MOLOSS, un solveur pour la satisfiabilité en logique modale

Rémi Delmas[1] **Christophe Garion**[2] **Josselin Giet**[3]

[1] ONERA Toulouse, Toulouse, France
[2] ISAE-SUPAERO, Université de Toulouse, Toulouse, France
[3] École Normale Supérieure, Paris, France
remi.delmas@onera.fr garion@isae-supaero.fr josselin.giet@ens.fr

Résumé

Cet article présente MOLOSS, un solveur pour la satisfiabilité en logique modale. MOLOSS implémente et étend le travail théorique d'Aceres et al. [1] dans lequel les auteurs définissent une procédure de décision basée SMT, pour les logiques modales. Cette procédure traduit classiquement une formule de la logique modale en une formule du premier ordre et instancie les quantificateurs de la formule résultante pour vérifier sa satifaisabilité en utilisant un solveur SAT. Notre implémantation permet de comparer la procédure d'Aceres et al. avec une autre dans laquelle les quantificateurs de la formule du premier ordre sont gérés directement par le solveur SMT utilisé.

Abstract

This paper introduces MOLOSS, a solver for satisfiability checking for modal logics. MOLOSS implements and extends theoretical work by Aceres et al. [1] in which the authors define an SMT-based decision procedure for modal logics. This procedure classically translates a modal logic formula into a first-order logic formula and instantiates the quantifiers in the resulting formula to check its satisfiability using a SAT solver. Our implementation allows to compare the procedure of Aceres et al. with another one in which the quantifiers of the first-order formula are directly managed by the SMT solver.

1 Introduction

Les logiques modales propositionnelles sont des extensions de la logique propositionnelle qui introduisent des opérateurs représentant la nécessité, la croyance, le temps, l'obligation etc. Pour vérifier la satisfiabilité de formules de telles logiques, on trouve trois types de méthodes :

— des méthodes par tableaux comme InKreSAT [15], FaCT++ [28], LoTREC [11], MetTeL2 [26], LWB [2]
— des méthodes par traduction du problème de satisfiabilité vers un problème de satisfiabilité dans une autre logique, comme S52SAT [9], Km2SAT [27], *SAT [13], KCSP [7], QMRES [25] ou MoSaiC [21]
— d'autres méthodes, comme KSP [24], KK [29] ou BDDTab [14] par exemple

Dans [1], Areces et al. définissent une procédure de décision basée sur des solveurs SMT pour la logique modale K, les modalités globales et la logique hybride. Cet article présente MOLOSS, une implémentation du travail théorique de Areces et al. MOLOSS étend également ce travail en définissant de nouvelles procédures d'instanciation pour des axiomes de cadre qui ne sont pas pris en compte dans [1].

Cet article est organisé comme suit. La section 2 décrit brièvement les logiques modales et leur traduction en logique du premier ordre. La section 3 présente l'algorithme défini dans [1] et la section 6 donne un aperçu de l'implémentation de MOLOSS, en particulier l'utilisation d'une procédure Max-SMT pour restreindre l'explosion du nombre de mondes dans le modèle de Kripke. La section 7 présente les résultats d'expérimentations sur différentes logiques modales et les différents modes opératoires de MOLOSS et la section 8 présente quelques perspectives.

2 Logique modale

La logique modale propositionnelle ou la logique modale basique (BML pour *Basic Modal Logic*) peuvent être vues comme une extension de la logique propositionnelle. Soit P un ensemble non vide de variables propositionnelles, la syntaxe de BML est définie comme $\varphi = p \mid \neg p \mid \varphi \rightarrow \varphi \mid \Box\varphi \mid \Diamond\varphi$ où $p \in P$. $\Diamond$ est l'opérateur dual de $\Box$, i.e. $\Diamond\varphi = \neg\Box\neg\varphi$. $\wedge$ et $\vee$ sont définis classiquement à partir de $\rightarrow$. On peut remarquer que dans [1], les formules sont considérées comme étant en forme normale négative, i.e. le symbole de négation $\neg$ ne peut être appliqué que sur

des propositions atomiques. La sémantique de BML est définie avec un *modèle de Kripke* [18] $\langle W, \mathcal{R}, V\rangle$ où W est un ensemble non vide de mondes, $\mathcal{R} \subseteq W \times W$ est une relation d'accessibilité binaire sur W définissant la sémantique de $\Box$ et appelée *relation d'accessibilité* et $V : P \mapsto \mathcal{P}(W)$ est une *fonction de valuation*. Étant donnés un modèle de Kripke $\mathcal{M}$ et un monde $w \in W$, la sémantique des formules de BML est définie comme suit [1] :

$$\begin{array}{lll}
\mathcal{M}, w \models p & \text{ssi} & w \in V(p) \\
\mathcal{M}, w \models \neg p & \text{ssi} & w \notin V(p) \\
\mathcal{M}, w \models \varphi \to \psi & \text{ssi} & \text{si } \mathcal{M}, w \models \varphi \text{ alors } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \Box\varphi & \text{ssi} & \text{pour tout } w' \in W \text{ t.q. } w\mathcal{R}w' \\
 & & \qquad \text{on a } \mathcal{M}, w' \models \varphi
\end{array}$$

Des contraintes sur $\mathcal{R}$ peuvent être définies par des *axiomes de cadre*. Les axiomes de cadre les plus utilisés sont représentés sur le tableau 1 (l'axiome (K) est admis dans toutes les logiques modales que nous considérerons).

Nous ne considérerons ici que les logiques modales pour lesquelles toute formule peut être traduite en une formule de la logique classique du premier ordre avec la signature suivante : les constantes représentent les mondes possibles, il n'y pas d'autre symbole de fonction, les prédicats unaires correspondent aux propositions et il y a un symbole de prédicat binaire, $\mathcal{R}$, représentant la relation d'accessibilité. Cette traduction peut être représentée par une fonction ST_w avec $w \in W$ des formules de la logique modale vers les formules de la logique classique du premier ordre [5]. Par exemple, $ST_w(\Box(a \wedge b)) = \forall x\, w\mathcal{R}x \to (A(w) \wedge B(w))$. Les règles définissant ST_w sont les suivnates :

$$\begin{aligned}
ST_w(p) &= P(w) \\
ST_w(\neg p) &= \neg ST_w(p) \\
ST_w(\varphi \to \varphi') &= ST_w(\varphi) \to ST_w(\varphi') \\
ST_w(\Diamond\varphi) &= \exists x\, w\mathcal{R}x \wedge ST_x(\varphi) \\
ST_w(\Box\varphi) &= \forall x\, w\mathcal{R}x \to ST_x(\varphi)
\end{aligned}$$

La propriété suivante est vraie : soient $\mathcal{M}$ un modèle de Kripke, $w \in W$ et φ une formule de BML, alors $\mathcal{M}, w \models \varphi$ ssi $\mathcal{M}', \sigma[x \mapsto w] \models ST_x(\varphi)$ où $\mathcal{M}'$ est une interprétation du premier ordre classique représentant $\mathcal{M}$ et σ est une affectation pour $\mathcal{M}'$. Pour utiliser un système d'axiomes particulier, une théorie du premier ordre classique utilisant les conditions de cadre doit être définie et $\mathcal{M}'$ doit être une interprétation dans cette théorie. La vérification de la satisfiabilité d'une formule modale peut donc être réduite à la vérification de la satisfiabilité d'une formule du premier ordre classique particulière dans une théorie particulière (cf. [19, 17] pour des détails, en particulier sur la complexité d'une telle transformation).

1. La satisfiabilité et la validité sont définies comme d'habitude.

3 Algorithme

3.1 Présentation de l'algorithme principal

L'algorithme décrit dans [1] et présenté sur l'algorithme 1 prend en entrée une formule de BML φ et construit un ensemble $\mathcal{S}$ de formules de la logique du premier ordre contenant les traductions de φ via la fonction ST_w comme présenté en section 2. $\mathcal{S}$ est alors instancié en utilisant une *fonction d'abstraction abs* « encadre » toutes les formules du premier ordre fermées comme suit :

$$\begin{aligned}
abs(P(x)) &= \boxed{P(x)} \\
abs(x\mathcal{R}y) &= \boxed{x\mathcal{R}y} \\
abs(\neg\varphi) &= \neg abs(\varphi) \\
abs(\varphi \to \psi) &= abs(\varphi) \to abs(\psi) \\
abs(\exists x\, w\mathcal{R}x \wedge \varphi(x)) &= \boxed{\exists x\, w\mathcal{R}x \wedge \varphi(x)} \\
abs(\forall x\, \neg w\mathcal{R}x \vee \varphi(x)) &= \boxed{\forall x\, \neg w\mathcal{R}x \vee \varphi(x)}
\end{aligned}$$

Par exemple, $abs((\forall x\ \neg w\mathcal{R}x \vee P(x)) \wedge P(a)) = \boxed{\forall x\, \neg w\mathcal{R}x \vee P(x)} \wedge \boxed{P(a)}$. La fonction inverse de *abs* est notée *conc* : elle « enlève » le cadre autour des littéraux.

Chaque formule du premier ordre encadrée est associée à une nouvelle variable propositionnelle. $abs(\mathcal{S})$ est alors envoyée à un solveur constitué d'un SAT solveur et éventuellement d'autres procédures de décision. Si $abs(\mathcal{S})$ est insatisfiable, alors $\mathcal{S}$ est également insatisfiable, sinon un *module d'instanciation* calcule des raffinements de $abs(\mathcal{S})$ en utilisant des *règles d'instanciation* en ajoutant de nouvelles formules dans $\mathcal{S}$. Le nouvel ensemble de formules est instancié et envoyé à son tour au solveur. Le but principal du module d'instanciation est de rejeter les modèles propositionnels produits par le solveur qui ne correspondent pas un à modèle premier ordre de $\mathcal{S}$.

Durant l'exécution de l'algorithme, une *configuration* $C = \langle \mathcal{S}, \Theta_\bullet, \ldots\rangle$, où $\Theta_\bullet$ est un ensemble contenant des informations sur les règles d'instanciation précédemment appliquées, est utilisée pour éviter d'appliquer la même règle plusieurs fois et ainsi de produire la même formule plusieurs fois. Il existe un ensemble $\Theta_\bullet$ pour chaque règle d'instanciation. Initialement, $C = \langle ST_w(\varphi), \varnothing, \ldots\rangle$ où φ est la formule modale dont la satisfiabilité doit être vérifiée.

3.2 Règles d'instanciation

Les règles d'instanciation sont utilisées pour gérer les formules quantifiées, en particulier pour générer de nouvelles constantes représentant des mondes et pour mettre à jour la relation d'accessibilité si nécessaire. Les règles sont des fonctions prenant comme paramètres une configuration et un modèle propositionnel Γ et renvoyant une nouvelle configuration. Elles fonctionnent toutes de la même façon :

nom	propriété	axiome	condition de cadre
K	distribution	$\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$	
T	réflexivité	$\Box\varphi \to \varphi$	$\forall w,\ w\mathcal{R}w$
4	transitivité	$\Box\varphi \to \Box\Box\varphi$	$\forall w,u,v,\ w\mathcal{R}w \wedge u\mathcal{R}v \to w\mathcal{R}v$
B	symétrie	$\varphi \to \Box\Diamond\varphi$	$\forall w,u,\ w\mathcal{R}u \leftrightarrow u\mathcal{R}w$
5	euclidienne	$\Diamond\varphi \to \Box\Diamond\varphi$	$\forall w,u,v,\ w\mathcal{R}u \wedge w\mathcal{R}v \to u\mathcal{R}v$
CD	fonctionnelle	$\Diamond\varphi \to \Box\varphi$	$\forall w,u,v,\ w\mathcal{R}u \wedge w\mathcal{R}v \to u = v$

TABLE 1 – Axiomes et conditions de cadre

Algorithm 1 : Algorithme pour décider de la satisfiabilité de la formule modale φ

$\mathcal{S} \leftarrow ST_w(\varphi)$;
$C \leftarrow \langle \mathcal{S}, \varnothing, \ldots \rangle$;
while *true* **do**
  envoyer $abs(\mathcal{S})$ au solveur ;
  if *le solveur répond* `UNSAT` **then**
    return `UNSAT` ;
  else
    récupérer le modèle Γ de $abs(\mathcal{S})$;
    if *aucune règle d'instanciation ne s'applique* **then**
      construire un modèle de Kripke $\mathcal{M}$ de φ à partir de Γ ;
      return `SAT` et $\mathcal{M}$;
    else
      mettre à jour C en utilisant la règle d'instanciation ;

— elles sélectionnent un littéral encadré particulier ε dans Γ qui n'a pas encore été choisi pour générer de nouvelles formules via la règle en cours d'exécution.
— $conc(\varepsilon)$, la version « non encadrée » de ε, est utilisée pour générer de nouveaux mondes où de nouvelles relations entre les mondes.
— les mondes utilisés pour cette instanciation particulière sont ajoutés à la configuration correspondante (possiblement avec ε) afin de se souvenir l'instanciation courante et d'éviter de la réutiliser par la suite.

Dans [1], les auteurs définissent deux règles pour BML, une pour chaque quantificateur du premier ordre.

Définition 1 (règle originelle $\exists$ de [1])
$f_\exists(\langle \mathcal{S}, \Theta_\exists, \ldots \rangle, \Gamma) = \langle \mathcal{S}', \Theta'_\exists, \ldots \rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{\exists y\, c\mathcal{R}y \wedge \varphi(y)}$ *dans* $\Gamma \setminus \Theta_\exists$ *et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \to (c\mathcal{R}d \wedge \varphi(d))\}$ *avec d une nouvelle constante et* $\Theta'_\exists = \Theta_\exists \cup \{\varepsilon\}$.

La règle $\exists$ est appliquée quand il existe un littéral encadré $\varepsilon = \boxed{\exists y\, c\mathcal{R}y \wedge \varphi(y)}$ dans le modèle propositionnel de $abs(S)$ qui n'a pas encore été utilisé pour générer un nouveau monde. Elle génère alors une nouvelle constante d représentant un monde et ajoute une formule $conc(\varepsilon) \to (c\mathcal{R}d \wedge \varphi(d))$ dans $\mathcal{S}$ spécifiant que le monde concerné par le quantificateur existentiel est instancié avec d. ε est alors ajoutée au sous-ensemble de configuration $\Theta_\exists$ pour éviter de la réutiliser par la suite.

Définition 2 (règle originelle $\forall$ de [1])
$f_\forall(\langle \mathcal{S}, \Theta_\exists, \ldots \rangle, \Gamma) = \langle \mathcal{S}', \Theta'_\forall, \ldots \rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{\forall y\, \neg c\mathcal{R}y \vee \varphi(y)}$ *dans* Γ *tel que* $\boxed{c\mathcal{R}d} \in \Gamma$, $(\varepsilon, d) \notin \Theta_\forall$ *et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \to (\neg c\mathcal{R}d \vee \varphi(d))\}$ *et* $\Theta'_\forall = \Theta_\forall \cup \{(\varepsilon, d)\}$.

La règle $\forall$ est appliquée quand une formule universellement quantifiée $\forall y\ \neg c\mathcal{R}y \vee \varphi(y)$ apparaît dans $\mathcal{S}$. Cette formule est la traduction d'une formule $\Box\varphi$, donc tous les mondes accessibles depuis c doivent « satisfaire » φ. Ainsi, si un tel monde c existe et n'a pas encore été utilisé pour instancié la formule, $conc(\varepsilon) \to (\neg c\mathcal{R}d \vee \varphi(d))$ est ajouté à $\mathcal{S}$. On peut remarquer que $\Theta_\forall$ ne contient pas simplement des formules encadrées, mais des couples constitués d'une formule encadrée et d'un monde.

Areces et al. ont proposé dans [1] deux extensions à BML : une pour les modalités globales et une pour la logique hybride (d'où l'utilisation d'un solveur SMT pour gérer les fonctions non interprétées). Nous proposons dans MOLOSS d'étendre les règles $\exists$ et $\forall$ pour prendre en compte des conditions de cadre supplémentaires : réflexivité, symétrie, transitivité et relations euclidiennes.

Définition 3 (règle *ref*)
$f_{ref}(\langle \mathcal{S}, \Theta_\forall, \Theta_{ref}, \ldots \rangle, \Gamma) = \langle \mathcal{S}', \Theta'_\forall, \Theta'_{ref}, \ldots \rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{\forall y\, \neg c\mathcal{R}y \vee \varphi(y)}$ *dans* $\Gamma \setminus \Theta_{ref}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \to \varphi(c)\}$, $\Theta'_{ref} = \Theta_{ref} \cup \{\varepsilon\}$ *et* $\Theta'_\forall = \Theta_\forall \cup \{(\varepsilon, c)\}$.

Cette règle assure que φ est vraie dans un monde c si $\Box\varphi$ est vraie dans c et représente donc la réflexivité. Une autre possibilité pour représenter cette propriété est de modifier la règle $\exists$ en ajoutant $d\mathcal{R}d$ dans $\mathcal{S}'$ pour chaque monde d nouvellement créé par la règle $\exists$ et en laissant la règle $\forall$ instancier des propriétés du type $\Box\varphi$ dans le monde, mais

des expérimentations ont montré que la règle proposée est plus rapide que la version modifiée de la règle $\exists$.

Définition 4 (règle *sym*)
$f_{sym}(\langle \mathcal{S}, \Theta_{sym}, \ldots\rangle, \Gamma) = \langle \mathcal{S}', \Theta'_{sym}, \ldots\rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{c\mathcal{R}d}$ *dans* Γ *tel que* $\{c, d\} \notin \Theta_{sym}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{c\mathcal{R}d \leftrightarrow d\mathcal{R}c\}$ *et* $\Theta'_{sym} = \Theta_{sym} \cup \{\{c, d\}\}$.

Cette règle représente la propriété de symétrie. Comme $\{c, d\}$ est un ensemble et non pas un tuple, il n'y a pas besoin d'ajouter $\{d, c\}$ dans Θ_{sym}.

Définition 5 (règle *trans*)
$f_{trans}(\langle \mathcal{S}, \Theta_{trans}, \ldots\rangle, \Gamma) = \langle \mathcal{S}', \Theta'_{trans}, \ldots\rangle$ *s'il existe deux littéraux encadrés* $\varepsilon = \boxed{c\mathcal{R}d}$ *et* $\varepsilon' = \boxed{d\mathcal{R}e}$ *dans* Γ *tels que* $(c, d, e) \notin \Theta_{trans}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{c\mathcal{R}d \wedge d\mathcal{R}e \rightarrow c\mathcal{R}e\}$ *et* $\Theta'_{trans} = \Theta_{trans} \cup \{(c, d, e)\}$.

Cette règle représente la propriété de transitivité. La configuration de la règle conserve les triplets de mondes impliqués dans la relation de transitivité.

Définition 6 (règle *euc*)
$f_{euc}(\langle \mathcal{S}, \Theta_{euc}, \ldots\rangle, \Gamma) = \langle \mathcal{S}', \Theta'_{euc}, \ldots\rangle$ *s'il y a deux littéraux encadrés* $\varepsilon = \boxed{c\mathcal{R}d}$ *et* $\varepsilon' = \boxed{c\mathcal{R}e}$ *dans* Γ *tels que* $(c, \{d, e\}) \notin \Theta_{euc}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{c\mathcal{R}d \wedge c\mathcal{R}e \rightarrow (d\mathcal{R}e \wedge e\mathcal{R}d)\}$ *et* $\Theta'_{euc} = \Theta_{euc} \cup \{(c, \{d, e\})\}$.

Cette règle représente la propriété euclidienne pour une relation.

Définition 7 (rule $\exists_{fonc}$) $f_{\exists}(\langle \mathcal{S}, \Theta_{\exists}, \Theta_{fun}, \ldots\rangle, \Gamma) = \langle \mathcal{S}', \Theta'_{\exists}, \Theta'_{fun}, \ldots\rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{\exists y\, c\mathcal{R}y \wedge \varphi(y)}$ *in* $\Gamma \setminus \Theta_{\exists}$ *et*

- *Soit il existe d tel que* $(c, d) \in \Theta_{fun}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \rightarrow (c\mathcal{R}d \wedge \varphi(d))\}$ *et* $\Theta'_{\exists} = \Theta_{\exists} \cup \{\varepsilon\}$.
- *Soit il n'existe pas un tel d et on définit une nouvelle constante d et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \rightarrow (c\mathcal{R}d \wedge \varphi(d))\}$, $\Theta'_{\exists} = \Theta_{\exists} \cup \{\varepsilon\}$ *et* $\Theta'_{fun} = \Theta_{fun} \cup \{(c, d)\}$

Cette réécriture de la règle $\exists$, représente une relation fonctionnelle. MOLOSS n'autorise qu'un seul successeur par état et stocke dans une table chaque couple (état, successeur). Toutefois cette règle est valide que si elle est utilisée sans les autres axiomes.

Enfin, on peut remarquer qu'il y a deux types de règles : des règles de *saturation* comme $\forall$, *ref*, *trans*, ou *euc* qui étendent le modèle et des règles d'*introduction* comme $\exists$ et $\exists_{fonc}$, qui ajoutent un nouveau monde avec les propriétés attendues.

4 Restreindre l'expansion du modèle

Dans notre contexte, les formules en logique modale vérifient la *propriété du modèle fini* [6] : si une formule modale est satisfiable, alors elle est satisfiable dans un modèle fini. Cependant, la borne supérieure du nombre de mondes nécessaires pour satisfaire une formule donnée dépend de la logique considérée : par exemple, le nombre de mondes pour S5 est linéaire en nombre d'opérateurs $\Diamond$ [9] alors que pour S4 elle est en $O(2^m)$ pour une formule de taille $O(m)$ [16]. La taille du modèle peut être limitée en tentant d'éviter d'introduire un nouveau monde à chaque fois qu'une formule $\exists y\, c\mathcal{R}y \wedge \varphi(y)$ est instanciée, en vérifiant d'abord si monde existant ne satisfait pas la formule (modulo une relation entre ce monde et y). Pour implémenter cette relation de priorité entre les mondes, MOLOSS utilise une procédure de type *Partial Weighted Max-SAT/SMT*. Un ensemble $\mathcal{S}_{soft}$ de contraintes souples de la forme (formules, poids) est ajouté à la configuration et chaque fois qu'une formule $\exists y\, w\mathcal{R}y \wedge \varphi(y)$ est instanciée et qu'un nouveau monde d est créé, une formule interdisant $d\mathcal{R}x$ pour tous les mondes x créés précédemment est ajoutée à S_{soft} avec un poids égal au nombre de mondes créés précédemment. Le solveur doit satisfaire les formules de $\mathcal{S}$ en minimisant la somme des poids des formules de $\mathcal{S}_{soft}$ qui ne sont pas satisfaites. Les poids des formules dans $\mathcal{S}_{soft}$ garantissent que le solveur utilise le monde le plus ancien possible pour satisfaire la formule.

Définition 8 (rule $\exists_{soft}$)
$f_{\exists_{soft}}(\langle \mathcal{S}, \mathcal{S}_{soft}, \Theta_{\exists}, \Theta_w, \ldots\rangle, \Gamma) = \langle \mathcal{S}', \mathcal{S}'_{soft}, \Theta'_{\exists}, \Theta'_w, \ldots\rangle$ *s'il existe un littéral encadré* $\varepsilon = \boxed{\exists y\, c\mathcal{R}y \wedge \varphi(y)}$ *dans* $\Gamma \setminus \Theta_{\exists}$ *et* $\mathcal{S}' = \mathcal{S} \cup \{conc(\varepsilon) \rightarrow (\bigvee_{w \in \Theta_w} c\mathcal{R}w \wedge \varphi(w)) \vee (c\mathcal{R}d \wedge \varphi(d)\}$ *avec d une nouvelle constante et* $\mathcal{S}'_{soft} = \mathcal{S}'_{soft} \cup \{(\bigwedge_{w \in \Theta_w} \neg w\mathcal{R}d, |\Theta_w| + 1)\}$, $\Theta'_{\exists} = \Theta_{\exists} \cup \{\varepsilon\}$ *et* $\Theta'_w = \Theta_w \cup \{d\}$.

Le solveur Z3 supporte directement les *assert-soft* [4]. Afin de pouvoir utiliser cette approche avec Minisat, les contraintes souples sont gérées en dehors du solveur. Au lieu de stocker les formules φ_{soft} dans $\mathcal{S}_{soft}$, nous stockons des contraintes dures de la forme $g \rightarrow \varphi_{soft}$ dans $\mathcal{S}$, où g est une nouvelle variable propositionnelle appelée *garde*. $\mathcal{S}_{soft}$ est une structure de file ne contenant que les variables de garde. Lorsque nous vérifions la satisfiabilité propositionnelle avec Minisat, les variables de garde sont forcées à `true`. Si la formule est `UNSAT`, les variables de garde sont défilées de $\mathcal{S}_{soft}$ une par une et mise à `false` dans la prochaine requête de satisfiabilité au solveur, « autorisant » ainsi l'utilisation du monde correspondant. Remarquons finalement que ces nouvelles règles ne sont utilisées qu'avec des logiques utilisant les axiomes (4) ou (5).

5 Validité et complétude

5.1 Complétude

Rappelons que $\mathcal{S}$ est un ensemble de formules du premier ordre. Nous définissons :

$$ST_{x,tot}(\varphi) := ST_x(\varphi) \wedge \text{axiomes de cadre}$$
$$\mathcal{S}^n_{tot} := \mathcal{S}^n \cup \{\text{axiomes de cadre}\}$$

Considérons deux ensembles $\mathcal{S}$ et $\mathcal{S}'$. $\mathcal{S} \rightarrow \mathcal{S}'$ dénote alors le fait que $\mathcal{S}'$ est obtenu en appliquant une règle sur une configuration contenant $\mathcal{S}$. Le lemme suivant est équivalent au lemme 2 de [1] :

Lemme 1 *Si $\mathcal{S}^n \rightarrow \mathcal{S}^{n+1}$, alors $\mathcal{S}^n_{tot}$ et $\mathcal{S}^{n+1}_{tot}$ sont équisatisfiables.*

On peut définir un lemme correspondant au lemme 3 de [1] de la même façon :

Lemme 2 *Si $\mathcal{M}$ est un modèle premier ordre pour $\mathcal{S}$, alors il existe un modèle Γ pour $abs(\mathcal{S})$.*

Preuve 1 *Γ est construit de la façon suivante : pour un littéral ψ, $\boxed{\psi} \in \Gamma$ si $\mathcal{M} \vDash \psi$ et $\neg\boxed{\psi} \in \Gamma$ si $\mathcal{M} \nvDash \psi$. $\Gamma \vDash abs(S)$ est alors prouvé par induction.*

Théorème 1 *Si l'algorithme termine et répond `UNSAT`, alors la formule est insatisfaisable.*

Preuve 2 *Si l'algorithme répond `UNSAT`, alors il existe une suite finie de configurations*

$$C^0 \rightarrow C^1 \rightarrow ... \rightarrow C^n.$$

telle que $abs(\mathcal{S}^n)$ est insatisfaisable. En utilisant le lemme 2, $\mathcal{S}^n$ est également insatisfaisable, donc $\mathcal{S}^n_{tot}$ est insatisfaisable. En itérant le lemme 1, on prouve donc que $\mathcal{S}^0_{tot}$ est insatisfaisable.

5.2 Validité

Théorème 2 *Si l'algorithme termine et répond `SAT`, alors $\mathcal{S}^0_{tot}$ est satisfiable.*

Preuve 3 *Si l'algorithme répond `SAT`, alors il existe une suite finie de configurations*

$$C^0 \rightarrow C^1 \rightarrow ... \rightarrow C^n$$

telle que $\mathcal{S}^n$ est satisfiable et aucune procédure d'instanciation ne peut être appliquée. Il existe donc un modèle Γ pour $abs(\mathcal{S}^n)$. Un modèle du premier ordre $\mathcal{M}$ est construit à partir de ce modèle propositionnel :

- *les constantes sont celles apparaissant dans S^n*
- *les prédicats sont construits de la façon suivante :*
  - *$P(w) :=$ true ssi $\boxed{P(w)} \in \Gamma$*
  - *$R(w, w') :=$ true ssi $\boxed{R(w, w')} \in \Gamma$.*

Les étapes 1 et 2 de la preuve du théorème 5 de [1] sont encore valides et donc $\mathcal{M} \vDash ST(\varphi)$.

Montrer que $\mathcal{M}$ est également un modèle pour les axiomes de cadre est immédiat par l'absurde, donc $\mathcal{M} \vDash ST_{tot}(\varphi)$.

6 Implémentation de MOLOSS

MOLOSS est implémenté en OCaml et est disponible sur [12] sous licence GPL. Voici quelques détails à propos de l'implémentation :

- MOLOSS prend en entrée le format InToHyLo utilisé dans de nombreux solveurs (cf. [20, 15])
- MOLOSS peut utiliser trois solveurs en back-end : Z3 [23], mSAT [8] et Minisat [10] (utilisé par défaut)
- il existe un mode `direct` qui envoie simplement la traduction en logique du premier ordre de la formule en logique modale vers Z3 fonctionnant ainsi en solveur SMT. Cela permet de laisser Z3 gérer les quantificateurs sans utiliser de règle d'instanciation.
- lorsque plusieurs règles d'instanciation peuvent être appliquées, l'ordre dans lequel elles le sont peut impacter sérieusement les performances du solveur. L'ordre suivant est utilisé dans MOLOSS : $\exists <$ *trans* $\sim$ *euc* $< \forall <$ *sym* $\sim$ *ref*

7 Expérimentations

MOLOSS a été testé sur un ordinateur avec 3 processeurs Intel XEON E-2670 et 64 Go de mémoire vive. Nous avons généré aléatoirement des formules de test de profondeur comprise entre 10 et 23 pour des logiques modales particulières (K, T, S4, ...).

Les différents modes de MOLOSS ont été comparés en utilisant ces formules de test : le mode `direct`, le mode utilisant des règles d'instanciation et Z3 comme solveur SAT et le mode utilisant des règles d'instanciation et Minisat comme solveur SAT. Un *timeout* de 900s a été fixé et seules les formules pour lesquelles au moins un des trois modes précédemment cités a émis un résultat sont considérées. Lorsque plusieurs modes renvoient un résultat pour la même formule, nous avons vérifié que les résultats étaient identiques.

Le tableau 2 présente les taux de succès pour les différents modes. Les colonnes Z3 et MS (pour Minisat) présentent la proportion de formules pour lesquelles MOLOSS avec le mode correspondant était plus rapide que le mode `direct`. La colonne `both` représente la proportion de formules pour lesquelles au moins un des solveurs avec règles d'instanciation était plus rapide que le mode `direct`. Enfin, la colonne MS/Z3 présente la proportion de formules pour lesquelles Minisat était plus rapide que Z3 comme solveur SAT. Le tableau 3 présente le taux de *timeout* pour chaque mode.

Les modes utilisant des procédures d'instanciation sont plus rapides que le mode `direct`. Il y a toutefois deux exceptions : les logiques T et S4. Une explication possible pourrait être le fait que T et S4 utilisent la propriété de réflexivité. On peut cependant noter que S5 utilise également la propriété de réflexivité, mais que cette propriété

logique	Z3	MS	both	MS/Z3
K	0.8049	0.9432	0.9892	0.9324
T	0.3180	0.2809	0.4259	0.2870
B	0.7159	0.8023	0.9040	0.7985
K4	0.4571	0.8857	0.8857	0.9429
S4	0.1873	0.7570	0.7570	0.9316
K5	0.5374	1	1	0.9962
S5	0.3351	0.9046	0.9072	0.9820

TABLE 2 – Comparaison des différents modes

logique	`direct`	Z3	MS
K	0.2442	0.0169	0.0492
T	0.2191	0.2347	0.5648
B	0.2342	0.0461	0.1459
K4	0.2555	0.3328	0.0571
S4	0.4886	0.5722	0.0582
K5	0.5624	0.3589	0
S5	0.7062	0.5129	0.0129

TABLE 3 – Taux de *timeout*

est contrebalancée pour la propriété euclidienne qui avantage MOLOSS. La figure 1 montre que pour la plupart des logiques considérées il existe deux nuages de points, un pour chaque oracle, et que ces nuages de points distincts n'apparaissent pas pour T, S4 et S5. Ce point devra être approfondi dans la suite de nos travaux.

Dans la plupart des cas, MOLOSS utilisant Minisat comme solveur SAT est plus rapide que MOLOSS utilisant Z3 comme solveur SAT. Plus précisément, pour les logiques K4, S4, K5 et S5, Minisat est plus rapide que Z3, mais pour K et B, Minisat atteint la limite de temps plus souvent que Z3. Enfin, il n'y a pas de gagnant entre Minisat et Z3 pour la logique T, même si Minisat atteint la limite de temps plus souvent que Z3. Ces résultats sont étranges car les procédures d'instanciation sont appelées sur un modèle et ne dépendent pas directement du solveur SAT utilisé.

8 Conclusion

Dans cet article nous avons décrit MOLOSS, une première implémentation et extension du travail théorique présenté dans [1]. MOLOSS traduit classiquement des formules de logique modale en formule en logique classique du premier ordre et permet pour vérifier la satisfiabilité de ces formules traduites d'utiliser soit Z3 comme solveur SMT ou d'utiliser des procédures d'instanciation et un solveur SAT. Nous avons également implémenté une procédure permettant d'éviter l'explosion du nombre de mondes, soit en utilisant la procédure Max-SMT de Z3, soit en utilisant une procédure incrémentale et une relaxation graduelle des contraintes pour des solveurs SAT.

Lors des tests de MOLOSS sur des formules générées aléatoirement pour différentes logiques modales, nous avons pu vérifier qu'utiliser les procédures d'instanciation était plus efficace que de laisser le solveur SMT gérer lui-même les quantificateurs du premier ordre. Cela était attendu, car les règles d'instanciation sont guidées par les axiomes de cadre et sont donc plus efficaces que l'algorithme d'E-matching utilisé dans Z3 pour gérer les quantificateurs.

Enfin, des campagnes de tests doivent être menées afin de comparer MOLOSS à des solveurs de l'état de l'art comme ceux cités dans la section 1 sur des benchmarks classiques comme ceux présentés dans [3, 22] et non pas des formules générées aléatoirement. Cela permettrait de montrer que l'approche prise par MOLOSS est intéressante et devrait nous permettre de l'améliorer.

Remerciements

Les auteurs tiennent à remercier les relecteurs anonymes pour leurs remarques et les nombreuses références supplémentaires qu'ils leur ont fournies.

Références

[1] Areces, Carlos, Pascal Fontaine et Stephan Merz : *Modal Satisfiability via SMT Solving*. Dans *Software, Services, and Systems.*, tome 8950 de *LNCS*, pages 30–45. Springer, 2015.

[2] Balsiger, Peter et Alain Heuerding : *Comparison of Theorem Provers for Modal Logics - Introduction and Summary*. Dans *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, TABLEAUX '98, pages 25–26, Berlin, Heidelberg, 1998. Springer-Verlag, ISBN 3-540-64406-7. `http://dl.acm.org/citation.cfm?id=646889.709131`.

[3] Balsiger, Peter, Alain Heuerding et Stefan Schwendimann : *A Benchmark Method for the Propositional Modal Logics K, KT, S4*. Journal of Automated Reasoning, 24(3):297–317, Apr 2000, ISSN 1573-0670. `https://doi.org/10.1023/A:1006249507577`.

[4] Bjørner, Nikolaj et Anh-Dung Phan : *νZ - Maximal Satisfaction with Z3*. Dans *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014*, pages 1–9, 2014.

[5] Blackburn, Patrick et Johan van Benthem : *1 Modal logic : a semantic perspective*. Dans Blackburn, Patrick, Johan Van Benthem et Frank Wolter (rédacteurs) : *Handbook of Modal Logic*, tome 3 de *Studies in Logic and Practical Reasoning*, pages 1 – 84. Elsevier, 2007. `http://www.sciencedirect.com/science/article/pii/S1570246407800048`.

[6] Blackburn, Patrick, Maarten de Rijke et Yde Venema : *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[7] Brand, Sebastian, Rosella Gennari et Maarten de Rijke : *Constraint Methods for Modal Satisfiability*. Dans Apt, Krzysztof R., François Fages, Francesca Rossi, Péter Szeredi et Josef Váncza (rédacteurs) : *Recent Advances in Constraints*, pages 66–86, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg, ISBN 978-3-540-24662-6.

[8] Bury, Guillaume : *mSat – A modular sat/smt solver with proof output*, 2017. `https://github.com/Gbury/mSAT`.

[9] Caridroit, Thomas, Jean Marie Lagniez, Daniel Le Berre, Tiago de Lima et Valentin Montmirail : *A SAT-Based Approach for Solving the Modal Logic S5-Satisfiability Problem*. Dans *Thirty-First AAAI Conference on Artificial Intelligence (AAAI17)*, pages 3864–3870, 2017.

[10] Eén, Niklas et Niklas Sörensson : *MiniSat*, 2017. `http://minisat.se/`.

[11] Gasquet, Olivier, Andreas Herzig, Dominique Longin et Mohamad Sahade : *LoTREC : Logical Tableaux Research Engineering Companion*. Dans Beckert, Bernhard (rédacteur) : *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 318–322, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg, ISBN 978-3-540-31822-4.

[12] Giet, Josselin, Rémi Delmas et Christophe Garion : *MOLOSS – a MOdal LOgic Satisfiability Solver*. 2018. `https://github.com/Meleagant/MOLOSS`.

[13] Giunchiglia, F., F. Giunchiglia, R. Sebastiani, A. Tacchella, Enrico Giunchiglia, Fausto Giunchiglia, Roberto Sebastiani et Armando Tacchella : *SAT vs. Translation Based decision procedures for modal logics : a comparative evaluation*. Journal of Applied NonClassical Logics, 10:2000, 1998.

[14] Goré, Rajeev, Kerry Olesen et Jimmy Thomson : *Implementing Tableau Calculi Using BDDs : BDDTab System Description*. Dans Demri, Stéphane, Deepak Kapur et Christoph Weidenbach (rédacteurs) : *Automated Reasoning*, pages 337–343, Cham, 2014. Springer International Publishing, ISBN 978-3-319-08587-6.

[15] Götzmann, Daniel, Mark Kaminski et Gert Smolka : *Spartacus : A Tableau Prover for Hybrid Logic*. Electronic Notes in Theoretical Computer Science, 262:127–139, 2010. Proceedings of the 6th Workshop on Methods for Modalities.

[16] Halpern, Joseph Y et Yoram Moses : *A guide to completeness and complexity for modal logics of knowledge and belief*. Artificial intelligence, 54(3):319–379, 1992.

[17] Halpern, Joseph Y. et Leandro Chaves Rêgo : *Characterizing the NP-PSPACE Gap in the Satisfiability Problem for Modal Logic*. Dans *Proceedings of IJCAI'07*, pages 2306–2311. Morgan Kaufmann, 2007.

[18] Kripke, Saul A. : *A Completeness Theorem in Modal Logic*. Journal of Symbolic Logic, 24(1):1–14, 1959.

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

[20] Lagniez, Jean Marie, Daniel Le Berre, Tiago De Lima et Valentin Montmirail : *On Checking Kripke Models for Modal Logic K*. Dans Fontaine, Pascal, Stefan Schulz et J. Urban (rédacteurs) : *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016)*, pages 69–81, 2016.

[21] Lagniez, Jean Marie, Daniel Le Berre, Tiago de Lima et Valentin Montmirail : *A Recursive Shortcut for CEGAR : Application To The Modal Logic K Satisfiability Problem*. Dans *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 674–680, 2017. `https://doi.org/10.24963/ijcai.2017/94`.

[22] Massacci, Fabio et Francesco M. Donini : *Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison*. Dans Dyckhoff, Roy (rédacteur) : *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 52–56, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg, ISBN 978-3-540-45008-5.

[23] Microsoft Research : *The Z3 Theorem Prover*, 2017. `https://github.com/Z3Prover/z3`.

[24] Nalon, Cláudia, Ullrich Hustadt et Clare Dixon : *[InlineEquation Not Available : See Fulltext.] : A Resolution-Based Prover for Multimodal K*. Dans *Proceedings of the 8th International Joint Conference on Automated Reasoning - Volume 9706*, pages 406–415, Berlin, Heidelberg, 2016. Springer-Verlag, ISBN 978-3-319-40228-4. `https://doi.org/10.1007/978-3-319-40229-1_28`.

[25] Pan, Guoqiang et Moshe Y. Vardi : *Symbolic Decision Procedures for QBF*. Dans Wallace, Mark (rédacteur) : *Principles and Practice of Constraint Programming – CP 2004*, pages 453–467, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg, ISBN 978-3-540-30201-8.

[26] Schmidt, Renate A. et Dmitry Tishkovsky : *A General Tableau Method for Deciding Description Logics, Modal Logics and Related First-Order Fragments*. Dans *Proceedings of the 4th International Joint*

Conference on Automated Reasoning, IJCAR '08, pages 194–209, Berlin, Heidelberg, 2008. Springer-Verlag, ISBN 978-3-540-71069-1. http://dx.doi.org/10.1007/978-3-540-71070-7_17.

[27] Sebastiani, Roberto et Michele Vescovi : *Automated Reasoning in Modal and Description Logics via SAT Encoding : The Case Study of KmALC-satisfiability*. J. Artif. Int. Res., 35(1):343–389, juin 2009, ISSN 1076-9757. http://dl.acm.org/citation.cfm?id=1641503.1641511.

[28] Tsarkov, Dmitry et Ian Horrocks : *FaCT++ Description Logic Reasoner : System Description*. Dans Furbach, Ulrich et Natarajan Shankar (rédacteurs) : *Automated Reasoning*, pages 292–297, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg, ISBN 978-3-540-37188-5.

[29] Voronkov, Andrei et Harald Ganzinger : *KK : a theorem prover for K*, tome 1632 de *Lecture Notes in Computer Science*, pages 383–387. Springer, Germany, 1999, ISBN 3-540-66222-7.

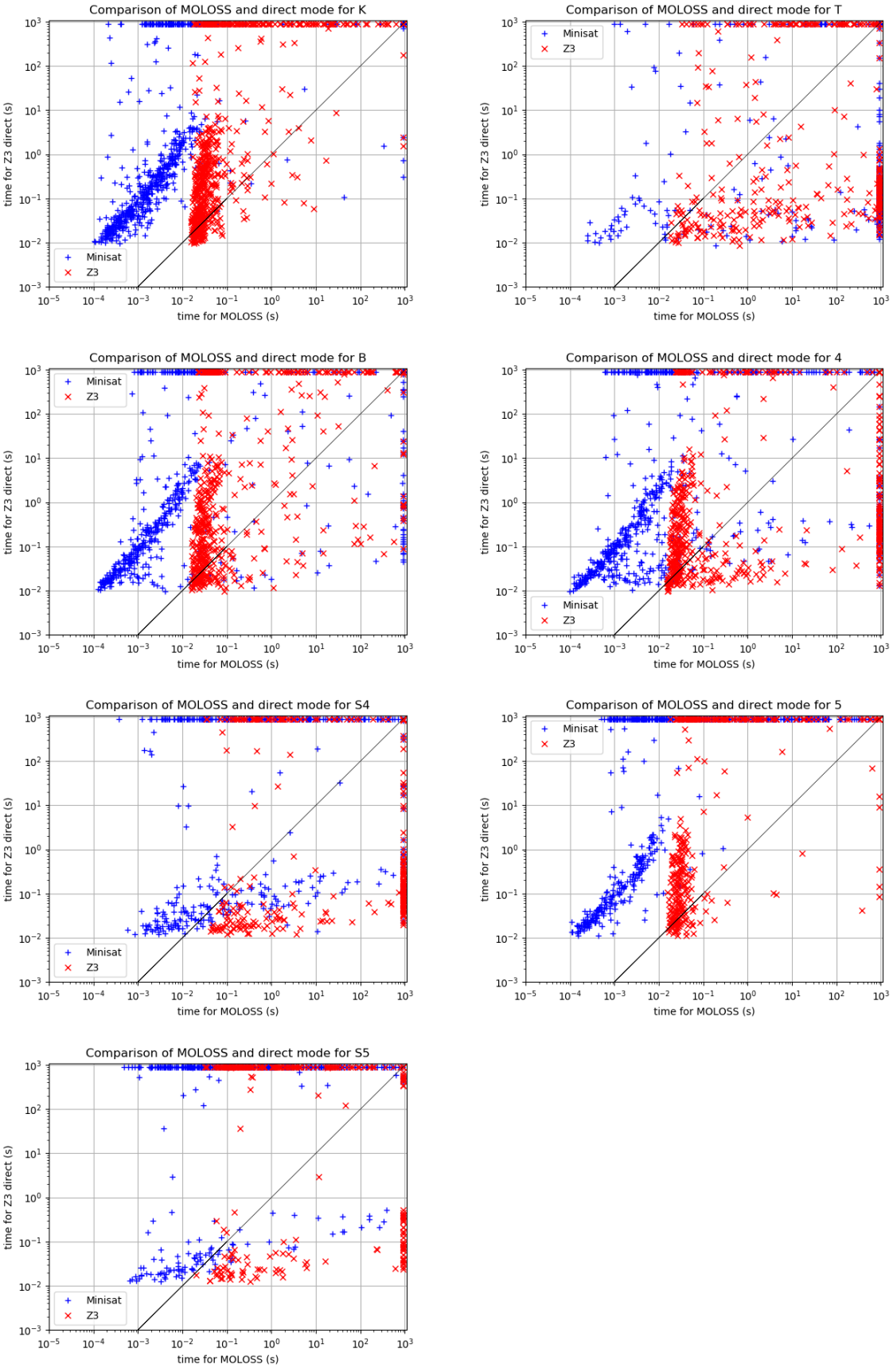

FIGURE 1 – Temps d'exécution de MOLOSS pour différents modes