Une approche basée sur SAT pour le problème de satisfiabilité en logique modale S5

Thomas Caridroit Jean-Marie Lagniez Daniel Le Berre
Tiago de Lima Valentin Montmirail *

CRIL, Univ. Artois et CNRS, F62300 Lens, France
{caridroit,lagniez,leberre,delima,montmirail}@cril.fr

Résumé

Nous présentons une approche basée sur SAT pour résoudre le problème de satisfiabilité en logique modale S5. Ce problème étant NP-complet, la traduction en SAT n'est pas une surprise. Notre contribution est de réduire considérablement le nombre de variables propositionnelles ainsi que de clauses nécessaires pour coder le problème. Nous présentons dans un premier temps une propriété syntaxique appelée *diamond degree*. Nous montrons que la taille d'un modèle S5 satisfaisant une formule ϕ peut être bornée par ce *diamond degree*. Une telle mesure peut ainsi être utilisée comme borne supérieure pour générer un codage SAT pour la satisfiabilité S5 de cette formule. Nous proposons ensuite un système de *caching* qui nous permet de réduire la taille de la formule propositionnelle. Nous avons implémenté une approche générique basée sur SAT dans le solveur S52SAT. Celle-ci nous permet de comparer expérimentalement notre nouvelle borne supérieure à celle connue antérieurement, c'est-à-dire le nombre de modalités de ϕ, ainsi que d'évaluer l'effet de notre technique de *caching*. Nous comparons également notre solveur avec des solveurs existants en logique modale S5. L'approche proposée surpasse les précédentes sur les benchmarks utilisés. Ces résultats prometteurs ouvrent des perspectives de recherche intéressantes pour la résolution pratique d'autres logiques modales (par exemple K, KT, S4).

Abstract

We present a SAT-based approach for solving the modal logic S5-satisfiability problem. That problem being NP-complete, the translation into SAT is not a surprise. Our contribution is to greatly reduce the number of propositional variables and clauses required to encode the problem. We first present a syntactic property called diamond degree. We show that the size of an S5-model satisfying a formula ϕ can be bounded by its diamond degree. Such a measure can thus be used as an upper bound for generating a SAT encoding for the S5-satisfiability of that formula. We also propose a lightweight caching system which allows us to further reduce the size of the propositional formula. We implemented a generic SAT-based approach within the modal logic S5 solver S52SAT. It allowed us to compare experimentally our new upper-bound against the previously known one, i.e. the number of modalities of ϕ and to evaluate the effect of our caching technique. We also compared our solver against existing modal logic S5 solvers. The proposed approach outperforms previous ones on the benchmarks used. These promising results open interesting research directions for practical reasoning in other modal logics (e.g. K, KT, S4).

Introduction

Au cours des vingt dernières années, les logiques modales ont été utilisées dans divers domaines de l'intelligence artificielle comme la vérification formelle [10], la théorie des jeux [25], la théorie des bases de données [11] et l'informatique distribuée [38]. Plus récemment, la logique modale S5 a été utilisée pour la planification contingente [29] et la compilation de connaissances [5]. Pour cette raison, le raisonnement automatisé en logiques modales a été largement étudié (par exemple [28, 34, 35]). Ladner [9, 24] a montré que le problème de satisfiabilité pour plusieurs logiques modales comprenant K, KT et S4 est PSPACE-complet alors qu'il est NP-Complet pour la logique S5 (voir [21] pour plus de détails). Puisque les solveurs SAT sont devenus des oracles NP assez efficaces pour de nombreux problèmes, nous nous intéressons à étudier l'encodage SAT pour le problème S5-SAT d'un point de vue pratique. Utiliser un oracle SAT dans le cadre de la logique modale n'est pas nouveau : un aperçu complet des travaux antérieurs peut être trouvé dans [33]. Cependant,

*Papier doctorant : Valentin Montmirail est auteur principal

la plupart d'entre eux s'attaquent à la satisfiabilité de la logique modale K. *SAT [13, 14, 15, 17] utilise un oracle SAT pour décider de la satisfaction de 8 logiques modales différentes, y compris K, mais pas S5. Dans le même esprit que notre travail, une traduction de la logique modale K vers SAT a été proposée dans Km2SAT [18, 34]. Plus récemment, le solveur InKreSAT [23] a proposé un système innovant basé sur SAT où le solveur SAT conduit le développement d'une méthode des tableaux. Sur un plan théorique, une approche basée sur SMT (Satisfiability Modulo Theory) [2] a également été proposée. Aucune de ces méthodes n'est applicable à la logique modale S5.

Le nombre de variables nécessaires pour réduire S5-SAT à SAT dépend d'une borne supérieure du nombre de mondes possibles à considérer dans le modèle S5. Minimiser cette borne supérieure est donc crucial pour obtenir des formules CNF de taille raisonnable. Nous proposons une nouvelle borne supérieure basée sur une propriété syntaxique de la formule que nous appelons *diamond degree*. Nous fournissons quelques preuves expérimentales que notre approche améliore significativement les solveurs S5 de la littérature.

Le reste de l'article est organisé comme suit : nous présentons dans un premier temps la logique modale S5 ainsi que les différentes borne sur la taille d'un modèle S5 ; nous détaillons ensuite la réduction de S5-SAT à SAT, paramétrée par le nombre de mondes à considérer. Nous présentons dans un deuxième temps deux améliorations pour réduire la taille de l'encodage SAT : une meilleure borne supérieure sur le nombre de mondes à considérer ainsi qu'un *caching* structurel. Enfin, nous comparons expérimentalement l'efficacité de notre approche aux solveurs S5 existants.

Préliminaires

Soit $\mathbb{P}$ un ensemble fini non vide de variables propositionnelles. Le langage $\mathcal{L}$ de la logique modale S5 est l'ensemble des formules ϕ définies par la grammaire suivante :

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Box\phi \mid \Diamond\phi$$

où p est une variable de $\mathbb{P}$. Une formule de la forme $\Box\phi$ (*box phi*) signifie que ϕ est nécessairement vrai. Une formule de la forme $\Diamond\phi$ (*diamond phi*) signifie que ϕ est possiblement vrai. Toute formule $\phi \in \mathcal{L}$ peut être convertie en une formule équivalente en forme normale négative (NNF), c'est-à-dire que les négations apparaissent seulement devant les variables propositionnelles (voir la définition dans [32]), notée $\text{nnf}(\phi)$. Cela peut se faire en temps polynomial. Les formules de $\mathcal{L}$ sont interprétées en utilisant des modèles S5 pointés. Une modèle S5 est une paire (W, V), où W est un ensemble non vide de mondes possibles et V est une fonction de valuation de W dans $(\mathbb{P} \to \{0, 1\})$. Un modèle S5 pointé est un triplet (W, V, w), où (W, V) est un modèle S5 et $w \in W$. La relation de satisfaction $\models$ entre des formules du langage $\mathcal{L}$ et des modèles S5 pointés est définie récursivement comme suit :

$(W, V, w) \models \top$
$(W, V, w) \models p$ ssi $V(w)(p) = 1$
$(W, V, w) \models \neg\phi$ ssi $(W, V, w) \not\models \phi$
$(W, V, w) \models \phi_1 \wedge \phi_2$ ssi $(W, V, w) \models \phi_1$ et $(W, V, w) \models \phi_2$
$(W, V, w) \models \phi_1 \vee \phi_2$ ssi $(W, V, w) \models \phi_1$ ou $(W, V, w) \models \phi_2$
$(W, V, w) \models \Box\phi$ ssi pour tout $w' \in W$, nous avons $(W, V, w') \models \phi$
$(W, V, w) \models \Diamond\phi$ ssi il existe un $w' \in W$ tel que $(W, V, w') \models \phi$

La validité et la satisfiabilité sont définies comme d'habitude. Une formule $\phi \in \mathcal{L}$ est valide, noté $\models \phi$, si et seulement si, pour tout modèle S5 pointé (W, V, w), nous avons $(W, V, w) \models \phi$. De plus, ϕ est satisfiable si et seulement si $\not\models \neg\phi$.

Exemple 1 *Voici un exemple d'une formule en logique modale S5* $\phi = (\Diamond(p_1 \wedge \Box p_2) \wedge \Diamond p_3)$ *ainsi qu'un exemple de modèle* (W, V, w) *de tel sorte que* $(W, V, w) \models \phi$

$$W = \{w, v, u\}$$
$$V(w) = \{(p_1, 1), (p_2, 1), (p_3, 0)\}$$
$$V(u) = \{(p_1, 0), (p_2, 1), (p_3, 0)\}$$
$$V(v) = \{(p_1, 0), (p_2, 1), (p_3, 1)\}$$

De S5-SAT à SAT

Il a été montré dans [24] que si une formule S5 ϕ contenant n connecteurs modaux est satisfiable, alors il existe un modèle S5 satisfaisant ϕ contenant au plus $n + 1$ mondes. Nous savons aussi qu'il existe un algorithme s'exécutant en temps polynomial capable de transformer un problème S5-SAT en un problème SAT (vu que S5-SAT est NP-complet [24]). Cependant, à notre connaissance, personne n'a évalué cette approche dans la pratique. Celle-ci n'a pas été comparée aux solveurs de la littérature jusqu'à présent. Pourtant, les approches basées sur SAT sont connues pour être très efficaces en pratique.

Un codage SAT est ici représenté par une fonction de traduction tr, qui prend en entrée une formule S5 ϕ ainsi qu'un nombre de mondes n dans un modèle S5 et qui produit une formule propositionnelle. Ceci est inspiré par la

traduction standard vers la logique du premier ordre [31]. [1]

$$\mathrm{tr}(\phi, n) = \mathrm{tr}'(\mathrm{nnf}(\phi), 1, n)$$
$$\mathrm{tr}'(\top, i, n) = \top$$
$$\mathrm{tr}'(\neg\top, i, n) = \neg\top$$
$$\mathrm{tr}'(p, i, n) = p_i$$
$$\mathrm{tr}'(\neg p, i, n) = \neg p_i$$
$$\mathrm{tr}'((\phi \wedge \cdots \wedge \delta), i, n) = \mathrm{tr}'(\phi, i, n) \wedge \cdots \wedge \mathrm{tr}'(\delta, i, n)$$
$$\mathrm{tr}'((\phi \vee \cdots \vee \delta), i, n) = \mathrm{tr}'(\phi, i, n) \vee \cdots \vee \mathrm{tr}'(\delta, i, n)$$
$$\mathrm{tr}'(\Box\phi, i, n) = \bigwedge_{j=1}^{n} (\mathrm{tr}'(\phi, j, n))$$
$$\mathrm{tr}'(\Diamond\phi, i, n) = \bigvee_{j=1}^{n} (\mathrm{tr}'(\phi, j, n))$$

La traduction ajoute de nouvelles variables booléennes p_i à la formule, désignant la valeur de vérité de p dans le monde w_i. Dans cette fonction, le paramètre i représente l'indice du monde. La fonction est définie sur une formule en forme normale négative (NNF) par souci de simplicité.

Exemple 2 *Soit la formule $\phi = \Diamond(a \wedge \Box b)$, la Figure 1 montre la traduction de la formule ϕ avec $n = 2$.*

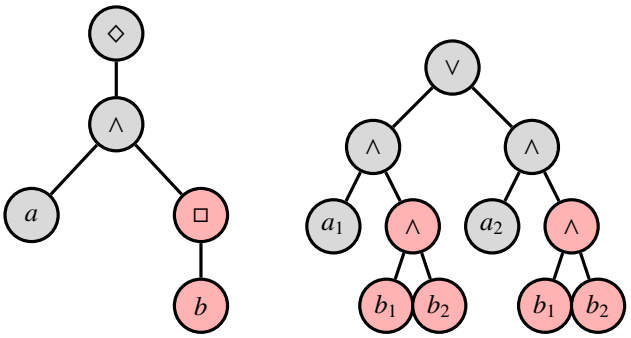


FIGURE 1 – De S5 ϕ (gauche) vers la logique propositionnelle avec n=2 (droite)

Si la valeur de n est une borne supérieure du nombre de mondes dans le modèle, alors la traduction et la formule S5 d'origine sont équi-satisfiables. C'est en particulier le cas pour la borne supérieure montrée dans [24] :

Définition 1 *Soit $\phi \in \mathcal{L}$, $\mathtt{nm}(\phi)$ désigne le nombre de connecteurs modaux contenus dans le formule ϕ.*

Théorème 1 *Une formule ϕ de $\mathcal{L}$ est satisfiable si et seulement si $tr(\phi, \mathtt{nm}(\phi) + 1)$ est satisfiable.*

Démonstration 1 *Le théorème 1 est prouvé de la même façon que la traduction standard vers la logique du premier ordre, plus le Lemme 6.1 de [24].*

1. Notons que la relation d'accessibilité n'a pas besoin d'être représentée dans un modèle S5, puisque celle-ci est une relation d'équivalence.

Notons que le résultat de la traduction n'est pas en CNF. Ainsi, la traduction classique en CNF utilisant l'algorithme de Tseitin [37] est nécessaire pour utiliser un oracle SAT.

Amélioration de la borne supérieure

La taille du codage dépend de $\mathtt{nm}(\phi)$. En pratique, une telle borne supérieure produit des formules déraisonnablement grandes. Une première étape consiste donc à améliorer cette borne supérieure. Nous proposons d'utiliser une nouvelle mesure appelée *diamond degree* comme nouvelle borne supérieure.

Définition 2 (*Diamond degree*) *Le diamond degree d'une formule ϕ de $\mathcal{L}$, noté $\mathtt{dd}(\phi)$, est défini récursivement comme suit :*

$$\mathtt{dd}(\phi) = \mathtt{dd}'(\mathrm{nnf}(\phi))$$
$$\mathtt{dd}'(\top) = 0$$
$$\mathtt{dd}'(\neg\top) = 0$$
$$\mathtt{dd}'(p) = 0$$
$$\mathtt{dd}'(\neg p) = 0$$
$$\mathtt{dd}'(\phi \wedge \psi) = \mathtt{dd}'(\phi) + \mathtt{dd}'(\psi)$$
$$\mathtt{dd}'(\phi \vee \psi) = \max(\mathtt{dd}'(\phi), \mathtt{dd}'(\psi))$$
$$\mathtt{dd}'(\Box\phi) = \mathtt{dd}'(\phi)$$
$$\mathtt{dd}'(\Diamond\phi) = 1 + \mathtt{dd}'(\phi)$$

Le calcul du *diamond degree* nécessite la conversion de ϕ en NNF. Comme mentionné dans la section précédente, cette opération peut être effectuée en temps et espace polynomial. Ainsi, le *diamond degree* de toute formule peut être calculé en temps polynomial. Sans perte de généralité, nous supposerons que ϕ est en NNF et considérerons $\mathtt{dd}'$ au lieu de $\mathtt{dd}$.

De façon informelle, le *diamond degree* représente une borne supérieure sur le nombre de diamants à prendre en compte pour satisfaire la formule. Pour montrer que le *diamond degree* est une borne supérieure valide pour notre codage SAT, nous utiliserons une méthode des tableaux. Nous avons donc besoin de définitions additionnelles. Soient ϕ une formule en NNF et $\mathrm{sub}(\phi)$ l'ensemble des sous-formules de ϕ. Un tableau pour ϕ est un ensemble non vide $T = \{s_0, s_1, \ldots, s_n\}$ tel que $s_i \in T$ est un sous-ensemble de $\mathrm{sub}(\phi)$ et $\phi \in s_0$. De plus, tout ensemble $s \in T$ satisfait les conditions suivantes :

1. $\neg\top \notin s$.
2. si $p \in s$ alors $\neg p \notin s$.
3. si $\neg p \in s$ alors $p \notin s$.
4. si $\psi_1 \wedge \psi_2 \in s$ alors $\psi_1 \in s$ et $\psi_2 \in s$.
5. si $\psi_1 \vee \psi_2 \in s$ alors $\psi_1 \in s$ ou $\psi_2 \in s$.
6. si $\Box\psi_1 \in s$ alors $\forall s' \in T$ nous avons $\psi_1 \in s'$.

7. si $\Diamond\psi_1 \in s$ alors $\exists s' \in T$ tel que $\psi_1 \in s'$.

Lemme 1 *Soit ϕ une formule en NNF. ϕ est satisfiable si et seulement si il existe un tableau pour ϕ.*

Démonstration 2 *Nous pouvons démontrer ce résultat de manière standard. De la gauche vers la droite, nous montrons que chaque règle du tableau préserve sa satisfiabilité. De la droite vers la gauche, nous montrons par induction sur la structure de ϕ que si le tableau pour ϕ existe, alors nous pouvons construire un modèle pour ϕ en utilisant chaque s comme monde possible de ce modèle.*

Lemme 2 *Soit ϕ une formule en NNF. Le nombre d'éléments de l'ensemble T créés en construisant le tableau de ϕ est borné par $\mathsf{dd}'(\phi) + 1$.*

Démonstration 3 *Soit une formule $\psi \in \mathrm{sub}(\phi)$. Soit $g(\psi)$ le nombre d'ensembles s ajouté à T par la présence de ψ. Autrement dit, $g(\psi)$ est le nombre de fois où la condition impliquant l'opérateur $\Diamond$ est déclenchée pour une sous-formule de ψ. Nous montrons que, pour toute formule $\psi \in \mathrm{sub}(\phi)$, nous avons $g(\psi) \leq \mathsf{dd}'(\psi)$.*

Pour ce faire, nous utilisons une induction sur la structure de ψ.

Base d'induction. Nous considérons quatre cas : (1) $\psi = \top$, (2) $\psi = \neg\top$, (3) $\psi = p$ et (4) $\psi = \neg p$. Dans chaque cas, le condition impliquant $\Diamond$ ne sera jamais déclenché par des formules de $\mathrm{sub}(\psi)$. *Ainsi, $g(\psi) = 0 \leq \mathsf{dd}'(\psi)$.*

Étape d'induction. Nous considérons quatre cas :

1. *$\psi = \psi_1 \wedge \psi_2$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute ψ_1 et ψ_2 à s. Par conséquent, $g(\psi)$ est borné par $g(\psi_1) + g(\psi_2)$. Ainsi, $g(\psi)$ est borné par $\mathsf{dd}'(\psi_1) + \mathsf{dd}'(\psi_2)$ (d'après l'hypothèse d'induction). D'où $g(\psi) \leq \mathsf{dd}'(\psi)$.*
2. *$\psi = \psi_1 \vee \psi_2$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute soit ψ_1 soit ψ_2 à s. Par conséquent, $g(\phi)$ est borné par $\max(g(\psi_1), g(\psi_2))$. Ce dernier est borné par $\max(\mathsf{dd}'(\psi_1), \mathsf{dd}'(\psi_2))$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*
3. *$\psi = \Box\psi_1$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, l'algorithme ajoute ψ_1 à tout $s' \in T$. $g(\psi)$ est donc borné par $g(\psi_1)$, qui est borné par $\mathsf{dd}'(\psi_1)$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*
4. *$\psi = \Diamond\psi_1$. Supposons que $\psi \in s$, pour un certain $s \in T$. Dans ce cas, s'il n'existe aucun s' contenant ψ_1, alors l'algorithme ajoute un nouvel ensemble s'', contenant ψ_1, à T. Par conséquent, $g(\psi)$ est borné par $1 + g(\psi_1)$. Ce dernier est borné par $1 + \mathsf{dd}'(\psi_1)$ (d'après l'hypothèse d'induction). Ainsi $g(\psi) \leq \mathsf{dd}'(\psi)$.*

Nous avons donc $|T| = 1 + g(\phi) \leq 1 + \mathsf{dd}'(\phi)$.

Ainsi, pour toute formule $\phi \in \mathcal{L}$, chaque ensemble s_i du tableau T correspond à un monde $w_i \in W$ dans le modèle S5. $|T| \leq \mathsf{dd}(\phi) + 1$ signifie que le nombre de mondes dans le modèle S5 est borné par $\mathsf{dd}(\phi) + 1$.

Théorème 2 *Une formule $\phi \in \mathcal{L}$ est satisfiable si et seulement si $tr(\phi, \mathsf{dd}(\phi) + 1)$ est satisfiable.*

Caching structurel

Le *caching* est un moyen classique d'éviter du travail redondant. *SAT effectue le *caching* en utilisant une « matrice de bits » [16]. L'implémentation efficace des bibliothèques de BDD [6] repose également sur le *caching*, pour construire un graphe explicite. Ces deux exemples nécessitent plus de temps et d'espace pour la recherche et la mise en cache de travaux exécutés ultérieurement. Ici, notre technique est un compromis « simple mais efficace ». Il ne mémorise pas le travail, de sorte qu'il ne peut pas mettre en cache toutes les formules possibles, mais il ne nécessite qu'un indicateur pour détecter le travail redondant. En d'autres termes, nous ne générons pas la formule redondante, mais grâce à un indicateur, nous sommes capable de détecter les sous-formules redondantes.

Dans l'exemple représenté sur figure 2.b, la sous-formule $(b_1 \wedge b_2)$ apparaît deux fois. La traduction du premier diamant crée deux sous-formules $(a_1 \wedge \Diamond b)$ et $(a_2 \wedge \Diamond b)$, où chaque $\Diamond b$ doit être traduit.

Puisque nous sommes en S5 (tous les mondes sont connectés), les traductions de $\Diamond b$ sur des mondes différents sont équivalentes, nous pouvons donc réutiliser la même sous-formule. Cela signifie qu'au lieu d'utiliser un arbre nous pouvons travailler avec un DAG, ce qui permet une traduction en CNF plus efficace.

Lemme 3 $\mathrm{tr}'(\circ\phi, i, n) = \mathrm{tr}'(\circ\phi, j, n)\ \forall i, j$ *et* $\circ \in \{\Diamond, \Box\}$

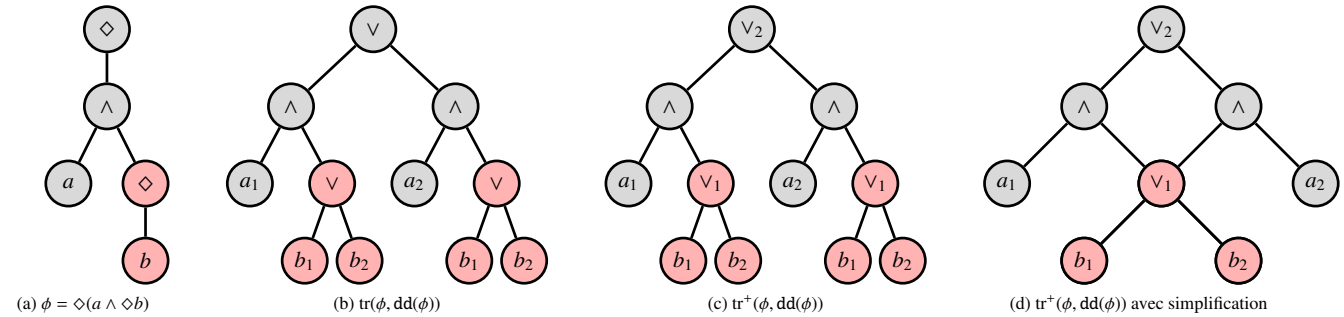
Démonstration 4 *nous avons deux cas à considérer :*

— *Si ($\circ = \Box$), alors* $\mathrm{tr}'(\Box\phi, i, n) = \bigwedge_{k=1}^{n}(\mathrm{tr}'(\phi, k, n))$

— *Si ($\circ = \Diamond$), alors* $\mathrm{tr}'(\Diamond\phi, i, n) = \bigvee_{k=1}^{n}(\mathrm{tr}'(\phi, k, n))$

Dans chaque cas, le résultat est indépendant de i, ainsi choisir j comme indice donne le même résultat.

De façon informelle, le Lemme 3 montre le fait que, quelle que soit la façon dont est incorporée la sous-formule modale, sa traduction donnera toujours le même résultat (indépendamment de l'indice *i*). Par conséquent, nous pouvons commencer par traduire la formule la plus profonde, marquer le nœud correspondant puis revenir en arrière. La formule résultante peut contenir plusieurs nœuds ayant le même marquage. Cela signifie que ces sous-formules sont syntaxiquement identiques (voir Figure 2.c). Ensuite, nous ne conservons qu'une occurence de la sous-formule, tranformant l'arbre en un DAG (voir Figure 2.d). Le *caching*



(a) $\phi = \Diamond(a \wedge \Diamond b)$ (b) $\mathrm{tr}(\phi, \mathrm{dd}(\phi))$ (c) $\mathrm{tr}^+(\phi, \mathrm{dd}(\phi))$ (d) $\mathrm{tr}^+(\phi, \mathrm{dd}(\phi))$ avec simplification

FIGURE 2 – Traduction de $\Diamond(a \wedge \Diamond b)$ (a), traduction initiale (b), formule marquée (c), traduction finale (d)

structurel est donc effectuée à la volée avant de traduire en CNF. La fonction de traduction utilisant cette technique est notée tr^+.

Expérimentations

Nous avons considéré LCKS5TabProver [1] et SPASS 3.7 [39] qui sont, à notre connaissance, à la pointe de la résolution du problème de satisfiabilité en logique modale S5. Nous les avons comparés à quatre configurations différentes de S52SAT : `nm`, `nm+` (avec *caching*), `dd`, `dd+` (avec *caching*) (`http://www.cril.univ-artois.fr/~montmirail/s52SAT`). Afin d'évaluer rigoureusement les solveurs, nous avons utilisé la même entrée pour chacun d'eux, au format InToHyLo [22]. Pour ce faire, nous avons modifié le code de LCKS5TabProver pour le rendre capable de lire ce format. Les modifications sont assez simples pour garantir que les performances du solveur ne sont pas affectées.

Nous avons utilisé SPASS 3.7 au lieu de la dernière version (3.9) car le premier lit le format `dfg`, qui peut être produit à partir de benchmarks InToHyLo en utilisant l'outil `ftt` (Fast Transformation Tool) intégré dans Spartacus [19]. La différence entre 3.7 et 3.9 est minimale selon le site web du solveur. Le temps de traduction est négligeable dans nos expérimentations. Nous avons également considéré MetTel2 [36] et LoTREC [12] pour les solveurs de l'état de l'art en logique modale S5, mais ils ne sont malheureusement pas conçus pour résoudre efficacement les problèmes de logique S5. Nous utilisons Glucose 4.0 [3] comme solveur SAT back-end.

Nous avons évalué ces solveurs sur des benchmarks bien établis de la logique modale : $\mathrm{3CNF}_K$ [30], MQBF_K [26], $\mathrm{TANCS2000}_K$ [27] et $\mathrm{LWB}_{K,KT,S4}$ [4].

Notons qu'ils sont conçus pour les logiques modales K, KT et S4. Par conséquent, certains d'entre eux sont triviaux dans le cadre S5. Toutefois, nous pensons que les résultats obtenus sur ces benchmarks demeurent importants. S5-SAT implique K, KT et S4-SAT, ainsi nous pourrions envisager de résoudre la satisfiabilité en S5 comme étape de prétraitement pour ces logiques modales.

Nous avons fixé la limite de mémoire à 8Go et la limite d'exécution à 900 secondes. Nous avons rarement atteint ce délai (804 fois sur 12444 essais). La plupart des benchmarks non résolus sont dus au manque de mémoire.

Dans les tables suivantes, nous fournissons le nombre de benchmarks résolus, en gras les meilleurs résultats d'une rangée/colonne donnée (selon l'orientation du tableau) ; et entre parenthèses, nous fournissons le nombre de benchmarks qui ne peuvent être résolus à cause du manque de mémoire.

3CNF_K : le *caching* n'aide pas

Solveur	d=2	d=4	d=6	Total
LckS5TabProver	0 (17)	0 (29)	0 (40)	0
S52SAT `nm`	21 (24)	0 (45)	0 (45)	21
S52SAT `nm+`	21 (24)	0 (45)	0 (45)	21
S52SAT `dd`	**45 (0)**	**8 (27)**	0 (45)	**53**
S52SAT `dd+`	**45 (0)**	**8 (27)**	0 (45)	**53**
SPASS 3.7	**45 (0)**	5 (40)	0 (45)	50

TABLE 1 – Nombre d'instances résolues en $\mathrm{3CNF}_K$

Les résultats sont affichés dans la table 1. `dd`, `dd+` et SPASS sont assez proches les uns des autres. Les formules se composent de grandes conjonctions où chaque conjoint a une profondeur modale d'au plus 2, 4 ou 6. Elles sont construites de telle manière que notre algorithme de *caching* ne trouve pas de redondances sur ces formules. C'est pourquoi le *caching* ne fournit aucun avantage sur ces benchmarks.

modKSSS et modKLadn : le *caching* aide

n représente le nombre de variables et a représente le nombre d'alternances dans le préfixe QBF d'origine, voir [26] pour plus de détails, et # correspond au nombre de benchmarks disponible pour une paire (n,a) donnée.

n,*a*	#	LckS5...	nm	nm+	dd	dd+	SPASS 3.7
4,4	40	0 (12)	32	**40**	**40**	**40**	**40**
4,6	40	0 (23)	32	**40**	**40**	**40**	32 (8)
8,4	40	0 (16)	32	**40**	39 (1)	**40**	16 (24)
8,6	40	0 (17)	24	**40**	**40**	**40**	10 (30)
16,4	40	0 (10)	22	**40**	**40**	**40**	8 (32)
16,6	40	0 (16)	19	**40**	39 (1)	**40**	2 (38)
total	240	0	161	**240**	238	**240**	108
4,4	40	**40**	0 (40)	**40**	0 (40)	**40**	0 (40)
4,6	40	14 (0)	0 (40)	32 (8)	0 (40)	**40**	0 (40)
8,4	40	2 (0)	0 (40)	8 (32)	0 (40)	**40**	0 (40)
8,6	40	0 (0)	0 (40)	0 (40)	0 (40)	**8** (32)	0 (40)
16,4	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
16,6	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
total	240	56	0	80	0	**128**	0

TABLE 2 – Au-dessus : modKSSS | En-dessous : modKLadn

Dans cette catégorie, nous pouvons voir que dd et dd+ sont beaucoup plus efficaces que SPASS. Les formules de ces benchmarks contiennent beaucoup de redondances : nous pouvons voir que l'utilisation du *caching* nous permet de résoudre tous les benchmarks modKSSS par exemple.

TANCS-2000 : Coûteux en mémoire

n,*a*	#	LckS5...	nm	nm+	dd	dd+	SPASS 3.7
4,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
4,6	40	0 (33)	**40**	**40**	**40**	**40**	**40**
8,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
8,6	40	0 (39)	**40**	**40**	**40**	**40**	**40**
16,4	40	0 (40)	**40**	**40**	**40**	**40**	**40**
16,6	40	0 (40)	**40**	**40**	**40**	**40**	35 (5)
total	240	0	**240**	**240**	**240**	**240**	235
4,4	40	23 (0)	3 (37)	**40**	**40**	**40**	**40**
4,6	40	3 (0)	0 (40)	**40**	**40**	**40**	31 (9)
total	80	26	3	**80**	**80**	**80**	71

TABLE 3 – TANCS-2000. Au-dessus : qbfMS | En-dessous : qbfML

Ici, nous pouvons voir que ces problèmes peuvent être résolus par presque tous les solveurs. Les instances non résolues par SPASS 3.7 le sont à cause du manque de mémoire. Comme indiqué dans la section suivante, en augmentant la limite mémoire à 32Go, ces instances sont résolues par SPASS.

LWB K, KT, S4 : dd et le *caching* aident

Les benchmarks sont à l'origine séparés sur les formules SAT/UNSAT pour les logiques K, KT et S4. Évidemment, la satisfaction dans S5 peut différer, nous avons donc supprimé la séparation SAT/UNSAT. Les résultats sont affichés dans la table 4. Là encore, dd+ est légèrement meilleur que SPASS. Les benchmarks qui ne peuvent être résolus sont un encodage spécifique en logique modale du principe de *pigeon hole* [20] dans la logique correspondante.

Solveur	Logique K	Logique KT	Logique S4	Total
LckS5TabProver	227 (73)	206 (102)	194 (111)	627
S52SAT nm	307 (66)	344 (33)	292 (86)	943
S52SAT nm+	351 (21)	363 (12)	349 (28)	1063
S52SAT dd	333 (40)	355 (23)	337 (40)	1025
S52SAT dd+	**357 (12)**	**364 (12)**	**363 (12)**	**1084**
SPASS 3.7	343 (16)	363 (15)	360 (17)	1066

TABLE 4 – Nombre d'instances résolues dans $LWB_{K,KT,S4}$

Résultats globaux sur tous les benchmarks

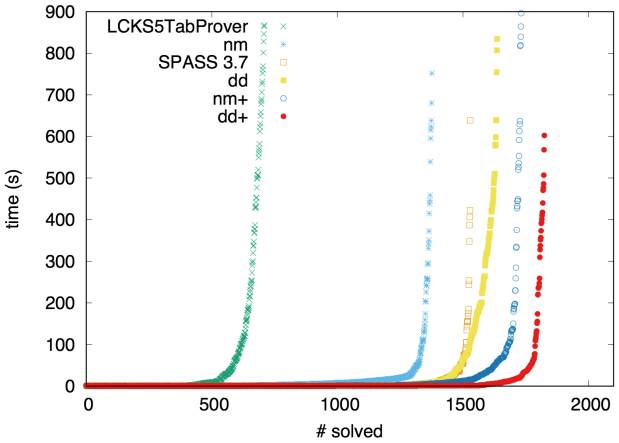


FIGURE 3 – Distribution des temps d'exécution

Solveur	# résolus	# SAT	MO	TO
LckS5TabProver	709	143	710	655
S52SAT nm	1377	411	667	30
S52SAT nm+	1733	452	292	49
S52SAT dd	1645	433	412	17
S52SAT dd+	**1834**	**460**	**203**	37
SPASS 3.7	1530	451	528	**16**

TABLE 5 – Résultats globaux sur tous les benchmarks

SPASS est moins rapide, en partie en raison de ses mauvais résultats sur les benchmarks modKSSS et modKLadn. Bien que l'encodage SAT par défaut épuise souvent la mémoire disponible, chacune des deux améliorations proposées réduit considérablement le nombre de *memory-out*, les meilleurs résultats étant obtenus lorsque les deux sont activés.

Il semble assez clair que le *caching* est essentiel dans notre approche pour résoudre efficacement ces benchmarks. Ceci est attesté par le diagramme de dispersion de la figure 4. L'axe des abscisses correspond au temps utilisé par dd et l'axe des ordonnées correspond au temps utilisé par dd+ pour résoudre le problème.

La principale raison de l'amélioration est la réduction de la taille du codage CNF, comme le montre la table 6. Nous

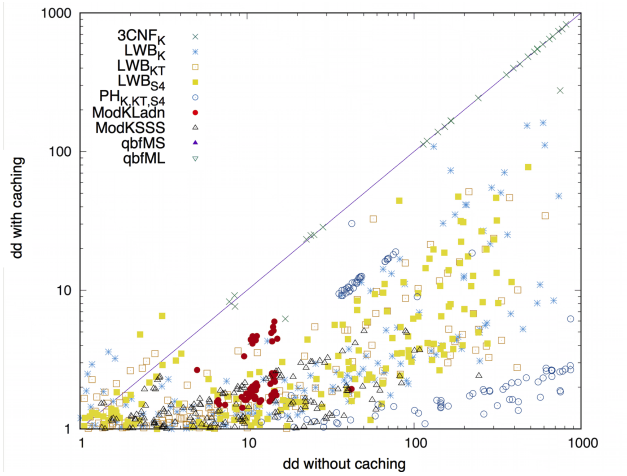


FIGURE 4 – Temps d'exécution de dd avec et sans *caching*

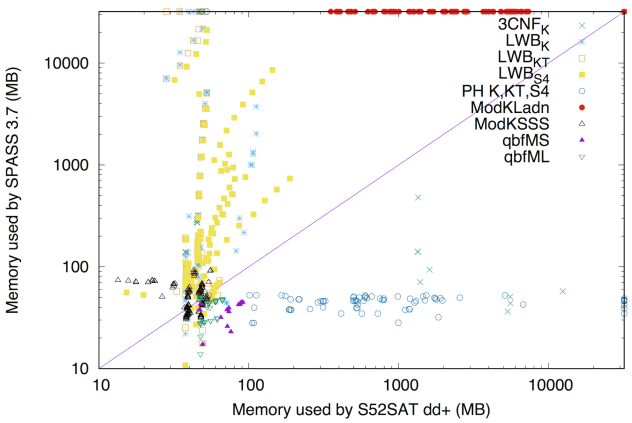


FIGURE 5 – Comparaison (32 Go) SPASS 3.7 vs dd+

Solveur	moy	médiane	max
nm	6 881 821	1 100 054	58 653 264
nm+	1 619 923	118 040	29 492 779
dd	2 385 515	169 324	55 813 557
dd+	269 891	27 090	22 914 442

TABLE 6 – Nombre de clauses dans les formules CNF générées

avions une valeur médiane de 1 100 054 clauses pour nm, et cette valeur tombe à 27 090 pour dd+ sur exactement les mêmes problèmes.

Importance de la mémoire

Nous avons répété les expérimentations avec 32Go de RAM. Notons que pour l'approche SAT, le manque de mémoire se produit pendant la phase de traduction, tandis que pour les autres, la mémoire est épuisée dans la phase de résolution.

Nous nous demandions ce que serait la performance avec plus de mémoire. La table 7 résume le nombre de problèmes résolus avec 8Go et 32Go par chaque solveur. Fournir 32Go à SPASS ne change pas les résultats de manière significative : il a résolu 30 instances supplémentaires. Notre approche SAT tire avantage de cette quantité de mémoire accrue, jusqu'à 98 benchmarks supplémentaires peuvent être résolus par nm sans *caching*. 32Go sont suffisants pour LckS5TabProver (sans *memory-out*), mais seulement une instance supplémentaire peut être résolue.

Solveur	8Go	32Go	MO avec 32Go
LckS5TabProver	709	710	0
SPASS 3.7	1530	1560	498
S52SAT nm	1377	1475	382
S52SAT nm+	1733	1800	166
S52SAT dd	1645	1672	317
S52SAT dd+	1834	1888	96

TABLE 7 – Nombre d'instances résolues avec 8Go et 32Go

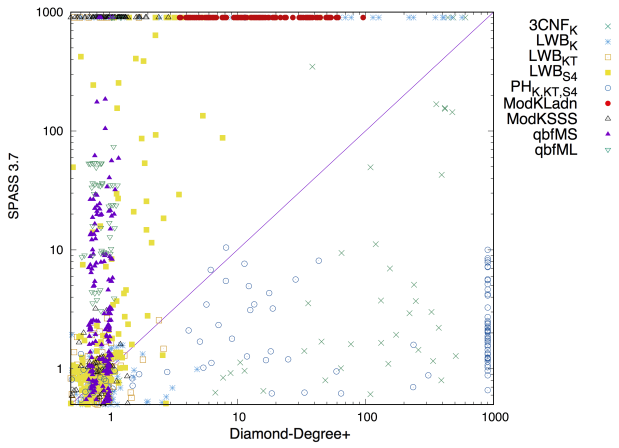


FIGURE 6 – Temps d'exécution de SPASS 3.7 vs. dd+

La figure 5 compare la consommation de mémoire de SPASS 3.7 et S52SAT dd+ en Go. SPASS nécessite généralement plus de mémoire que dd+. Le temps d'exécution et la consommation de mémoire sont légèrement en corrélation (le coefficient de corrélation de Pearson pour dd+ est égal à 0.58 et celui de SPASS est de 0.57). Dans la catégorie modKLadn, SPASS manque de mémoire, même avec 32Go.

Une perspective intéressante ici serait d'établir une heuristique qui détermine, avant même de réaliser la traduction, que la génération de la CNF va dépasser la limite de mémoire autorisée. Une telle heuristique pourrait être '$(\text{len}(\phi) \times \text{dd}(\phi)) > \mathit{BIGNUMBER}$'. On pourrait ensuite

faire le choix par exemple, de ne prendre que $\frac{\mathrm{dd}(\phi)}{2}+1$ comme borne supérieure, si nous trouvons une solution, c'est une solution de la formule complète, sinon il faudrait augmenter au fur et à mesure le nombre de monde autorisée jusqu'à arriver soit à une solution, soit au MO qui aurait du se produire sans l'heuristique. Une telle approche se rapprocherait très fortement des approches CounterExample Guided Abstration Refinement (CEGAR) [8].

Comparaison avec la littérature

La figure 6 montre une comparaison détaillée du temps d'exécution entre dd+ et SPASS sur tous les benchmarks. Dans la plupart des cas, notre approche surpasse SPASS. Les deux cas où elle est moins efficace sont des problèmes *pigeon hole* UNSAT combinatoires durs et des benchmarks dits « 3CNF » pour lesquels ni le *diamond degree* ni le *caching* ne sont utiles.

Conclusion

Nous avons présenté un nouvel encodage SAT pour résoudre le problème S5-SAT en utilisant un solveur SAT. Il est basé sur une réduction de S5-SAT à SAT avec deux améliorations : une meilleure borne supérieure sur le nombre de mondes requis et un *caching* structurel. Nous avons comparé notre approche aux solveurs représentant, à notre connaissance, l'état de l'art pour la résolution pratique de S5-SAT, sur un large éventail de benchmarks. L'approche basée sur SAT avec toutes les améliorations a permis de surclasser tous ces solveurs.

Même si les benchmarks peuvent ne pas être représentatifs de l'utilisation de S5 en pratiques puisqu'ils proviennent d'autres logiques modales (K, KT, S4), ces résultats ouvrent des perspectives intéressantes. En effet, prouver la satisfiabilité d'une formule de la logique modale S5 implique que la formule est également satisfiable dans des systèmes moins restrictifs (c'est-à-dire dans K, KT, S4). Puisque notre solveur S5 fournit un modèle S5 en quelques secondes (temps médian de 2,06s), nous pourrions parfaitement l'utiliser comme une étape de prétraitement pour résoudre les benchmarks dans d'autres logiques modales.

Les résultats préliminaires dans cette direction sont assez encourageants : sur 276 benchmarks satisfiables en logique S5 (donc en K), S52SAT surpasse Spartacus [19] sur 158 d'entre eux. Une autre perspective serait d'adapter S52SAT afin de résoudre les problèmes KD45-SAT étant donné que KD45 est également NP-complet [9].

Remerciements

Nous remercions Renate Schmidt pour son aide dans le recherche de solveurs S5. Une partie de ces travaux ont étés soutenues par le Ministère de l'Enseignement Supérieur et de la Recherche et le Conseil Régional Nord-Pas de Calais par le biais du Contrat de Plan d'Epargne (CPER) ainsi que par une subvention EC FEDER.

Informations

Ces travaux ont déjà étés publiés à la conférence AAAI'17 [7].

Références

[1] Pietro Abate, Rajeev Goré, and Florian Widmann. Cut-free single-pass tableaux for the logic of common knowledge. In *Workshop on Agents and Deduction at TABLEAUX'07*, 2007.

[2] Carlos Areces, Pascal Fontaine, and Stephan Merz. Modal Satisfiability via SMT Solving. In *Software, Services, and Systems - Essays Dedicated to Martin Wirsing*, pages 30–45, 2015.

[3] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[4] Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *J. Autom. Reasoning*, 24(3) :297–317, 2000.

[5] Meghyn Bienvenu, Hélène Fargier, and Pierre Marquis. Knowledge Compilation in the Modal Logic S5. In *Proc. of AAAI'10*, 2010.

[6] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8) :677–691, 1986.

[7] Thomas Caridroit, Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima, and Valentin Montmirail. A SAT-based Approach For Solving The Modal Logic S5-Satisfiability Problem. In *Proc. of AAAI'17*, 2017.

[8] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5) :752–794, 2003.

[9] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[10] Matt Fairtlough and Michael Mendler. An Intuitionistic Modal Logic with Applications to the Formal Verification of Hardware. In *Proc. of CSL'94*, pages 354–368, 1994.

[11] Melvin Fitting. Modality and Databases. In *Proc. of TABLEAUX'00*, pages 19–39, 2000.

[12] Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade. LoTREC : Logical Tableaux

Research Engineering Companion. In *Proc. of TABLEAUX'05*, pages 318–322, 2005.

[13] Enrico Giunchiglia, Fausto Giunchiglia, and Armando Tacchella. The SAT-Based Approach for Classical Modal Logics. In *Proc. of AI*IA'99*, pages 95–106, 1999.

[14] Enrico Giunchiglia, Roberto Sebastiani, Fausto Giunchiglia, and Armando Tacchella. SAT vs. Translation based decision procedures for modal logics : a comparative evaluation. *Journal of Applied Non-Classical Logics*, 10(2) :145–172, 2000.

[15] Enrico Giunchiglia and Armando Tacchella. System description : *SAT : A platform for the development of modal decision procedures. In *Proc. of CADE'00*, pages 291–296. Springer, 2000.

[16] Enrico Giunchiglia and Armando Tacchella. A Subset-Matching Size-Bounded Cache for Testing Satisfiability in Modal Logics. *Ann. Math. Artif. Intell.*, 33(1) :39–67, 2001.

[17] Enrico Giunchiglia, Armando Tacchella, and Fausto Giunchiglia. SAT-Based Decision Procedures for Classical Modal Logics. *J. Autom. Reasoning*, 28(2) :143–171, 2002.

[18] Fausto Giunchiglia and Roberto Sebastiani. Building Decision Procedures for Modal Logics from Propositional Decision Procedures : The Case Study of Modal K(m). *Inf. Comput.*, 162(1-2) :158–178, 2000.

[19] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus : A Tableau Prover for Hybrid Logic. *Electr. Notes Theor. Comput. Sci.*, 262 :127–139, 2010.

[20] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(0) :297 – 308, 1985.

[21] Joseph Y. Halpern and Leandro Chaves Rêgo. Characterizing the NP-PSPACE Gap in the Satisfiability Problem for Modal Logic. In *Proc. of IJCAI'07*, pages 2306–2311, 2007.

[22] Guillaume Hoffmann. *Tâches de raisonnement en logiques hybrides*. PhD thesis, Henri Poincaré University, Nancy, 2010.

[23] Mark Kaminski and Tobias Tebbi. InKreSAT : Modal Reasoning via Incremental Reduction to SAT. In *Proc. of CADE'13*, pages 436–442, 2013.

[24] Richard E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM J. Comput.*, 6(3) :467–480, 1977.

[25] Emiliano Lorini and François Schwarzentruber. A Modal Logic of Epistemic Games. *Games*, 1(4) :478–526, 2010.

[26] Fabio Massacci. Design and Results of the TABLEAUX-99 Non-classical (Modal) Systems Comparison. In *Proc. of Tableaux'99*, pages 14–18, 1999.

[27] Fabio Massacci and Francesco M. Donini. Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. In *Proc. of Tableaux'00*, pages 52–56, 2000.

[28] Fabio Masssacci. Single step tableaux for modal logics : Methodology, computations, algorithms. *Journal of Autom.*, 24(3) :319–364, 2000.

[29] Alexandre Niveau and Bruno Zanuttini. Efficient Representations for the Modal Logic S5. In *Proc. of IJCAI'16*, pages 1223–1229, 2016.

[30] Peter F. Patel-Schneider and Roberto Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *J. Artif. Intell. Res.*, 18 :351–389, 2003.

[31] Johan van Benthem Patrick Blackburn and Frank Wolter. *Handbook of Modal Logic*. Elsevier, 2007.

[32] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[33] Roberto Sebastiani and Armando Tacchella. SAT Techniques for Modal and Description Logics. *Handbook of Satisfiability*, 185 :781–824, 2009.

[34] Roberto Sebastiani and Michele Vescovi. Automated Reasoning in Modal and Description Logics via SAT Encoding : the Case Study of K(m)/ALC-Satisfiability. *JAIR*, 35(1) :343, 2009.

[35] Roberto Sebastiani and Adolfo Villafiorita. SAT-Based Decision Procedures for Normal Modal Logics : A Theoretical Framework. In *Proc. of AIMSA'98*, pages 377–388, 1998.

[36] Dmitry Tishkovsky, Renate A. Schmidt, and Mohammad Khodadadi. The Tableau Prover Generator MetTeL2. In *Proc. of JELIA'12*, pages 492–495, 2012.

[37] G. S Tseytin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer, 1983.

[38] Tom Murphy VII, Karl Crary, and Robert Harper. Distributed Control Flow with Classical Modal Logic. In *Proc. of CSL'05*, pages 51–69, 2005.

[39] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *Proc. of CADE'09*, pages 140–145, 2009.