M \models CTR$ et $I \subseteq M$. Comme CTR contient uniquement des littéraux de $P^{\neg}$, $M \subseteq I \cup (P\backslash I)^{\neg}$. Donc $I \cup (P\backslash U)\neg$ satisfait CTR.

Exemple : prenons $BC = \{ \quad A\ D, \quad B\ D, \quad C\ E\}$ et $P = \{A, B, C, D, E\}$. Les impliquants [P-restreints][46] premiers de BC sont $\{A, B, C\}^{\wedge}$, $\{C, D\}^{\wedge}$, $\{A, B, E\}^{\wedge}$, $\{E, D\}^{\wedge}$. Si nous voulons calculer les impliquants P-restreints premiers de BC ne contenant pas E, il nous suffit de calculer les impliquants P-restreints premiers de $BC \cup \{E\}$. On trouve bien $\{A,B,C\}^{\wedge}$ et $\{C,D\}^{\wedge}$.

Il est donc facile de prendre en compte les contraintes d'absence dans MPL lorsque l'on doit effectuer un calcul de P-impliquants premiers.

Malheureusement, ce résultat ne se généralise pas lorsque CTR est une base de clauses quelconques. Produire tous les impliquants P-restreints premiers en ne gardant que ceux qui « satisfont » CTR n'est pas une solution envisageable car elle ne permet aucunement de réduire l'espace de recherche (qui reste le même) ni l'espace mémoire (qui augmente, car il faut absolument ajouter les impliquants P-restreints trouvés dans la base de minimisation pour garantir leur primarité, il faut donc maintenir aussi un ensemble de solutions).

Cette dernière remarque n'est pas vraie si l'on cherche à filtrer des P-impliquants premiers. En effet, en utilisant la notion d'ensemble de littéraux nécessaires, la primarité est testée indépendamment des P-impliquants premiers déjà trouvés. Il est donc possible de restreindre l'espace de recherche sans perdre la preuve de la primarité.

On en déduit donc, à partir de la définition de filtrage, un algorithme calculant uniquement les impliquants d'une base de clauses pure BC satisfaisant une base de clauses CTR.

```
Castell(BC,IP,P,CTR)
// Calcule les P-impliquants premiers de BC
// en utilisant le filtrage CTR
// BC ne doit contenir que des littéraux de P
// IP l'ensemble des littéraux satisfaits
// P un ensemble consistant de littéraux
// retourne les P-impliquants premiers de BC.
       // il ne faut pas falsifier les contraintes
       Si    ∈ BC ou    ∈ CTR alors retourner ∅ Finsi ;
       // on restreint la recherche de littéraux nécessaires aux
       // seules clauses de BC
```

[46] Comme la base est pure, ces deux notions sont équivalentes.

```
Si IP∩P n'est pas nécessaire dans BC alors retourner ∅ Finsi ;
Si BC = ∅ alors
        // on a trouve un P-impliquant premier
        I = IP∩P ;
        Si I∪(P\I)¬ ⊨ CTR alors
                // il satisfait toutes les contraintes
                retourner {(IP∩P)^}
        Finsi ;
        Retourner ∅ ;
Finsi ;
l ← LittéralPourSimplifier(BC) ;
Si l ≠ null alors // si l existe
        retourner Castell(BC[l],IP∪{l},P,CTR)
Finsi ;
l ← choisirLiteral(BC) ;
retourner Castell(BC[¬l],IP∪{¬l},P) ∪ Castell(BC[l],IP∪{l},P,CTR)
```

Algorithme 18 Filtrage de P-impliqués par la méthode des littéraux nécessaires

Nous rappelons que ce filtrage ne peut être utilisé que sur une base de clauses pure, c'est à dire lors de la seconde phase du calcul de P-impliqué premier.

2.6 Et aussi …

Que faire si notre relation de préférence ne nous permet pas d'énumérer les impliquants P-restreints des plus préférés au moins préférés ? Par exemple, nous pouvons disposer pour chaque élément de P de probabilités (de pannes, d'occurrences), de poids (des coûts, des pénalités, des gains), etc… et d'une fonction d'agrégation (fonction d'évaluation). On cherche alors les impliquants P-restreints premiers les plus probables, ceux minimisant/maximisant l'agrégation des poids, des probabilités. On se trouve alors en présence d'un problème d'optimisation, dont l'approche arborescente est le *Branch & Bound* [Lawler/Wood 1966]. La principale différence entre les algorithmes présentés dans cette thèse et les algorithmes de type *B&B* concerne le moment où une solution optimale du problème est trouvée : dans le premier cas, dès qu'une solution est trouvée, elle est optimale, alors que dans le second cas, il faut attendre la fin de l'algorithme pour faire la preuve de l'optimalité.

Partie V : Autres Travaux

1 Algorithme incomplet pour le raisonnement non monotone

Nous avons présenté dans cette thèse différentes utilisations d'une variante d'un algorithme « complet » de résolution de SAT, notamment dans le cadre du raisonnement non monotone. Nous avons montré comment appliquer un calcul de P-impliqués premiers pour détecter les sous-bases minimales incohérentes d'une base de croyances moyennant un codage de cette base. Or cette approche a ses limites :

- nous introduisons une nouvelle variable par clauses, ce qui peut augmenter de façon considérable la taille du problème initial pour un algorithme énumératif,
- nous devons effectuer deux appels à MPL.

Les travaux que nous allons présenter sont basés sur l'utilisation d'une méthode stochastique pour approcher l'ensemble des sous-bases minimales incohérentes d'une base de croyances. Contrairement à notre approche, aucun codage de la base n'est nécessaire. Seule la « trace » de l'algorithme de recherche locale suffit ([Mazure, et al. 1996], voir aussi page 42).

[Bessant, et al. 1998] utilisent cette technique pour la révision de croyances, [Grégoire, et al. 1998] dans le cadre de l'inférence non monotone.

1.1 Application à la révision de croyances

Soit une base de croyances *CR* décrivant un système, et *f* une formule de « révision » du système, c'est à dire qui apporte de nouvelles informations sur l'état du système. Cette formule *f* peut être inconsistante avec *CR* ($CR \wedge f \models \bot$) : il faut alors faire une révision *sévère*. L'une des solutions pour résoudre ce problème est d'écrire *CR* à l'aide de règles à exceptions. [Bessant, et al. 1998] pensent qu'il n'est pas très naturel pour un ingénieur de la connaissance d'écrire l'ensemble des règles à exception nécessaires pour enlever toutes les incohérences, mais que le système devrait pouvoir, en cas de besoin, créer ces règles.

La base de croyance est écrite en calcul propositionnel sous forme CNF. Elle contient deux ensembles disjoints de symboles d'anormalité avec priorités [McCarthy 1986] P_{CR} et $P_{Révision}$ dont les éléments sont notés Ab_i et Ab_{r_i}. Les règles à exceptions de la forme $\alpha \wedge \neg Ab_i \rightarrow \beta$ seront fournies par l'ingénieur de la connaissance. Celles de la forme $\alpha \wedge \neg Ab_{r\_i} \rightarrow \beta$ seront générées automatiquement lors du processus de révision. Le raisonnement sur *CR* est basé sur la notion de modèles préférés définie par [Shoham 1987b] : $CR \mid\sim f$ ssi *f* est vraie dans tous les modèles préférés de *CR*. Ici, les modèles préférés sont ceux minimisant les symboles d'anormalité assignés à *vrai* (ce qui correspond à des impliquants Ab-restreints premiers, avec $Ab = P_{CR} \cup P_{révision}$).

Une révision *sévère* revient à ajouter un symbole d'anormalité Ab_{r_i} à certaines clauses de *CR* pour lever l'inconsistance. On dit alors que ces clauses sont *affaiblies* par l'ajout du symbole Ab_{r_i}.

Il existe plusieurs manières de lever l'inconsistance de la base (voir [Gärdenfors 1988] par exemple pour de plus amples détails). La solution la plus « propre » (celle qui affaiblit le moins de clauses) serait de rajouter un symbole d'anormalité dans une clause de chaque sous-base minimale inconsistante avec *f* : c'est l'approche de la révision *maxi-choix* (*maxi-choice*). Le problème réside dans le choix de la clause à sélectionner dans chaque sous-base : comment faire pour choisir parmi ces clauses si aucune autre connaissance n'est fournie ? Une autre approche, plus prudente, est la révision par pleine rencontre (*full meet*) : il s'agit d'affaiblir toutes les clauses contenues dans les sous-bases minimales inconsistantes. D'où l'idée de travailler sur l'ensemble des clauses concernées par l'inconsistance, le noyau.

Définition 76 Noyau [Bessant, et al. 1998]
Soit *CR* une base de clauses. Le noyau de *CR* selon *f* (noté Ker(*CR*,*f*)) est la conjonction des clauses de *CR* contenues dans une sous-base minimale inconsistante de $CR \wedge f$. Nous noterons Res(*CR*,f) la conjonction des clauses de *CR* qui n'apparaissent pas dans Ker(*CR*,*f*).

Remarque : nous avons montré précédemment que l'on pouvait calculer les sous-bases minimales inconsistantes d'une base de croyances à l'aide d'un calcul de nogoods sur un encodage de cette base (ici, seul *CR* sera encodée, pas *f*). De plus, nous avons montré comment calculer ces nogoods à partir de deux calculs d'impliquants P-restreints premiers. Le résultat du premier calcul d'impliquants P-restreints premiers suffit pour calculer exactement Ker(*CR*,*f*). En effet, le second calcul d'impliquants P-restreints premiers est un calcul de *minimal hitting set* utilisé pour construire les nogoods (sous-bases minimales incohérentes) de la base. Toute hypothèse (clause) appartenant à un nogood (à une sous-base incohérente) apparaît donc dans le résultat du premier calcul (si H = {A,B,C,D} et CR est une base dont les nogoods sont {B,C} et {B,D}, alors le résultat de la première passe sera {B} et {C,D}, dont on peut déduire l'union des nogoods, {B,C,D}).

Il est alors possible de définir la notion de révision par pleine rencontre :

Définition 77 Révision par pleine rencontre [Bessant, et al. 1998]
Supposons que *CR* soit le résultat de *i-1* révisions sévères précédentes :
$CR \circ^{\text{pleine rencontre}} f = (\text{Ker}(CR,f) \vee \text{Abr\_i}) \wedge \text{Res}(CR,f) \wedge f$

Res(*CR*,*f*) étant consistant, comme il est toujours possible de satisfaire Ab_{r_i} on obtient le résultat suivant :

Propriété 9 [Bessant, et al. 1998]
$CR \circ^{\text{pleine rencontre}} f$ est toujours consistante lorsque *f* est consistante.

Ce résultat est valable indépendamment de la forme de Ker(*CR*,*f*) : on peut donc imaginer approximer Ker(*CR*,*f*), en garantissant seulement qu'une clause au moins de chaque sous-base minimale inconsistante est contenue dans cette approximation (pour garantir Res(*CR*,*f*) consistant). C'est ce que proposent [Bessant, et al. 1998]. Ils utilisent une méthode de recherche locale pour détecter un sous-ensemble de clauses inconsistantes selon la méthodologie de [Mazure, et al. 1996], qui se fonde sur l'observation qu'une clause souvent falsifiée par une méthode de recherche locale appartient généralement à une sous base minimale inconsistante. L'idée est d'approcher *Ker(CR,f)* par l'ensemble des clauses les plus falsifiées de *CR* lorsque l'on cherche la consistance de $CR \wedge f$. Si la base de clauses est localement inconsistante, un ensemble de clauses se dégage clairement de la base, et il correspond à peu près à Ker(*CR*,*f*).

Il est ainsi « facile » d'effectuer la révision pleine rencontre de *CR* avec *f* : il suffit de lancer l'algorithme de recherche locale sur $CR \wedge f$ et si l'algorithme ne trouve pas de solution au bout d'un certain temps, arrêter l'algorithme et rajouter les symboles d'anormalité aux clauses les plus falsifiées.

Il apparaît empiriquement que la distance du résultat obtenu par rapport à Ker(*CR*,*f*) dépend du temps de calcul alloué pour rechercher une solution (ce qui donne un aspect « anytime » à la méthode).

Notre méthode complète ne semble pas adaptée à la révision sous pleine-rencontre. Nous pouvons utiliser le résultat de notre premier calcul d'impliquants P-restreints premiers pour calculer exactement *Ker(CR,f)* mais les performances de cette approche ne pourront sans doute pas égaler ceux obtenus par [Bessant, et al. 1998]. L'une des raisons est que nous obtenons plus d'informations que ces derniers : nous connaissons la valeur exacte de Ker(CR,f) et nous obtenons des informations sur les sous-bases minimales inconsistantes. Par exemple, le premier impliquant P-restreint premier trouvé par $\text{MPL}(\text{Encode}(\text{CR},\text{A}) \wedge \text{f}, \varnothing, \text{A}^{\neg})$ nous permet de restaurer la cohérence de $\text{CR} \wedge \text{f}$. Ce qui laisse entrevoir une application de nos travaux dans le cadre de la révision maxi-choix : dès qu'un impliquant P-restreint préféré est calculé, nous pouvons réviser notre base à l'aide de l'information obtenue. Le problème est de trouver une relation de préférence entre sous-bases maximales consistantes compatible avec une énumération arborescente des impliquants P-restreints.

1.2 Application à un processus d'inférence non monotone

On trouve dans [Grégoire, et al. 1998] une méthode d'inférence dans le cadre de modèles préférés utilisant une relation de préférence proche de la préférence *Worst In* présentée dans la partie précédente.

Soit S un ensemble de symboles propositionnels et Ab = {Ab1<Ab2<…Abn} un ensemble de symboles d'anormalité ordonné. Abi<Abj signifie que Abi est une anormalité plus probable que Abj (dans le cadre du diagnostic par exemple, Abi est une panne plus probable que Abj). On dispose d'une base de clauses *BC* construite à partir de $S \cup Ab$. Les symboles d'anormalité sont utilisés pour exprimer des règles à exception (modèle de bon fonctionnement du système comme dans le cadre du diagnostic consistant de Reiter).

Un modèle préféré de *BC* est intuitivement un modèle de *BC* qui ne satisfait pas de symboles d'anormalité (un modèle de bon fonctionnement) ou qui en contient un minimum.

Définition 78 (modèle préféré[Grégoire, et al. 1998])
M est un modèle préféré de *BC* ssi :

- M ne contient pas de symboles d'anormalité Abi à vrai
- M contient au moins un symbole d'anormalité Abi à vrai et
  - ∄ M' un modèle de *BC* tq M' ne contienne aucun Abj à vrai
  - ∄ M'' un modèle de *BC* tq M'' contienne un Abj à vrai et ∀Abi à vrai dans M, j<i.

Ils définissent alors la préférence entre 2 modèles par :

Définition 79 (préférence entre modèles [Grégoire, et al. 1998])
M est préféré (au sens large) à M' ssi

- M ne contient pas de symboles d'anormalités.
- ou M' ne contient pas de symboles d'anormalités ; dans ce cas, M ne contient pas de symboles d'anormalité.
- M et M' contiennent au moins un symbole d'anormalité et ∀Abj ∈ M', ∃ Abi ∈ M tq i≤j

On cherche donc des modèles ayant un minimum de variables d'anormalité assignées à vrai : dans notre terminologie, il s'agit d'impliquants Ab-restreints premiers. Les impliquants Ab-restreints sont préférés uniquement à partir de leur élément le plus probable. Ce que nous pouvons rapprocher de la préférence *Worst In* (au sens large) que nous présentions page 128, qui ordonne les impliquants P-restreints uniquement selon leur strate la moins certaine. On montre en effet que l'ordre sur les symboles d'anormalité correspond à une stratification des croyances sans utiliser de symboles d'anormalité.

Il suffit de partitionner *BC* en <*KB ;CR*> telles que *KB* contient toutes les règles « classiques » de *BC*, les connaissances, et *CR* est une base de clauses stratifiée construite à partir des règles à exception, qui contient des croyances. L'ordre total entre les Abi nous permet de construire un préordre total entre les clauses à exception. Nous en déduisons une stratification *CR* = <*CR1 ;CR2 ;CR3 ;… ;CRN*> avec CRi ={Ci | Ci Abi ∈ CR} et N=|Ab|.

On peut considérer que les règles a exception plus l'ordre entre Abi est un encodage d'une base de croyances stratifiée dans le cadre du raisonnement consistant. Le codage que nous avons proposé page 125 correspond à celui des bases stratifiées pour le raisonnement abductif.

On montre alors qu'un modèle M est préféré à un modèle M' dans le cadre de la Définition 79 (nous le noterons M GMS-préféré à M') ssi M'∩P est WI-préféré (au sens large) à M∩P en utilisant la stratification de littéraux P = <{Abn} ;… ;{Ab1}>.

En effet, [Amgoud 1999] a prouvé que H est strictement WI-préféré à H' ssi niveau(H)<niveau(H') ssi ∃h'∈H' tel que ∀h∈H h>h' (h est strictement plus certaine que h'). Soit au sens large H est WI-préféré à H' ssi ∃h'∈H' tel que ∀h∈H h'≤h. Ce qui signifie en terme de stratification ∃h'∈H' tel que ∀h∈H, Abk = h∩P∈Pi , Abl = h'∩P∈Pj et i≤j. Donc l≤k. Donc M est WI-préféré à M' ssi ∃Abl∈M' tel que ∀Abk∈M l≤k. Donc M' est GMS-préféré à M.

Nous avons proposé un algorithme de calcul d'impliquants P-restreints WI-préférés. [Grégoire, et al. 1998] proposent une méthode d'inférence fondée sur leur notion de modèles préférés.

Définition 80 [Grégoire, et al. 1998]
Soit f une formule. *BC* ⊢ f ssi f est satisfaite dans tous les modèles préférés de *BC* ssi ∃M un modèle de *BC* qui satisfait f et ∄ un modèle M' de *BC* préféré (au sens large) à M tel que M' ne satisfait pas f.

L'une des solutions pour déterminer si f est inférée préférentiellement par *BC* est de calculer l'intersection des modèles préférés de *BC* (comme nous l'avons proposé pour GCWA page 97). [Grégoire, et al. 1998] montrent comment cette inférence non monotone peut être effectuée en quelques tests de satisfiabilité (en pratique 5) en exploitant la nature de la relation de préférence.

Il faut d'abord tester si f et ¬f ne peuvent pas être satisfaites par un modèle de fonctionnement normal, c'est à dire tester la consistance de BC ∧ f (resp. ¬f) ∧{¬Abi | Abi ∈ Ab}^. Ce test est effectué par la méthode de recherche locale TSAT car si un tel modèle existe, cette procédure s'avère en général plus efficace qu'une méthode systématique pour trouver ce modèle. La preuve de l'inconsistance est effectuée par un algorithme systématique sur les clauses les plus falsifiées par TSAT. A l'aide de la définition précédente, f est inférée préférentiellement ssi il existe un modèle de bon fonctionnement pour f et pas pour ¬f.

Si cette première étape ne suffit pas à conclure, il faut chercher des modèles contenant des symboles d'anormalité satisfaits. L'étape précédente a permis de mettre en évidence un ensemble de clauses inconsistantes dans BC ∧ f ∧{¬Abi | Abi ∈ Ab}^. Parmi ces clauses, celles qui contiennent des Abi peuvent être satisfaites en satisfaisant ces Abi. Si l'ensemble de ces clauses est Ker(BC,f), alors seuls ces Abi sont à satisfaire pour trouver une modèle de BC ∧ f (car Ker(BC,f) contient toutes les manières de restaurer la cohérence de la base). D'où l'idée de chercher un modèle de BC ∧ f qui satisfait en priorité l'un de ces Abi.
La seconde étape peut se schématiser par l'algorithme suivant : il retourne vrai ssi BC ⊢~ f

Soit L = {Abi_1< …< Abi_m, Abi_{m+1} <…,< Abi_n} l'ensemble des symboles d'anormalité.
Abi_1…Abi_m dénote l'ensemble des symboles d'anormalité apparaissant dans les clauses le plus souvent falsifiées dans la première étape.
Abi_{m+1}…Abi_n dénote le reste des symboles d'anormalité.

```
Tant que L ≠ ∅ faire
       Abi = pop(L) ; // retourne le premier élément de L
       Si BC ∧ f ∧ {Abj | Abj∈ Ab, j<i }^ ∧ Abi consistant alors
       // il existe un modèle de BC ∧ f contenant Abi
       // il faut vérifier qu'il n'en existe pas de préféré pour ¬f
              Si BC ∧ ¬f ∧ {Abj | Abj∈ Ab, j ≤ i }^  est consistant alors       (1)
              // il existe un modèle de BC ∧ ¬f contenant Abi ou un Abj préféré à Abi.
              // on limite le reste de la recherche aux Abk strictement préférés à Abj
                     L = L \ { Abk | Abj ≤ Abk } ;
              Sinon
              // il existe un modèle de BC ∧ f tel qu'il n'existe pas de modèle de
              // BC ∧ ¬f qui lui est préféré.
                     Retourner vrai
              Finsi
       Finsi
Fin tant que
Retourner faux
```

L'une des astuces de cet algorithme pour limiter le nombre de tests de satisfaction est de remplacer les i preuves de l'inconsistance de BC ∧ ¬f ∧ {Abj | Abj ∈ Ab et j ≤ k} pour k = 1 à i par un seul test de consistance (1).

La plus importante est surtout l'utilisation de la trace de TSAT sur BC ∧ f ∧{¬Abi | Abi ∈ Ab}^ pour sélectionner un sous-ensemble des symboles d'anormalité sur lesquels appliquer d'abord la recherche « systématique ». Il s'agit d'une heuristique permettant de résoudre en pratique plus rapidement le problème.

Dans notre algorithme WI (comme dans tous les autres), nous ne retirons aucune information de la preuve de l'inexistence d'un modèle de bon fonctionnement (de l'impliquant P-restreint premier ∅). On peut imaginer effectuer la même technique que [Grégoire, et al. 1998] pour améliorer l'heuristique de branchement sur *P*. L'information retournée dans notre cas sera du type « appartient à un impliquant P-restreint premier » (car Kernel(BC,f) correspond à l'union des impliquants P-restreints premiers de BC∧f). On distinguera donc les littéraux de *P* apparaissant dans les impliquants P-restreints et ceux n'y apparaissant pas. Nous pensons qu'effectuer d'abord le branchement sur des variables ayant peu de chances d'apparaître dans un impliquant P-restreint permettra de réduire la taille de l'arbre de recherche. En effet, en supposant que ces littéraux n'apparaissent effectivement pas dans les impliquants P-restreints premiers, alors les sous-arbres développés par leur satisfaction devraient être élagués par les clauses de minimisations. De plus, lorsque nous encodons une base de clauses pour calculer ses sous-bases minimales incohérentes, la même information peut être obtenue à partir de la base originale et interprétée sur les variables ajoutées. D'un autre coté, le temps passé à calculer cette heuristique ne sera

pas forcément inférieur au temps gagné sur le calcul d'impliquants P-restreints premiers. Il s'agit donc d'une voie à explorer pour améliorer le comportement de notre méthode systématique.

2 Calcul de modèles minimaux par des tableaux analytiques

Une méthode proche de nos travaux est présentée par [Niemelä 1996]. Il s'agit d'une adaptation de la méthode des tableaux analytiques dans le cadre du calcul de modèles <P ;Z>-minimaux en calcul propositionnel et en logique du premier ordre dans [Niemelä 1996b]. A la différence de nos travaux, le but n'est pas ici de calculer des modèles minimaux mais de déterminer si une formule est conséquence logique de tous les modèles minimaux d'une base. Une modification de la méthode de preuve permet néanmoins de les obtenir.

Cette version de la méthode des tableaux analytiques [Smullyan 1968] s'applique comme l'algorithme de Davis et Putnam à un ensemble de clauses, c'est à dire une formule CNF. Les clauses ont la forme $b_1\ b_2\ b_3 \ldots b_n \quad a_1\ a_2\ a_3 \ldots a_m$. On dira plus généralement qu'elles sont de la forme A B avec A = $\{a_1,a_2,\ldots,a_m\}$ et $B=\{b_1,b_2,\ldots,b_m\}$ deux ensembles de variables.

La construction d'un tableau est basée sur l'utilisation de deux règles :

$$(H)\ \frac{b_1 b_2 \ldots b_n \quad a_1 a_2 \ldots a_m \qquad b_1, \quad b_2,\ldots, \quad b_n}{a_1 \quad ,a_2 \quad ,\ldots,a_{i-1} \quad ,a_{i+1} \quad ,\ldots,a_m} \qquad (C)\ \frac{b_1 b_2 \ldots b_n \quad a_1 a_2 \ldots a_m \qquad b_1, \quad b_2,\ldots, \quad b_n}{a_i \quad \neg a_i}$$

La première règle (H), est une règle d'hyper-résolution restreinte, que l'on peut comparer à la propagation des clauses unitaires dans un algorithme de type Davis et Putnam. Attention toutefois : seules les clauses unitaires positives sont propagées. L'auteur signale néanmoins que cette règle peut être modifiée pour propager aussi les clauses unitaires négatives sans changer la validité de ses travaux.

La seconde est une version affaiblie de la règle de branchement utilisée dans la procédure de Davis et Putnam. On effectue le branchement uniquement parmi des variables apparaissant positivement dans une clause. Nous verrons par la suite que cette règle est très importante pour nous car elle va nous permettre d'utiliser une version affaiblie de la propagation des littéraux purs dans notre algorithme MPL.

L'utilisation de ces règles peut être restreinte de manière à éviter l'introduction répétitive de littéraux : la règle (H) est utilisable seulement si a_i n'apparaît pas déjà sur la branche et la règle (C) est applicable seulement si ni a_i ni $\neg a_i$ ne sont présents sur la branche.

Définition 81 (tableau analytique)
Une branche est dite *close* si elle contient une paire de littéraux complémentaires ou il y a une clause de la forme b1 b2 …bn sur la branche telle que b1, b2, …, bn apparaissent aussi sur la branche. Si une branche n'est pas close, alors elle est dite *ouverte*. Un tableau est ouvert s'il a une branche ouverte, clos sinon.
Une branche est dite *terminée* quand pour toute clause b1 … bn a1…am sur la branche, si b1, …,bn sont sur la branche alors il existe un ai, i∈{1..m} sur la branche. Un tableau est *terminé* si toutes ses branches sont terminées ou closes.

Cette méthode permet de déterminer l'inconsistance d'une base de clauses.

Proposition 43 (méthode des tableaux saine et complète pour la satisfiabilité [Niemelä 1996])
Soit BC une base de clauses.
i) S'il existe un tableau clos pour BC, alors BC est inconsistante.
ii) Si BC est inconsistante, alors tout tableau terminé est clos.

Un algorithme capable de résoudre SAT basé sur le principe des tableaux est le suivant :

```
Tableau(BC,T)
        // retourne close si la base est inconsistante
        // ouvert sinon
        Tn ← doHrule(BC,T) ; // application de la règle H
```

```
si Tn = close alors retourner close
sinon
        a ← doCrule(BC,Tn) ; // application de la règle C
        si a=null alors retourner ouvert
        sinon
                si tableau(BC,Tn∪{a   }) = ouvert retourner ouvert
                sinon retourner tableau(BC,Tn∪{   a})
```

Algorithme 19 Méthode des tableaux analytiques

On note que cette méthode ressemble beaucoup à la procédure de Davis et Putnam : il suffit de remplacer ouvert par consistant (ou vrai) et close par inconsistant (ou faux) pour retrouver l'algorithme présenté page 26. Notons que la méthode développée par [Crawford/Auton 1993] classée parmi les bonnes implantations de DP est au départ une méthode basée sur la méthode des tableaux !

- doHrule propage sur la branche les clauses unitaires positives à l'aide de la règle H. Si la branche est close, alors doHrule retourne close sinon elle retourne la nouvelle branche.
- doCrule retourne une variable apparaissant positivement dans une clause de la branche, null sinon.

Définition 82 (inférence minimale)
Soit $BC \in F_{PS}$ une base de clauses. Une interprétation M est un modèle minimal de BC ssi M est un modèle de BC et $\nexists$ M' un modèle de BC tel que $M' \cap S \subset M \cap S$. BC infère minimalement α ($BC \vdash_{min} \alpha$) ssi α est satisfaite par tous les modèles minimaux de BC.

Proposition 44 [Niemelä 1996]
Soit $BC \in F_{PS}$ une base de clauses. Une interprétation M est un modèle minimal de BC ssi M est un modèle de BC et pour toute variable a, $M \models a$ implique $BC \cup N(BC,M) \models a$ avec $N(BC,M) = \{\neg v \mid v$ est une variable apparaissant positivement dans une clause de BC et $M \not\models v\}$.

L'algorithme tableau est alors modifié pour retourner close si la formule α est inférée minimalement par BC. D'après le théorème de la déduction, on peut remplacer la preuve de $BC \models \alpha$ par un test de la consistance de $BC \wedge \neg\alpha$. L'auteur veut ici remplacer la preuve de $BC \vdash_{min} \alpha$ par la construction d'un tableau pour $BC \wedge \neg\alpha$. Si celui-ci est clos, alors la formule α est vraie dans tous les modèles minimaux de BC. Une branche ouverte correspond à un contre exemple, un modèle minimal de BC qui ne satisfait pas α. Comme l'inférence minimale permet de déduire plus d'informations que l'inférence classique[47], il faut restreindre la production des modèles aux seuls modèles minimaux de BC à l'aide de la propriété précédente. Ainsi, dès qu'un modèle de $BC \wedge \neg\alpha$ est trouvé, on vérifie si c'est un modèle minimal de BC. Si oui, alors on a trouvé un contre exemple de $BC \vdash_{min} \alpha$. Sinon, on continue la recherche. Cela donne l'algorithme suivant : BC est la base de clause de départ, BC' est une CNF logiquement équivalente à $BC \wedge \neg\alpha$ et T est la branche du tableau initialisée à $\varnothing$.

```
mm(BC, BC', T)
Tn ← doHrule(BC',T) ; // application de la règle H
si Tn = close alors retourner close
sinon
        a ← doCrule(BC',Tn) ; // application de la règle C
        si a=null alors
                si pour tout b∈Tn, BC ∪ N(BC,Tn) ⊨ b alors retourner ouvert
                sinon retourner close
        sinon
                si mm(BC',Tn∪{a  }) = ouvert retourner ouvert
                sinon retourner mm(BC',Tn∪{  a})
```

Algorithme 20 Inférence minimale par la méthode des tableaux

L'algorithme peut être utilisé pour renvoyer les modèles minimaux de BC en effectuant une petite modification après le test de minimalité (les modèles minimaux sont ajoutés dans min) :

```
mm'(BC, BC', T, min)
Tn ← doHrule(BC',T) ; // application de la règle H
si Tn = close alors retourner close
```

[47] si l'on considère un langage contenant deux variables propositionnelles *a* et *b*, la formule *BC = {a}* permet classiquement de déduire *a*, alors qu'elle permet de déduire minimalement *a* et $\neg b$

```
sinon
        a ← doCrule(BC',Tn) ; // application de la règle C
        si a=null alors
                si pour tout b∈Tn, BC ∪ N(BC,Tn) ⊨ b alors
                        min ← min ∪ {{b | b ∈ Tn}} ;
                retourner close
        sinon
                si mm(BC',Tn∪{a   })= ouvert retourner ouvert
                sinon retourner mm(BC',Tn∪{   a})
```

Algorithme 21 Calcul de modèles minimaux par la méthode des tableaux

On peut noter que toutes les branches sont systématiquement explorées car l'algorithme ne retourne jamais ouvert.

Par rapport à notre méthode MPL, la principale différence se situe au niveau du test de minimalité. Ici il s'agit d'un test de consistance effectué à chaque fois que l'on se trouve en présence d'un modèle de BC'. En notant M l'ensemble des variables satisfaites dans la branche Tn, il suffit donc de tester la consistance de $BC \cup \{M^{\neg\vee}\} \cup N(BC,M)$ à l'aide ne n'importe quel prouveur SAT. L'auteur utilise … une méthode par tableaux. On peut optimiser ce test en prenant pour *M* l'ensemble des variables satisfaites sur la branche Tn produites par la règle de branchement : en effet, si celles là sont vraies, les autres le sont aussi.
Dans le cadre de Davis et Putnam, il s'agirait de tester pour un impliquant *I* de *BC* la consistance de $BC \cup (I \cap P)^{\neg\vee} \cup (P \setminus I)^{\neg\wedge}$. Comme aucune information de type ajout de clause de minimisation n'est ajoutée dans la base une fois une solution minimale trouvée, ce test sera effectué à chaque impliquant de la base, ce qui ne paraît pas très efficace.

Pour notre part, nous avons choisi une règle de branchement nous garantissant une énumération compatible avec la minimalité des modèles de BC. Ainsi, le premier modèle trouvé est un modèle minimal.

Les implantations de ces méthodes par l'auteur sont en réalité des méthodes dérivées de la procédure de Davis et Putnam car comme dans cette dernière, la base de clauses initiale est réduite par propagation unitaire à chaque appel de la fonction doHrule. L'auteur note cette correspondance en soulignant que la propagation des littéraux purs ne peut pas être effectuée car elle «manque »[48] des modèles minimaux. Une autre différence concerne la règle de branchement (C) qui est plus restrictive que la règle de branchement de la méthode de Davis et Putnam. Si aucune variable n'apparaît positivement dans une clause, alors un modèle est trouvé. Cela permet de réduire le nombre de modèles trouvés sans perdre de modèles minimaux. En effet, si BC = {a b, b a}, une méthode de type Davis et Putnam sans arrêt au premier modèle trouvé donnerait comme résultat ∅ et {a,b}. L'application de la règle (C) étant impossible, seul le modèle minimal ∅ est trouvé par la méthode des tableaux.

Nous allons montrer que cette règle de branchement est équivalente dans le cadre d'une méthode de Davis et Putnam à propager les littéraux négatifs purs. Nous étendrons cette propriété au cas d'un ensemble consistant de littéraux *P* ou la propagation des littéraux purs de $P^{\neg}$ ne modifiera pas le nombre de modèles minimaux.

Proposition 45
Les deux fonctions suivantes sont équivalentes (on suppose que BC est réduite par doHrule comme dans un algorithme de type DP) :

- fonction doCrule(BC) { si ∃ x tel que x apparaît positivement dans une clause de BC retourner x sinon retourner null }
- fonction doDPruleWithNPLP(BC) { Tant qu'il existe x un littéral positif tel que ¬x est pur satisfaire ¬x ; Si ∃ x un littéral positif de BC tel que x n'est pas assigné retourner x sinon null}

Preuve :

) Supposons qu'il existe x apparaissant positivement dans BC. Alors il sera retourné par doCrule. De plus, ¬x n'est pas pur, donc il sera aussi retourné par doDPruleWithNPLP.
Si une telle variable n'existe pas, deux solutions :
Soit toutes la variables sont assignées, auquel cas les deux fonctions retournent null

[48] *« This rule cannot be used for generating minimals models because some minimal models might then be lost. For example, for* $\Sigma = \{ a \vee b\}$ *the Davis-Putnam procedure with the affirmative-negative rule returns that* Σ *is satisfiable with (implicitly) assuming both a and b true and, thus, misses the two minimal models {a} and {b} ».*

Soit toutes les variables de BC apparaissent exclusivement négativement. Dans ce cas, elles seront toutes assignées par doDPruleWithNPLP donc les deux fonctions retournerons là encore null.
⇐) Supposons qu'il existe x de BC qui ne soit pas assignée. Cette variable apparaît positivement car sinon elle aurait été assignée par propagation des littéraux purs négatifs. Donc elle sera retournée par les deux fonctions.
Supposons qu'une telle variable n'existe pas. Toutes les variables sont donc assignées.
Soit par propagation des littéraux purs négatifs,
Soit par ce qu'ils étaient déjà tous satisfaits dans BC.
Dans les deux cas les deux fonctions retournent null.

Nous avons donc ajouté cette propriété à MPL généralisée au cadre d'un ensemble consistant de littéraux. Si un élément de $P^{\neg}$ est pur, il est propagé.

Ainsi la seule différence entre nos deux algorithmes devient le test de minimalité. En ce qui concerne les heuristiques de branchement utilisées dans doCrule, rien n'est précisé.

Les seuls tests présentés par l'auteur sont ceux concernant deux bases de clauses :

Test1 = { a1 b1 c1 d1 e1 ; a2 b2 c2 d2 e2 ; a3 b3 c3 d3 e3 ; a4 b4 c4 d4 e4 ;
a1 a4 ; a4 a3 ; a3 a2}

Test2 = { a1 b1 c1 d1 e1 ; a2 b2 c2 d2 e2 ; a3 b3 c3 d3 e3 ; a4 b4 c4 d4 e4 ;
a1 a4 ; a4 a3 ; a3 a2 ;
b1 b4 ; b4 b3 ; b3 b2 ;
c1 c4 ; c4 c3 ; c3 c2 ;
d1 d4 ; d4 d3 ; d3 d2 }

La première admet 341 modèles minimaux et la seconde 17. Le prouveur de l'auteur (mm) est disponible sur son site. Malheureusement, étant implanté en ECLIPSe Prolog, un prolog commercial, nous n'avons pu le tester sur notre machine. Nous nous contenterons donc des résultats présentés par l'auteur : les 341 modèles minimaux de Test1 sont trouvés en 2s et les 17 modèles minimaux de Test2 sont trouvés en 0,5s sur une Sparc 4.

Nous avons implanté cette méthode à partir de notre Davis et Putnam (sans propagation des littéraux purs). Nous choisissons pour le branchement uniquement un littéral apparaissant positivement. Si un tel littéral n'existe pas, nous effectuons un test de minimalité. Nous l'effectuons à l'aide du même algorithme de Davis et Putnam (sans UP) que celui utilisé dans MPL. On trouve les 341 solutions en 7,4 secondes (796 nœuds) et les 17 solutions en 6,2 secondes (671 nœuds). A titre d'indication, dans le premier test, seuls des modèles minimaux sont trouvés. Dans le second, 204 modèles non minimaux sont trouvés.

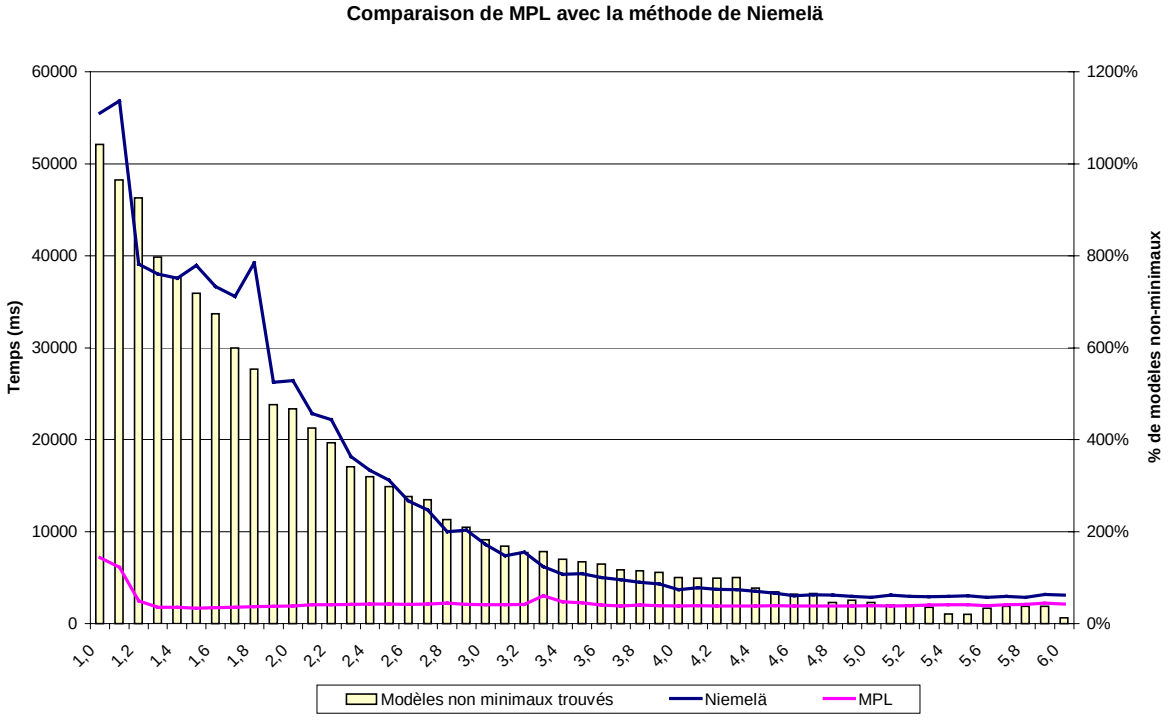
Nous avons lancé notre algorithme MPL (sans la propagation des littéraux purs négatifs) sur ces deux tests. Le comportement de l'algorithme est mauvais sur la première base : comme il y a beaucoup plus de modèles minimaux que de clauses dans la base, la prise en compte des clauses de minimisation dans l'heuristique perd l'algorithme (plus de 25000 nœuds développés !). Cela est vérifié par l'utilisation de MPL sans utiliser les clauses de minimisation dans l'heuristique : le même résultat est trouvé en développant uniquement 792 nœuds !. Ce comportement est inversé dans le cadre de la deuxième base : la prise en compte des clauses de minimisation dans l'heuristique permet de développer moins de nœuds (213 contre 429). Il semble donc que la prise en compte des clauses de minimisation dans l'heuristique est efficace lorsque le nombre de solutions est inférieur ou proche du nombre de clauses de la base de départ. Cela semble confirmé par les résultats empiriques obtenus où nous considérions des bases de clauses dont le nombre de modèles minimaux était inférieur au nombre de clauses.

Nous avons ensuite utilisé sur ces deux tests la propagation des littéraux purs négatifs : 1005 nœuds développés dont 213 propagations de littéraux purs négatifs pour le premier, 204 nœuds développés dont 35 propagations de littéraux purs négatifs pour le second. Ce sont donc des instances pour lesquelles la propagation des littéraux purs négatifs est importante.

En ce qui concerne les temps d'exécution de ces tests, nous les donnons dans le tableau suivant : comme il s'agit d'une implantation en Java, nous donnons le temps nécessaire à la résolution du problème la première fois que l'algorithme est appelé (la compilation à la volée est incluse dans ce temps) puis entre parenthèse le temps minimum obtenu en faisant 5 appels successifs au prouveur sur cette instance (on peut considérer que tout le code est traduit en code natif, c'est à dire équivalent à du C++).

	mm (Sparc 4, ECLIPSe Prolog)	MPL classique (P133, Java 1.2)	MPL sans clauses de min. dans l'heuristique	MPL + propagation des littéraux purs négatifs
Test 1	2s	15s (7,8s) (25822 nœuds)	4s (1,1s) (792 nœuds)	2,8s (0,6s) (1005 nœuds)
Test 2	0,5s	0,39s (0,22s) (213 nœuds)	0,82s (0,65s) (429 nœuds)	0,33s (0,16s) (204 nœuds)

Nous avons testé notre implantation de cette méthode sur un calcul de modèles minimaux, quand SUB=VAR (en effet, dans cette version de mm, l'auteur ne prends pas en compte un ensemble de circonscription). Pour des questions de temps de calcul, nous avons pris VAR = 15. Nous présentons les temps de calculs cumulés pour résoudre 100 bases aléatoires de même ratio. Nous avons aussi ajouté le pourcentage de modèles non minimaux trouvés par rapport au nombre de modèles minimaux.



MPL semble avoir un comportement constant : c'est parce que les instances sont très petites. On note par contre que la méthode de Niemelä n'est pas très performante sur les instances sous-contraintes. Cela est du au nombre élevé de modèles non minimaux trouvés. Quand il y a moins de modèles, ceux trouvés sont souvent minimaux.

Nos deux approches sont donc très similaires : utilisation d'un prouveur SAT pour calculer des modèles minimaux. Pourquoi avoir choisi la méthode des tableaux analytiques, quand le meilleur prouveur SAT complet depuis quelques années est un algorithme de type Davis et Putnam (surtout quand l'implantation de la méthode des tableaux ressemble à s'y méprendre à un DP) ?

Notre argument sur ce point est de pouvoir utiliser les différentes heuristiques et amélioration autour de Davis et Putnam.

Il semble que celui de l'auteur soit tout autre : cette approche est étendue au cadre du premier ordre et au calcul de modèles <P ;Z> minimaux dans [Niemelä 1996b]. La méthode des tableaux, si elle n'est pas la meilleure dans le cadre du calcul propositionnel, à un équivalent en logique du premier ordre. Ce n'est pas le cas pour la méthode de Davis et Putnam : on préférera utiliser la résolution par exemple.

Conclusion

Cette thèse présente une modélisation logique d'un concept utilisé dans divers raisonnements en logique propositionnelle. Il s'agit de restreindre des impliquants à un ensemble consistant P de littéraux : les impliquants P-restreints. Cette notion se retrouve par exemple dans celle de modèles minimaux de Herbrand ou de diagnostic consistant. L'hypothèse de restriction à un ensemble *consistant* de littéraux permet d'utiliser la meilleure (à l'heure actuelle) procédure de recherche systématique pour SAT pour explorer l'espace des impliquants P-restreints : l'algorithme de Davis et Putnam.

L'algorithme de calcul d'impliquants P-restreints premiers d'une base de clauses qui est proposé, MPL, utilise les mêmes concepts que celui de Davis et Putnam :

- propagation des clauses unitaires,
- propagation des littéraux purs dont la négation est dans P,
- utilisation des heuristiques utilisées dans les meilleures implantations de Davis et Putnam.

Afin de garantir la primarité des solutions obtenues, nous avons également introduit :

- une énumération des impliquants P-restreints compatible avec l'inclusion grâce à un ordre sur les littéraux de branchement. Ceci garantit que le premier impliquant trouvé est un impliquant P-restreint premier.
- l'ajout d'une nouvelle clause dans la base lorsqu'une solution est trouvée : cette clause empêche la production d'impliquants P-restreints sous-sommés par l'impliquant P-restreint premier trouvé.

A partir de cet algorithme de calcul d'impliquants P-restreints premiers, nous avons montré comment calculer des P-impliquants et P-impliqués premiers. Dans le premier cas, il suffit d'appliquer MPL sur une base de clauses filtrée sur P. Dans le second cas, il s'agit d'appliquer deux fois MPL : la première fois pour obtenir les solutions sous une forme « compilée », la seconde fois pour les expliciter. Cette deuxième phase peut être effectuée par n'importe quel algorithme de calcul d'impliquants premiers.

L'étude du comportement de nos algorithmes sur des problèmes générés aléatoirement montre que l'ajout de clauses de minimisation, outre de garantir la primarité des solutions, permet généralement de réduire la taille de l'arbre de recherche grâce aux clauses unitaires produites et à la prise en compte de ces clauses dans l'heuristique. Elle amène également à limiter l'utilisation de MPL pour effectuer la deuxième phase du calcul de P-impliqués premiers aux situations où le nombre de solutions est comparable à la taille de la base. Dans les autres cas, le nombre de clauses à ajouter dans la base devenant important, on utilisera une autre technique pour déterminer la primarité des solutions (par exemple la notion d'ensemble de littéraux nécessaires proposée par [Castell 1997]).

Le calcul d'impliquants P-restreints et de P-impliqués premiers a été appliqué à divers domaines : ATMS/ACMS, raisonnement en monde clos, décision dans l'incertain en logique propositionnelle, argumentation. Les notions d'impliquant P-restreint et de P-impliqué premier lorsque P est un ensemble consistant de littéraux sont à la base de nombreux problèmes en raisonnement propositionnel.

Le passage obligatoire par une première étape de compilation avant de pouvoir obtenir un P-impliqué premier constitue l'un des défauts majeurs de notre méthode. En effet, ce calcul intermédiaire s'avère en pratique plus difficile que l'extraction des solutions qui est immédiate (leur nombre peut être exponentiel par rapport à la taille de la formule compilée).

Nous proposons donc la notion d'impliquant P-restreint préféré pour « filtrer » l'ensemble des impliquants P-restreints premiers d'une base de connaissances. Des algorithmes de calcul d'impliquants P-restreints préférés pour quelques relations de préférence telles que la cardinalité ou des relations basées sur la notion de niveau de

certitude sont présentés. On en déduit la notion de P-impliqué préféré, en effectuant l'explicitation des solutions par un algorithme de calcul d'impliquant P-restreint préféré.

Notre approche présente certaines limites :

- Le calcul des P-impliqués premiers d'une base de P-impliqués par notre méthode en « deux passes » s'avère désastreux car la taille de la base « compilée » est exponentielle par rapport à la taille de la base de départ.

- Nous avons proposé ce calcul de P-impliqués à l'aide d'un algorithme unique de calcul d'impliquant P-restreint premier car, pour l'explicitation des solutions (cas particulier d'une base de littéraux purs), il est équivalent de calculer les P-impliquants ou les impliquants P-restreints premiers de la base. Ainsi toute amélioration de l'algorithme se répercute sur le calcul des impliquants P-restreints et des P-impliqués premiers. Toutefois, notre méthode ne tient pas compte des spécificités du calcul de P-impliquants : la technique de calcul de P-impliquants premiers proposée par Thierry Castell (adaptée au cas particulier où P est consistant) est, en général, plus efficace que notre méthode de calcul d'impliquants P-restreints premiers pour effectuer l'explicitation des solutions.

- Notre méthode de calcul d'impliquants P-restreints premiers est fondée sur une énumération des variables de *P* à la racine de l'arbre, puis l'utilisation d'un algorithme de satisfaction lorsque toutes les variables de *P* ont une valeur de vérité. Cette approche s'est avérée mauvaise dans le cadre du problème Distance-SAT car cette contrainte perturbe l'heuristique.

Le cas de Distance-SAT est particulier. Il s'agit d'un problème de décision alors que nous travaillons sur un problème d'optimisation (nous devons trouver toutes les solutions minimales). De plus, il est possible de caractériser les solutions minimales pour la cardinalité par une propriété commune : elles ont la même taille. [Castell 1997] a montré une propriété similaire pour les P-impliquants premiers : leurs littéraux sont des ensembles de littéraux nécessaires. Cette caractéristique permet de savoir si une solution est minimale indépendamment des autres solutions déjà trouvées. Il n'existe pas, à notre connaissance, de propriété similaire pour les impliquants P-restreints premiers. Il est donc nécessaire de connaître tous les impliquants P-restreints pour trouver les impliquants P-restreints premiers. Dans ces conditions, l'approche que nous proposons dans MPL nous semble intéressante car elle évite d'effectuer explicitement les tests de minimalité et les solutions déjà trouvées permettent de guider l'heuristique.

Plusieurs points restent à développer :

- Nos résultats expérimentaux semblent montrer que l'heuristique utilisée dans notre algorithme de calcul d'impliquants P-restreints premiers est perturbée lorsque P contient environ 20 % de variables du langage. [Bailleux/Marquis 1999] ont présenté une heuristique appelée *lasso* pour le problème DistanceSAT qui permet de réduire de manière significative le nombre de nœuds explorés. L'idée consiste à chercher les variables de branchement parmi celles des clauses falsifiées par l'interprétation partielle de référence (pour nous, $P^{\neg}$). Son adaptation au calcul d'impliquants P-restreints premiers contribuera peut-être à améliorer le comportement de notre algorithme dans ce cas. Il faut cependant noter que les auteurs montrent que cette heuristique est performante pour des interprétations de référence complètes, mais ne garantissent pas ce résultat lorsque elles sont partielles.

- Une autre approche pour améliorer la recherche d'impliquants P-restreints premiers consiste peut être à utiliser une méthode de type « retour arrière intelligent » comme celle de Rel-SAT [Bayardo/Schrag 1997] par exemple lors de l'énumération des impliquants P-restreints. On constate empiriquement que ces techniques permettent de corriger une mauvaise heuristique : si l'heuristique est bonne, il y a peu de retours arrières intelligents. Comme le branchement est forcé sur des variables particulières au début de l'arbre de recherche, il y a de fortes chances que ces choix ne soient pas optimaux.

- Une politique de révision « maxi-choix » consiste à affaiblir dans une base de croyances inconsistante une clause de chaque sous-base inconsistante. Nous avons noté dans la partie V que ces clauses correspondent à la notion d'impliquant P-restreint premier, via un encodage de la base. Le problème est de savoir quelle clause choisir dans chaque sous-base. Nous souhaitons étudier les relations de préférence existant dans ce cadre pour proposer un système de révision basé sur la notion d'impliquant P-restreint préféré.

L'ensemble des travaux présentés a conduit à l'implantation d'un prototype intégrant les différents algorithmes développés. Ses performances « brutes » ne sont pas satisfaisantes : par exemple, les prouveurs SAT implantés ont des temps de résolution médiocres par rapport aux prouveurs actuels. Outre le langage choisi, Java, ces mauvaises performances s'expliquent avant tout par l'objectif de ce prototype : faciliter le développement et l'intégration de divers algorithmes sur une même plate-forme (les différents algorithmes pouvant interagir), et permettre de manipuler facilement les données calculées. Par exemple, tous les algorithmes sont écrits de manière récursive (cela facilite leur lecture et leur conception) et utilisent fortement les notions d'héritage et de surcharge permises par les langages objets. Nous utilisons rarement des tableaux mais des vecteurs (plus souples d'utilisation). De plus, comme tous nos algorithmes ne sont pas dédiés au problème SAT, nous avons fait des compromis sur les représentations des données. Ce prototype peut être amélioré en utilisant des structures de données plus fines comme les TRIES pour les clauses, et des mécanismes de propagation plus performants, comme ceux que l'on retrouve pour SATO.

Nous pensons néanmoins que ces choix nous ont permis de mieux comprendre le comportement des différents algorithmes utilisés. Ils nous ont permis de rapidement mettre en œuvre une méthode (comme par exemple celle de Niemelä, ou l'algorithme DistanceSAT) pour la comparer, sur la base d'une même implantation, avec l'un de nos algorithmes. Ce prototype permet de tester sur de petits exemples (comme c'est souvent le cas lorsque l'on étudie une nouvelle approche), et de manière relativement conviviale, les diverses techniques fondées sur le raisonnement consistant ou abductif présentées dans cette thèse.

Références bibliographiques

[Allais 1953] Allais M., *Le comportement de l'homme rationnel devant le risque : critique des postulats et axiomes de l'école américaine*, Dans Econometrica, n° 21, pp. 503-546, 1953.

[Amgoud 1999] Amgoud Leila, *Contribution à l'intégration des préférences dans le raisonnement argumentatif*, Thèse de Doctorat de l'Université Paul Sabatier, Toulouse, 1999.

[Amgoud, et al. 1996] Amgoud L., Cayrol C. and Le-Berre D., *Comparing Arguments Using Preference Orderings for Argument-based Reasoning*, In Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence, pp. 400-403, 1996.

[Audemard, et al. 1999] Audemard G., Benhamou B. et Siegel P., *La méthode d'avalanche AVAL : une méthode énumérative pour SAT*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'99), pp. 17-25, 1999.

[Bailleux/Marquis 1999] Bailleux O. and Marquis P., *DISTANCE-SAT: Complexity and Algorithms*, In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), pp. 642-647, 1999. Voir aussi *DISTANCE-SAT : complexité et algorithmes*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'99), pp. 199-206, 1999.

[Bayardo/Schrag 1997] Bayardo R.J.J. and Schrag R.C., *Using CSP Look-Back Techniques to Solve Real-World SAT Instances*, In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), pp. 203-208, 1997.

[Benhamou/Saïs 1994] Benhamou B. and Saïs L., *Tractability Through Symmetries in Propositional Calculus*, In Journal of Automated Reasoning, no 12, pp. 89-102, 1994.

[Bessant, et al. 1998] Bessant B., Grégoire E., Marquis P. and Saïs L., *Syntax-based belief revision through local search*, In Proceedings of Belief Revision (BR'98), 1998.

[Billionnet/Sutter 1992] Billionnet A. and Sutter A., *An efficient algorithm for the 3 satisfiability problem*, In Operation Research Letters, no 12, pp. 29-36, 1992.

[Bonet/Geffner 1999] Bonet B. and Geffner H., *Planning as heuristic search : New Results*, In Proceedings of European Conference on Planning (ECP'99), http://www.ldc.usb.ve/~hector, 1999.

[Boufkhad 1996] Boufkhad Y., *Aspects probabilistes et algorithmiques du problème de satisfiabilité*, Thèse de doctorat de l'Université de Paris 6, 1996.

[Bouquet 1999] Bouquet F., *Gestion de la dynamicité et énumération d'impliquants premiers : une approche fondée sur les Diagrammes de Décision Binaire.*, Thèse de l'Université de Provence (Aix-Marseille I), 1999.

[Brisoux 1999] Brisoux-Devandville L., *Satisfaisabilité propositionnelle en informatique : aspects algorithmiques et extensions du formalisme*, Thèse de doctorat de l'Université d'Artois, 1999.

[Bryant 1986] Bryant R.E., *Graph-based Algorithms for Boolean Function Manipulation*, In IEEE Transactions in Computers, vol. 8, n° 35, pp. 677-691, 1986.

[Cadoli 1995] Cadoli M., *Language restriction : complexity of minimal reasoning*, In Tractable Reasoning in Artificial Intelligence, Springer Verlag, pp. 27-65, 1995.

[Cadoli, et al. 1998] Cadoli M., Giovanardi A. and Schaerf M., *An algorithm to Evaluate Quantified Boolean Formulae*, In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), pp. 262-267, 1998.

[Castell 1997] Castell T., *Consistance et déduction en logique propositionnelle*, Thèse de doctorat de l'Université Paul Sabatier, Toulouse, 1997.

[Castell, et al. 1996] Castell T., Cayrol C., Cayrol M. and Le-Berre D., *Using the Davis and Putnam procedure for an efficient computation of preferred models*, In Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96), pp. 350-354, 1996.

[Castell, et al. 1998] Castell T., Cayrol C., Cayrol C. et Le-Berre D., *Modèles P-restreints : application à l'inférence propositionnelle*, Dans Actes du Onzième Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA'98), pp. 205-214, 1998.

[Castell/Cayrol 1996] Castell T. and Cayrol M., *Computation of Prime Implicates and Prime Implicants by the Davis and Putnam Procedure*, In Proceedings of the ECAI'96 Workshop on Advances in Propositional Deduction, pp. 5-10, 1996.

[Castell/Fargier 1998] Castell T. et Fargier H., *Entre SAT et CSP : Problèmes de Satisfaction Propositionnels et CSPs clausaux*, Dans Actes du Onzième Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA'98), pp. 117-126, 1998.

[Cayrol, et al. 1993] Cayrol C., Royer V. and Saurel C., *Management of preferences in Assumption-based Reasoning*, In Lecture Notes In Computer Science n° 682, pp. 13-22, 1993.

[Chang/Lee 1973] Chang C. and Lee R., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[Chatalic/Simon 1999] Chatalic P. et Simon L., *Davis et Putnam, 40 ans après : une première expérimentation*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'99), pp. 9-16, 1999.

[Console/Torasso 1991] Console L. and Torasso P., *A spectrum of logical definitions of model-based diagnosis*, In Computational Intelligence, vol. 3, no 7, pp. 133-141, 1991.

[Cook 1971] Cook S., *The complexity of theorem-proving procedures*, In Proceedings of the Third IEEE Symposium on the Foundations of Computer Science, pp. 151-158, 1971.

[Crawford, et al. 1996] Crawford J., Ginsberg M., Luks E. and Roy A., *Symmetry-Breaking Predicates for Search Problems*, In Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), pp. 148-159, 1996.

[Crawford/Auton 1993] Crawford J.M. and Auton L.D., *Experimental results on the crossover point in satisfiability problems*, In Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93), pp. 21-27, 1993.

[Davis, et al. 1962] Davis M., Logemann G. and Loveland D., *A machine program for theorem proving*, In Communications of the ACM, no 5, pp. 394-397, 1962.

[Davis/Putnam 1960] Davis M. and Putnam H., *A computing procedure for quantification theory*, In Journal of the ACM, no 7, pp. 201-215, 1960.

[Dechter/Rish 1994] Dechter R. and Rish I., *Directional Resolution: the Davis-Putnam Procedure, Revisited*, In Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Revision (KR'94), pp. 134-145, 1994.

[de-Kleer 1986] de-Kleer J., *An Assumption-based TMS*, In Artificial Intelligence, no 28, pp. 127-162, 1986.

[de-Kleer 1986b] de-Kleer J., *Extending the ATMS*, In Artificial Intelligence, no 28, pp. 163-196, 1986.

[de-Kleer 1988] de-Kleer J., *A general Labeling Algorithm for Assumption-based Truth Maintenance*, In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'88), pp. 188-192, 1988.

[de-Kleer 1991] de-Kleer J., *Focusing in probable Diagnoses*, In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91),842-848, 1991.

[de-Kleer 1992] de-Kleer J., *An improved incremental algorithm for generating prime implicates*, In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), pp. 780-785, 1992.

[de-Kleer, et al.1992] de-Kleer J., Mackworth A.K. and Reiter R., *Characterizing Diagnoses and systems*, In Artificial Intelligence, no 56, pp. 197-222, 1992.

[de-Kleer/Williams 1987] de-Kleer J. and Williams B.C., *Diagnosing multiple faults*, In Artificial intelligence, no 32, pp. 97-130, 1987.

[Dempster 1967] Dempster A.P., *Upper and lower probabilities induced by a multivalued mapping*, In Annals of Mathematical Statistics, n° 38, pp. 325-339, 1967.

[Dubois, et al. 1990] Dubois D., Lang J. and Prade H., *Handling Uncertain Knowledge in an ATMS using Possibilistic Logic*, In Methodologies for Intelligent Systems, 5, pp. 252-259, 1990.

[Dubois, et al. 1996] Dubois O., André P., Boufkhad Y. and Carlier J., *SAT versus UNSAT*, In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, pp. 415-436, 1996.

[Dubois, et al. 1998] Dubois D., Le-Berre D., Prade H. and Sabbadin R., *Logical representation and computation of optimal decisions in a qualitative setting*, In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), pp. 588-593, 1998.

[Dubois, et al. 1999] Dubois D., Le-Berre D., Prade H. and Sabbadin R., *Using Possibilistic Logic for Modeling Qualitative Decision: ATMS-based Algorithms*, In Fundamenta Informaticae, no 37, pp. 1-30, 1999.

[Ellsberg 1961] Ellsberg D., *Risk, ambiguity and the savage axioms*, In Quarterly Journal of Economics, no 75, pp. 643-669, 1961.

[Elvang-Goransson, et al. 1993] Elvang-Goransson M., Fox J. and Krause P., *Dialectic reasoning with inconsistent information*, In Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI'93), pp. 114-121, 1993.

[Even, et al. 1976] Even S., Ital A. and Shamir A., *On the complexity of timetable and multicommodity flow problems*, In SIAM J. on Computing., no 5, pp. 691-703, 1976.

[Forbus/De-Kleer 1988] Forbus K.D. and De-Kleer J., *Focusing the ATMS*, In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'88), pp. 193-198, 1988.

[Franco 1991] Franco J., *Elimination of infrequent variables improves average case performance of satisfiability algorithms*, In SIAM J. on Computing, no 20, pp. 1119-1127, 1991.

[Fratta/Montanari 1973] Fratta L. and Montanari U., *Boolean Algebra Methods to Probabilistic Communications Networks*, In IEEE Trans. Circuit Theory, no 20, pp. 203-211, 1973.

[Freeman 1995] Freeman J.W., *Improvements to Propositional Satisfiability Search Algorithms*, PhD thesis, Departement of computer and Information science, Univ. of Pennsylvania, 1995.

[Gärdenfors 1988] Gärdenfors P., *Knowledge in Flux : Modeling the Dynamics of Epistemics States*, 1988.

[Garey/Johnson 1979] Garey M.R. and Johnson D.S., *Computers and intractability : A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

[Gelfond, et al. 1989] Gelfond M., Przymusinska H. and Przymusinski T., *On the relationship between circumscription and negation as failure*, In Artificial Intelligence, no 38, pp. 49-73, 1989.

[Gelfond/Przymusinska 1986] Gelfond M. and Przymusinska H., *Negation as failure : Careful closure procedure*, In Artificial Intelligence, no 30, pp. 273-287, 1986.

[Génisson/Siegel 1994] Génisson R. and Siegel P., *A polynomial method for sub-clauses production*, In Proceedings of the Sixth International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'94), pp. 25-34, 1994.

[Gent/Walsh 1999] Gent I.P. and Walsh T., *Beyond NP: the QSAT phase transition*, In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), pp. 648-653, 1999.

[Glover/Laguna 1993] Glover F. and Laguna M., *Tabu Search*, In Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publishing, pp. 70-141, 1993.

[Gomes, et al. 1998] Gomes C.P., Selman B. and Kautz H., *Boosting Combinatorial Search Through Randomization*, In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), pp. 431-437, 1998.

[Grégoire 1999] Grégoire E., *Gestion efficace de l'inconsistance dans les bases de connaissances stratifiées*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'99), pp. 121-127, 1999.

[Grégoire, et al. 1998] Grégoire E., Mazure B. and Saïs L., *Logically-complete local search for propositional nonmonotonic knowledge bases*, In Proceedings of the KR'98 Workshop on Computational Aspects of Nonmonotonic Reasoning, pp. 37-45, 1998.

[Greiner, et al. 1989] Greiner R., Smith B.A. and Wilkerson R.W., *A Correction to the Algorithm in Reiter's Theory of Diagnosis*, In Artificial Intelligence, vol. 41, no 1, pp. 79-88, 1989.

[Groote/Warners 1999] Groote J.F. and Warners J.P., *The propositional formula checker HeerHugo*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1999.

[Gu 1988] Gu J., *How to solve Very Large-Scale Satisfiability problem.*, Technical Report UUCS-TR-88-032, 1988.

[Gu, et al. 1997] Gu J., Purdom P.W., Franco J. and Wah B.W., *Algorithms for the satisfiability (SAT) Problem: A survey*, In Satisfiability Problem: Theory and applications, Du D.-Z., Gu J. et Pardalos P., American Mathematical Society, pp. 19-152, 1997.

[Hamadi/Merceron 1997] Hamadi Y. and Merceron D., *Reconfigurable Architectures: A new Vision for Optimization Problems*, In Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97), Lecture Notes in Computer Science N°1330, pp. 209-221, 1997.

[Harrison 1996] Harrison J., *Stålmarck's Method as a HOL Derived Rule*, In Proceedings of the 9th international Conference on Theorem Proving in Higher Order Logics (TPHOLs'96), volume 1125 of Lecture Note in Computer Science, pp. 221-234, 1996.

[Harvey/Ginsberg 1995] Harvey W.D. and Ginsberg M.L., *Limited Discrepancy Search*, In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), pp. 607-613, 1995.

[Hooker/Vinay 1995] Hooker J.N. and Vinay V., *Branching Rules for Satisfiability*, In Journal of Automated Reasoning, no 15, pp. 359-383, 1995.

[Jeannicot, et al. 1988] Jeannicot S., Oxuzoff L. et Rauzy A., *Evaluation sémantique en calcul propositionnel : une propriété de coupure pour rendre plus efficace la procédure de Davis et Putnam*, Dans La Revue d'Intelligence Artificielle, vol.. 1, n° 2, pp. 41-60, 1988.

[Jeroslow/Wang 1990] Jeroslow R.J. and Wang J., *Solving propositional satisfiability problems*, In Annals of Mathematics and Artificial Intelligence, no 1, pp. 167-188, 1990.

[Johnson/Trick 1996] Johnson D.S. et Trick M.A. (eds.), Second DIMACS Implementation Challenge : Cliques, coloring and Satisfiability, vol. 26 (http://dimacs.rutgers.edu/challenges/), American Mathematical Society , 1996.

[Kautz/Selman 1996] Kautz H. and Selman B., *Pushing the Envelope : Planning, Propositional Logic, and Stochastic Search*, In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'96), pp. 1194-1201, 1996.

[Kautz/Selman 1999] Kautz H. and Selman B., *Unifying SAT-based and Graph-based Planning*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), pp. 318-325, 1999.

[Kean/Tsiknis 1990] Kean A. and Tsiknis G., *Assumption based Reasoning and Clause Management Systems*, In Technical Report 90-9, The University of British Columbia - Departement of Computer Science, 1990.

[Kean/Tsiknis 1992] Kean A. and Tsiknis G., *Assumption-based reasoning and clause management systems*, In Computational Intelligence, vol. 1, no 8, pp. 1-24, 1992.

[Kim/Zhang 1994] Kim S. and Zhang H., *ModGen: Theorem proving by model generation*, In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94), pp. 162-167, 1994.

[Korf 1996] Korf R.E., *Improved Limited Discrepancy Search*, In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), pp. 286-291, 1996.

[Kowalsky/Kuehner 1971] Kowalsky R.A. and Kuehner D., *Linear Resolution with Selection Function*, In Artificial Intelligence, no 2, pp. 227-260, 1971.

[Lawler/Wood 1966] Lawler E.L. and Wood D.E., *Branch and Bound method : A survey*, In Operations Research, vol. 14, pp. 699-719, 1966.

[Le-Berre 1996] Le-Berre D., *Le calcul de modèles possibles basé sur la procédure de Davis et Putnam, et ses applications*, Rapport IRIT / 96-32-R, Toulouse, 1996.

[Li 1999] Li C.M., *Equivalency Reasoning to Solve a Class of Hard Sat Problems*, Non publié. Disponible sur http://www.research.att.com/~selman/challenge, 1999.

[Li/Anbulagan 1997] Li C.M. and Anbulagan, *Heuristics Based on Unit Propagation for Satisfiability Problems*, In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), pp. 366-371, 1997.

[Lifschitz 1985] Lifschitz, *Computing Circumscription*, In the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85), pp. 121-127, 1985.

[Lin/Reiter 1994] Lin F. et Reiter R., *Forget it !*, In AAAI Fall Symposium on Relevance, pp. 154-159, 1994.

[Littman 1999] Littman M.L., *Initial Experiments in Stochastic Satisfiability*, In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), pp. 667-672, 1999.

[Lobjois/Lemaître 1998] Lobjois L. and Lemaître M., *Branch and Bound Algorithm Selection by Performance Prediction*, In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), pp. 353-358, 1998.

[Locks 1987] Locks M.O., *A minimizing Algorithm for Sum of Disjoint Products*, In IEEE Trans. Reliability, no 36, pp. 445-453, 1987.

[Marques-Silva/ Sakallah 1996] Marques-Silva J.P. and Sakallah K.A., *GRASP - A New Search Algorithm for Satisfiability*, In Proceedings of the International Conference on Computer-Aided Design, pp. 220-227, 1996.

[Marquis 1999] Marquis P., *Consequence finding algorithms*, In Volume 5 : Algorithms for Uncertain and Defeasible Reasoning, Kholas J. et Moral S., Kluwer Academic, 1999.

[Massacci 1999] Massacci F., *Using Walk-SAT and Rel-SAT for Cryptographic Key Search*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), pp. 290-295, 1999.

[Mazure 1999] Mazure B., *De la Satisfiabilité à la Compilation de Bases de Connaissances Propositionnelles*, Thèse de doctorat de l'Université d'Artois, 1999.

[Mazure, et al. 1996] Mazure B., Saïs L. and Grégoire E., *Detecting logical inconsistencies*, In Proceedings of Mathematics and Artificial Intelligence Symposium (AI-MATH'96), pp. 116-121, 1996.

[Mazure, et al. 1997] Mazure B., Saïs L. and Grégoire E., *Tabu search for SAT*, In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), pp. 281-285, 1997.

[Mazure, et al. 1998] Mazure B., Saïs L. and Grégoire E., *Boosting Complete Techniques thanks to local search methods*, In Annals of Mathematics and Artificial Intelligence, no 22, pp. 319-331, 1998.

[Mazure/Marquis 1996] Mazure B. and Marquis P., *Theory Reasoning within Implicant Cover Compilation*, In Proceedings of ECAI'96 Workshop on Advances in Propositionnal Deduction, pp. 65-69, 1996.

[McAllester, et al. 1997] McAllester D., Selman B. and Kautz H., *Evidence for Invariants in Local Search*, In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), pp. 321-326, 1997.

[McCarthy 1986] McCarthy J., *Applications of circumscription to formalizing common-sense knowledge*, In Artificial Intelligence, vol. 28, pp. 89-116, 1986.

[McCune 1994] McCune W., *A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problem*, Technical Memorandum ANL/MCS-TM-194, 1994.

[Minker 1982] Minker J., *On indefinite databases and the closed world assumption*, In Proceedings of the Sixth Conference on Automated DEduction (CADE-82), pp. 292-308, 1982.

[Monien/Speckenmeyer 1985] Monien B. and Speckenmeyer E., *Solving Satisfiability in less than 2n steps*, In Discrete Applied Mathematics, no 10, pp. 287-295, 1985.

[Ngair/Provan 1993] Ngair T.H. and Provan G., *A lattice-theoretic analysis of ATMS problem solving*, In Proceedings of ECSQARU'93, pp. 284-289, 1993.

[Niemelä 1996] Niemelä I., *A tableau calculus for minimal model reasoning*, In Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods, Springer-Verlag, pp. 278-294, 1996.

[Niemelä 1996b] Niemelä I., *Implementing Circumscription Using a Tableau Method*, In Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96), pp. 80-84, 1996.

[Oxusoff/Rauzy 1989] Oxusoff L. et Rauzy A., *L'évaluation sémantique en calcul propositionnel*, Thèse de doctorat de l'Université de Provence, GIA-Luminy, 1989.

[Palmade 1992] Palmade O., *Etude et implémentation de nouveaux mécanismes d'inférence en logique propositionnelle. Application aux ATMS*, Thèse de doctorat de l'Université Paul Sabatier, Toulouse, 1992.

[Palopoli, et al. 1999] Palopoli L., Pirri F. and Pizzuti C., *Algorithms for Selective Enumeration of Prime Implicants*, In Artificial Intelligence, vol.. 2, n° 111, pp. 41-72, 1999.

[Parkes/Walser 1996] Parkes A.J. and Walser J.P., *Tuning Local Search for Satisfiability Testing*, In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), pp. 356-362, 1996.

[Pearl 1988] Pearl J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.

[Prawitz 1965] Prawitz D., *Natural Deduction: A Proof Theoretical Study*, Almquist & Wiksell, 1965.

[Provan 1989] Provan G.M., *An analysis of ATMS-based techniques for computing Dempster-Shafer belief functions*, In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89), pp. 1115-1120, 1989.

[Provan 1996] Provan G.M., *A formal analysis of cost-functions for ATMS focusing*, In Proceedings of the Mathematics and Artificial Intelligence Symposium (AI-MATH'96), pp. 130-133, 1996.

[Raiman/de-Kleer 1992] Raiman O. and de-Kleer J., *A Minimality Maintenance System*, In Proceedings of the Third International Conference on Principles on Knowledge Representation and Reasoning (KR'92), pp. 532-538, 1992.

[Rauzy, et al. 1999] Rauzy A., Saïs L. et Brisoux L., *Calcul propositionnel : vers une extension du formalisme*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99), pp. 189-198, 1999.

[Reiter 1978] Reiter R., *On closed world data bases*, In Logic and Data Bases, Gallaire H. et Minker J., Plenum, pp. 119-140, 1978.

[Reiter 1980] Reiter R., *A logic for default reasoning*, In Artificial Intelligence, no 13, pp. 81-132, 1980.

[Reiter 1987] Reiter R., *A theory of diagnosis from first principles*, In Artificial Intelligence, no 32, pp. 57-95, 1987.

[Reiter/de-Kleer 1987] Reiter R. and de-Kleer J., *Foundations of Assumption-based truth maintenance systems : preliminary report*, In Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI'87), pp. 183-188, 1987.

[Rintanen 1999] Rintanen J., *Improvement to the Evaluation of Quantified Boolean Formulae*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 1999.

[Robinson 1965] Robinson J.A., *A machine-oriented logic based on the resolution principle*, In Journal of the ACM, no 12, pp. 23-41, 1965.

[Robinson 1983] Robinson J.A., *Automatic deduction with hyperresolution*, In Automation of Reasoning-Classical Papers on Computational Logic, vol. 1 and 2, Springer, 1983.

[Rymon 1992] Rymon R., *Search through Systematic Set Enumeration*, In Proceedings of the Third International Conference on Principles on Knowledge Representation and Reasoning (KR'92), pp. 539-550, 1992.

[Sabbadin 1998] Sabbadin R., *Une approche ordinale de la décision dans l'incertain : axiomatisation, représentation logique et application à la décision séquentielle*, Thèse de doctorat de l'Université Paul Sabatier, 1998.

[Savage 1954] Savage L.J., *The Foundations of Statistics*, Wiley and Sons, 1954.

[Schrag 1996] Schrag R., *Compilation of critically Constrained Knowledge Bases*, In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), pp. 510-515, 1996.

[Schrag/Crawford 1996] Schrag R.C. and Crawford J.M., *Implicates and Prime Implicates in Random 3SAT*, In Artificial Intelligence, no 81, pp. 199-222, 1996.

[Selman, et al. 1992] Selman B., Levesque H. and Mitchel D., *A new method for solving hard satisfiability problems*, In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), pp. 459-465, 1992.

[Selman, et al. 1994] Selman B., Kautz H.A. and Cohen B., *Noise strategies for improving local search*, In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94), pp. 337-343, 1994.

[Selman, et al. 1997] Selman B., Kautz H.A. and McAllester D.A., *Ten Challenges in Propositionnal Reasoning and Search*, In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), pp. 50-54, 1997.

[Selman/Kautz 1993] Selman B. and Kautz H., *Domain-independant Extensions to GSAT : Solving Large Structured Variables*, In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (JCAI'93), pp. 290-295, 1993.

[Shafer 1976] Shafer G., *A Mathematical Theory of Evidence*, In Princeton University Press, 1976.

[Sheeran/Stålmark 1998] Sheeran M. and Stålmark G., *A tutorial on Stålmark's proof procedure for propositional logic*, In Volume 152 of Lecture Notes in Computer Science, Springer Verlag, pp. 82-99, 1998.

[Shoham 1987] Shoham Y., *Non-monotonic logics: Meaning and utility*, In Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI'87), pp. 388-393, 1987.

[Shoham 1987b] Shoham Y., *A semantical approach to non-monotonic logics*, In Readings in Non-monotonic Reasoning, Ginsberg M.L. ed., Morgan Kaufmann, 1987.

[Siegel 1987] Siegel P., *Représentation et utilisation de la connaissance en calcul propositionnel*, Thèse d'état, Université de Provence, GIA-Luminy, Marseille, 1987.

[Smets 1993] Smets P., *Probability of deductibility and Belief Functions*, In ECSQARU (LNCS 747), 1993.

[Smullyan 1968] Smullyan R.M., *First-Order Logic*, Springer-Verlag, 1968.

[Stålmarck 1989] Stålmarck G., *A system for determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are generated from a formula*, European Patent N° 0403 454 (1995), US Patent N° 5 276 897, Swedish Patent N° 467 076 (1989), 1989.

[Tatar 1994] Tatar M.M., *Combining the Lazy Label Evaluation with Focusing Techniques in an ATMS*, In Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94), pp. 160-164, 1994.

[Tayrac 1990] Tayrac P., *Etude de nouvelles stratégies de résolutions. Application à l'ATMS.*, Thèse de doctorat de l'Université Paul Sabatier, 1990.

[Verfaillie/Lobjois 1999] Verfaillie G. et Lobjois L., *Problèmes incohérents : expliquer l'incohérence, restaurer la cohérence*, Dans Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC'99), pp. 111-120, 1999.

[Wang 1997] Wang J., *Branching rules for propositionnal satisfiability test*, In Satisfiability Problem: Theory and applications, Du D.-Z., Gu J. et Pardalos P., American Mathematical Society, pp. 351-364, 1997.

[Warners/Van-Maaren 1998] Warners J.P. and Van-Maaren H., *A two phase algorithm for solving a class of hard satisfiability problems*, In Operations Research letters, no 23, pp. 81-88, 1998.

[Wolfman/Weld 1999] Wolfman S.A. and Weld D.S., *The LPSAT Engine and its Application to Resource Planning*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), pp. 310-316, 1999.

[Yacoub 1997] Yacoub R., *Stratégies de résolution pour focaliser un système de maintien de la cohérence*, Thèse de doctorat de l'Université de Paris 6, 1997.

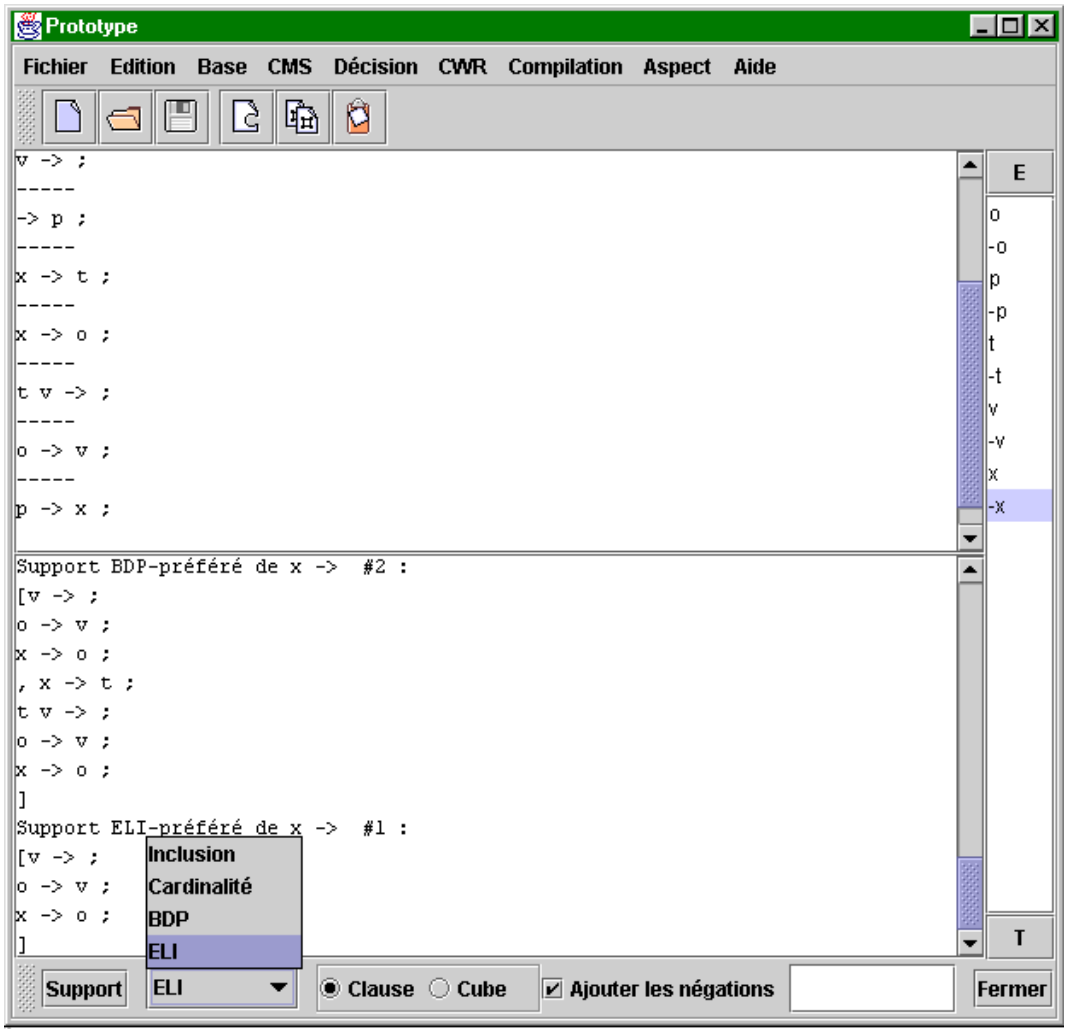
[Yahya/Henschen 1985] Yahya A. and Henschen L.J., *Deduction in non-horn databases*, In Journal of Automated Reasoning, no 1, pp. 141-160, 1985.

[Yokoo, et al. 1996] Yokoo M., Suyama T. and Sawada H., *Solving Satisfiability Problems Using Field Programmable Gate Arrays: First Results*, In Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP'96), pp. 495-509, 1996.

[Zhang/Stickel 1994] Zhang H. and Stickel M., *Implementing the Davis-Putnam algorithm by Tries*, In Technical Report, Dept. of Computer Science, University of Iowa, 1994.

[Zhang/Stickel 1996] Zhang H. and Stickel M.E., *An efficient Algorithm for Unit Propagation*, In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), 1996.

Annexe I - Utilisation du prototype



<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Lancement du programme

```
java Prototype [fichier.base]
```

Lance l'interface graphique du prototype. Si un fichier de base est donné, il est automatiquement chargé.

```
java Prototype -e fichier.XXX
```

Lance l'interface graphique du prototype en le forçant à charger fichier.XXX. Pratique lorsque l'on désire travailler "à la main" sur une base qui se trouve dans un fichier de script.

```
java Prototype [-a Algo] fichier.cnf fichier2.cnf ...
```

Utilise l'outil comme prouveur SAT. Si la paramètre -a est utilisé, alors Algo est utilisé pour résoudre le problème. Algo doit bien sur être le nom d'un algorithme de satisfaction, complet ou incomplet. Si Algo n'existe pas, DavPut est utilisé par défaut.

```
java Prototype -t fichier.cnf
```

Pour transformer un fichier au format DIMACS en un fichier au format HeerHugo. Le résultat s'affiche sur l'entrée standard. Pas très performant car on charge d'abord la base, et on demande ensuite à l'objet de se représenter dans le format HeerHugo.

```
java Prototype -t2 fichier.cnf
```

Même chose que précédemment mais pour obtenir une base de clauses sous forme implicative, c'est à dire notre représentation. Même remarque sur les performances.

```
java Prototype fichier.scp
```

Exécute le script fichier.scp.

java Prototype fichier.test

Exécute une série de tests.

```
java Prototype -
```

Lit les informations sur l'entrée standard (un script ou une série de test).
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Format des bases de clauses (CNF)

Les bases de clauses utilisées par notre prototype utilisent la syntaxe suivante :

```
(%CLAUSE%)+
```

avec :

```
%CLAUSE% = ((%Var%|%var%)* "->" ("%Var%|%var%)* ";"
%var% une chaine de caractères commençant par une minuscule.
%Var% une chaine de caractères commençant par une majuscule.
```

Par exemple :

```
Toto -> titi tutu ;
tata -> Toto tux ;
```

est une base de clauses reconnue par notre prototype.

Certaines variables d'une base de clauses ont quelquefois besoins d'être différenciées (les hypothèses dans le cadre d'un ATMS par exemple).Un identificateur de variable *commençant par une majuscule* indiquera ces variables. Dans notre exemple, les variables sont divisées en {Toto} et {titi, tutu, tata, tux}.

Les commentaires sont de la forme :

```
// commentaire sur une ligne
%% un autre commentaire d'une ligne
/*
 ...commentaire sur plusieurs lignes
*/
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
```

Bases de clauses stratifiées

Dans ces bases de clauses, on distingue deux types de formules : les connaissances et les croyances. Les premières sont certaines et ne peuvent pas être remises en cause. Les secondes peuvent être inconsistantes avec les connaissances, voire entre elles.

Le format de ces bases dans le prototype est le suivant :

```
(%CLAUSE%)*  // connaissances
("-----"
(%CLAUSE%)*  // croyances
)*
```

Par exemple :

```
-> b ;   // connaissance
-----    // ici débutent les croyances
b -> c ; // strate 1
-----
c -> ;   // strate 2
```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Bases de clauses ordonnées

Ce sont des bases de clauses stratifiées particulières, car chaque strate est étiquetée par une chaine de caractères.
Le format de ces bases dans le prototype est le suivant :

```
("=====" %ID%  // un identificateur de l'échelle
(%CLAUSE%)*)+  // toute la base est sous forme clausale
"=====" %ID0% // on termine toujours l'échelle par sa valeur minimale
```

Par exemple :

```
===== TOP
-> b ;
===== TUX
b -> c ;
===== TITI
c -> ;
===== BOTTOM
```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Données numériques

Il est possible d'associer des poids, probabilités, etc. à des symboles hypothèses. Dans tous les cas, les valeurs sont exprimées par des flottants, par exemple : 3. , 1.27, 0.35, .67.

Lorsqu'il s'agit de probabilités, il suffit de fixer la valeur d'un des littéraux hypothèses. La valeur de sa négation est déduite.
La syntaxe est la suivante :

```
// la base de clause se trouve ici
"_EXTRA"
("_MASS" %FLOAT% (["-"]%Var%)+)+
"_EXTRA"
```

Exemple :

```
// la base contient les variables hypothèses A1, A2, A3, A4
_EXTRA
_MASS .5 A1 A2 -A3
_MASS 3. A4
_EXTRA
```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Base de préférences

Ces bases sont seulement utilisées dans le cadre de la décision.
Elles ont presque la même syntaxe que les bases ordonnées, mais :

- chaque strate ne peut contenir qu'une formule
- une formule est soit une clause soit un cube.

Ce qui se traduit par :

```
("=====" ID  // un identificateur d'échelle
%CLAUSE% | %CUBE% )+  // chaque préférence est soit un cube soit une clause.
"=====" ID0
```

avec

```
%CUBE% = "[" (["-"](%Var%|%var%))+"]"
```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Format des scripts

Les scripts sont des fichiers contenant des commandes exécutables par le prototype. Elles suivent le plus souvent la description d'une base de connaissance.
Les scripts utilisés par notre prototype utilisent la syntaxe suivante :

```
(
%Une base de clauses%
// pour tester un ou plusieurs algorithmes sur une base particulière
("_TEST" %ALGO%
          // pour passer des paramètres à l'algorithme.
          // Attention, si l'algorithme n'accepte pas les paramètres,
          // retourne un message d'erreur et s'arrête
          [%ENTIER% | %MODGEN% %ENTIER%]
          ["_RESTRICTION" ("NegativeLiteral" | "PositiveLiteral")])+
|
Requêtes ATMS/CMS
|
Requêtes Raisonnement en monde clos.
)
// Dans ce cadre les bases de clauses ont une forme particulière.
Requêtes Décision dans l'incertain
```

Les commentaires sont de la forme :

```
// commentaire sur une ligne
%% autre commentaire sur une ligne
/*
 ...commentaire sur plusieurs lignes
*/
```

On peut afficher une chaîne de caractères en utilisant la commande `"_ECHO"` et la base de clauses lue à l'aide de la commande `"_THIS"`.
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Commandes autour de l'ATMS/CMS (Clause Management System)

```
"_Goal"
(
"_INFOS"    // pour passer en mode informations
"_NOINFOS"  // pour ne pas afficher d'informations
"_NOGOODS"  // pour afficher les nogoods de la base de clauses
"_IMPLCVR"  // pour calculer une couverture d'impliquants de la base
["_CARD"|"_1CARD"] (%CLAUSE%|%CUBE%) // pour calculer le label d'une formule
// CARD : tous les environnements du label minimaux pour la cardinalité
// 1CARD : 1 seul des environnements du label minimal pour la cardinalité
// RIEN : label classique
// %CLAUSE% le label de la clause (ex: a b -> c ;)
// %CUBE% la label du cube (ex: [-a -b c])
)+
```

Requêtes en raisonnement non monotone (modèles [P-]minimaux)

```
"_CWR"
// Retourne les littéraux ou ensembles de littéraux Free For Negation selon les
// différentes politiques de CWR
("_CWA" | "_GCWA" | "_EGCWA" | "_CCWA" | "_ECWA" |
// Retourne les modèles minimaux de Herbrand de la base de clauses
"_MODMIN" |
```

```
// Retourne les modèles P-minimaux de Lifschitz de la base de clauses
"_MODPMIN" |
// Retourne les modèles <P;Q;Z>-minimaux de Lifchitz. Les variables de P
// (à minimiser), sont ceux qui commencent par une majuscule. Les variables
// de Q (dont la valeur de vérité doit être identique pour que deux modèles
// soient comparables) sont données avant le mot clef.
(%var%)* "_PQZMIN"
// Pour savoir si un Cube ou une clause est minimalement
// conséquence logique de la base
("_MINENTAILS" | "_MINPENTAILS") (%CLAUSE% | %CUBE%)
)+
```

Exemple :

Si le script suivant se trouve dans CWR.scp

```
r -> A1 h ;
q -> A2 d ;
h d -> ;
h -> p ;
d -> p ;
-> r ;
-> q
_CWR
_CWA
_GCWA
_EGCWA
_CCWA
_ECWA
_MODMIN
_MODPMIN
r q _PQZMIN
_MINENTAILS  h d ;
_MINPENTAILS [ ->h d]
```

Son exécution par :

```
java Prototype CWR.scp
```

donne le résultat suivant :

```
Traitement du fichier : CWR.scp
CWA : {-d, -p, -h, -A1, -A2}
GCWA : {}
EGCWA : [{-p, -A1, -A2}, {-A1, -h}, {-h, -d}, {-A2, -d}]
CCWA : {}
ECWA : [{-A2, -A1}]
Modeles Minimaux : [{A2, A1, r, q, -d, -p, -h}, {h, A2, p, r, q, -d, -A1}, {A1,p, d, r, q, -h,
-A2}]
Modeles P-Minimaux : [{A1, -A2}, {A2, -A1}]
Modeles <P;Q;Z-Minimaux : [{A1, r, q, -A2}, {A2, r, q, -A1}]
MinEntails -> h d  : false
MinPEntails (-h d) : false
fin du traitement du fichier : CWR.scp
```

Décision dans l'incertain

Dans ce cadre, les informations sont situées dans deux bases de connaissances : les connaissances et les préférences (buts). De plus, il existe une échelle unique sur ces bases. Les connaissances situées en haut de l'échelle sont les plus certaines, les préférences situées en haut de l'échelle les plus prioritaires.
Trouver une solution d'un point de vue optimiste est : trouver une solution cohérente avec les connaissances et les buts à atteindre.
Trouver une solution d'un point de vue pessimiste est : trouver une solution satisfaisant tous les buts.

Syntaxe :

```
"_KNOWLEDGE"  // la base de connaissance
%BASE_DE_CLAUSES_ORDONNEE%
"_PREFERENCES"  // les préférences
%BASE_DE_CLAUSES_DE_PREFERENCE%
["_OPTIMISTIC_DECISION"]
["_PESSIMISTIC_DECISION"]
```

Exemple :

Si le fichier Robots.scp contient le script suivant :

```
_KNOWLEDGE
===== AA
Tuer_homme - nuire_humain ;
Tuer_homme - obeir_homme ;
- No_op Tuer_homme ;
se_proteger No_op - ;
===== BB
No_op Tuer_homme - ;
No_op nuire_humain - ;
===== ZZ
===== QQ
_PREFERENCES
===== AA
nuire_humain - ;
===== BB
[obeir_homme]
===== ZZ
- se_proteger ;
===== QQ
_OPTIMISTIC_DECISION
_PESSIMISTIC_DECISION
```

Alors le résultat de la commande

```
java Prototype Robots.csp
```

est le suivant :

```
Traitement du fichier : Robots.scp
 Utilité optimiste : BB
 Solutions : [(No_op)]
 Utilité pessimiste : ZZ
 Solutions : [(No_op)]
fin du traitement du fichier : Robots.scp
```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Format des fichiers de tests

Les fichiers de tests utilisés par notre prototype permettent de tester un algorithme sur des bases 3-SAT générées aléatoirement selon différents critères. Ils utilisent la syntaxe suivante :

```
// Nom de l'algorithme à tester et ses paramètres.
"_TEST" %NOM_ALGO% [%ENTIER% | %MODELGEN% %ENTIER%]
// Nombre de bases ayant ces caractéristiques à tester
"_PRECISION" %ENTIER%
// Permet de sélectionner les littéraux à restreindre parmi le sous langage.
// Par défaut, ce sont les littéraux négatifs qui sont sélectionnés.
["_RESTRICTION" ("PositiveLiteral" | "NegativeLiteral")]
// Indique le nombre de variables.
"_VAR" %ENTIER% ["_TO" %ENTIER% "_STEP" %ENTIER%]
// Indique la proportion de variables dans le sous langage.
// choisir SUB entre 0. et 1. (.4 par exemple)
["_SUB" %FLOAT% ["_TO" %FLOAT "_STEP" %FLOAT%]]
// On indique enfin le nombre de clauses,
// soit par le ratio (CLAUSES/VAR)
("_RATIO" %FLOAT%  ["_TO" %FLOAT "_STEP" %FLOAT%]
// soit en donnant directement le nombre de clauses
| "_CLAUSES" %ENTIER% ["_TO" %ENTIER% "_STEP" %ENTIER%]
Les commentaires sont de la forme :
// commentaire sur une ligne
%% autre commentaire sur une ligne
/*
 ...commentaire sur plusieurs lignes
*/
```

Lorsque le script commence, les caractéristiques de la machine sont affichées, ainsi que les principaux paramètres.
Chaque algorithme affiche ses propres données : on retrouve le plus souvent CU (nombre de propagation de clauses unitaires), LP (propagation de littéraux purs), NODES (nœuds de branchement ou nombre d'appels récursifs à l'algorithme), #SOL (nombre de solutions) etc.

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Algorithmes disponibles

{PRIVATE}DistanceSAT, DistanceSATPL, DistanceSATPLOrd, LDS	**Satisfaction**	**Impliquants/Impliqués**
Complets	SimpleDavPut, DavPut, DavPutPreProcess	MPL, MPLNoHeur, MPLNoNPLP, PIP, PIPQNTP, ILDSMPL, Niemela, Castell, DavPutModel
Incomplets	GSAT, WSAT, Novelty	IncompleteMPL, AnytimeMPL, ModelMinimisation,ILDS

{PRIVATE}Simple-DavPut	Algorithme de type Davis et Putnam avec une Heuristique de type MOMS. Pas d'heuristique de type UP. Développe plus de nœuds que la version suivante mais n'est pas beaucoup moins rapide.
DavPutModel	Le même algorithme mais renvoie un impliquant de la base s'il y en a un ou null sinon.
DavPut	Algorithme de type Davis et Putnam avec une Heuristique de type UP (choix de la variable satisfaisant le plus de clauses parmi les variables apparaissant au moins 3 fois dans des clauses positives et ou négatives) plus une heuristique de type MOMS. Pour l'instant, le nombre de nœuds de l'arbre de recherche est considérablement réduit mais ce n'est pas vrai pour le temps d'exécution. Cela changera peut être en utilisant les collections fournies par le JDK1.2
DavPutPreProcess	Le même qu précédemment avec en prétraitement la production de résolvantes à partir des singletons de la base (comme C-SAT).
MPL	L'algorithme de calcul des impliquants P-restreints premiers.
MPLNoHeur	Le même que précédemment mais sans tenir compte des clauses de minimisation dans l'heuristique. Permet de résoudre des instances où il y a beaucoup plus de solutions que de clauses dans la base de départ (cela perturbe l'heuristique de l'algorithme précédent).
MPLNoNPLP	Algorithme de calcul des impliquants P-restreints premiers sans propagation des littéraux purs de non-P (propagation des littéraux purs dont le littéral complémentaire est dans P). Selon l'instance à traiter, cette méthode peut s'avérer plus rapide que MPL.
IncompleteMPL	MPL avec arrêt au bout d'un nombre de solutions fixé (par défaut, 10) paramètre optionnel : %ENTIER%
AnytimeMPL	MPL avec arrêt au bout d'un temps fixé (par défaut, 1 heure) paramètre optionnel : %ENTIER% Attention, il s'agit de millisecondes.
ModelMinimisation	Calcul d'impliquants P-restreints premiers par minimisation d'un modèle fourni par un algorithme de recherche de modèle (par defaut DavPutModel). Arrêt au bout d'un nombre de solutions donné (par défaut, 10). paramètres optionnels : %ENTIER% %MODELGEN% %ENTIER% Attention : %MODELGEN% est soit DavPutModel soit une méthode de recherche locale
ILDS	Algorithme "Improved Limited Discrepancy Search" de Korf appliqué au cadre de la recherche d'un impliquant P-restreint minimal pour la cardinalité.
ILDSMPL	Le même que précédemment mais complet : il donne tous les impliquants P-restreints minimaux pour la cardinalité.
PIP	Calcul de P-impliqués premiers (P étant un ensemble consistant de littéraux). Effectue deux appels à MPL.
PIPQNTP	Calcul de P-impliquants premiers. Effectue un filtrage des clauses avant d'appeler MPL.
Castell	Calcul de P-impliquants d'une base de clauses en utilisant la notion de littéral nécessaire pour effectuer le test de primarité. Cet l'algorithme est plus efficace que le précédent (mais la base de clauses doit être pure car on ne filtre pas la base de clauses dans l'algorithme).
Niemela	Calcul de modèles minimaux basé sur les travaux de Niemela (méthode des tableaux analytiques revue à la sauce DP)
GSAT	Le plus connu des algorithmes de recherche locale. Cette version contient un "Random Walk" de 0,5.
WSAT	Une variante de GSAT. La prochaine variable a flipper est choisie parmi une clause falsifiée

	choisie aléatoirement. Contient un "Noise" de 0,5.
Novelty	Une variante de WSAT. Critère d'antériorité dans le choix des variables de la clause falsifiée. Contient un "Noise" de 0,6.
DistanceSAT	Implantation personnelle de l'algorithme présenté par Olivier Bailleux et Pierre Marquis. N'utilise pas (encore) l'heuristique LASSO présentée par les deux auteurs. paramètre : %ENTIER% la distance par rapport à l'interprétation (partielle) de référence. L'interprétation partielle est construite sur les variables hypothèses avec la négation des littéraux indiqués car les autres algorithmes utilisent des restrictions et non des références.
DistanceSATPL	Le même que précédemment avec propagation de certains littéraux purs.
DistanceSATPLOrd	Le même que précédemment avec ajout de l'ordonnancement des littéraux lorsqu'une variable de l'interprétation de référence est choisie
LDS	L'algorithme LDS modifié pour résoudre la problème DistanceSAT (on fait un seul appel à LDS_Probe). paramètre : %ENTIER% la distance maximale autorisée avec l'interprétation (partielle) de référence.

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Présentation de l'interface

L'interface se compose :

- de la barre de menu, classique,
- d'une barre d'outils,
- d'une fenêtre d'édition,
- et d'une fenêtre de résultats.

On retrouve dans le menu les opérations classiques de manipulations de fichiers et d'édition, plus un menu aspect qui permet de changer le « look » de l'application (attention, les « look » Mac et Windows ne sont disponibles que sur ces systèmes), et un menu d'aide.
Chaque autre menu correspond à des taches bien particulières :

- travail d'édition et de test sur les bases de clauses
- le raisonnement autour de l'ATMS/CMS
- la théorie de la décision dans l'incertain
- le raisonnement en monde clos
- la compilation de solutions

.

Menu Base

Ces actions agissent sur la sélection courante ou sur le contenu de la fenêtre d'édition si rien n'est sélectionné.

{PRIVATE}Commenter	commenter les lignes (ajout de `"// "`)
Décommenter	enlève les commentaires (retrait de `"// "`)
Ajouter hypothèses	ajoute un symbole hypothèse `"AssX"` devant chaque ligne non vide
Enlever hypothèses	enlève les symboles `"AssX"`
Traduc. règles planif.	utilisé dans l'équipe pour transformer des règles de type `ACT : p1 p2 -> +a1 +a2 -r1 -r2` en autant de clauses, ACT étant le nom de l'action, pX les préconditions, aX les ajouts et rX les retraits.
Tester consistance	calcule et affiche si la base est consistante ou inconsistante (algorithme DavPut)
Couverture d'impliquants	calcul d'une couverture d'impliquants de la base à l'aide d'un algorithme de Davis et Putnam sans propagation des littéraux purs ni arrêt au premier impliquant.
impliqués P-restreints	Calcul des impliquants P-restreints premiers de la base (algorithme MPL).

P-impliquants	Calcul des P-impliquants premiers de la base (algorithme PIPQNTP)
P-impliqués	Calcul des P-impliqués premiers de la base (algorithme PIP)

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Menu CMS

Différentes versions de (Assumption) Clause Management System.Certaines de ces actions sont disponibles en ligne de commande.
Les différentes versions du CMS utilisent la même interface. Sur la droite, l'ensemble des littéraux de la base. Par défaut, seuls les littéraux non hypothèses positifs sont affichés. Les autres littéraux peuvent être affichés en cochant les cases "ajouter hypothèses" et "ajouter négations". La sélection des littéraux se fait de manière analogue au système utilisé (sur PC, utilisation des touches Shift et Ctrl pour faire des sélections multiples). La sélection correspond soit à une requête, soit à un filtre, selon la position du bouton radio associé. La requête est interprétée comme un clause ou un cube selon la position du bouton radio correspondant (par défaut, on considère des clauses).
Lorsqu'un filtre est défini, seuls les littéraux du filtres apparaissent dans le résultat.

{PRIVATE}Normal	On retrouve le CMS/ACMS classique avec calcul des nogoods, labels, interprétations, contexte, extensions ...). A utiliser avec des bases de clauses contenant des littéraux hypothèses. Certaines de ses fonctionnalités sont implantées pour effectuer des expérimentations dans l'équipe.
Avec Belief	Calcul du degré de croyance d'une formule. Nécessite une base de clauses **contenant des littéraux hypothèses** plus des probabilités sur chacun des littéraux hypothèses. Attention : il ne s'agit pas de calculer les environnements les plus probables du label d'une donnée (cf. suivant).
Branch & Bound	Calcul des environnements les plus probables. Comme son nom l'indique, c'est un algorithme de Branch & Bound qui est utilisé. Nécessite une base de clauses **contenant des littéraux hypothèses** plus une probabilité pour chaque littéral hypothèses.
Argumentation	Utilisation du CMS pour raisonner sur des clauses et non sur des littéraux grâce à un encodage de la base transparent pour l'utilisateur. Possibilité de choisir parmi 4 préférences pour le calcul des supports préférés (Inclusion, Cardinalité, WI, ELI). Nécessite une base de clauses **classique ne contenant pas de symbole hypothèse** ou une base de clauses stratifiée (obligatoire pour utiliser WI et ELI).
Sous-bases maximales consistantes	Utilise la notion d'interprétation de l'ATMS et le codage pour calculer les sous bases maximales d'une base de clauses **ne contenant pas de symbole hypothèse.**
Sous-bases minimales inconsistantes	Même chose à partir des nogoods de l'ATMS.

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Menu Décision

Ces calculs peuvent s'effectuer en ligne de commande.

{PRIVATE}Charger connaissances	Lit une base de clauses représentant des croyances plus ou moins certaines. Nécessite une base de clauses ordonnée.
Charger préférences	Lit une base de préférences contenant des buts plus ou moins prioritaires. Attention : il faut utiliser les mêmes identificateurs d'échelle entre les deux bases.
Afficher connaissances	Affiche la base de connaissances
Afficher préférences	Affiche la base de préférences
Afficher échelle	Affiche l'échelle de la base de connaissance
Utilité optimiste	Calcule une décision ayant une utilité optimale selon le point de vue optimiste (décision consistante avec les buts)

Utilité pessimiste	Calcule une décision ayant une utilité optimale selon le point de vue pessimiste (décision satisfaisant les buts)

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Menu CWR

Toutes ces actions, et d'autres, sont disponibles en ligne de commande.
Calcul des formules "free for negation" de différentes politiques du "Closed World Reasoning". Il s'agit de littéraux négatifs ou de conjonctions de littéraux négatifs.

{PRIVATE}Herbrand	Calcul des modèles minimaux de Herbrand. Pas de différence entre les littéraux hypothèses et non hypothèses.
CWA	Closed World Assumption. Littéraux négatifs de l'union des modèles minimaux de Herbrand.
GCWA	Generalized CWA. Littéraux négatifs de l'intersection des modèles minimaux de Herbrand.
EGCWA	Extended GCWA. "hitting set minimal" sur les littéraux négatifs des modèles minimaux de Herbrand.
CCWA	Careful Closed World Assumption. Comme GCWA sur les variables "hypothèses" uniquement.
ECWA	Extended CWA. Comme EGCWA sur les variables "hypothèses" uniquement.

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

Menu compilation

Lorsque l'on cherche les P-impliqués premiers d'une base, nous calculons d'abord une formule ne contenant que les littéraux de P, à partir de laquelle nous extrayons les P-impliqués premiers (ici P est l'ensemble des symboles hypothèses négatifs). Nous appellons le calcul de la première formule "Compilation" car le plus souvent un nombre important de P-impliqués sont "cachés" dans quelques clauses.

{PRIVATE**}**Compiler	Calcule la première formule et l'ajoute à la fenêtre d'édition. Nécessite une base de clauses contenant des littéraux hypothèses.
Inclusion	Calcul des P-impliqués premiers. Base de clauses classique avec littéraux hypothèses.
Cardinalité	Calcul des P-impliqués CARD-préférés. Base de clauses classique avec littéraux hypothèses.
WI	Calcul des P-impliqués WI-préférés. Nécessite une base de clauses stratifiée.
ELI	Calcul des P-impliqués ELI-préférés. Nécessite une base de clauses stratifiée.
Filtrage	Filtrage des solutions parmi les P-impliqués premiers d'une formule. Le filtre est exprimé sous la forme d'une base de clauses.

Annexe II - Sources des algorithmes

Nous ajoutons ici une implantation des algorithmes présentés dans le document en Java. Chaque algorithme est un objet Java. Les méthodes d'un objet qui ne sont pas présentées n'apportent aucune information relative à l'algorithme : par exemple, pour être intégré dans notre outil, chaque algorithme doit avoir des méthodes permettant de lancer un calcul selon différents paramètres. Elles n'apportent aucune aide à la compréhension de l'algorithme. Elles sont donc omises.

SimpleDavPut

```
/**
 * Une implantation personnelle de la fameuse procédure de Davis et
 * Putnam. L'heuristique utilisée est de type MOMS.
 */
public class SimpleDavPut implements SatisfiabilityAlgorithm {

    /**
     * Retourne le meilleur littéral de la base pour le prochain branchement.
     * @param base la base de clauses.
     * @return un littéral de la base.
     */
    protected Literal chooseLiteral(Base base) {
       Literal lmax=null;
       Symbol s;
       long max = 0,poids;

       Enumeration e = base.getSymbols().elements();
       while (e.hasMoreElements()) {
           s = (Symbol)e.nextElement();
           poids = s.getPoidsMax();
           if (max<poids) {
              max = poids;
              lmax = s.getBestLiteral();
           }
       } // end of while ()
       return lmax;
    }

    /**
     * Propage les clauses unitaires de la base.
     * @param une base de clauses
     * @return <code>true</code> ssi la base est consistante.
     */
    protected boolean propagateUC(Base base) {
       /* tant qu'il y a des clauses unitaires, la base ne peut pas être
          satisfaite. On teste seulement si elle est falsifiée. */
       if (base.isFalsified()) {
           return false;
       } // end of if ()
       if (base.hasUnitClauses()) {
           Literal lit = (Literal)base.getUnitClauses().pop();
           lit.satisfy();
           boolean res = propagateUC(base);
           lit.undefine();
           return res;
       } // end of if ()
       /* Si il n'y a plus de clauses unitaires, on propage les littéraux
          purs */
       return propagatePL(base);
    }

    /**
```

```
         * Propage les littéraux purs de la base.
         * @param base une base de clauses.
         * @return <code>true</code> ssi la base est consistante.
         */
        protected boolean propagatePL(Base base) {
           /* on ne peut pas falsifier la base, on teste seulement si celle-ci
              est satisfaite. */
           if (base.isSatisfied()) {
               return true;
           } // end of if ()
           if (base.hasPureLiterals()) {
               Literal lit = (Literal)base.getPureLiterals().pop();
               lit.satisfy();
               boolean res = propagatePL(base);
               lit.undefine();
               return res;
           } // end of if ()
           /* il ne reste ni clauses unitaires ni littéraux purs, on effectue un
              branchement.*/
           return branch(base);
        }

        /**
         * Effectue un branchement en utilisant une heuristique de type MOMS.
         * @param base une base de clauses
         * @return <code>true</code> ssi la base est consistante.
         */
        protected boolean branch(Base base) {
           return branch(base,chooseLiteral(base));
        }

        /**
         * Effectue un branchement sur un littéral donné.
         * @param base une base de clauses
         * @param lit le littéral de branchement.
         * @return <code>true</code> ssi la base est consistante.
         */
        protected boolean branch(Base base, Literal lit) {
           if (lit!=null) {
               base.getUnitClauses().push(lit);
               if (!propagateUC(base)) {
                  base.getUnitClauses().push(lit.negation());
                  return propagateUC(base);
               } // end of if ()
           } // end of if ()
           return true;
        }

        /**
         * La procedure de Davis et Putnam elle même.
         * @param base une base de clauses
         * @return <code>true</code> si la base est consistante, false sinon
         */
        public boolean run(Base base) {
           return propagateUC(base);
        }
    }
```

MPL

```
    /**
     * Algorithme décrit dans [CCC96] a une variante près : ici, on ne
     * retourne que les ensembles minimaux, alors que dans [CCC96] on
     * retourne des P-modèles.
     */
    public class MPL implements EnumerativeAlgorithm {
      protected SatisfiabilityAlgorithm sat;

        public MPL() {
           sat = new SimpleDavPut();
        }

      /**
       * Choisis le littéral sur lequel va s'effectuer le branchement.
```

```
 * Dans cette version très simple, on se base uniquement sur
 * l'heuristique des littéraux.
 * Comme notre choix s'effectue parmi les littéraux de
 * p, il faut vérifier à chaque fois si le litéral n'a pas déja
 * été assigné.
 * @param base une base de clauses
 * @param p l'ensemble de littéraux parmi lesquels il faut chercher.
 * @return le meilleur littéral de <code>l</code>.
 */
protected Literal chooseLiteral(Base base,LiteralSet p) {
  Literal lmax=null;
  Literal l;
  long max = 0;

  for (int i=0;i<p.size();i++) {
      l = (Literal)p.elementAt(i);
      if (l.getSymbol().isUndefined()) {
          long poids = l.getSymbol().getPoids();
          if (max<poids) {
              max = poids;
              lmax = l.negation();
          }
      } // end of if ()
  } // end of for ()
  return lmax;
}

/**
 * Choisis un littéral dont la satisfaction conserve l'équivalence
 * logique restreinte à P entre la base de départ et la base simplifiée
 * (par exemple, un littéral apparaissant dans une clause unitaire).
 * @param base une base de clause
 * @param p un ensemble consistant de littéraux.
 * @return un littéral permettant de simplifier la base tout en conservant
 *         l'équivalence logique.
 */
  protected Literal simplified(Base base,LiteralSet p) {
      if (base.hasUnitClauses()) {
          return (Literal)base.getUnitClauses().pop();
      }
      for (int i=0;i<p.size();i++) {
          Literal lit = (Literal)p.elementAt(i);
          if (lit.negation().isPureLiteral()) {
              return lit.negation();
          } // end of if ()
      } // end of for ()
      return null;
  }

  protected Literal simplified(Base base, Base basea,LiteralSet p) {
      if (base.hasUnitClauses()) {
          return (Literal)base.getUnitClauses().pop();
      }
      for (int i=0;i<p.size();i++) {
          Literal lit = (Literal)p.elementAt(i);
          if (lit.negation().isPureLiteral()) {
              return lit.negation();
          } // end of if ()
      } // end of for ()
      if (basea.hasUnitClauses()) {
          return (Literal)basea.getUnitClauses().pop();
      }
      return null;
  }

/**
 * La procédure elle même.
 * Le résultat se trouve dans la base de clauses : ce sont les
 * clauses ajoutées (ou plus précisément, les négations des clauses
 * ajoutées).
 */
  public void run(Base base, Base basea, LiteralSet p) {
      run(base,basea,new LiteralSet(),p);
  }

  protected void run(Base base, Base basea,LiteralSet ip,LiteralSet p) {
      if (base.isFalsified()||basea.isFalsified()) return;
```

```
            if (base.isSatisfied()) {
                basea.addClause(ip.inter(p).complement());
                return;
            }
            Literal l = simplified(base,basea,p);
            if (l!=null) {
                l.satisfy(); ip.addElement(l);
                run(base,basea,ip,p);
                ip.removeElement(l); l.undefine();
                return;
            }
            l = chooseLiteral(base,p);
            if (l!=null) {
                l.satisfy(); ip.addElement(l);
                run(base,basea,ip,p);
                ip.removeElement(l); l.flip();
                ip.addElement(l.negation());
                run(base,basea,ip,p);
                ip.removeElement(l.negation()); l.negation().undefine();
                return;
            } else {
                if (sat.run(base)) {
                   basea.addClause(ip.inter(p).complement());
                } // end of if ()
            } // end of else
        }

    }
```

PIPQNTP

```
    public class PIPQNTP extends MPL {

        public void run(Base base,LiteralSet p) {
            Base newbase = base.inter(p);
            Base basea = new Base(newbase.getSymbols());
            LiteralSet pp = newbase.getSameLiterals(p);
            run(newbase,basea,pp);
        }

    } // PIPQNTP
```

PIP

```
    public class PIP implements EnumerativeAlgorithm {
        protected MPL mpl;

        public PIP() {
            mpl = new MPL();
        }

        public void run (Base base, Base res) {
            LiteralSet p = base.getSubLanguageLiterals(new NegativeLiteral());
            run(base,res,p);
        }

        public void run(Base base, Base res, LiteralSet p) {
            Base basea = new Base(base.getSubLanguageSymbols());
            mpl.run(base,basea,p);
            basea.clean();
            if (!p.isEmpty()) {
                p = basea.getLiterals(((Literal)p.firstElement()).negation());
            }
            mpl.run(basea,res,p);
            res.clean();
        }
    } // PIP
```

DualMPL

```
// DualMPL est basé sur une version de MPL ne propageant pas de littéraux purs
public class DualMPL extends MPLNoNPLP {
    public void run(Base base, Base basea,LiteralSet ip,LiteralSet p) {
        try {
            if (base.isSatisfied()||basea.isFalsified()) return;
            if (base.isFalsified()) {
                basea.addClause(ip.complement());
                return;
            }
            Literal l = chooseLiteral(base,p);
            if (l!=null) {
                p.removeElement(l.negation());
                run(base,basea,ip,p);
                l.negation().satisfy(); ip.addElement(l.negation());
                run(base,basea,ip,p);
                ip.removeElement(l.negation()); l.negation().undefine();
                p.addElement(l.negation());
                return;
            } else {
                if (!sat.run(base)) {
                    basea.addClause(ip.complement());
                }
            } // end of else
        } catch (Exception ex) {
            System.out.println("Ex   "+ex+"p   "+p+"ip   "+ip+"Base   :   "+base+"   basea
"+basea);
        }
    }
} // DualMPL
```

Castell

```
/**
 * Calcul de P-impliquants premiers basé sur la technique de
 * minimisation de Thierry Castell
 */
public class Castell extends MPL {

    /**
     * Retourne le meilleur littéral de la base pour le prochain branchement.
     * @param base la base de clauses.
     * @return un littéral de la base.
     */
    protected Literal chooseLiteral(Base base) {
        Literal lmax=null;
        Symbol s;
        long max = 0,poids;

        Enumeration e = base.getAllSymbols().elements();
        while (e.hasMoreElements()) {
            s = (Symbol)e.nextElement();
            poids = s.getPoidsMax();
            if (max<poids) {
                max = poids;
                lmax = s.getBestLiteral();
            }
        } // end of while ()
        return lmax;
    }

    /**
     * Test de minimalité : on vérifie que tous les littéraux
     * de l'ensemble de littéraux sont bien nécessaires.
     * @param ip un ensemble de littéraux
     * @param <code>true</code> ssi cet impliquant est minimal.
     */
    protected boolean isMinimal(LiteralSet ip) {
        for (int i=0;i<ip.size();i++) {
            if (!((Literal)ip.elementAt(i)).isNecessaire()) {
```

```
                return false;
            } // end of if ()
        } // end of for ()
        return true;
    }

    protected boolean isMinimal(LiteralSet ip, Base base) {
        for (int i=0;i<ip.size();i++) {
            if (!((Literal)ip.elementAt(i)).isNecessaire(base.getClauses())) {
                return false;
            } // end of if ()
        } // end of for ()
        return true;
    }

    /**
     * La procédure elle même.
     * @param base une base de clauses qui ne contient que des
     *             littéraux purs.
     * @param basea une base de clauses qui contiendra les
     *              impliquants premiers de <code>base</code>.
     */
    public void run(Base base, Base basea, LiteralSet p) {
        ReactiveVector sols = new ReactiveVector();
        sols.setNewClauseEventListener(basea.getNewClauseEventListener());
        run(base,new LiteralSet(),p,sols);
        basea.and(sols);
    }


    protected void run(Base base, LiteralSet ip,LiteralSet p,ReactiveVector sols) {
        if (base.isFalsified()||!isMinimal(ip.inter(p))) return;
        if (base.isSatisfied()) {
            sols.addClause(new Clause(ip.inter(p).complement()));
            return;
        }
        Literal l = simplified(base,p);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            run(base,ip,p,sols);
            ip.removeElement(l); l.undefine();
            return;
        }
        l = chooseLiteral(base);

        l.satisfy(); ip.addElement(l);
        run(base,ip,p,sols);
        ip.removeElement(l); l.flip();
        ip.addElement(l.negation());
        run(base,ip,p,sols);
        ip.removeElement(l.negation()); l.negation().undefine();
    }

    /**
     * Seul algorithme capable de faire du filtrage.
     *
     */
    public    void    run(Base    base,LiteralSet    ip,LiteralSet    p,Base    contrain-
tes,ReactiveVector sols) {
        if
(base.isFalsified()||!isMinimal(ip.inter(p),base)||contraintes.isFalsified()) return;
        if (base.isSatisfied()&&contraintes.isSatisfied()) {
            sols.addClause(new Clause(ip.inter(p).complement()));
            return;
        }
        Literal l = simplified(base,p);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            run(base,ip,p,contraintes,sols);
            ip.removeElement(l); l.undefine();
            return;
        }
        l = chooseLiteral(base);

        l.satisfy(); ip.addElement(l);
        run(base,ip,p,contraintes,sols);
        ip.removeElement(l); l.flip();
```

```
            ip.addElement(l.negation());
            run(base,ip,p,contraintes,sols);
            ip.removeElement(l.negation()); l.negation().undefine();
        }

        public void run(Base base, Base basea, LiteralSet p,Base contraintes) {
            ReactiveVector sols = new ReactiveVector();
            sols.setNewClauseEventListener(basea.getNewClauseEventListener());
            run(base,new LiteralSet(),p,contraintes,sols);
            basea.and(sols);
        }

    }
```

ATMS/ACMS

```
    public class ATMS  {
        protected static MPL mpl = new MPL();
        protected EnumerativeAlgorithm first,second;
        protected Base nogoods;
        protected Vector interpretations;
        protected SubLanguageBase base;
        protected Vector ncel;

        protected ATMS() {
            first = mpl;
            second = mpl;
            ncel = new Vector();
        }

        public ATMS(SubLanguageBase base) {
            this();
            this.base = base;
            nogoods = nogoods((SubLanguageBase)base.clone());
        }

        public Base nogoods(SubLanguageBase base) {
            LiteralSet p = base.getSubLanguageLiterals(new NegativeLiteral());
            Base basea = new Base(base.getSubLanguageSymbols());
            // 1ere passe
            mpl.run(base,basea,p);
            basea.clean();
            // on calcule les interprétations
            interpretations = new Vector();
            Enumeration it = basea.getClauses().elements();
            while (it.hasMoreElements()) {
                LiteralSet ls = ((Clause)it.nextElement()).getLiterals();
                interpretations.addElement(p.complement().minus(ls));
            } // end of while ()
            p = basea.getLiterals(new PositiveLiteral());
            Base res = new Base(basea.getSymbols());
            // 2eme passe
            mpl.run(basea,res,p);
            res.clean();
            return res;
        }

        public Vector getNogoods() {
            Vector v = new Vector();
            Enumeration e = nogoods.getClauses().elements();
            while (e.hasMoreElements()) {
                v.addElement(((Clause)e.nextElement()).getLiterals().complement());
            } // end of while ()
            return v;
        }

        public Vector getInterpretations() {
            return interpretations;
        }

        public Vector label(SubLanguageBase base) {
            LiteralSet p = base.getSubLanguageLiterals(new NegativeLiteral());
            Base basea = new Base(base.getSubLanguageSymbols());
            // 1ere passe
```

```
            first.run(base,basea,p);
            basea.clean();
            // on ajoute la negation des nogoods
            basea.and(nogoods);
            p = basea.getLiterals(new PositiveLiteral());
            Base res = new Base(basea.getSymbols());
            // 2eme passe
            second.run(basea,res,p);
            res.clean();
            Vector v = new Vector();
            Enumeration e = res.getClauses().elements();
            while (e.hasMoreElements()) {
                v.addElement(((Clause)e.nextElement()).getLiterals().complement());
            } // end of while ()
            return v;
        }

        public LiteralSet contexte(SubLanguageBase base) {
            return contexte(base,base.getLiterals(new PositiveLiteral()));
        }

        public LiteralSet contexte(SubLanguageBase base, LiteralSet p) {
            Base basea = new Base(base.getAllSymbols());
            mpl.run(base,basea,p);
            basea.clean();
            LiteralSet ls = (LiteralSet)p.clone();
            Enumeration it = basea.getClauses().elements();
            while (it.hasMoreElements()) {
                LiteralSet lsn = ((Clause)it.nextElement()).getLiterals();
                ls = ls.inter(lsn.complement());
            } // end of while ()
            return ls;
        }


        public Vector label(Clause cl) {
            SubLanguageBase nbase = (SubLanguageBase)base.clone();
            nbase.and(cl.negation());
            return label(nbase);
        }

        public Vector label(Cube cube) {
            SubLanguageBase nbase = (SubLanguageBase)base.clone();
            nbase.and(cube.negation());
            return label(nbase);
        }

        public LiteralSet contexte(LiteralSet ls) {
            SubLanguageBase nbase = (SubLanguageBase)base.clone();
            nbase.and(new Cube(ls));
            return contexte(nbase);
        }

        public Vector contexteEnsemble(Vector ens) {
            Vector sol = new Vector();
            Enumeration it = ens.elements();
            while (it.hasMoreElements()) {
                LiteralSet ls = (LiteralSet)it.nextElement();
                sol.addElement(contexte(ls));
            } // end of while ()
            return sol;
        }


        public Vector extensions() {
            return contexteEnsemble(interpretations);
        }
    } // ATMS
```

CWR

```
    public class CWR  {
        protected static MPL mpl = new MPL();
        protected static PQZMPL pqz = new PQZMPL();
```

```
public static Vector ModMinHerbrand(Base base) {
   return ModMin(base,base.getLiterals(new PositiveLiteral()));
}

public static Vector ModMinLifschitz(Base base) {
   return ModMin(base,base.getSubLanguageLiterals(new PositiveLiteral()));
}

public static Vector ModMin(Base base, LiteralSet p) {
   Base basea = new Base();
   mpl.run(base,basea,p);
   basea.clean();
   Vector resultat = new Vector();
   Enumeration e = basea.getClauses().elements();
   while (e.hasMoreElements()) {
       LiteralSet ls = ((Clause)e.nextElement()).getLiterals();
       // comme les clauses de minimisations contiennent des negations
       // d'elements de p, on complete par p et on complemente le tout
       // pour obtenir les modèles minimaux de Herbrand.
       resultat.addElement(ls.complete(p).complement());
   } // end of while ()
   return resultat;
}

public static Vector ModMinPQZLifschitz(Base base, LiteralSet q) {
   LiteralSet p = base.getSubLanguageLiterals(new PositiveLiteral());
   return ModMinPQZ(base,p.minus(q),q);
}
public static Vector ModMinPQZ(Base base, LiteralSet p, LiteralSet q) {
   Base basea = new Base();
   pqz.run(base,basea,p,q);
   basea.clean();
   Vector resultat = new Vector();
   Enumeration e = basea.getClauses().elements();
   while (e.hasMoreElements()) {
       LiteralSet ls = ((Clause)e.nextElement()).getLiterals();
       // comme les clauses de minimisations contiennent des negations
       // d'elements de p, on complete par p et on complemente le tout
       // pour obtenir les modèles minimaux de Lifschitz.
       resultat.addElement(ls.complete(p).complement());
   } // end of while ()
   return resultat;
}

public static LiteralSet CWA(Base base) {
   LiteralSet p = base.getLiterals(new PositiveLiteral());
   Base basea = new Base(base.getSymbols());
   mpl.run(base,basea,p);
   basea.clean();
   Vector resultat = basea.getClauses();
   LiteralSet ls = new LiteralSet();
   while (!resultat.isEmpty()) {
       Clause cl = (Clause)resultat.firstElement();
       resultat.removeElement(cl);
       ls = ls.union(cl.getLiterals().complete(p).inter(p).complement());
   } // end of while ()
   return ls;
}

public static LiteralSet GCWA(Base base) {
   LiteralSet p = base.getLiterals(new PositiveLiteral());
   Base basea = new Base(base.getSymbols());
   mpl.run(base,basea,p);
   basea.clean();
   Vector resultat = basea.getClauses();
   LiteralSet ls;
   if (!resultat.isEmpty()) {
       Clause cl = (Clause)resultat.firstElement();
       resultat.removeElement(cl);
       ls = cl.getLiterals().complete(p).inter(p).complement();
       while (!resultat.isEmpty()) {
           cl = (Clause)resultat.firstElement();
           resultat.removeElement(cl);
           ls = ls.inter(cl.getLiterals().complete(p).inter(p).complement());
       } // end of while ()
   } else {
```

```
            ls = new LiteralSet();
        } // end of else
        return ls;
    }

    public static Vector EGCWA(Base base) {
        LiteralSet p = base.getLiterals(new PositiveLiteral());
        Base basea = new Base(base.getSymbols());
        mpl.run(base,basea,p);
        basea.clean();
        basea = basea.getBaseMax(p);
        p = basea.getLiterals(new NegativeLiteral());
        Base res = new Base(basea.getSymbols());
        mpl.run(basea,res,p);
        res.clean();
        Vector resultat = new Vector();
        Enumeration e = res.getClauses().elements();
        while (e.hasMoreElements()) {
            resultat.addElement(((Clause)e.nextElement()).getLiterals().complement());
        } // end of while ()
        return resultat;
    }

    public static LiteralSet CCWA(SubLanguageBase base) {
        LiteralSet p = base.getSubLanguageLiterals(new PositiveLiteral());
        Base basea = new Base(base.getSymbols());
        mpl.run(base,basea,p);
        basea.clean();
        Vector resultat = basea.getClauses();
        LiteralSet ls;
        if (!resultat.isEmpty()) {
            Clause cl = (Clause)resultat.firstElement();
            resultat.removeElement(cl);
            ls = cl.getLiterals().complete(p).inter(p).complement();
            while (!resultat.isEmpty()) {
                cl = (Clause)resultat.firstElement();
                resultat.removeElement(cl);
                ls = ls.inter(cl.getLiterals().complete(p).inter(p).complement());
            } // end of while ()
        } else {
            ls = new LiteralSet();
        } // end of else
        return ls;
    }

    public static Vector ECWA(SubLanguageBase base) {
        LiteralSet p = base.getSubLanguageLiterals(new PositiveLiteral());
        Base basea = new Base(base.getSymbols());
        mpl.run(base,basea,p);
        basea.clean();
        basea = basea.getBaseMax(p);
        p = basea.getLiterals(new NegativeLiteral());
        Base res = new Base(basea.getSymbols());
        mpl.run(basea,res,p);
        res.clean();
        Vector resultat = new Vector();
        Enumeration e = res.getClauses().elements();
        while (e.hasMoreElements()) {
            resultat.addElement(((Clause)e.nextElement()).getLiterals().complement());
        } // end of while ()
        return resultat;
  }
} // CWR
```

EntailsMPL

```
/**
 * Inférence minimale à l'aide de MPL.
 * On utilise le même principe que Niemela avec sa méthode des
 * tableaux.
 *
 */
public class EntailsMPL extends MPL {

    /**
     * On vérifie que l'ensemble de littéraux est bien consistant
```

```
   * avec la base.
   * Contrairement à l propagation de clauses unitaires, il faut
   * tester à chaque fois si la base est satisfaite ou falsifiée.
   */
  protected boolean isConsistant(Base base,LiteralSet ls) {
     if (base.isFalsified()) {
         return false;
     } // end of if ()
     if (base.isSatisfied()) {
         return true;
     } // end of if ()
     if (!ls.isEmpty()) {
         Literal lit = (Literal)ls.removeLast();
         lit.satisfy();
         boolean res = isConsistant(base,ls);
         lit.undefine();
         return res;
     } // end of if ()
     return sat.run(base);
  }

/**
 * La procédure elle même.
 * on est obligé de changer de nom de méthode car on retourne vrai ou faux.
 */
public boolean run2(Base base, Base basea,Base formule,LiteralSet ip,LiteralSet p) {
    if (base.isFalsified()||basea.isFalsified()) return true;
    if (base.isSatisfied()) {
       // on a trouvé un modèle P-minimal
       LiteralSet min = ip.inter(p).complement();
       // on l'ajoute dans la base de minimisation.
       basea.addClause(min);
       // on vérifie si la formule est falsifiée par ce modèle
       if (!formule.isFalsified()) {
           // comme les littéraux de IP sont déjà satisfaits, il suffit
           // de propager les littéraux de p.complement() ni satisfait
           // ni falsifiés.
           return !isConsistant(formule,p.complement().minus(ip).minus(min));
       }
       return true;
    }
    Literal l = simplified(base,p);
    if (l!=null) {
       l.satisfy(); ip.addElement(l);
       boolean res = run2(base,basea,formule,ip,p);
       ip.removeElement(l); l.undefine();
       return res;
    }
    l = chooseLiteral(base,p);
    if (l!=null) {
       l.satisfy(); ip.addElement(l);
       boolean res = run2(base,basea,formule,ip,p);
       ip.removeElement(l); l.undefine();
       if (res) {
           ip.addElement(l.negation());l.negation().satisfy();
           res = run2(base,basea,formule,ip,p);
           ip.removeElement(l.negation()); l.negation().undefine();
       } // end of if ()
       return res;
    } else {
       if (sat.run(base)) {
           LiteralSet min = ip.inter(p).complement();
           if (!formule.isFalsified()) {
              return !isConsistant(formule,p.complement().minus(ip).minus(min));
           }
           basea.addClause(min);
       }
       return true;
    } // end of else
}

  public void run(Base base, Base basea, Base formule, LiteralSet p) {
     boolean res = run2(base,basea,formule,new LiteralSet(),p);
  }

  /**
   * @param base une base de clauses
```

```
     * @param basea une base de minimisation
     * @param formule une formule sous forme CNF
     * @param p un ensemble consistant de littéraux
     * @return <code>true</code> ssi la formule est inconsistante avec
     *         tous les modèles P-minimaux de base, c'est à dire que
     *         la négation de la formule est minimalement inférée par base.
     *         Dans ce cas, basea contient tous les modèles P-minimaux de BC.
     *         Sinon, la dernière clause de basea est un contre-exemple.
     */
    public boolean isEntailed(Base base,Base basea, Base formule, LiteralSet p) {
       return run2(base,basea,formule,new LiteralSet(),p);
    }

    /**
     * @param base une base de clauses
     * @param formule une formule sous forme CNF
     * @param p un ensemble consistant de littéraux
     * @return <code>true</code> ssi la formule est inconsistante avec
     *         tous les modèles P-minimaux de base, c'est à dire que
     *         la négation de la formule est minimalement inférée par base.
     */
    public boolean isEntailed(Base base,Base formule,LiteralSet p) {
       Base basea = new Base();
       return isEntailed(base,basea,formule,p);
    }

}
```

Decision dans l’incertain

```
public class Decision  {
    protected MPL mpl = new MPL();

    public String optimisticUtility(StratifiedBase K,PreferenceBase P,Vector solutions)
{
       Vector scale = K.getScale();
       int size =scale.size();
       int i=size-1;
       boolean cont = true;
       String key="";
      Vector sol = new Vector();

       while ((i>0)&&cont) {
           key = (String)scale.elementAt(i--);
           K.setStrateOver(key);
           K.and(P.getStrateOver(key));
           LiteralSet d = K.getSubLanguageLiterals(new PositiveLiteral());
           sol = mpl.getSolutions(K,d);
           cont = sol.isEmpty();
       } // end of while ()
       Enumeration enum = sol.elements();
       while (enum.hasMoreElements()) {
           solutions.addElement(enum.nextElement());
       } // end of while ()
       if (cont) {
           return (String)scale.lastElement();
       } else {
           return (String)scale.elementAt(size-2-i);
       } // end of else
    }

    public  String  pessimisticUtility(StratifiedBase  K,PreferenceBase  P,Vector  solu-
tions) {
       Vector  scale = K.getScale();
       int size =scale.size();
       int i=0;
       boolean cont = true;
       String alpha="";
       PIP pip = new PIP();
       Base inter2 = new Base();

       while ((i<size-1)&&cont) {
```

```
            alpha = (String)scale.elementAt(i++);
            K.setStrateUpTo(alpha);
            ATMS atms = new ATMS(K);
            Base nogoods = atms.getNegatedNogoods();
            Base inter = new Base(K.getSubLanguageSymbols());
            int j = 0;
            String beta = (String)scale.elementAt(j);
            K.setStrateUpTo(alpha);
            K.and(P.getNegatedStrate(beta));
            LiteralSet d = K.getSubLanguageLiterals(new NegativeLiteral());
            mpl.run(K,inter,d);
            inter.clean();
            d = inter.getLiterals(new PositiveLiteral());
            inter2 = new Base(inter.getSymbols());
            inter.and(nogoods);
            mpl.run(inter,inter2,d);
            inter2.clean();
            while ((j<size-i-1)&&!inter2.isEmpty()) {
                beta = (String)scale.elementAt(++j);
                K.setStrateUpTo(alpha);
                K.and(P.getNegatedStrate(beta));
                Base inter3 = new Base(K.getSubLanguageSymbols());
                d = K.getSubLanguageLiterals(new NegativeLiteral());
                mpl.run(K,inter3,d);
                inter3.clean();
                inter.and(inter3);
                inter2 = new Base(inter.getSymbols());
                d = inter.getLiterals(new PositiveLiteral());
                mpl.run(inter,inter2,d);
                inter2.clean();
            } // end of while ()
            if (cont = inter2.isEmpty()) {
                String nextalpha = K.getNextNonEmptyStrateAfter(alpha);
                alpha  =  scale.indexOf(nextalpha)<size-j-1  ?  nextalpha  :  (String)
scale.elementAt(size-j-1) ;
                i = scale.indexOf(alpha);
            } // end of if ()
        } // end of while ()
        Enumeration enum = inter2.negation().elements();
        while (enum.hasMoreElements()) {
            solutions.addElement(enum.nextElement());
        } // end of while ()
        return alpha ;
    }

} // Decision
```

ModelMinimization

```
public class ModelMinimization implements EnumerativeAlgorithm {

    ModelGenerator gen;
    SimpleDavPut sat;
    int maxsol;

    public ModelMinimization() {
        this(new DavPutModel(),1);
    }

    public ModelMinimization(Integer nbsol) {
        this(new DavPutModel(),nbsol.intValue());
    }

    public ModelMinimization(ModelGenerator modgen) {
        this(modgen,1);
    }

    public ModelMinimization(ModelGenerator modgen, int nbsol) {
        gen = modgen;
        sat = new SimpleDavPut();
        maxsol = nbsol;
    }
```

```
    public void run(Base base, Base basea, LiteralSet p) {
       LiteralSet ls = gen.getModel(base);
       while ((basea.getClausesCount()<maxsol)&&(ls!=null)) {
           LiteralSet E = ls.inter(p);
           boolean change;
           Stack EE = new Stack();
           LiteralSet nls = new LiteralSet();
           LiteralSet mls = new LiteralSet();
           EE.addAll(E);
           while (!EE.empty()) {
              Literal l = (Literal)EE.pop();
              nls = mls.union(p.minus(E).complement());
              nls.addElement(l.negation());
              if (!sat.propagate(base,nls)) {
                  mls.addElement(l);
              } else {
                  mls.addElement(l.negation());
              } // end of else
           }
           E = mls.inter(p);
           basea.addClause(basea.newClauseForBase(E.complement()));
           if (E.isEmpty()) {
              return;
           } // end of if ()
           base.addClause(base.newClauseForBase(E.complement()));
           if (basea.getClausesCount()<maxsol) {
              ls =  gen.getModel(base);
           } // end of if ()
       } // end of while ()
    }
} // ModelMinimization
```

DSMPL

```
public class DSMPL extends MPL implements ValuedEnumerativeAlgorithm {

    /**
     * On ne peut pas propager les littéraux purs comme dans MPL.
     */
    protected Literal simplified(Base base,LiteralSet p) {
       if (base.hasUnitClauses()) {
           return (Literal)base.getUnitClauses().pop();
       }
       return null;
    }

    public float vrun(Base base, Base basea, LiteralSet ip, LiteralSet p,Hashtable va-
lues) {
       if (base.isFalsified()) return 0.0f;
       if (base.isSatisfied()) {
           if (!basea.isFalsified()) {
              basea.addClause(ip.inter(p).complement());
           } // end of if ()
           return 1.0f;
       }
       Literal l = simplified(base,p);
       if (l!=null) {
           l.satisfy(); ip.addElement(l);
           float res = vrun(base,basea,ip,p,values);
           ip.removeElement(l); l.undefine();
           res = res*((Float)values.get(l.toString())).floatValue();
           return res;
       }
       l = chooseLiteral(base,p);
       if (l!=null) {
           l.satisfy(); ip.addElement(l);
           float resg = vrun(base,basea,ip,p,values);
           ip.removeElement(l); l.undefine();

           l.negation().satisfy(); ip.addElement(l.negation());
           float resd = vrun(base,basea,ip,p,values);
           ip.removeElement(l.negation()); l.negation().undefine();

           float res = resd*((Float)values.get(l.negation().toString())).floatValue();
```

```
            res += resg*((Float)values.get(l.toString())).floatValue();
            return res;
        } else {
            if (sat.run(base)) {
                if (!basea.isFalsified()) {
                    basea.addClause(ip.inter(p).complement());
                }
                return 1.0f;
            } // end of if ()
            return 0.0f;
        } // end of else {
    }

    public float vrun(Base base, Base basea, LiteralSet p,Hashtable values) {
        return vrun(base,basea,new LiteralSet(),p,values);
    }
} // DSMPL
```

WI

```
/**
 * Calcul des impliquants P-restreints WI-préférés [ACL96].
 */
public class WI extends MPL
    implements ValuedEnumerativeAlgorithm {

    public WI() {
        sat = new SimpleDavPut();
    }

    /**
     * Choisis le littéral de niveau maximal.
     * @param p un ensemble de littéraux.
     * @param values le numéro de strates associé à chaque symbole de p.
     * @return un littéral de p ayant le numéro de strate le plus élevé.
     */
    protected Literal chooseLiteral(LiteralSet p, Hashtable values) {
        Literal lmax=null;
        Literal l;
        float max = 0.0f;

        for (int i=0;i<p.size();i++) {
            l = ((Literal)p.elementAt(i));
            if (l.getSymbol().isUndefined()) {
                float poids = ((Float)values.get(l.toString())).floatValue();
                if (max<poids) {
                    max = poids;
                    lmax = l.negation();
                } // else {
            } // end of if ()
        }
        return lmax;
    }

    /**
     * Pour les niveaux, la fonction d'agrégation est le max.
     */
    protected float agregate(float v1,float v2) {
        return Math.max(v1,v2);
    }

    /**
     * On préfère les ensemble ayant le niveau le plus bas.
     */
    protected boolean best(float v1,float v2) {
        return v1<v2;
    }

    public float vrun(Base base, Base basea,LiteralSet ip,LiteralSet p,Hashtable va-
lues,float ipval,float borne) {
        if (base.isFalsified()||basea.isFalsified()||best(borne,ipval)) return borne;
        if (base.isSatisfied()) {
            basea.addClause(ip.inter(p).complement());
```

```
            return ipval;
        }
        Literal l = simplified(base,p);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            float val ;
            if (p.contains(l)) {
                val = agregate(ipval,((Float)values.get(l.toString())).floatValue());
            } else {
                val = ipval;
            } // end of else

            float res = vrun(base,basea,ip,p,values,val,borne);
            ip.removeElement(l); l.undefine();
            return res;
        }
        l = chooseLiteral(p,values);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            float res = vrun(base,basea,ip,p,values,ipval,borne);
            ip.removeElement(l); l.undefine();
            l.negation().satisfy(); ip.addElement(l.negation());
            res                                                                    =
vrun(base,basea,ip,p,values,agregate(ipval,((Float)values.get(l.negation().toString()))
.floatValue()),res);
            ip.removeElement(l.negation()); l.negation().undefine();
            return res;
        } else {
            base.getPureLiterals().removeAllElements();
            if (sat.run(base)) {
                basea.addClause(ip.inter(p).complement());
                return ipval;
            } else {
                return borne;
            } // end of else
        } // end of else
    }

    public float vrun(Base base, Base basea, LiteralSet p,Hashtable values) {
        return vrun(base,basea,new LiteralSet(),p,values,0.0f,Float.MAX_VALUE);
    }
} // WI
```

ELI

```
public class ELI extends WI {

    public ELI() {
        sat = new SimpleDavPut();
    }

    public float vrun(Base base, Base basea,LiteralSet ip,LiteralSet p,Hashtable va-
lues,float borne) {
        if (base.isFalsified()||basea.isFalsified()) return borne;
        if (base.isSatisfied()) {
            basea.addClause(ip.inter(p).complement());
            return 0.0f;
        }
        Literal l = simplified(base,p);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            float res = vrun(base,basea,ip,p,values,borne);
            ip.removeElement(l); l.undefine();

            if (p.contains(l)) {
                float val = ((Float)values.get(l.toString())).floatValue();
                return agregate(res,val);
            } else {
                return res;
            } // end of else
        }
        l = chooseLiteral(p,values);
        if (l!=null) {
```

```
            float val = ((Float)values.get(l.negation().toString())).floatValue();
            l.satisfy(); ip.addElement(l);
            float res = vrun(base,basea,ip,p,values,borne);
            ip.removeElement(l); l.undefine();
            if (!best(res,val)) {
                l.negation().satisfy(); ip.addElement(l.negation());
                res = vrun(base,basea,ip,p,values,res);
                ip.removeElement(l.negation()); l.negation().undefine();
                return agregate(res,val);
            } // end of if ()
            return res;
        } else {
            if (sat.run(base)) {
                base.getPureLiterals().removeAllElements();
                basea.addClause(ip.inter(p).complement());
                return 0.0f;
            } else {
                return borne;
            } // end of else
        } // end of else
    }

    public float vrun(Base base, Base basea,LiteralSet p,Hashtable values) {
        return vrun(base,basea,new LiteralSet(),p,values,Float.MAX_VALUE);
    }
} // ELI
```

ILDSMPL

```
public class ILDSMPL extends ILDS implements ValuedEnumerativeAlgorithm {
    public void LDSprobe(Base base,Base basea, LiteralSet ip, LiteralSet p, int k, int
depth) {
        if (base.isFalsified()) return;
        if (base.isSatisfied()) {
            basea.addClause(ip.inter(p).complement());
            return;
        }
        Literal l = simplified(base,p);
        if (l!=null) {
            l.satisfy(); ip.addElement(l);
            if (p.contains(l.negation())) {
                if (depth>k) {
                    LDSprobe(base,basea,ip,p,k,depth-1);
                } // end of if ()
            } else {
                if (p.contains(l)) {
                    if (k>0) {
                        LDSprobe(base,basea,ip,p,k-1,depth-1);
                    } // end of if ()
                } else {
                    LDSprobe(base,basea,ip,p,k,depth);
                } // end of else
            } // end of else
            ip.removeElement(l); l.undefine();
            return;
        } // end of if ()
        l = chooseLiteral(p);
        if (l!=null) {
            if (depth>k) {
                l.satisfy(); ip.addElement(l);
                LDSprobe(base,basea,ip,p,k,depth-1);
                ip.removeElement(l); l.undefine();
            }
            if (k>0) {
                l.negation().satisfy(); ip.addElement(l.negation());
                LDSprobe(base,basea,ip,p,k-1,depth-1);
                ip.removeElement(l.negation()); l.negation().undefine();
            } // end of if ()
        } else {
            if (sat.run(base)) {
                basea.addClause(ip.inter(p).complement());
            } // end of if ()
        }
    }
```

TITLE : Beyond SAT : P-restricted implicant, algorithms and applications

---

ABSTRACT :

Our work concernes two major problems in propositional logic : the satisfiability of a boolean formula (SAT problem) and the computation of its prime implicants/implicates. The first one is widely studied in the Artificial Intelligence community and some recent results show that SAT algorithms can be used efficiently to solve problems in domains where specialized algorithms were used (planning or diagnosis for example). The second one is very important because it characterizes abductive reasoning (implicate framework). We propose to modify a SAT algorithm (the Davis and Putnam procedure) to compute formulas used by Assumption-based Truth Maintenance Systems. We formalize our method in terms of prime P-restricted implicants : the intersection between a model and a consistent set of literals. Then we apply our formalization in two examples of non-monotonic reasoning: Closed World Reasoning and Argumentation. We introduce preference relations between P-restricted implicants. For example, in diagnosis, minimal cardinality explanations are sufficient. If other informations are available (fault probabilities for each component for instance), most probable explanations are preferred. We take two examples of such preference relations in argumentation. Finally, we show that qualitative decision theory in a logical framework can also be modeled with P-restricted implicants.

KEYWORDS

Artificial Intelligence, Propositional Logic, SAT, Prime Implicant/Implicate, ATMS, Nonmonotonic Reasoning, Algorithms.

AUTEUR : Daniel LE BERRE

TITRE : Autour de SAT : le calcul d'impliquant P-restreint, algorithmes et applications

DIRECTEUR DE THESE : Michel CAYROL

LIEU ET DATE DE SOUTENANCE : IRIT-UPS, le 12 janvier 2000

RESUME

Les travaux présentés concernent les deux problèmes fondamentaux de la logique propositionnelle : la satisfiabilité d'une formule logique (problème SAT) et la détermination de ses impliquants/impliqués premiers. Le premier problème est largement étudié par la communauté IA et les progrès récents sont si importants que l'on utilise actuellement des algorithmes de résolution de SAT pour traiter des problèmes dans des domaines jusqu'alors réservés à des prouveurs spécialisés (planification ou diagnostic par exemple). Le second est très important car il caractérise le raisonnement abductif (notion d'impliqué). Nous proposons de modifier une méthode de résolution de SAT (la procédure de Davis et Putnam) afin de produire des formules utilisées par des outils comme l'ATMS. Nous avons formalisé cette méthode par la notion d'impliquant P-restreint premier : intuitivement, il s'agit de l'intersection d'un modèle avec un ensemble consistant de littéraux. Nous utilisons ensuite cette notion dans le cadre du raisonnement non monotone : en raisonnement en monde clos (Closed World Reasoning) et dans le cadre de l'argumentation. Nous avons étendu notre formalisation afin d'utiliser différentes relations de préférence entre impliquants P-restreints (notion d'impliquant P-restreint préféré). Dans le cadre du diagnostic, on cherchera souvent les explications mettant en cause le plus petit nombre de composants. Si l'on dispose de connaissances exogènes (des probabilités de panne pour chaque composant par exemple), on souhaite obtenir le diagnostic de panne le plus probable. Dans le cadre de l'argumentation, nous donnons deux exemples de ces relations de préférence. Ces travaux s'éloignent du domaine de la satisfaction pour rejoindre les travaux sur l'optimisation. Nous montrons enfin dans le cadre de l'approche logique de la décision qualitative comment profiter de certaines particularités de notre méthode de calcul d'impliquant P-restreint pour calculer des décisions optimales.

MOTS-CLES

Intelligence Artificielle, Logique Propositionnelle, SAT, Impliquant/Impliqué Premier, ATMS, Raisonnement Non Monotone, Algorithmes.

DISCIPLINE ADMINISTRATIVE

Informatique

INTITULE ET ADRESSE DE L'U.F.R. OU DU LABORATOIRE

Institut de Recherche en Informatique de Toulouse, UMR 5505 CNRS - INP - UPS, Université Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse cedex 4