Une approche DPLL pour l'abduction

Florian Letombe, Bruno Zanuttini

Une approche DPLL pour l'abduction

Florian Letombe[1] **Bruno Zanuttini**[2]

[1]CRIL, CNRS FRE 2499 [2]GREYC, CNRS UMR 6072
Université d'Artois Université de Caen
letombe@cril.univ-artois.fr bruno.zanuttini@info.unicaen.fr

Résumé

L'abduction est le processus consistant à expliquer une observation q étant donnée une base de connaissances KB, c'est-à-dire à trouver un ensemble de faits E, appelé une explication, tel que KB et E impliquent conjointement q. De plus, les explications sont contraintes par un ensemble d'hypothèses H, c'est-à-dire un ensemble de littéraux sur lesquels elles doivent être formées. Ce problème est reconnu pour avoir de nombreuses applications, en particulier en intelligence artificielle (IA), et a beaucoup été étudié des points de vue de l'IA comme de la complexité.

Dans cet article, nous présentons un algorithme à la DPLL adapté aux problèmes d'abduction. En particulier, cet algorithme nous permet de proposer une approche heuristique pour ces problèmes, et de développer des techniques *ad hoc* d'élagage de l'arbre de recherche.

Nous développons un prouveur, dédié aux problèmes d'abduction et baptisé `zlas`, et nous proposons une comparaison de notre méthode à celle du projet QUIP, qui utilise une traduction en une formule booléenne quantifiée (QBF) pour résoudre ce type de problèmes. En plus du prouveur QBF inclus dans QUIP, nous enrichissons les tests avec les meilleurs prouveurs QBF actuels. Nous montrons que pour la plupart des tests proposés, notre algorithme permet d'améliorer significativement les résultats obtenus avec une modélisation en QBF.

Abstract

Abduction is the process of explaining an observation q given a knowledge base KB, i.e., of finding a set of facts E, called an explanation, such that KB and E together entail q. Explanations are moreover constrained by a set of hypotheses H, that is, a set of literals over which they must be built. This problem is well-known to have many applications, in particular in Artificial Intelligence (AI), and has been widely studied from both an AI and a complexity-theoretic point of view.

In this paper, we present a DPLL-like algorithm, adapted to abduction problems. In particular, it allows us to propose a heuristic approach to abduction, and to develop ad hoc pruning techniques of the search tree.

We develop a solver, dedicated to abduction problems and called `zlas`, and we propose a comparison of our method to the one envisioned in the QUIP project, which uses a translation into a quantified Boolean formula (QBF) to solve this kind of problems. In addition to the QBF solver included in QUIP, we enhance tests with the best state-of-the-art current QBF solvers. We show that for most of the proposed tests, our algorithm allows to significantly improve results obtained with a QBF modeling.

1 Introduction

QBF, le problème de la validité pour les formules booléennes quantifiées (QBF), revêt une importance grandissante en intelligence artificielle (IA). Cela peut s'expliquer par le fait que, en tant que problème PSPACE-complet canonique, de nombreux problèmes d'IA peuvent s'y réduire en temps polynomial. En outre, des résultats expérimentaux obtenus dans divers domaines de l'IA (dont la planification, le raisonnement non-monotone, l'inférence paraconsistante) montrent qu'une approche basée sur la traduction peut s'avérer plus efficace que des algorithmes dépendants du domaine et dédiés à ces tâches (voir [11, 14, 2, 27, 22]). Ainsi, de nombreux prouveurs QBF ont été conçus et évalués ces dernières années (voir principalement [3, 15, 28, 17, 23, 1, 29]).

Parmi les problèmes intéressants qui peuvent être traduits en QBF, l'abduction est un problème très important. Alors que de nombreuses études sur l'abduction adoptent le point de vue de la complexité et ne donnent des algorithmes que pour les classes polynomiales, nous nous intéressons aux algorithmes génériques pour les problèmes d'abduction. Cependant, le problème général (sans restriction sur la base

de connaissances) est Σ_2^P-complet, et reste complet même avec des restrictions telles que des requêtes atomiques et des ensembles d'hypothèses clos par négation. Il s'agit donc de trouver des solutions efficaces en pratique. L'utilisation d'algorithmes pour le problème QBF est une direction prometteuse, puisque la recherche dans ce domaine est actuellement très active.

Néanmoins, des études d'algorithmes génériques pour l'abduction existent, sous la forme d'algorithmes pour le calcul d'impliqués (*consequence-finding*). Des travaux dans cette direction ont été menés essentiellement par Simon et del Val [7, 31], et nous conseillons au lecteur l'état de l'art de Marquis [18]. Zanuttini a également proposé un algorithme générique [32], qui utilise principalement les notions d'oubli de variables (projection) et de satisfiabilité. Finalement, Egly *et al.* [11] proposent des méthodes pour traduire un problème d'abduction en une QBF.

Nous proposons une autre approche, basée sur un algorithme à la DPLL pour explorer l'espace de recherche des explications candidates. Notre approche est donc dédiée à l'abduction, et permet d'utiliser des méthodes heuristiques spécialisées pour choisir les hypothèses, ainsi que des conditions d'élagage fortes pour l'espace de recherche. L'une de ces conditions est celle provoquant un retour en arrière (*backtrack*) dès que l'explication candidate n'est pas cohérente avec la base de connaissances. Cette condition peut bénéficier de prouveurs très efficaces pour la satisfiabilité (nous utilisons actuellement MINISAT [9]).

Nous comparons notre approche à celle basée sur une traduction en QBF sur des jeux d'essais générés à partir de problèmes de satisfiabilité, et nous montrons que, comme on pouvait s'y attendre, notre prouveur est plus performant que ceux basés sur une telle transformation, et ce, même lorsque ces derniers utilisent les meilleurs prouveurs QBF actuels.

Le papier est organisé comme suit. Le paragraphe 2 donne des définitions préliminaires. Dans le paragraphe 3, nous présentons les méthodes connues pour résoudre des problèmes d'abduction. Nous présentons notre algorithme à la DPLL dans le paragraphe 4, et commentons les résultats expérimentaux dans le paragraphe 5. Dans le paragraphe 6, nous concluons et donnons des perspectives de travail.

2 Définitions

2.1 Notations

Si φ est une formule propositionnelle, ℓ est un littéral et L est un ensemble de littéraux, $\mathcal{V}(\varphi)$ (resp. $\mathcal{V}(\ell)$, $\mathcal{V}(L)$) désigne l'ensemble des variables apparaissant dans φ (resp. ℓ, L). De même, si φ est sous forme normale négative, $\mathcal{L}(\varphi)$ désigne l'ensemble des littéraux y apparaissant. Si ℓ est un littéral, $\overline{\ell}$ désigne le littéral opposé. Si $L = \{\ell_1, \ldots, \ell_k\}$ est un ensemble de littéraux, $\bigwedge L$ désigne la formule propositionnelle $\ell_1 \wedge \cdots \wedge \ell_k$. Les affectations sont notées comme l'ensemble des littéraux qu'elles satisfont. Si φ et φ' sont deux formules propositionnelles, nous notons $\varphi \models \varphi'$ si φ implique φ', c'est-à-dire si tout modèle de φ est un modèle de φ'.

2.2 Problèmes d'abduction

Nous utilisons la définition standard suivante d'un problème d'abduction. Dans la définition comme dans le reste de l'article, V représente un ensemble de variables, H un ensemble d'hypothèses, q une requête, φ une base de connaissances et E une explication.

Définition 1 (problème d'abduction) *Une instance du problème d'abduction est un n-uplet (V, H, q, φ), où V est un ensemble de variables propositionnelles, H est un ensemble de littéraux formés sur V, q est une variable de $V \setminus \mathcal{V}(H)$ et φ est une formule propositionnelle formée sur V.*

Une solution *d'une telle instance est un ensemble de littéraux E (aussi appelé une* explication*) tel que :*

1. $E \subseteq H$;
2. $\varphi \wedge \bigwedge E$ *est satisfiable ;*
3. $\varphi \wedge \bigwedge E \models q$.

Le problème d'abduction consiste, étant donnée une telle instance, à décider s'il existe une explication et, si c'est le cas, à en calculer une.

Dans la pratique, nous nous intéressons à des bases de connaissances sous forme normale conjonctive.

Exemple 2 (problème d'abduction) *Soient $V = \{h_1, h_2, h_3, x_1, x_2, q\}$, $H = \{h_1, \neg h_1, h_2, \neg h_2, h_3\}$ et la formule propositionnelle suivante sur V :*

$$\begin{aligned}\varphi = \quad & (h_1 \vee \neg h_2 \vee q) \wedge (h_1 \vee \neg h_2 \vee \neg x_1) \wedge \\ & (h_1 \vee \neg h_2 \vee x_1) \wedge (h_3 \vee x_1 \vee x_2) \wedge \\ & (h_1 \vee h_2 \vee q).\end{aligned}$$

Alors le n-uplet (V, H, q, φ) est une instance du problème d'abduction.

L'ensemble $E = \{\neg h_1, \neg h_2\}$ est une explication pour cette instance, puisque l'on a $E \subseteq H$, que $\varphi \wedge \bigwedge E$ est satisfiable (comme en témoigne l'affectation $\{\neg h_1, \neg h_2, h_3, x_1, x_2, q\}$), et que $\varphi \wedge \bigwedge E$ implique q (comme en témoigne la dernière clause). Notons que $\{\neg h_1, \neg h_2, h_3\}$ est aussi une explication.

En revanche, $\{\neg h_1, \neg h_2, \neg h_3\}$ n'est pas une explication, puisque l'on a $\neg h_3 \notin H$. L'ensemble $\{\neg h_1, h_2\}$ n'en est pas une non plus, même s'il implique q

conjointement à φ ; en effet, il n'est pas cohérent avec φ. Finalement, l'ensemble $\{h_1, h_3\}$ n'est pas une explication, puisqu'il n'implique pas q conjointement à φ ; cela peut se voir, par exemple, avec l'affectation $\{h_1, h_2, h_3, x_1, x_2, \neg q\}$.

2.3 Restrictions du problème

Dans la littérature, il existe de nombreuses autres formalisations du problème d'abduction. Elles diffèrent en particulier quant à la représentation syntaxique de la requête, aux propriétés imposées à l'ensemble d'hypothèses et à la classe de formules considérée pour les bases de connaissances (KB), ainsi qu'aux propriétés imposées aux explications calculées.

Les restrictions imposées à l'ensemble d'hypothèses H sont importantes. Cet ensemble est souvent restreint à être clos par négation, c'est-à-dire à contenir soit h et $\neg h$, soit aucun des deux, pour toute variable h.

Les études cherchant à identifier des classes polynomiales de KB pour ce problème imposent également diverses restrictions. Par exemple, Selman et Levesque [30] et Eiter et Gottlob [12] considèrent des KB de Horn (en CNF), et Zanuttini [32] considère des classes de formules DNF et affines. Des études ont également classifié la complexité du problème, en fonction du langage de contraintes autorisé pour la KB, pour des restrictions spécifiques sur H et q [21, 6].

Cependant, lorsque la classe précise de la KB n'importe pas, ce qui est le cas lorsque l'on étudie des approches génériques, plusieurs restrictions se réduisent les unes aux autres. En particulier, nous supposons ici que la requête q est un littéral, et ce sans perte de généralité. En effet, si la requête est un terme ou une clause ψ, on peut ajouter des clauses exprimant $q' \leftrightarrow \psi$ à φ en temps polynomial et remplacer la requête ψ par q' (où q' est une nouvelle variable). Notons que la littérature ne considère pas, en général, des requêtes plus complexes (telles que des formules CNF quelconques), puisque les requêtes formalisent habituellement des observations simples sur l'environnement.

Nous supposons également que q n'est pas une hypothèse ($q \notin H$). À nouveau, c'est sans perte de généralité, puisque si $q \in H$, on peut introduire une nouvelle variable q', ajouter des clauses exprimant $q \leftrightarrow q'$ à φ, et remplacer la requête q par q'.

Il est aussi important de noter que nous nous restreignons au problème de décision, c'est-à-dire que nous cherchons une explication quelconque, pas forcément, par exemple, minimale pour l'inclusion. En effet, pour ce dernier critère, on voit qu'il existe une explication minimale si et seulement s'il existe une explication quelconque. Mais il est également facile de voir que la partie difficile est celle consistant à trouver une explication ; la minimisation est en général plus facile (en ce qui concerne la minimalité pour l'inclusion), car elle peut être réalisée de façon gloutonne. Pour une étude détaillée des divers critères de minimalité et certaines implications algorithmiques, nous renvoyons le lecteur à [12].

2.4 Formules booléennes quantifiées

Nous comparons notre approche principalement à celle du projet QUIP, qui utilise une réduction du problème d'abduction au problème QBF, le problème de la validité des formules booléennes quantifiées. Nous définissons ce dernier problème dans la suite.

Une *formule booléenne quantifiée* (QBF) est une formule construite sur un ensemble de symboles propositionnelles avec les connecteurs usuels $\neg$, $\vee$, $\wedge$ et $\rightarrow$ et les quantificateurs $\forall$ et $\exists$ sur les variables. L'ensemble de toutes les variables apparaissant dans une QBF Σ est noté $\mathcal{V}(\Sigma)$, et une variable $x \in \mathcal{V}(\Sigma)$ est dite *libre* dans Σ si et seulement si toutes ses occurrences ne sont pas sous la portée d'un quantificateur.

Une QBF Σ est dite *prénexe* si elle est de la forme $\Sigma = Qx_1 \ldots Qx_n.\phi$, où chaque occurrence de Q désigne $\forall$ ou $\exists$, et ϕ ne contient aucun quantificateur. ϕ est appelée la *matrice* de Σ, et la séquence de quantifications $Qx_1 \ldots Qx_n$ est appelée son *préfixe*. Une formule est dite *fermée* si et seulement si elle n'a aucune variable libre.

Une QBF est dite *satisfiable* si elle a au moins un modèle selon la sémantique usuelle des connecteurs et des quantificateurs, et *valide* si toute interprétation la satisfait. Nous ne manipulons ici que des formules QBF fermées.

Nous utiliserons la formalisation suivante du problème de validité pour les QBF.

Définition 3 (QBF) *Une instance du problème de validité pour les QBF est simplement une QBF Σ. Le problème consiste à décider si Σ est valide.*

3 Méthodes pour résoudre des problèmes d'abduction génériques

3.1 Calcul d'impliqués

Le *consequence-finding* est essentiellement le problème consistant à générer les impliqués premiers d'une formule donnée. Diverses propriétés peuvent être imposées sur les impliqués premiers que l'on cherche à générer ; Marquis [18] donne un état de l'art, notamment sur l'utilisation de la génération d'impliqués premiers pour l'abduction. La caractérisation donnée par Marquis est la suivante [18, Proposition 4.2].

Proposition 4 *Un ensemble E est une explication minimale pour l'inclusion d'une instance (V, H, q, φ)*

du problème d'abduction si et seulement si la clause $\bigvee_{\ell \in E} \overline{\ell}$ *est un impliqué premier de* $\varphi \wedge \neg q$ *qui n'est pas impliqué par* φ *et dont tous les littéraux sont dans* $\overline{H} = \{\overline{\ell} \mid \ell \in H\}$. *Ces impliqués premiers sont appelés les* impliqués $\langle \overline{H}, \varphi \rangle$-premiers de $\neg q$.

Il y a d'autres caractérisations des explications en termes d'impliqués premiers (voir par exemple [26]).

Proposition 5 *Un ensemble* E *est une explication minimale pour l'inclusion d'une instance* (V, H, q, φ) *du problème d'abduction si et seulement si la clause* $(q \vee \bigvee_{\ell \in E} \overline{\ell})$ *est un impliqué premier de* φ.

La caractérisation de Marquis suggère donc de calculer des impliqués $\langle \overline{H}, \varphi \rangle$-premiers de $\neg q$. Celle de Reiter et de Kleer's suggère de calculer des impliqués premiers de φ dont les littéraux sont dans $\overline{H} \cup \{q\}$, puis de ne retenir que ceux qui contiennent q. Cette dernière idée semble moins efficace, mais le problème de génération correspondant a été plus largement étudié dans la littérature.

Cependant, dans ce rapport préliminaire, nous ne comparons pas notre approche à celle basée sur la génération d'impliqués premiers. Cette comparaison est en effet relativement complexe, puisqu'en général, les algorithmes de génération d'impliqués premiers sont conçus et optimisés pour calculer *tous* les impliqués premiers d'une CNF donnée, ce qui n'est pas notre but (cela correspondrait à générer toutes les explications minimales pour un problème d'abduction). En particulier, dans la conception de tels algorithmes l'accent est souvent mis sur des représentations efficaces de très grands ensembles d'impliqués (comme par exemple dans `zres`, de Simon et del Val [31]). Cela est nécessaire pour la génération, mais induit clairement des surcoûts inutiles pour l'abduction.

3.2 Traductions en QBF

Nous comparons notre approche avec celle par traduction en QBF, développée par le projet QUIP à Vienne [11]. L'idée est d'utiliser l'expressivité du problème de validité des QBF pour encoder divers problèmes de raisonnement non-monotone, puis d'utiliser les meilleurs prouveurs QBF pour les résoudre.

Pour l'abduction, l'approche est la suivante. La QBF qui encode une instance du problème d'abduction reflète simplement chaque condition dans la définition de l'abduction. Néanmoins, comme cela est fait dans l'implémentation de `QUIP`, nous restreignons la définition à des hypothèses positives. Mais cette restriction n'induit aucune perte de généralité, puisqu'une hypothèse négative $\neg h$ peut être remplacée par une hypothèse positive h', où h' est une nouvelle variable, du moment que l'on ajoute $h' \leftrightarrow \neg h$ à la KB.

Définition 6 (QBF pour l'abduction)
Soit $\Pi = (V, H, q, \varphi)$ *une instance du problème d'abduction, où* H *ne contient que des littéraux positifs. Notons* $H = \{h_1, \ldots, h_n\}$. *La QBF* Σ_Π *est définie comme :*

$$\exists e_1, \ldots, e_n \left[\begin{array}{l} \exists V \left(\varphi \wedge \bigwedge_{i=1}^{n} e_i \rightarrow h_i \right) \wedge \\ \forall V \left((\varphi \wedge \bigwedge_{i=1}^{n} e_i \rightarrow h_i) \rightarrow q \right) \end{array} \right].$$

Intuitivement, pour tout i, e_i devine si h_i est dans l'explication. En effet, on voit que les contraintes sur les e_i sont de la forme $e_i \rightarrow h_i$; autrement dit, h_i est pris en compte si et seulement si e_i est positif. Les deux termes dans la portée de $\exists e_1, \ldots, e_n$ expriment les conditions 2 et 3 de la définition 1, respectivement.

Egly *et al.* [11] montrent que, pour une instance Π du problème d'abduction, Π a une explication si et seulement si Σ_Π est valide. De fait, cette transformation effectuée, tout prouveur QBF peut être utilisé pour résoudre un problème d'abduction.

Malheureusement, la plupart des prouveurs QBF n'acceptent pas des QBF quelconques, mais requièrent qu'elles soient données sous *forme normale conjonctive prénexe*. Afin d'éviter une explosion en taille de la formule, des transformations en forme normale préservant la structure peuvent être utilisées pour mettre la QBF sous cette forme [24, 8, 10]. Contrairement aux transformations usuelles, qui exploitent les lois de distributivité des connecteurs, ces transformations introduisent de nouvelles variables servant de noms pour des occurrences de sous-formules, et sont polynomiales en la taille de la formule donnée.

Les tests reportés dans [11] ne concernent pas seulement des problèmes d'abduction, mais aussi d'autres problèmes de raisonnement non-monotone. Les auteurs comparent leur outil, baptisé `QUIP`, avec `DeRes` [5], `dlv` [13], `smodels` [20] et `Theorist` [25]. Quatre des cinq jeux d'essais viennent de `Theory-Base` [4], un ensemble de tests classique pour les formalismes non-monotones. Le dernier jeu consiste en des problèmes de diagnostic abductif pour des additionneurs n-bits.

4 Une approche DPLL pour l'abduction

4.1 Présentation

Nous décrivons maintenant notre approche DPLL pour résoudre un problème d'abduction. L'idée de base est d'explorer l'espace de recherche, c'est-à-dire toutes les combinaisons de littéraux de H, à la DPLL. À chaque nœud, soit une hypothèse est éliminée, soit elle est choisie et affectée à vrai dans la base de connaissances. De fait, une explication candidate courante E

est implicitement maintenue par l'algorithme, et la base de connaissances courante représente $KB \wedge \bigwedge E$.

Néanmoins, puisqu'en général les variables ne peuvent pas être toutes affectées de cette manière, il faut des conditions d'arrêt plus complexes que pour la satisfiabilité. Lorsqu'on atteint une feuille, c'est-à-dire lorsque tout littéral de H est soit éliminé, soit affecté et propagé dans la base de connaissances, on décide simplement si la base de connaissances courante est satisfiable et implique la requête. Si c'est le cas, le candidat courant E est une explication. Sinon, un retour en arrière est nécessaire. Cette condition d'arrêt peut donc bénéficier de prouveurs efficaces pour la satisfiabilité.

Nous insistons sur le fait que contrairement au cas du problème de satisfiabilité, chaque hypothèse est soit éliminée, soit choisie à chaque nœud. Il n'y a pas d'affectation à vrai ou faux, et le cas d'une hypothèse apparaissant positivement et négativement dans H est simplement pris en compte par la procédure « éliminer ou affecter ». Pour ce qui est de cette dernière décision, nous testons simplement s'il existe une explication contenant l'hypothèse courante, et sinon, lors du retour en arrière, nous décidons s'il en existe une ne la contenant pas ; autrement dit, nous l'éliminons et passons à l'hypothèse suivante.

Certaines conditions peuvent être testées presque sans surcoût à chaque nœud, et peuvent permettre l'élagage de tout le sous-arbre ayant ce nœud pour racine. Nous en présentons trois.

Tout d'abord, notre algorithme maintient la base de connaissances courante saturée pour la résolution unitaire. Si, à un nœud donné, la clause vide est détectée, on peut en déduire que la formule $KB \wedge \bigwedge E$ courante n'est pas satisfiable, et donc que E ne peut pas être étendue en une explication. Donc, dans ce cas, on peut revenir sur le dernier choix d'hypothèse.

Ensuite, si la base de connaissances courante ne contient plus aucune occurrence de la requête (et même si elle contient des occurrences de sa négation), on peut en déduire qu'il n'existe pas d'explication étendant l'explication courante. En effet, si $KB \wedge \bigwedge E$ ne contient pas d'occurrence de q, alors soit elle est insatisfiable, soit elle est satisfiable mais ses modèles peuvent être étendus avec q ou $\overline{q}$ indifféremment. Dans tous les cas, on voit aisément qu'il n'y a pas d'explication étendant E. En outre, cette condition peut encore être décidée efficacement.

Pour terminer, si la clause unitaire (q) est dans la base de connaissances courante, cela signifie que la formule $KB \wedge \bigwedge E$ implique q. Dans ce cas, on décide si la base de connaissances courante est satisfiable. Si c'est le cas, on a trouvé une explication, et sinon on revient sur le dernier choix d'hypothèse.

4.2 L'algorithme

L'algorithme de base mettant en œuvre les principes présentés ci-dessus est donné sur l'algorithme 1. Il prend en entrée une instance (V, H, q, φ) du problème d'abduction, et retourne, soit une explication pour l'instance, soit $\{\{\}\}$, ce qui signifie qu'il n'existe pas d'explication. La formule propositionnelle φ est supposée en CNF.

Algorithme 1 : Algorithme DPLL pour l'abduction

fonction ABDUCTION
Données : une instance (V, H, q, φ) du problème d'abduction.
Résultat : une explication E pour l'instance, s'il en existe une, $\{\{\}\}$ sinon.
début
  / Cas de base : */*
  si *φ contient une clause vide* **alors**
    / $\varphi \wedge \bigwedge E$ est insatisfiable : */*
    retourner $\{\{\}\}$;
  si $q \notin \mathcal{L}(\varphi)$ **alors**
    / $\varphi \wedge \bigwedge E \wedge \bigwedge E'$ ne peut impliquer q pour aucun E' : */*
    retourner $\{\{\}\}$;
  si *la clause unitaire (q) est dans φ* **alors**
    / $\varphi \wedge \bigwedge E$ implique q : */*
    si *φ est satisfiable* **alors retourner** E;
    retourner $\{\{\}\}$;
  si $H = \{\}$ **alors**
    / Plus de branchement possible : */*
    si *φ est satisfiable et $\varphi \wedge \neg q$ ne l'est pas* **alors**
      retourner E;
    retourner $\{\{\}\}$;
  / Branchement : */*
  $\ell =$ un littéral de H;
  $E = E \cup \{\ell\}$;
  affecter ℓ à *vrai* dans φ;
  saturer φ pour la résolution unitaire;
  $H' = H \setminus \{\ell\}$;
  $res =$ ABDUCTION$((V, H', q, \varphi))$;
  si $res \neq \{\{\}\}$ **alors retourner** res;
  / Brancher sans ℓ : */*
  $E = E \setminus \{\ell\}$;
  annuler la propagation unitaire de ℓ dans φ;
  retourner ABDUCTION*$((V, H', q, \varphi))$*;
fin

L'algorithme est récursif. Chaque appel récursif définit un nouveau nœud dans l'arbre de recherche. Afin de simplifier la présentation, l'explication candidate courante E est supposée stockée dans une variable globale.

Il est important de noter que des règles de simplification usuelles peuvent être utilisées, comme c'est le cas lors de la détection d'une clause vide, mais que d'autres ne peuvent pas. La propagation de littéraux apparaissant dans des clauses unitaires peut aussi être utilisée, puisqu'elle ne change pas l'ensemble de modèles de la base de connaissances. En revanche, on ne peut pas simplifier les littéraux purs. Par exemple, si $V = \{h, q\}$, $\varphi = \{\neg h \vee q\}$ et $H = \{h\}$, la règle des littéraux purs affecterait h à $faux$, et l'explication (unique) $E = \{h\}$ disparaîtrait.

Nous donnons maintenant un exemple du fonctionnement de l'algorithme.

Exemple 7 (suite de l'exemple 2) *Soient à nouveau* $H = \{h_1, \neg h_1, h_2, \neg h_2, h_3\}$ *et*

$$\begin{array}{rl} \varphi = & (h_1 \vee \neg h_2 \vee q) \wedge (h_1 \vee \neg h_2 \vee \neg x_1) \wedge \\ & (h_1 \vee \neg h_2 \vee x_1) \wedge (h_3 \vee x_1 \vee x_2) \wedge \\ & (h_1 \vee h_2 \vee q). \end{array}$$

L'algorithme branche d'abord sur h_1 *à vrai ; on obtient donc* $\varphi = (h_3 \vee x_1 \vee x_2)$*, et puisque cette formule ne contient pas* q*, cette branche est une impasse.*

L'algorithme revient donc sur son choix et branche sur $\neg h_1$*. On obtient alors :*

$$\begin{array}{rl} \varphi = & (\neg h_2 \vee q) \wedge (\neg h_2 \vee \neg x_1) \wedge (\neg h_2 \vee x_1) \wedge \\ & (h_3 \vee x_1 \vee x_2) \wedge (h_2 \vee q). \end{array}$$

Cela ne donne pas de réponse. Il faut donc continuer à descendre dans l'arbre de recherche.

L'ensemble H *est maintenant* $\{h_2, \neg h_2, h_3\}$*. L'algorithme branche sur* h_2 *avec la valeur vrai, et on obtient :*

$$\varphi = (q) \wedge (\neg x_1) \wedge (x_1) \wedge (h_3 \vee x_1 \vee x_2).$$

La formule contient la clause unitaire (q)*, mais n'est pas satisfiable, donc cette branche est une impasse.*

L'algorithme revient donc sur son dernier choix et branche sur $\neg h_2$ *; on obtient alors* $\varphi = (h_3 \vee x_1 \vee x_2) \wedge (q)$*, qui contient la clause unitaire* (q) *et est satisfiable. L'algorithme retourne donc l'explication* $\{\neg h_1, \neg h_2\}$*.*

4.3 Heuristiques

Comme nous l'avons vu, notre approche permet d'utiliser une approche heuristique pour sélectionner l'hypothèse la plus intéressante à chaque nœud. Du fait de la structure de la recherche, une telle heuristique doit être capable de sélectionner une hypothèse qui fait très probablement partie d'une explication, c'est-à-dire dont l'affectation à vrai rend la requête « plus impliquée » mais n'affecte « pas trop » la satisfiabilité de la base de connaissances.

Dans cet esprit, nous avons expérimenté plusieurs heuristiques, et il s'est avéré que la plus efficace est une fonction inspirée de l'heuristique bien connue pour SAT et dite de Jeroslow-Wang [16].

Cette heuristique vise à promouvoir la découverte d'une explication pour le problème en préférant les hypothèses dont la négation apparaît souvent dans des petites clauses avec la requête. L'idée est que, s'il existe une clause $(\neg h \vee q)$ dans la base de connaissances, alors, h est clairement une hypothèse prometteuse. Un raffinement est aussi effectué, qui pénalise les hypothèses dont la négation apparaît également souvent dans les mêmes clauses que la *négation* de la requête. Ensuite, avec des poids plus faibles, les co-occurrences de l'hypothèse avec la requête ou sa négation sont prises en compte. Tous ces poids cherchent donc à évaluer dans quelle mesure brancher sur l'hypothèse considérée « implique plus » la requête. Les égalités sont arbitrées par de petits poids qui visent à préférer les hypothèses qui mèneront moins probablement à l'insatisfiabilité.

Pour résumer, pour chaque hypothèse h nous additionnons les poids suivants, calculés pour toutes les clauses $C \in \varphi$. L'heuristique est statique, c'est-à-dire que les hypothèses sont ordonnées une fois pour toute. $|C|$ désigne la taille de la clause C.

$$\omega_h^C = \begin{cases} +2048/|C| \text{ si } q \in C \text{ et } \neg h \in C \\ -1024/|C| \text{ si } q \in C \text{ et } h \in C \\ -2048/|C| \text{ si } \neg q \in C \text{ et } \neg h \in C \\ +1024/|C| \text{ si } \neg q \in C \text{ et } h \in C \\ -2/|C| \text{ si } h \in C \\ +1/|C| \text{ si } \neg h \in C \\ 0 \text{ si } h, \neg h \notin C \end{cases}$$

Le poids final de chaque hypothèse h dans φ est donc par définition $\sum_{C \in \varphi} \omega_h^C$.

Un autre type d'heuristique que nous avons expérimenté donnait la préférence aux hypothèses proches de la requête dans le graphe d'implication de la base de connaissances. Autrement dit, h était évalué à 1 s'il existait une clause de la forme $(\neg h \vee \cdots \vee q)$, à 2 s'il n'en existait pas mais qu'il y avait deux clauses de forme respective $(\neg h \vee \cdots \vee \ell)$ et $(\bar{\ell} \vee \cdots \vee q)$, etc. Cependant, les expériences menées ont montré que cette heuristique n'était pas aussi efficace que la précédente, qu'elle fût utilisée statiquement ou dynamiquement.

4.4 Autres conditions d'élagage

Dans l'algorithme de base, nous avons donné quelques conditions sans surcoût pour élaguer des parties de l'arbre de recherche. Nous avons également expérimenté des conditions plus coûteuses, que nous décrivons maintenant.

Tout d'abord, nous avons expérimenté un test de satisfiabilité de la formule $\varphi \wedge \bigwedge E$ courante à chaque

nœud de l'espace de recherche, ce qui constitue évidemment une condition plus forte que la détection des clauses vides. Dès que la formule $\varphi \wedge \bigwedge E$ courante est insatisfiable, on peut revenir sur le dernier choix[1]. En poursuivant dans cette direction, nous avons également expérimenté un test à chaque nœud pour déterminer si $\varphi \wedge \bigwedge E$ implique la requête. Si c'est le cas, on peut décider si $\varphi \wedge \bigwedge E$ est satisfiable, ce qui décide alors si E est une explication pour le problème d'abduction.

Ces deux conditions, et leurs combinaisons, ont été expérimentées avec un prouveur SAT, en l'occurrence MINISAT [9]. Il s'est avéré que tester l'implication à chaque nœud induit un surcoût important à chaque nœud, tout en permettant peu d'élagage. En revanche, il s'est avéré que tester la satisfiabilité à chaque nœud accélère la recherche globale, tant en termes de temps que de nombre de nœuds explorés. Les expériences rapportées dans le paragraphe suivant ont donc été menées avec cette condition.

5 Expérimentations

5.1 Cadre expérimental

Les résultats empiriques présentés dans ce paragraphe ont été obtenus sur un PIV 3 GHz muni de 512 Mo de RAM sous Linux.

Nous avons implémenté un prouveur d'abduction appelé `zlas` (version 0.0.6)[2]. Il prend en entrée des problèmes d'abduction dans notre propre format inspiré des formats DIMACS et appelé *ADIMACS*.

Exemple 8 (suite de l'exemple 2) *Soient à nouveau* $H = \{h_1, \neg h_1, h_2, \neg h_2, h_3\}$ *et*

$$\begin{aligned}\varphi = \quad & (h_1 \vee \neg h_2 \vee q) \wedge (h_1 \vee \neg h_2 \vee \neg x_1) \wedge \\ & (h_1 \vee \neg h_2 \vee x_1) \wedge (h_3 \vee x_1 \vee x_2) \wedge \\ & (h_1 \vee h_2 \vee q).\end{aligned}$$

Dans le format DIMACS, les variables sont représentées par des nombres. Posons $h_1 = 1$, $h_2 = 2$, $h_3 = 3$, $x_1 = 4$, $x_2 = 5$, *et* $q = 6$. *Alors ce problème d'abduction dans le format ADIMACS est comme suit :*

```
c Ceci est un commentaire
c #VARIABLES=6, #CLAUSES=5
p cnf 6 5
q 6
h 1 -1 2 -2 3 0
1 -2 6 0
1 -2 -4 0
1 -2 4 0
3 4 5 0
1 2 6 0
```

Notons que la différence avec le format DIMACS classique réside dans les lignes débutant par `q` *et* `h`*, qui représentent respectivement la requête et les hypothèses.*

L'objectif est de comparer `zlas` avec d'autres outils. Plus précisément, nous utilisons `quip_filter` [11], un outil pour construire une QBF à partir d'un problème de raisonnement (ici l'abduction) avec la définition 6, et nous comparons le comportement de `boole` (le prouveur QBF de `QUIP`) avec `zlas`. Nous comparons également, en plus de `boole`, deux des meilleurs prouveurs QBF actuels, `sKizzo` et `2clsQ` [19]. Ces deux derniers prouveurs, contrairement à `boole`, nécessitent une transformation sous forme normale, ce qui est réalisé par `qst`[3]. `qst` est développé par Martina Seidl et Michael Zolda, et il est décrit dans le mémoire de Michael Zolda [33].

Les trois prouveurs QBF sont comme suit :

boole est un prouveur propositionnel basé sur les diagrammes de décision binaires[4] ; l'un de ses avantages est qu'il ne nécessite pas de transformation sous forme normale supplémentaire [11] ;

sKizzo (version 0.8.2) est basé sur une nouvelle technique, appelée skolémisation symbolique, et sur une forme de raisonnement symbolique ; cette approche le démarque des autres prouveurs QBF [1], mais il n'en est pas moins compétitif ;

2clsQ (1re place à la compétition QBF 2006) est un prouveur QBF basé sur un raisonnement sur les clauses binaires [29].

5.2 Jeux d'essais

Tous les jeux d'essais générés le sont à partir d'instances DIMACS écrites pour SAT[5]. Nous réalisons un premier tri afin de ne conserver que les instances satisfiables, puisque ce sont les seules susceptibles d'être intéressantes. Nous travaillons ainsi sur 148 des 237 instances, et 8 des 12 familles de problèmes. Nous utilisons un générateur de problèmes d'abduction, baptisé `agen`, qui génère trois problèmes d'abduction à partir d'une seule instance SAT comme suit :

- choisir une variable pour requête,
- générer trois ensembles différents d'hypothèses formés de 10, 50 ou 100 % des variables,

[1] Cette technique est à rapprocher de techniques des prouveurs QBF comme le test de *vérité triviale*.

[2] Documentation complète et exécutable disponibles sur `www.info.unicaen.fr/~zanutti/zlas`.

[3] Exécutable disponible sur `http://www.kr.tuwien.ac.at/research/eq/qst`.

[4] Le programme, ainsi que le code source, peuvent être téléchargés sur `http://www.cs.cmu.edu/~modelcheck/bdd.html`.

[5] Disponibles sur `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`.

– conserver telle qu'elle la formule CNF comme base de connaissances du problème.

Précisons que les requêtes et les hypothèses ne sont générées qu'à partir de littéraux positifs afin de respecter le formalisme de `QUIP`. De plus, seules les instances que `quip_filter` est capable de filtrer sont conservées, même si `zlas` est capable de les résoudre.

Chaque instance est générée aux formats *ADIMACS* et *QUIP*. Notons que les instances au format *QUIP* sont générées avec l'option `SET DETAIL OFF`. Un total de $101 \times 3 = 303$ instances, à partir de 6 familles de problèmes SAT, sont générées.

5.3 Résultats

Dans le tableau 1, la colonne `Bench` donne le nom du type d'instances DIMACS original pour SAT. La troisième colonne (`Hyp`) est une indication sur la manière dont les hypothèses sont choisies : 00 (resp. 50, 90) signifie que chaque variable a 0 % (resp. 50 %, 90 %) de chances de *ne pas* apparaître dans le problème d'abduction considéré, c'est-à-dire que 100 % (resp. 50 %, 10 %) des variables, exceptée la requête, ont une chance d'apparaître dans l'ensemble d'hypothèses. La dernière colonne indique le nombre de problèmes ayant une explication ou pas d'explication. Il est ainsi possible de déduire, à partir du nombre d'instances (colonne `#inst`), pour combien d'entre elles la solution est inconnue, i.e. combien sont irrésolues par tous les prouveurs.

Enfin, les six colonnes restantes donnent des temps en secondes. Les colonnes Filtre et `qst` donnent juste une indication sur le temps nécessaire respectivement à `quip_filter` pour construire une QBF, et `qst` pour en calculer la forme normale. Finalement, les colonnes `boole`, `sKizzo`, `2clsQ`, et `zlas` donnent les temps cumulés pour les prouveurs `boole`, `sKizzo`, `2clsQ`, et notre prouveur pour résoudre l'ensemble d'instances correspondant. Nous donnons enfin, entre parenthèses, le nombre de timeouts ou problèmes irrésolus, par exemple pour des raisons de manque d'espace mémoire, pour le prouveur correspondant. La limite de temps est fixée à 60 secondes de temps CPU pour chaque prouveur.

Les meilleurs résultats, c'est-à-dire ceux dont le nombre de timeouts ou d'instances irrésolues est le plus petit, sont en caractère gras. `2clsQ` obtient des résultats généraux légèrement meilleurs, puisque son nombre de timeouts est plus petit. Cependant, excepté pour les instances `aim`, notre algorithme est clairement plus efficace que tous les autres testés.

Assez naturellement, nous observons que `zlas` est mois efficace sur les problèmes comportant le plus grand nombre d'hypothèses (i.e. avec une colonne « Hyp » petite). Parmi les 101 instances pour chaque taux d'hypothèses, il nécessite 374.85 secondes pour en résoudre 59 avec toutes les variables dans l'ensemble d'hypothèses, alors qu'il n'a besoin que de 72.79 secondes pour résoudre 94 problèmes lorsque seules 10 % des variables apparaissent dans l'ensemble d'hypothèses. Cette tendance est encore plus frappante pour les problèmes `aim`. En effet, alors que `zlas` peine réellement à les résoudre en général, il s'avère ne pas être loin de devenir le meilleur pour ces instances avec 10 % des variables dans l'ensemble d'hypothèses.

Nous avons aussi calculé des temps moyens. Les résultats sont à nouveau assez encourageants. Alors que `zlas` a besoin de 2.32 secondes en moyenne pour résoudre l'une des 223 instances, `boole` (resp. `sKizzo`, `2clsQ`) nécessitent 3.01 (resp. 4.68, 12.47) secondes pour résoudre l'une des 174 (resp. 162, 233) instances, en considérant les temps moyens dûs au filtre (resp. au filtre (0.09 seconds) et à `qst` (3.05 seconds)).

6 Conclusion et perspectives

Dans cet article, nous avons traité d'un point de vue pratique et expérimental de la résolution de problèmes d'abduction. Nous avons, dans un premier temps, décrit un nouvel algorithme dédié à l'abduction, et présenté une heuristique intéressante. Les résultats expérimentaux rapportés donnent un sens à cette étude et encouragent une recherche dans cette direction.

Nous pouvons constater que, bien que la recherche sur les prouveurs QBF suscite un grand intérêt de nos jours, ils ne sont pas encore aussi efficaces qu'ils le pourraient. Nous pouvons également conclure que les techniques d'encodage ne sont pas assez puissantes. En effet, cette étude tend à montrer l'inefficacité des outils d'encodage et de résolution de QBF pour l'abduction.

Clairement, cette étude n'est pas complète d'un point de vue expérimental. Dans un proche avenir, nous envisageons de comparer notre approche avec des algorithmes de consequence-finding efficaces (cf. la discussion dans cet article). Nous projetons également de comparer ces approches à l'aide d'autres types de jeux d'essais. De même, il apparaît nécessaire de déterminer quels jeux d'essais sont difficiles, ce qui nous semble complexe, étant donné le manque de maturité actuel de la recherche sur les prouveurs d'abduction. Enfin, ces expérimentations auraient besoin d'être approfondies par des comparaisons avec d'autres transformations en des QBF sous forme normale.

Des perspectives concernant notre prouveur sont également envisagées. Nous projetons d'expérimenter de nouvelles heuristiques, en particulier basées sur un apprentissage durant la recherche. Nous essaierons aussi d'utiliser des conditions d'élagage plus sophistiquées, par exemple en utilisant un prouveur SAT moins

			Temps						
Bench	#inst	Hyp	Filtre	`qst`	`boole`	`sKizzo`	`2clsQ`	`zlas`	oui/non
aim	48	00	0.34	28.70	**2.73(0)**	85.38(0)	81.95(0)	270.48(22)	20/28
	48	50	0.30	23.84	**2.46(0)**	71.61(0)	130.50(1)	26.17(14)	21/27
	48	90	0.30	20.91	**2.52(0)**	75.99(0)	337.10(1)	3.26(0)	22/26
	144	all	0.94	73.45	**7.71(0)**	232.98(0)	549.55(2)	299.91(36)	63/81
ii	16	00	5.00	137.41	0(16)	0(16)	**132.68(5)**	0.50(6)	12/0
	16	50	5.00	129.33	8.04(15)	4.36(15)	108.69(9)	**0.41(6)**	10/0
	16	90	5.00	124.24	7.14(15)	1.44(15)	203.67(9)	**17.81(5)**	10/2
	48	all	15.00	390.98	15.18(46)	5.80(46)	445.04(23)	**18.72(17)**	32/2
jnh	16	00	0.92	26.14	0(16)	0(16)	**78.92(2)**	0.05(9)	9/5
	16	50	0.69	22.83	0(16)	0(16)	**301.09(2)**	0.07(5)	12/3
	16	90	0.81	22.46	0(16)	0(16)	281.19(12)	**0.36(0)**	10/6
	48	all	2.42	71.43	0(48)	0(48)	661.20(16)	**0.48(14)**	31/14
par	17	00	1.54	79.00	149.30(8)	3.71(12)	6.11(7)	**103.28(4)**	8/8
	17	50	1.48	60.37	186.92(7)	3.27(12)	132.90(7)	**42.61(5)**	5/8
	17	90	1.51	50.96	148.53(8)	3.23(12)	194.75(8)	**50.55(1)**	5/11
	51	all	4.53	190.33	484.75(23)	10.21(36)	333.76(22)	**196.44(10)**	18/27
ssa	3	00	1.12	72.08	0(3)	0.18(2)	3.81(2)	**0.54(0)**	2/1
	3	50	1.32	59.91	0(3)	0(3)	120.40(1)	**0.89(0)**	3/0
	3	90	1.05	37.14	0(3)	0(3)	59.41(1)	**0.81(0)**	3/0
	9	all	3.49	169.13	0(9)	0.18(8)	183.62(4)	**2.24(0)**	8/1

TAB. 1 – Résultats (temps et nombre de timeouts ou échecs mémoire) résumés par type d'instance sur `boole`, `sKizzo`, `2clsQ`, et `zlas`. Les temps sont donnés en secondes.

souvent mais en conservant son pouvoir élaguant.

Remerciements

Les auteurs tiennent à remercier S. Woltran, M. Seidl, M. Zolda, L. Simon et A. del Val pour leur avoir gracieusement fourni leurs implementations et pour les avoir aidés à les utiliser, et les relecteurs anonymes pour leurs commentaires. Ce travail a bénéficié du soutien de l'IUT de Lens, de l'Université d'Artois et de l'Université de Caen Basse-Normandie.

Références

[1] M. Benedetti. sKizzo : a suite to evaluate and certify QBFs. In *Proc. 20th International Conference on Automated Deduction (CADE'05)*, pages 369–376, 2005.

[2] P. Besnard, T. Schaub, H. Tompits, and S. Woltran. Paraconsistent reasoning via quantified Boolean formulas, I : axiomatising signed systems. In *Proc. Journées Européennes sur la Logique en Intelligence Artificielle (JELIA'02)*, pages 320–331, 2002.

[3] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proc. 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, 1998.

[4] P. Cholewinski, V. W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proc. 12th International Conference on Logic Programming (ICLP'95)*, pages 267–282, 1995.

[5] P. Cholewinski, V. W. Marek, and M. Truszczyński. Default reasoning system DeReS. In *Proc. 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 518–528, 1996.

[6] N. Creignou and B. Zanuttini. A complete classification of the complexity of propositional abduction. *SIAM Journal on Computing*, 36(1) :207–229, 2006.

[7] A. del Val. The complexity of restricted consequence finding and abduction. In *Proc. 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 337–342, 2000.

[8] E. Eder. *Relative complexity of first-order calculi.* Artificial Intelligence. Vieweg Verlag, 1992.

[9] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[10] U. Egly. On different structure-preserving translations to normal form. *Journal of Symbolic Computation*, 22(2) :121–142, 1996.

[11] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified Boolean formulas. In *Proc. 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 417–422, 2000.

[12] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1) :3–42, 1995.

[13] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In *Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, pages 364–375, 1997.

[14] H. Fargier, J. Lang, and P. Marquis. Propositional logic and one-stage decision making. In *Proc. 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, pages 445–456, 2000.

[15] R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified Boolean formulae. In *Proc. 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 285–290, 2000.

[16] R. J. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1 :167–188, 1990.

[17] R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *Proc. 11th Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'02)*, pages 160–175, 2002.

[18] P. Marquis. Consequence finding algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems (DRUMS)*, volume 5, pages 41–145. Kluwer Academic, 2000.

[19] M. Narizzano, L. Pulina, and A. Tacchella. The QBFEVAL web portal. In *Proc. Journées Européennes sur la Logique en Intelligence Artificielle (JELIA'06)*, pages 494–497, 2006.

[20] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proc. 1996 Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, pages 289–303, 1996.

[21] G. Nordh and B. Zanuttini. Propositional abduction is almost always hard. In *Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 534–539. Professional Book Center, 2005.

[22] G. Pan, U. Sattler, and M. Y. Vardi. BDD-based decision procedures for K. In *Proc. 18th International Conference on Automated Deduction (CADE'02)*, pages 16–30, 2002.

[23] G. Pan and M.Y. Vardi. Optimizing a BDD-based modal solver. In *Proc. 19th International Conference on Automated Deduction (CADE'03)*, pages 75–89, 2003.

[24] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3) :293–304, 1986.

[25] D. Poole. Explanation and prediction : An architecture for default and abductive reasoning. *Computational Intelligence*, 5(1) :97–110, 1989.

[26] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems : preliminary report. In *Proc. 6th National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188, 1987.

[27] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10 :323–352, 1999.

[28] J. Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified Boolean formulae. In *Proc. 1st International Conference on Quantified Boolean Formulae (QBF'01)*, pages 84–93, 2001.

[29] H. Samulowitz and F. Bacchus. Binary clause reasoning in QBF. In *Proc. 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 353–367, 2006.

[30] B. Selman and H. J. Levesque. Abductive and default reasoning : a computational core. In *Proc. 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 343–348, 1990.

[31] L. Simon and A. del Val. Efficient consequence finding. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 359–365. Morgan Kaufman, 2001.

[32] B. Zanuttini. New polynomial classes for logic-based abduction. *Journal of Artificial Intelligence Research*, 19 :1–10, 2003.

[33] M. Zolda. Comparing different prenexing strategies for quantified Boolean formulas. Master's thesis, Technische Universität Wien, 2005.