Codage SMT dans un espace de plans (liens causaux) pour la planification temporelle en temps continu

Frédéric Maris, Maël Valais, Julien Vianey

Codage SMT dans un espace de plans (liens causaux) pour la planification temporelle en temps continu

Frédéric Maris Maël Valais Julien Vianey
IRIT, Université de Toulouse

{maris, mael.valais, julien.vianey}@irit.fr

Résumé

L'amélioration considérable des solveurs SAT a permis de les utiliser pour la résolution de divers problèmes d'intelligence artificielle, et en particulier la génération automatique de plans d'actions. Récemment, des solveurs SMT (SAT Modulo Theory) prometteurs basés sur des solveurs SAT efficaces ont été développés. Il est donc intéressant d'utiliser le langage SMT qui nous permet de produire directement des codages de problèmes de planification temporelle sans avoir à discrétiser le temps ou à utiliser un gestionnaire de contraintes temporelles externe (Time Map Manager). Cette approche a déjà été explorée, mais la plupart des codages existants considèrent une modélisation discrète du temps. Nous introduisons dans cet article une nouvelle traduction de problèmes de planification temporelle en temps continu, basée sur un codage dans un espace de plans (liens causaux), en formules propositionnelles modulo QF-RDL (Quantifier Free Rational Difference Logic).

Mots Clef

Planification, codages, temps, SAT, SMT.

1 Introduction

Dans le planificateur SATPLAN [8], un problème de planification est transformé en une formule propositionnelle dont les modèles, correspondant aux plans solution, peuvent être trouvés en utilisant un solveur SAT. Cette approche SAT recherche un plan solution de longueur fixe k. En cas d'échec à trouver un tel plan, cette longueur est augmentée avant de recommencer la recherche d'une solution. Cette approche bénéficie directement des améliorations des solveurs SAT [1]. L'exemple le plus évident est le planificateur BLACKBOX [10, 11] (et ses successeurs SATPLAN'04 [7] et SATPLAN'06 [12]). Ces planificateurs ont obtenu la première place dans la catégorie planificateur optimal (en termes de nombre d'étapes du plan) des compétitions internationales de planification [2] IPC-2004 et IPC-2006. C'était inattendu car ces planificateurs étaient essentiellement des mises à jour de BLACKBOX et n'incluaient aucune réelle nouveauté : l'amélioration des performances était principalement due aux progrès du solveur SAT sous-jacent.

De nombreuses améliorations de cette approche originale ont été proposées, notamment via le développement de codages plus compacts et efficaces [9, 3, 13, 14, 18, 19, 20]. Suite à ces travaux, de nombreuses autres techniques similaires pour le codage de problèmes de planification ont été développées : programmation linéaire (LP) [23], problèmes de satisfaction de contraintes (CSP) [2]. En particulier, différents types de codages SMT ont aussi été proposés pour résoudre des problèmes de planification temporelle. Les codages de [22] et [21] se basent sur une représentation discrète du temps, alors que dans [17, 16] nous utilisons des atomes de QF-RDL (Quantifier Free Rational Difference Logic) pour coder un temps continu. Dans la section 2, nous définissons notre modèle de planification temporelle, précédemment introduit dans [1]. Nous décrivons dans la section 3 un nouveau codage SAT basé sur des conditions ouvertes (liens causaux) pour la planification classique. Ensuite, dans la section 4, nous proposons une adaptation SMT de ce codage pour la planification temporelle en temps continu.

2 Planification temporelle

Nous étudions la planification temporelle propositionnelle dans un langage basé sur les aspects temporels de PDDL2.1 [4]. Un fluent est une proposition atomique positive ou négative. Comme dans PDDL2.1, nous considérons que les changements des valeurs des fluents sont instantanés mais que les conditions sur la valeur des fluents peuvent être imposées sur un intervalle. Une action a est un quadruplet $\langle Cond(a), Add(a), Del(a), Constr(a)\rangle$, où l'ensemble des conditions $Cond(a)$ est l'ensemble des fluents qui doivent être vrais pour que a soit exécutée, l'ensemble des ajouts $Add(a)$ est l'ensemble des fluents qui sont établis par a, l'ensemble des retraits $Del(a)$ est l'ensemble des fluents qui sont détruits par a, et l'ensemble de contraintes $Constr(a)$ est un ensemble de

1. http://www.satcompetition.org/
2. http://www.icaps-conference.org/index.php/Main/Competitions

contraintes entre les temps relatifs des événements qui se produisent pendant l'exécution de a. Un événement correspond à l'une des quatre possibilités : l'établissement ou la destruction d'un fluent par une action a, ou le début ou la fin d'un intervalle sur lequel un fluent est requis par une action a. Dans PDDL2.1, les événements ne peuvent se produire qu'au début ou à la fin des actions, mais nous relâchons cette hypothèse de sorte que les événements peuvent se produire à tout moment à condition que les contraintes $Constr(a)$ soient satisfaites. Notons que $Add(a) \cap Del(a)$ peut être non vide. En effet, il n'est pas inhabituel pour une action durative d'établir un fluent au début de l'action et de le détruire à sa fin. Nous pouvons également observer que la durée d'une action, le temps entre le premier et le dernier événement de l'action, n'a pas besoin d'être explicitement stockée.

Nous utilisons la notation $a \rightarrow f$ pour désigner l'événement que l'action a établit le fluent f, $a \rightarrow \neg f$ pour désigner l'événement que a détruit f, et $f \mid\!\rightarrow a$ et $f \rightarrow\!\mid a$, respectivement, pour indiquer le début et la fin de l'intervalle sur lequel a requiert la condition f. Si f est déjà vrai (respectivement, faux) lorsque l'événement $a \rightarrow f$ ($a \rightarrow \neg f$) se produit, nous considérons toujours que a établit (détruit) f. Un plan temporel peut contenir plusieurs instances de la même action, mais par simplicité de notation, nous ferons seulement la distinction entre actions et instances d'action si cela est absolument nécessaire. Nous utilisons la notation $\tau(e)$ pour représenter l'instant dans un plan où un événement e se produit. Pour une action donnée (ou une instance d'action) a, $Events(a)$ représente les différents événements qui constituent sa définition, à savoir $a \rightarrow f$ pour tout f dans $Add(a)$, $a \rightarrow \neg f$ pour tout f dans $Del(a)$, $f \mid\!\rightarrow a$ et $f \rightarrow\!\mid a$ pour tout f dans $Cond(a)$. La définition d'une action a inclut les contraintes $Constr(a)$ sur les instants relatifs aux événements dans $Events(a)$. Comme dans PDDL2.1, nous considérons que la durée entre événements dans $Events(a)$ n'est pas nécessairement fixe et que $Constr(a)$ est un ensemble de contraintes d'intervalle sur des paires d'événements, tels que $\tau(f \rightarrow\!\mid a) - \tau(f \mid\!\rightarrow a) \in [\alpha, \beta]$ pour des constantes α, β. Nous utilisons $[\alpha_a(e_1, e_2), \beta_a(e_1, e_2)]$ pour désigner l'intervalle de valeurs possibles pour la distance relative entre les événements e_1, e_2 de l'action a. Une durée fixe entre les événements $e_1, e_2 \in Events(a)$ peut, bien sûr, être modélisée en définissant $\alpha_a(e_1, e_2) = \beta_a(e_1, e_2)$. De même, l'absence de contrainte peut être modélisée par l'intervalle $[-\infty, +\infty]$. Nous introduisons maintenant deux contraintes fondamentales que tous les plans temporels doivent satisfaire :

— les *contraintes inhérentes* à l'ensemble des instances d'action $\mathcal{A}$: pour toute $a \in \mathcal{A}$, a satisfait $Constr(a)$, c'est-à-dire pour toutes les paires d'événements $e_1, e_2 \in Events(a)$ nous avons $\tau(e_1) - \tau(e_2) \in [\alpha_a(e_1, e_2), \beta_a(e_1, e_2)]$;

— les *contraintes d'effets contradictoires* sur l'ensemble des instances d'action $\mathcal{A}$: pour toutes $a_i, a_j \in \mathcal{A}$, pour tout fluent positif $f \in Del(a_i) \cap Add(a_j)$ nous avons $\tau(a_i \rightarrow \neg f) \neq \tau(a_j \rightarrow f)$.

Les contraintes inhérentes définissent la structure interne de chaque instance d'action, alors que les contraintes d'effets contradictoires assurent que la valeur de vérité de chaque fluent ne soit jamais indéfinie lors de l'exécution d'un plan temporel.

Définition 1 *Un problème de planification temporelle $\langle I, \mathcal{A}, G\rangle$ consiste en un ensemble d'actions $\mathcal{A}$, un état initial I et un but G, où I et G sont des ensembles de fluents.*

Notation Si $\mathcal{A}$ est un ensemble d'instances d'actions, alors $Events(\mathcal{A})$ est l'union des ensembles $Events(a)$ (pour toutes les instances d'action $a \in \mathcal{A}$).

Définition 2 *$P = \langle \mathcal{A}, \tau\rangle$, où $\mathcal{A}$ est un ensemble fini d'instances d'actions $\{a_1, \ldots, a_n\}$ et τ est une fonction à valeurs réelles sur $Events(\mathcal{A})$, est un plan temporel pour le problème $\langle I, \mathcal{A}', G\rangle$ si*

(1) $\mathcal{A} \subseteq \mathcal{A}'$, et

(2) P vérifie les contraintes inhérentes et contradictoires sur $\mathcal{A}$;

et lorsque P est exécuté (c'est-à-dire que les fluents sont établis ou détruits aux instants donnés par τ) à partir de l'état initial I :

(3) pour toute $a_i \in \mathcal{A}$, chaque $f \in Cond(a_i)$ est vrai lorsqu'il est requis, et

(4) tous les fluents $g \in G$ sont vrais à la fin de l'exécution de P.

(5) P est robuste sous des changements infinitésimaux dans les temps de démarrage des actions.

Les événements sont instantanés, alors que les actions ont non seulement une durée mais celle-ci peut aussi être de longueur variable. Ainsi, un plan temporel P ne planifie pas directement ses instances d'action mais planifie tous les événements dans ses instances d'actions. La condition (5) de la définition 2 signifie que nous interdisons les plans qui exigent une synchronisation parfaite entre différentes actions. Cette condition peut être imposée dans PDDL2.1 [5]. Nous exigeons que dans tous les plans, les fluents soient établis strictement avant le début de l'intervalle sur lequel ils sont requis. La seule exception à cette règle est lorsqu'un fluent f est établi et requis par la même action a. Nous permettons la possibilité d'une parfaite synchronisation au sein d'une action, ce qui signifie que nous pouvons avoir $\tau(a \rightarrow f) = \tau(f \mid\!\rightarrow a)$. De même, les fluents ne peuvent être détruits qu'après la fin de l'intervalle sur lequel ils sont requis. La seule exception à cette regle est quand un fluent f est requis

et détruit par une action a, auquel cas on peut avoir $\tau(f \rightarrow| \ a) = \tau(a \rightarrow \neg f)$.

Définition 3 *Un problème de planification temporelle $\langle I, \mathcal{A}, G\rangle$ est positif s'il n'y a pas de fluents négatifs dans les conditions d'actions ni dans le but G.*

Dans cet article, nous ne considèrerons que des problèmes de planification temporelle positifs $\langle I, \mathcal{A}, G\rangle$. Il est bien connu que tout problème de planification peut être transformé en un problème positif équivalent en temps linéaire par l'introduction, pour chaque fluent positif f, d'un nouveau fluent pour remplacer les occurrences de $\neg f$ dans les conditions d'actions [6]. En supposant que tous les problèmes sont positifs, G et $Cond(a)$ (pour toute action a) sont composés de fluents positifs. Par convention, $Add(a)$ et $Del(a)$ sont aussi composés exclusivement de fluents positifs. L'état initial I, cependant, peut contenir des fluents négatifs.

3 Codage SAT pour la planification classique

Dans cette section, nous considérons la planification classique comme restriction de la planification temporelle décrite dans la section précédente. Un problème de planification positif est donc ici un triplet $\langle I, \mathcal{A}, G\rangle$ tel que toutes les actions de $\mathcal{A}$ sont instantanées (c'est-à-dire $\forall a \in \mathcal{A}, \forall e_1, e_2 \in Events(a), \tau(e_1) = \tau(e_2)$). De plus, nous considérons pour tout plan $P = \langle \mathcal{A}, \tau\rangle$, que τ est une fonction à valeurs entières de $Events(\mathcal{A})$ dans $N = \{1, \ldots, n\}$ l'ensemble ordonné des index d'étapes de P.
Les codages SAT dans les espaces de plans existants introduits par [15] se réfèrent tous à trois étapes indexées (pas nécessairement consécutives) du plan. Dans notre nouveau codage, nous allons nous référer seulement à deux étapes consécutives. Pour chaque action $a \in \mathcal{A}$ et chaque étape i, nous créons une variable propositionnelle a_i pour déterminer qu'une instance de a doit être planifiée à l'étape i. Pour chaque fluent $f \in \mathcal{F}$ et chaque étape i, nous créons une variable propositionnelle $open_{f,i}$ pour exprimer que f se maintient à l'étape précédente $i-1$ et doit être protégé au moins jusqu'à l'étape i.

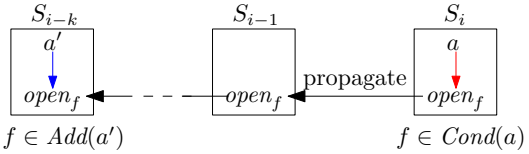

FIGURE 1 – Lien causal : l'action a' produit f à l'étape S_{i-k} pour l'action a qui nécessite f à l'étape S_i.

Dans la figure 1, la variable f est une *condition ouverte* à l'étape S_i, impliquant que $f \in I$ ou une action a' qui ajoute f est exécutée dans une étape précédente S_{i-k}. Les conditions ouvertes doivent être propagées vers l'arrière jusqu'à l'état initial ou une étape dans laquelle elles sont ajoutées par une action.
Dans la suite, nous donnons notre codage pour une longueur de plan fixée *length*. Une borne supérieure pour *length* est le nombre total d'états possibles, soit $2^{|\mathcal{F}|}$.

Conditions ouvertes Si une action a est exécutée dans une étape du plan, alors chaque condition de a doit être une condition ouverte à cette étape (c'est-à-dire qu'un lien causal est requis pour cette condition).

$$\bigwedge_{\mathbf{i}\in[1..length]} \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i} \Rightarrow \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} open_{\mathbf{f},\mathbf{i}} \right)$$

Dans la dernière étape du plan menant au but tous les fluents du but doivent être des conditions ouvertes ou ajoutées par des actions exécutées dans cette étape.

$$\bigwedge_{\mathbf{f}\in\mathbf{G}} \left(open_{\mathbf{f},length} \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Add(\mathbf{a})}} \mathbf{a}_{length} \right)$$

Propagation et fermeture Aucune condition ne doit rester ouverte dans la première étape du plan si elle n'est pas fournie dans l'état initial.

$$\bigwedge_{\mathbf{f}\in\mathcal{F}\setminus\mathbf{I}} \neg open_{\mathbf{f},1}$$

Toute condition ouverte dans une étape doit soit rester ouverte soit être ajoutée (fermée) par une action à l'étape précédente.

$$\bigwedge_{\mathbf{i}\in[2..length]} \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(open_{\mathbf{f},\mathbf{i}} \Rightarrow \left(open_{\mathbf{f},\mathbf{i}-1} \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Add(\mathbf{a})}} \mathbf{a}_{\mathbf{i}-1} \right) \right)$$

Protection des conditions ouvertes Une condition ouverte dans une étape donnée ne peut pas être supprimée à l'étape précédente. Cela garantit de ne rompre aucun lien de causalité dans le plan.

$$\bigwedge_{\mathbf{i}\in[2..length]} \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(open_{\mathbf{f},\mathbf{i}} \Rightarrow \bigwedge_{\substack{\mathbf{a}\in\mathcal{A} \\ \mathbf{f}\in Del(\mathbf{a})}} \neg \mathbf{a}_{\mathbf{i}-1} \right)$$

Prévention des interactions négatives Si une action supprime un fluent qui est nécessaire ou est ajouté par une autre action, alors ces deux actions ne peuvent pas être exécutées toutes les deux dans une même étape.

$$\bigwedge_{\mathbf{i}\in[1..\mathbf{length}]}\ \bigwedge_{\mathbf{a}\in\mathcal{A}}\ \bigwedge_{\mathbf{f}\in(Add(\mathbf{a})\cup Cond(\mathbf{a}))}\ \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ \mathbf{a}\neq\mathbf{b}\wedge\mathbf{f}\in Del(\mathbf{b})}} (\neg\mathbf{a_i}\vee\neg\mathbf{b_i})$$

Dans la section suivante, nous étendons ce codage SAT pour la planification classique en un codage SMT pour la planification temporelle en temps continu.

4 Codage SMT pour la planification temporelle

Nous introduisons une adaptation SMT du codage SAT basé sur les conditions ouvertes que nous avons introduit dans la section précédente pour la planification classique. Ici, les actions ne sont plus instantanées et des contraintes sur les instants $\tau(e)$ auxquels se produisent des événements $e \in Events(\mathcal{A})$ doivent explicitement être ajoutées.
Considérons donc maintenant un problème de planification temporelle positif $\langle I, \mathcal{A}, G\rangle$. Pour chaque instance d'action a et chaque fluent $f \in Cond(a)$, nous introduisons deux variables $\tau_s(open_f)$ et $\tau_e(open_f)$ qui nous permettent de définir un intervalle temporel de protection de lien causal depuis l'état initial ou une étape contenant une action qui produit f vers une étape qui contient l'instance d'action a. Durant cet intervalle temporel, aucune action ne pourra détruire f.

Conditions ouvertes Si une action a est exécutée dans une étape du plan, alors chaque condition de a doit être une condition ouverte à cette étape (c'est-à-dire qu'un lien causal est requis pour cette condition). De plus, le début de l'intervalle sur lequel cette condition est requise se trouve dans l'intervalle de protection du lien causal $[\tau_s(open_f); \tau_e(open_f)]$.

$$\bigwedge_{\mathbf{i}\in[1..\mathbf{length}]}\ \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i} \Rightarrow \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} \begin{pmatrix} open_{\mathbf{f},\mathbf{i}} \\ \wedge\,(\tau(\mathbf{f}\mid\!\rightarrow \mathbf{a_i}) \geq \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \wedge\,(\tau(\mathbf{f}\mid\!\rightarrow \mathbf{a_i}) \leq \tau_e(open_{\mathbf{f},\mathbf{i}})) \end{pmatrix}\right)$$

Dans la dernière étape du plan menant au but tous les fluents du but doivent être des conditions ouvertes ou ajoutées par des actions exécutées dans cette étape. Dans le cas où une action a ajoute une condition ouverte f, le début de l'intervalle de protection du lien causal correspondant est fixé à l'instant où a produit f.

$$\bigwedge_{\mathbf{f}\in\mathbf{G}} \left(\begin{array}{l} open_{\mathbf{f},length} \\ \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Add(\mathbf{a})}} \begin{pmatrix} \mathbf{a}_{length} \\ \wedge\,(\tau(\mathbf{a}_{length}\rightarrow\mathbf{f}) = \tau_s(open_{\mathbf{f},length})) \\ \wedge\,(\tau_{Goal} = \tau_e(open_{\mathbf{f},length})) \end{pmatrix} \end{array}\right)$$

Propagation et fermeture Aucune condition ne doit rester ouverte dans la première étape du plan si elle n'est pas fournie dans l'état initial.

$$\bigwedge_{\mathbf{f}\in\mathcal{F}\setminus\mathbf{I}} \neg open_{\mathbf{f},1}$$

Si une condition ouverte est fournie par l'état initial, alors le début de l'intervalle de protection du lien causal correspondant est fixé à l'instant initial τ_{Init}.

$$\bigwedge_{\mathbf{f}\in\mathbf{I}} (open_{\mathbf{f},1} \Rightarrow (\tau_{Init} = \tau_s(open_{\mathbf{f},1})))$$

Toute condition ouverte f dans une étape doit à l'étape précédente : (1) soit rester ouverte et dans ce cas l'intervalle de protection du lien causal correspondant reste le même pour ces deux étapes, (2) soit être ajoutée (fermée) par une instance d'action a et dans ce cas le début de l'intervalle de protection du lien causal est fixé à l'instant de production de f par a.

$$\bigwedge_{\mathbf{i}\in[2..length]}\ \bigwedge_{\mathbf{f}\in\mathcal{F}} \left(\begin{array}{l} open_{\mathbf{f},\mathbf{i}} \Rightarrow \\ \left(\begin{array}{l} \begin{pmatrix} open_{\mathbf{f},\mathbf{i}-1} \\ \wedge\,(\tau_s(open_{\mathbf{f},\mathbf{i}-1}) = \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \wedge\,(\tau_e(open_{\mathbf{f},\mathbf{i}-1}) = \tau_e(open_{\mathbf{f},\mathbf{i}})) \end{pmatrix} \\ \vee \bigvee_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Add(\mathbf{a})}} (\mathbf{a_{i-1}} \wedge (\tau(\mathbf{a_{i-1}}\rightarrow\mathbf{f}) = \tau_s(open_{\mathbf{f},\mathbf{i}}))) \end{array}\right) \end{array}\right)$$

Protection des conditions ouvertes Une condition ouverte dans une étape donnée ne peut pas être supprimée à l'intérieur de l'intervalle de protection du lien causal correspondant $[\tau_s(open_f); \tau_e(open_f)]$. Cela garantit de ne rompre aucun lien de causalité dans le plan.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}}\ \bigwedge_{\mathbf{f}\in\mathcal{F}}\ \bigwedge_{\substack{\mathbf{a}\in\mathcal{A}\\ \mathbf{f}\in Del(\mathbf{a})}} \left((open_{\mathbf{f},\mathbf{i}} \wedge \mathbf{a_j}) \Rightarrow \begin{pmatrix} (\tau(\mathbf{a_j}\rightarrow\neg\mathbf{f}) < \tau_s(open_{\mathbf{f},\mathbf{i}})) \\ \vee\,(\tau_e(open_{\mathbf{f},\mathbf{i}}) < \tau(\mathbf{a_j}\rightarrow\neg\mathbf{f})) \end{pmatrix}\right)$$

Prévention des interactions négatives Si une action b supprime, à un instant $\tau(b \rightarrow \neg f)$, un fluent f qui est ajouté par une autre action a à un instant $\tau(a \rightarrow f)$, alors ces deux instants sont différents.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}}\ \bigwedge_{\mathbf{a}\in\mathcal{A}}\ \bigwedge_{\mathbf{f}\in Add(\mathbf{a})}\ \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ ((\mathbf{i}\neq\mathbf{j})\vee(\mathbf{a}\neq\mathbf{b}))\wedge\mathbf{f}\in Del(\mathbf{b})}} ((\mathbf{a_i}\wedge\mathbf{b_j}) \Rightarrow (\tau(\mathbf{a_i}\rightarrow\mathbf{f}) \neq \tau(\mathbf{b_j}\rightarrow\neg\mathbf{f})))$$

De même, si une action b supprime, à un instant $\tau(b \rightarrow \neg f)$, un fluent f qui est nécessaire à une autre action a sur un intervalle temporel $[\tau(f \mid\rightarrow a), \tau(f \rightarrow\mid a)]$, alors cet instant ne peut être contenu dans cet intervalle.

$$\bigwedge_{\substack{\mathbf{i}\in[1..length]\\ \mathbf{j}\in[1..length]}} \bigwedge_{\mathbf{a}\in\mathcal{A}} \bigwedge_{\mathbf{f}\in Cond(\mathbf{a})} \bigwedge_{\substack{\mathbf{b}\in\mathcal{A}\\ ((\mathbf{i}\neq\mathbf{j})\vee(\mathbf{a}\neq\mathbf{b}))\wedge\mathbf{f}\in Del(\mathbf{b})}} \left((\mathbf{a_i}\wedge\mathbf{b_j})\Rightarrow\begin{pmatrix}(\tau(\mathbf{f}\rightarrow\mid\mathbf{a_i})<\tau(\mathbf{b_j}\rightarrow\neg\mathbf{f}))\\ \vee(\tau(\mathbf{b_j}\rightarrow\neg\mathbf{f})<\tau(\mathbf{f}\mid\rightarrow\mathbf{a_i}))\end{pmatrix}\right)$$

Bornes du plan temporel Enfin, nous devons ajouter une contrainte pour maintenir le plan dans un intervalle de temps borné par l'étape initiale qui produit l'état initial I et l'étape finale qui nécessite tous les fluents du but G.

$$\bigwedge_{\mathbf{i}\in[1..length]} \bigwedge_{\mathbf{a}\in\mathcal{A}} \left(\mathbf{a_i}\Rightarrow\begin{pmatrix}\bigwedge_{\mathbf{f}\in Cond(\mathbf{a})}\begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{f}\mid\rightarrow\mathbf{a_i}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{f}\rightarrow\mid\mathbf{a_i}))\end{pmatrix}\\ \wedge\bigwedge_{\mathbf{f}\in Add(\mathbf{a})}\begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{a_i}\rightarrow\mathbf{f}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{a_i}\rightarrow\mathbf{f}))\end{pmatrix}\\ \wedge\bigwedge_{\mathbf{f}\in Del(\mathbf{a})}\begin{pmatrix}(\tau_{Init}\leq\tau(\mathbf{a_i}\rightarrow\neg\mathbf{f}))\\ \wedge(\tau_{Goal}\geq\tau(\mathbf{a_i}\rightarrow\neg\mathbf{f}))\end{pmatrix}\end{pmatrix}\right)$$

5 Conclusion et perspectives

Après avoir introduit notre modèle de problèmes de planification temporelle basé sur les aspects temporels de PDDL2.1, nous avons présenté un nouveau codage SAT dans un espace de plans basé sur des conditions ouvertes (liens causaux) pour la planification classique. Nous avons proposé une adaptation SMT de ce codage SAT pour la planification temporelle en temps continu en introduisant des atomes de QF-RDL. Il est maintenant nécessaire de tester ce codage sur les problèmes de référence des compétitions internationales de planification (IPC) et de le comparer avec les autres approches SMT existantes.
Une autre voie à explorer serait d'intégrer cette modélisation des conditions ouvertes dans notre planificateur TLP-GP [17, 16] qui couple un algorithme de recherche dans un graphe de planification temporel à la résolution de contraintes SMT.

Références

[1] Martin C. Cooper, Frederic Maris, and Pierre Régnier. Monotone temporal planning : Tractability, extensions and applications. *J. Artif. Intell. Res. (JAIR)*, 50 :447–485, 2014.

[2] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP. *Artif. Intell.*, 132(2) :151–182, 2001.

[3] M. Ernst, T. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, 1997.

[4] Maria Fox and Derek Long. PDDL2.1 : an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20 :61–124, 2003.

[5] Maria Fox, Derek Long, and Keith Halsey. An investigation into the expressive power of PDDL2.1. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–342. IOS Press, 2004.

[6] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[7] H. Kautz. Satplan'04 : Planning as satisfiability. In *Abstracts of the 4th International Planning Competition, IPC-04*, 2004.

[8] H. Kautz and B. Selman. Planning as satisfiability. In *ECAI-92*, pages 359–363, 1992.

[9] H. Kautz and B. Selman. Pushing the envelope : Planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[10] H. Kautz and B. Selman. BLACKBOX : A new approach to the application of theorem proving to problem solving. In *Proc. of AIPS-98 Workshop on Planning as Combinatorial Search*, 1998.

[11] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In *Proc. IJCAI-99*, pages 318–325, 1999.

[12] H. Kautz, B. Selman, and J. Hoffmann. Satplan'06 : Planning as satisfiability. In *Abstracts of the 5th International Planning Competition, IPC-06*, 2006.

[13] A. Mali and S. Kambhampati. Refinement-based planning as satisfiability. In *Proc. Workshop planning as combinatorial search, AIPS-98*, 1998.

[14] A. Mali and S. Kambhampati. On the utility of plan-space (causal) encodings. In *Proc. AAAI-99*, pages 557–563, 1999.

[15] Amol Dattatraya Mali and Subbarao Kambhampati. On the utility of plan-space (causal) encodings. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 557–563. AAAI Press / The MIT Press, 1999.

[16] Frédéric Maris. *Planification SAT et planification temporellement expressive : les systèmes TSP et TLP-GP.* Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 2009.

[17] Frederic Maris and Pierre Régnier. TLP-GP : new results on temporally-expressive planning benchmarks. In *(ICTAI 2008), Vol. 1*, pages 507–514. IEEE Computer Society, 2008.

[18] J. Rintanen. Symmetry reduction for sat representations of transition systems. In *Proc. ICAPS-03*, 2003.

[19] J. Rintanen, K. Heljanko, and Niemelä. Parallel encodings of classical planning as satisfiability. In *Proc. European Conference on Logics in Artificial Intelligence, JELIA-04*, 2004.

[20] J. Rintanen, K. Heljanko, and Niemelä. Planning as satisfiability : parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(1213) :1031–1080, 2006.

[21] Jussi Rintanen. Discretization of temporal models with application to planning with SMT. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3349–3355. AAAI Press, 2015.

[22] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artif. Intell.*, 166(1-2) :194–253, 2005.

[23] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proc. 16th International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, 1999*, pages 310–317. Morgan Kaufmann, 1999.