Vérification d'invariants pour des systèmes spécifiés en logique de réécriture

Vlad Rusu, Manuel Clavel

Vérification d'invariants pour des systèmes spécifiés en logique de réécriture

Vlad Rusu* — **Manuel Clavel****

* *Inria Rennes Bretagne-Atlantique [3pt]*
rusu@irisa.fr *[6pt]*
** *Universidad Complutense de Madrid, Spain [3pt]*
clavel@sip.ucm.es

RÉSUMÉ. *Nous présentons une approche basée sur la preuve inductive pour vérifier des invariants de systèmes spécifiés en* logique de réécriture*, un langage de spécification formelle implémenté dans l'outil Maude. Un invariant est une propriété qui est vraie dans tous les états atteignables à partir d'une certaine classe d'états initiaux. Notre approche consiste à coder les propriétés d'invariance de la logique de réécriture en* logique équationnelle avec appartenance*, une sous-logique de la logique de réécriture également implémentée dans Maude. Ce codage nous permet ensuite de prouver les propriétés d'invariance à l'aide d'un assistant de preuve disponible pour la logique équationnelle de Maude. Nous montrons que notre codage est correct, pour un sous-ensemble bien identifié (et suffisant en pratique) de systèmes et de propriétés d'invariance, et illustrons notre approche sur une version à n processus de l'algorithme Bakery.*

ABSTRACT. *We present an approach based on inductive theorem proving for verifying invariants of dynamic systems specified in rewriting logic, a formal specification language implemented in the Maude system. An invariant is a property that holds on all the states that are reachable from a given class of initial states. Our approach consists in encoding the semantic aspects that are relevant for our task (namely, verifying invariance properties of the specified systems) in membership equational logic, a sublogic of rewriting logic. The invariance properties are then formalized over the encoded rewrite theories and are proved using an inductive theorem prover for membership equational logic also implemented in the Maude system using its reflective capabilities. We illustrate our approach by verifying mutual exclusion properties of a readers-writers system and of an n-process version of the Bakery algorithm.*

MOTS-CLÉS : *Vérification, logique de réécriture, Maude*

KEYWORDS: *Verification, Rewriting Logic, Maude*

1. Introduction

La logique de réécriture (*Rewriting Logic* [1], ou RL dans la suite de cet article) est un langage de spécification formelle dans lequel la dynamique d'un système est exprimée à l'aide de *règles de réécriture* qui agissent sur l'*état* du système, lui-même défini dans une *logique équationnelle* sous-jacente à la logique de réécriture. De nombreux auteurs ont mis en évidence l'adéquation de la logique de réécriture pour modéliser de nombreux types de systèmes dynamiques, comme par exemple les réseaux actifs [2], les sytèmes biologiques [3], ou la sémantique des langages de programmation [4]. Plusieurs systèmes implémentent des versions de cette logique : Maude [5], Elan [6], et CAFEOBJ [7].

La logique équationnelle sous-jacente à la logique de réécriture de Maude est la logique équationnelle avec appartenance (*Membership Equational Logic* [8], ou MEL dans la suite de cet article), qui permet d'exprimer, en plus des équations usuelles entre termes, *l'appartenance* d'un terme à une sorte. Notre approche pour la preuve inductive d'invariants utilise de telles appartenances, qui nous permettent de définir la sorte des termes accessibles à partir d'une classe de termes initiaux dans RL.

Le système Maude [5] donne une syntaxe concrète aux logiques MEL et RL, et fournit des outils qui permettent d'exécuter ces spécifications, de les analyser, et de les vérifier. Parmi ces ces outils, on trouve un explorateur exhaustif (à profondeur bornée) de l'ensemble des termes accessibles à partir d'un terme initial, et un *model checker* pour des propriétés exprimées dans la logique temporelle linéaire [9]. Ces deux outils sont limités, par nature, à des systèmes finis. Certaines classes de systèmes infinis peuvent aussi être vérifiées, en passant par des *abstractions équationnelles* [10] qui réduisent un système infini vers un système fini, donc vérifiable par *model checking*. Cependant, ces abstractions ne préservent pas les propriétés d'invariance, en général, mais seulement dans des cas particuliers [10].

Les limites de la vérification automatisée justifient notre approche, basée sur la preuve assistée de théorèmes, pour prouver des propriétés d'invariance de systèmes dynamiques spécifiés en Maude. Notre approche consiste à traduire, de manière automatique, les propriétés d'in-

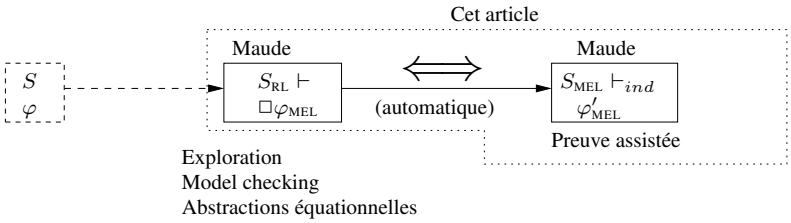

Figure 1 – Notre approche dans le contexte Maude.

variance de spécifications en RL vers des propriétés inductives de spécifications MEL. Nous démontrons que la traduction est correcte, en ce sens qu'elle transforme un invariant qui est *vrai* sur une spécification RL donnée, vers une propriété (inductivement) *vraie* sur la théorie MEL obtenue par traduction. Ceci permet la vérification des invariants en utilisant des prouveurs inductifs pour MEL tel que l'outil ITP [11].

L'approche proposée complète donc la panoplie d'outils de vérification disponible pour Maude. Elle peut être utilisée en combinaison avec ces autres outils (cf. Figure 1). Une utilisation typique est la suivante : étant donné un système dynamique S et une propriété φ sur l'état du système, on commence par les formaliser, respectivement, par une spécification RL S_{RL} et par une formule de MEL φ_{MEL}. Ensuite, pour vérifier que φ_{MEL} est un invariant de S_{RL} (partant d'un ensemble d'états initiaux donnés) l'utilisateur peut, d'une part, essayer de falsifier l'invariance $S_{\text{RL}} \vdash \Box\varphi_{\text{MEL}}$ en utilisant le *model checker* de Maude, ou plus simplement l'exploration bornée des termes accessibles ; ou bien, l'utilisateur peut essayer de prouver la propriété avec le model checker, éventuellement en passant par des abstractions équationnelles, ou encore en utilisant notre approche basée sur la preuve assistée.

Le reste de l'article est organisé de la façon suivante. Dans la section 2 nous présentons brièvement les logiques MEL et RL. Nous définissons ensuite la notion d'*invariant φ (exprimé en logique* MEL*) d'une spécification $\mathcal{R}$ (exprimée en* RL*) à partir d'une classe initiale d'états t_0(représentée par un terme t_0, qui peut contenir des variables libres)*, ainsi : $\varphi(t)$ est prouvable dans le modèle standard de la sous-théorie MEL de $\mathcal{R}$, pour tous terme t accessible depuis t_0 par *réécriture close*

à la racine dans $\mathcal{R}$. Nous notons l'invariance par $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$. La réécriture close à la racine veut dire, intuitivement, que le filtrage se fait à la racine, et que les substitutions utilisées sont closes, pour toutes les variables apparaîssant dans le terme réécrit et dans la régle de réécriture. Nous donnons des arguments en faveur de l'adéquation de ce type de réécriture pour exprimer la dynamique de systèmes exprimés en RL, et plus particulièrement lorsque les états initiaux sont exprimés par des termes à variables libres et les règles de réécriture comportent des variables supplémentaires en partie droite et/ou dans les conditions associées. Ces aspects sont essentiels lorsqu'on souhaite modéliser des systèmes à nombre paramétrique de processus, tel que l'algorithme Bakery à n processus qui constitue notre application.

Le cœur de notre approche est présenté en section 3. D'abord, on définit une traduction automatique qui prend une théorie RL $\mathcal{R}$ et un terme t_0, et qui engendre une théorie MEL $\mathcal{M}(\mathcal{R}, t_0)$ qui enrichit la sous-théorie MEL de $\mathcal{R}$ avec une nouvelle sorte, appelée *Reachable*, définie inductivement à partir des règles de $\mathcal{R}$ et du terme t_0. Ensuite, on démontre que les affirmations *être de la sorte Reachable dans* $\mathcal{M}(\mathcal{R}, t_0)$ et *être accessible dans* $\mathcal{R}$ *à partir de* t_0 *par réécriture close à la racine* sont équivalentes. Puis, nous donnons une définition alternative à l'invariance dans $\mathcal{R}$ à partir de t_0, comme suit : $\varphi(t)$ *est vraie dans le modèle standard (appelé aussi le modèle initial) de la théorie* $\mathcal{M}(\mathcal{R}, t_0)$ *construite précédemment, et ce, pour tous les termes clos ayant la sorte Reachable dans le modèle initial*. Enfin, nous prouvons que les deux définitions d'invariance sont équivalentes ; l'avantage de la deuxième définition est qu'elle permet l'emploi de prouveurs inductifs pour la logique MEL, tel que l'outil ITP. L'utilisation de notre approche pour prouver l'exclusion mutuelle de l'algorithme Bakery à n processus est décrite en section 4. Nous concluons et présentons des travaux connexes et futurs en section 5.

2. Logique équationnelle avec appartenance et logique de réécriture

Nous présentons ici brièvement ces deux logiques ; des présentations plus complètes existent [1, 8]. Une *signature* en MEL est un tuple

(K, Σ, S), où K est un ensemble de *kinds*[1], Σ est une famille indexée sur $K^* \times K$ de symboles de fonction : $\Sigma = \{\Sigma_{w,k}\}_{(w,k)\in K^*\times K}$, et $S = \{S_k\}_{k\in K}$ est une famille, indexée par K, de *sortes* - où S_k représente l'ensemble des sortes du kind k. Une signature (K, Σ, S) est souvent notée par sa seule composante Σ ; T_Σ est alors l'ensemble de termes clos sur Σ. Etant donné un ensemble de paires $X = \{x_1 : k_1, \ldots, x_n : k_n\}$ de variables, chacune munie de son kind, $T_\Sigma(X)$ représente alors l'ensemble des termes dont les variables libres sont dans l'ensemble X. De manière similaire, $T_{\Sigma,k}$ et $T_{\Sigma,k}(X)$ représentent, respectivement, l'ensemble des termes clos du kind k, et l'ensemble des termes clos du kind k aux variables libres dans X. Une formule atomique de MEL dans la signature (K, Σ, S) est soit une *équation* $t = t'$, avec $t, t' \in T_{\Sigma,k}(X)$ et $k \in K$, soit *un axiome d'appartenance* $t : s$, avec $t \in T_{\Sigma,k}(X)$, s une sorte dans S_k, et $k \in K$. Une *phrase* dans MEL est une clause de Horn, implicitement universellement quantifiée sur les variables apparaîssant dans les formules atomiques :

$$(\forall X)t = t' \textit{ if } C, \text{ or} \tag{1}$$

$$(\forall X)t : s \textit{ if } C \tag{2}$$

où la *condition* C est de la forme suivante, avec I, J deux ensembles finis d'indices :

$$\bigwedge_{i\in I}(u_i = v_i) \wedge \bigwedge_{j\in J}(w_j : s_j)$$

Les phrases de la forme (1) sont appelées *équations conditionnelles*, et les phrases de la forme (2) sont des *appartenances conditionnelles*. Une équation ou une appartenance sont inconditionnelles lorsqu'elles sont atomiques. Une théorie en MEL est une couple $\mathcal{M} = (\Sigma, E)$ composé d'une signature MEL Σ et d'un ensemble E de phrases sur Σ. La logique MEL possède un système de déduction, donné en Définition 1

1. Nous gardons le nom anglais faute de traduction satisfaisante. La raison d'être des kinds (par rapport aux sortes) est de contenir des termes syntaxiquement corrects mais sémantiquement incorrects, comme par exemple l'"entier" 1/0.

ci-dessous, qui est complet et cohérent [8], en ce sens qu'une formule atomique φ est déductible dans une théorie (Σ, E) - ce qu'on note par $(\Sigma, E) \vdash \varphi$, ou plus simplement $E \vdash \varphi$, si et seulement φ est sémantiquement valide : elle est vraie dans tous les modèles de la théorie (Σ, E)

Définition 1 *Soit une théorie $\mathcal{M} = (\Sigma, E)$ et une formule atomique e, de la forme $(\forall X)t = t'$ ou $(\forall X)t : s$. Alors, e est déductible dans $\mathcal{M}$, noté $\mathcal{M} \vdash e$ ou simplement $E \vdash e$ lorsque la signature résulte du contexte, si e peut être obtenue en appliquant les règles suivantes :*

1) Reflexivité : $\frac{t \in T_\Sigma(X)}{E \vdash (\forall X) t = t}$

2) Appartenance : $\frac{E \vdash (\forall X) t' = t \quad E \vdash (\forall X) t : s}{E \vdash (\forall X) t' : s}$

3) Symmétrie : $\frac{E \vdash (\forall X) t' = t}{E \vdash (\forall X) t = t'}$

4) Transitivité : $\frac{E \vdash (\forall X) t_1 = t_2 \quad E \vdash (\forall X) t_2 = t_3}{E \vdash (\forall X) t_1 = t_3}$

5) Congruence :

$$\frac{f \in \Sigma_{k_1,\ldots,k_n,k} \quad t_1,\ldots t_n \in T_{\Sigma_{k_i}}(X) \quad \textit{pour } i \in [1,n] \quad t'_i \in T_{\Sigma_{k_i}}(X) \quad E \vdash (\forall X) t_i = t'_i}{E \vdash (\forall X) f(t_1,\ldots,t_i,\ldots,t_n) = f(t_1,\ldots,t'_i,\ldots,t_n)}$$

6) Remplacement$_1$ $\frac{(\forall X) t = t' \textit{ if } C \ \in E \quad \sigma : X \mapsto T_\Sigma(Y) \quad E \vdash (\forall Y) C\sigma}{E \vdash (\forall Y) t\sigma = t'\sigma}$

7) Remplacement$_2$ $\frac{(\mu)\ (\forall X) t : s \textit{ if } C \ \in E \quad \sigma : X \mapsto T_\Sigma(Y) \quad E \vdash (\forall Y) C\sigma}{E \vdash (\forall Y) t\sigma : s}$

où $\sigma : X \mapsto T_\Sigma(Y)$ sont des substitutions préservant les kinds, et pour une condition $C : \bigwedge_{i \in I} (\forall X)(u_i = v_i) \wedge \bigwedge_{j \in J} (\forall X)(w_j : s_j)$, la notation $E \vdash (\forall Y) C\sigma$ est une abbréviation de la conjonction $\bigwedge_{i \in I} E \vdash (\forall Y)(u_i\sigma = v_i\sigma) \wedge \bigwedge_{j \in J} E \vdash (\forall Y)(w_j\sigma : s_j)$.

Il convient de remarquer que la déductibilité (et, de manière équivalente, la validité sémantique) ne sont pas toujours la notion de vérité attendue. En effet, beaucoup de propriétés interessantes ne sont pas vraies dans *tous* les modèles d'une théorie MEL, mais seulement dans certains, dont le *modèle initial* [8]. Dans ce modèle, les sortes sont interprétées comme les plus petits ensembles satisfaisant les axiomes (équations et appartenances), et l'égalité est la plus petite congruence satisfaisant ces mêmes axiomes. On écrit $E \vdash_{ind} (\forall X)\varphi$ pour dire que la phrase φ est vraie dans le modèle initial de (Σ, E).

Exemple 1 *Soit la théorie* MEL *NAT constituée de :*

– $K_{NAT} = \{Nat?\}$.
– $\Sigma_{NAT_{(\lambda,Nat?)}} = \{0\}$.
– $\Sigma_{NAT_{(Nat?,Nat?)}} = \{s\}$.
– $\Sigma_{NAT_{w,Nat?}} = \emptyset$, *pour* $w \notin \{\lambda, Nat?\}$.
– $S_{Nat?} = \{Nat\}$.
– $E_{NAT} = \{0 : Nat, (\forall N)\ s(N) : Nat\ if\ N : Nat\}$.

Le modèle initial de NAT est l'ensemble des nombres naturels $\mathbb{N}$. Mais ils existe de nombreux autres modèles, parmi lesquels, par exemple, il y a tous les ensembles d'entiers modulo n, pour tout $n \geq 2$.

Une théorie en logique de réécriture (ou simplement, une théorie de réécriture, ou de RL) est un tuple $\mathcal{R} = (K, \Sigma, S, E, R)$, où (K, Σ, S, E) est une théorie MEL, et R est en ensemble de règles de réécriture

$$(\rho) \quad (\forall X)\ l \rightarrow r \quad \textit{if} \quad C \tag{3}$$

où la condition C est de la forme $\bigwedge_{i\in I}(u_i = v_i) \wedge \bigwedge_{j\in J}(w_j : s_j)$ avec I et J des ensemble finis d'indices. Dans sa forme la plus générale [12], la logique RL permet également des *réécritures dans les conditions des règles* et des *arguments gelés*, deux constructions avancées de RL qu'on ne considère pas ici.

Définition 2 *Etant donnée une théorie $\mathcal{R} = (\Sigma, E, R)$ et deux termes $t, t' \in T_\Sigma(X)$, on dit que t' est accessible $\mathcal{R}$ depuis t par réécriture à la racine, noté par $\mathcal{R} \vdash (\forall X)t \longrightarrow t'$, si la formule $(\forall X)t \longrightarrow t'$ peut être obtenue en appliquant les rgles suivantes :*

1) Reflexivité : $\frac{t\in T_\Sigma(X)}{\mathcal{R}\vdash(\forall X)t\longrightarrow t}$

2) Transitivité : $\frac{\mathcal{R}\vdash(\forall X)t_1\longrightarrow t_2 \quad \mathcal{R}\vdash(\forall X)t_2\longrightarrow t_3}{\mathcal{R}\vdash(\forall X)t_1\longrightarrow t_3}$

3) Egalité : $\frac{E\vdash(\forall X)t=u \quad \mathcal{R}\vdash(\forall X)u\longrightarrow u' \quad E\vdash(\forall X)u'=t'}{\mathcal{R}\vdash(\forall X)t\longrightarrow t'}$

4) Remplacement : $\frac{(\rho)\ (\forall X)\ l\longrightarrow r\ if\ C\in R \quad \sigma:X\mapsto T_\Sigma(Y) \quad E\vdash(\forall Y)C\sigma}{\mathcal{R}\vdash(\forall Y)l\sigma\longrightarrow r\sigma}$

On notera que dans sa forme la plus générale [12], la logique comporte aussi une règle de Congruence. Eliminer cette règle revient à forcer la réécriture à la racine.

Nous introduisons maintenant la réécriture *close* à la racine. Intuitivement, lorsqu'on réécrit de cette manière un terme $(\forall X)t$ par une règle $(\forall Y)l \rightarrow r\ if\ C$, on imposera que le terme t soit d'abord transformé en un terme clos $\sigma(t)$ au moyen d'une substitution close σ de ses variables ; et seulement ensuite, le terme $\sigma(t)$ pourra être réécrit à la racine, en imposant que le filtrage avec le membre gauche l de la règle se fasse par une substitution close de *toutes* les variables présentes dans l, r, et C.

Définition 3 *Etant donnée une théorie $\mathcal{R} = (\Sigma, E, R)$ et deux termes $t \in T_\Sigma(X)$ et $t' \in T_\Sigma$, on dira que t' est accessible depuis t par réécriture close à la racine, noté $\mathcal{R} \vdash\!\downarrow\ t \overrightarrow{\ }t'$, s'il existe une substitution close $\sigma : X \mapsto T_\Sigma$ et une dérivation $\mathcal{R} \vdash \sigma(t) \overrightarrow{\ }t'$ à la racine (cf. Définition 2) dans laquelle toutes les applications de la règle de* Remplacement *utilisent des substitutions closes.*

Exemple 2 *Pour illustrer notre notion de réécriture close à la racine, considerons le terme $s(W)$ dans la théorie NAT et la règle $s(X) \rightarrow s(s(Y))$. Le terme $s(W)$ peut être réécrit par substitution close à la racine, en une étape, en $s(s(0))$: on transforme $s(W)$ en $s(0)$ par la substitution close $W \leftarrow 0$; et on réécrit $s(0)$ en $s(s(0))$ avec la règle, au moyen de la substitution close $X \leftarrow 0, Y \leftarrow 0$.*

Il convient de remarquer que la réécriture habituelle (non-close), du terme donné avec la règle donnée, contiendra forcément des séquences de termes non-clos, e.g., $s(W), s(s(W)), \ldots, s^n(W), \ldots$, alors que la réécriture close engendre seulement des termes clos. C'est la raison principale pour laquelle la réécriture close est préférable à la réécriture standard pour exprimer la dynamique de systèmes modélisés en RL *(et, spécialement, lorsque l'ensemble d'états initiaux est représenté par un terme non-clos et/ou les règles de réécriture comportent des variables supplémentaires en partie droite et/ou dans la condition) : la réécriture close engendre seulement des termes clos, qui dénotent des états, alors que les termes non-clos ne dénotent pas des états, mais des ensembles d'états. Les systèmes paramétriques, tel l'algorithme Bakery à n processus que nous présentons en section 4, utilisent naturellement des terme non-clos pour dénoter l'ensemble de leurs états initiaux, ainsi que des règles de réécriture avec variables suppplémentaires en partie droite et dans les conditions pour exprimer leur dynamique.*

L'exemple suivant est simple (terme initial clos, pas de variable supplementaire a droite des règles) ; il nous sert comme exemple de motivation, et illustre aussi l'adéquation de la réécriture *à la racine* pour exprimer la dynamique de (nombreux) systèmes exprimés en RL.

Exemple 3 *Le problème des lecteurs-écrivains est un problème classique en programmation concurrente. Il s'agit d'assurer l'accès en lecture/écriture à un fichier, de telle manière qu'il y ait exclusion mutuelle entre lecteurs et écrivains et entre écrivains. L'accès simultané de plusieurs lecteurs au fichier est permis. Une spécification de ce système en* RL*, appelée* $ReadersWriters$*, inclut la déclaration d'une sorte* $State$ *et du kind correspondant* $[State]$ *pour décrire les états du système ainsi que la déclaration d'un constructeur* $\langle ., . \rangle$ *qui prend deux nombres naturels (le nombre de lecteurs et le nombre d'écrivains, respectivement) et qui rend un* $State$*. L'évolution du système est décrite par les règles de réécriture suivantes :*

$$R_{ReadersWriters} = \begin{cases} \langle 0, 0 \rangle \rightarrow \langle 0, s(0) \rangle, \\ \langle R, s(W) \rangle \rightarrow \langle R, W \rangle, \\ \langle R, W \rangle \rightarrow \langle s(R), W \rangle \ if \ W = 0, \\ \langle s(R), W \rangle \rightarrow \langle R, W \rangle \end{cases}$$

La première règle spécifie que depuis un état $\langle 0, 0 \rangle$ *dans lequel in n'y a ni lecteur, ni écrivain, le système peut accepter un écrivain. La seconde règle permet à un écrivain de quitter le fichier ; la troisième permet d'accepter un lecteur supplémentaire, pourvu qu'il n'y ait pas d'écrivains ; et la quatrième règle permet à un lecteur de sortir du fichier. Cette spécification à un nombre d'états qui est infini (le nombre de lecteurs pouvant croître indéfiniment par la troisième règle). Nous allons illustrer notre approche (présentée dans la section suivante) pour vérifier l'exclusion mutuelle entre lecteurs et écrivains.*

Enfin, remarquons le fait que la réécriture dans $ReadersWriters$ *s'effectue toujours à la racine : les parties gauches des règles filtrent des termes du kind* $[State]$*, et les réécrivent en termes du même kind ; et ces termes n'ont pas de sous-termes stricts du kind* $[State]$ *; par conséquent, la réécriture ne peut avoir lieu en dessous de la racine. Ce type de réécriture est souvent suffisant en pratique pour spécifier des systèmes en* RL*, comme remarqué également par d'autres auteurs [13].*

L'invariance pour les systèmes spécifiés en RL

Nous continuons cette section par une définition de la notion d'invariance pour des systèmes spécifiés en RL. Intuitivement, un prédicat sur les états d'un système est un invariant si le prédicat est vrai dans tous les états du système qui sont accessibles à partir d'une certaine classe d'états initiaux. Afin de formaliser cette notion pour les systèmes spécifiés en RL nous devons préciser les aspects suivants :

1) quels sont les états et la dynamique du système ?

2) comment spécifier les prédicats sur les états ?

3) quand ces prédicats sont-ils *vrais* dans un état donné ?

Nous proposons que les états du système soient représentés par des termes clos d'un certain kind $[State]$, et que la dynamique du système soit définie par la réécriture close à la racine, partant d'un ensemble d'états initiaux représentés par un terme t_0, qui peut avoir des variables libres, et qui est également du kind $[State]$. Nous avons discuté ci-dessus l'adéquation de la réécriture close à la racine (Définition 3) pour exprimer la dynamique de systèmes exprimés en RL, tout particulièrement lorsque le terme initial comporte des variables libres et/ou les règles de réécriture comportent des variables supplémentaires en partie droite et/ou dans la condition, dont nous verrons l'utilité en section 4.

Concernant les prédicats d'état, ils seront formalisés par des phrases de la logique MEL de la forme $(\forall x : [State])(\forall Y)\varphi$, i.e., des phrases ayant une variable x du kind $[State]$, ainsi qu'optionnellement d'autres variables Y, telles que $x \notin Y$. La variable x du kind $[State]$ fait qu'on puisse parle de φ comme prédicate d'*état* ; les autres variables servent à quantifier universellement sur les composantes de l'état ; nous en verrons l'utilité également en section 4.

Enfin, un prédicat d'état $(\forall x : [State])(\forall Y)\varphi$ est vrai dans un état t lorsque le le prédicat $(\forall Y)\varphi(t/x)$, obtenu à partir de φ en substituant la variable x par le terme t, est vrai dans le modèle initial de la sous-théorie MEL E de la spécification RL $\mathcal{R}$ du système : $E \vdash_{ind} (\forall Y)\varphi(t/x)$.

En conclusion, lorsqu'un système est spécifié en RL, on formalise l'idée qu'un invariant est un prédicat d'état qui est vrai dans touts

les états du système qui sont accessibles à partir d'une certaine classe d'états initiaux, ainsi :

Définition 4 *Soient* $\mathcal{R} = (K, S, \Sigma, E, R)$ *une théorie de réécriture,* $[State] \in K$ *un kind,* $(\forall X)t_0 \in T_{\Sigma,k}(X)$ *un terme, et* $(\forall x : [State])(\forall Y)\varphi$ *une phrase* MEL. *Alors,* φ *est un invariant de* $\mathcal{R}$ *en partant de* t_0*, noté* $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$*, si pour tout* $t \in T_{\Sigma,[State]}$, $\mathcal{R} \vdash\!\downarrow t_0 \overrightarrow{\ } t$ *implique* $E \vdash_{ind} (\forall Y)\varphi(t/x)$.

Exemple 4 *Considérons la spécification* $ReadersWriters$ *donnée dans l'exemple 3. Supposons qu'un prédicat* $>$ *ait été défini dans la sous-théorie* MEL *NAT, et supponsons aussi que les deux accesseurs standard fst et snd aux éléments d'une paire sont définis. Nous décrivons l'exclusion mutuelle entre lecteurs et écrivains par le prédicat mutex, défini par les équations suivantes :*

$$(\forall\, x : [State])mutex(x) = true \textit{ if } fst(x) = 0$$
$$(\forall\, x : [State])mutex(x) = true \textit{ if } snd(x) = 0$$
$$(\forall\, x : [State])mutex(x) = false \textit{ if } fst(x) > 0 \wedge\ snd(x) > 0$$

On remarque que mutex est une formule atomique de MEL*, qui contient une seule variable* x *du kind* $[State]$*. L'invariance de mutex sur la théorie* $ReadersWriters$ *partant du terme initial (clos)* $\langle 0, 0 \rangle$ *est notée* $\langle$*ReadersWriters*$, \langle 0, 0 \rangle\rangle \vdash \Box mutex$ *et définie par le fait que, pour tout terme clos* t *du kind* $[State]$*, on a* [*ReadersWriters* $\vdash\!\downarrow \langle 0, 0 \rangle \overrightarrow{\ } t]$ *implique* $[NAT \vdash_{ind} mutex(t/x)]$.

Falsification automatique d'invariants

Avant d'aborder la preuve interactive d'invariants dans la section suivante, nous concluons cette section sur la falsification automatique d'invariants. La preuve et la falsification sont évidemment des techniques complémentaires, et disposer d'une technique automatique pour falsifier un invariant est utile avant de démarrer une preuve interactive qui, dans le cas d'une propriété fausse, n'aboutira pas.

Nous allons considerer des énoncés d'invariance $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ dans lesquelles la théorie $\mathcal{R}$ est *exécutable*[2] et tels que le terme intial t_0 est clos et le prédicat φ a une seule variable, qui est du kind $[State]$, comme dans l'exemple $\langle$*ReadersWriters*$, \langle 0, 0 \rangle \rangle \vdash \Box mutex$. Nous montrons que si ces contraintes sont satisfaites, il existe une procédure qui est correcte et complète pour la falsification d'invariants. Cette procédure est implémentée par la commande `search` de l'outil Maude. Les contraintes sur le terme initial et le prédicat d'état peuvent être éliminées, au prix de la perte de complétude de la procédure, qui reste cependant correcte. Nous présentons ici les aspects de la commande `search` qui sont utiles pour nos besoins ; une description complète est disponible dans [5].

Soit un prédicat d'état $(\forall x : [State])\varphi$ où $\varphi \triangleq \varphi_0 \; if \; \varphi_1 \ldots \varphi_n$, et φ_i sont des équations ou des appartenances atomiques, pour $i = 0, \ldots, n (n \in \mathbb{N})$. Nous utilisons des commandes `search` de la forme

$$\texttt{search } t_0 \texttt{ =>* } x \texttt{ such that } \varphi_1(x) \wedge \ldots \wedge \varphi_n(x) \wedge \neg\varphi_0(x) \quad (4)$$

où t_0 est un terme clos, et est x une variable, tous les deux du kind $[State]$. La commande ci-dessus effectue une recherche en largeur des termes accessibles par réécriture depuis t_0[3]. La commande *termine avec succès* si elle retourne au moins un terme accessible depuis t_0 et satisfaisant les conditions de la clause `such that`. Le fait que la commande `search` (4) soit une procédure correcte et complète pour la falsification d'invariants est précisé comme suit :

Observation 1 *Soit une propriété d'invariance de la forme* $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ *telle que : la théorie* RL $\mathcal{R}$ *est exécutable ; le terme* t_0 *est clos ; et le prédicat d'état* φ *a une seule variable libre, qui est du kind* $[State]$. *Alors, la command* `search` *(4) se termine avec succès si et seulement si* $\langle \mathcal{R}, t_0 \rangle \nvdash \Box\varphi$.

---

2. Cette condition exige, entre autres, qu'il n'y ait pas de variable supplémentaire en partie droite et dans les conditions ; le système peut quand même avoir un nombre infini d'états, cf. [5] pour une définition complète.
3. Nous supposons que la réécriture se fait à la racine ; sinon, il est toujours possible d'*encapsuluer* le terme et les règles pour forcer la réécriture à la racine [13].

Cette observation est vraie pour les raisons suivantes : les contraintes d'exécutabilité garantissent que, d'une part, les conditions de la clause `such that` sont décidables et que, d'autre part, la commande `search` trouve tout état *accessible* depuis t_0 en un temps fini [5] ; et par ailleurs, lorsque le terme initial est clos et la spécification est exécutable (donc elle ne contient pas de variables supplémentaires en parties droites et dans les conditions de règles) la réécriture à la racine est close par construction.

Exemple 5 *La spécification* RL $ReadersWriters$ *et le prédicat mutex de l'exemple 4 satisfont les contraintes de l'observation 1 ci-dessus. On peut tenter de falsifier l'énoncé* $\langle$*ReadersWriters*$, \langle 0, 0\rangle\rangle \vdash \Box mutex$ *ainsi :* `search` $\langle 0, 0\rangle$ `=>*` x `such that` $\neg mutex(x)$.

La commande `search` *ci dessus ne termine pas, car l'énoncé* $\langle$*ReadersWriters*$, \langle 0, 0\rangle\rangle \vdash \Box mutex$ *est vrai, comme nous le démontrons dans la section suivante. Un exemple de falsification réussie est celle du prédicat* same-number*, défini par* same-number$(x) = true$ *if* $fst(x) = snd(x)$*,* same-number$(x) = false$ *if* $fst(x) < snd(x)$*, et* same-number$(x) = false$ *if* $fst(x) > snd(x)$*. Dans ce cas, la commande* `search` *retourne* $x = \langle 0, 0\rangle$*, qui constitue un contre-exemple à l'invariance.*

Enfin, remarquons que, bien que la commande `search` exige un terme initial t_0 clos et une seule variable dans le prédicat d'état φ dont on cherche à falsifier l'invariance, on peut, en sacrifiant la complétude, instancier "à la main" les variables supplémentaires éventuelles dans t_0 and φ avant de lancer la commande `search`, sans compromettre la correction de la procédure de falsification.

3. Preuve assistée pour les propriétés d'invariance

Dans cette section nous présentons notre approche basée sur la preuve assistée pour prouver des propriétés d'invariance de spécifications en RL. D'abord, nous définissons une traduction automatique qui prend en entrée une théorie RL $\mathcal{R}$ et un terme t_0 et engendre une théorie MEL $\mathcal{M}(\mathcal{R}, t_0)$, qui enrichit la sous-théorie MEL de $\mathcal{R}$ avec une nouvelle sorte, appelée *Reachable*, et avec des axiomes d'appartenance à

cette sorte. Ensuite, nous démontrons qu'*être accessible dans $\mathcal{R}$ à partir de t_0* est équivalent à *être de la sorte Reachable dans la théorie* MEL $\mathcal{M}(\mathcal{R}, t_0)$. Puis, nous montrons un corrolaire de ce résultat, qui dit que pour tout prédicat d'état $(\forall x : [State])(\forall X)\varphi$, l'invariance $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ est équivalente au fait que l'implication $(\forall x : [State])(\forall X)(x : \textit{Reachable} \Rightarrow \varphi)$ est vraie dans le modèle initial de la théorie $\mathcal{M}(\mathcal{R}, t_0)$. Enfin, en remarquant que la preuve par induction est *correcte* (en anglais : *sound*) dans les modèle initiaux des théories MEL, nous établissons la correction de notre approche pour prouver des invariants par preuve inductive, grâce à l'équivalence établie ci-dessus.

Définition 5 *Soit une théorie* RL $\mathcal{R} = (K, \Sigma, S, E, R)$ *munie d'une sorte* $State \in S$ *au kind [State]* $\in K$*, et un terme* $t_0 \in T_{\Sigma,[State]}(X)$*. On note par* $\mathcal{M}(\mathcal{R}, t_0)$ *la théorie* MEL $(K_{\mathcal{M}(\mathcal{R},t_0)}, \Sigma_{\mathcal{M}(\mathcal{R},t_0)}, S_{\mathcal{M}(\mathcal{R},t_0)}, E_{\mathcal{M}(\mathcal{R},t_0)})$ *construite ainsi :*

– $K_{\mathcal{M}(\mathcal{R},t_0)} = K$

– $\Sigma_{\mathcal{M}(\mathcal{R},t_0)} = \Sigma$

– $S_{\mathcal{M}(\mathcal{R},t_0)} = S \cup S'_{[State]}$ *avec* $S'_{[State]} = S_{[State]} \cup \{\textit{Reachable}\}$ *et* $\textit{Reachable} \notin S$

– $E_{\mathcal{M}(\mathcal{R},t_0)} = E \cup \{(\forall X)t_0 : \textit{Reachable}\} \cup \{\mu(\rho) | (\rho) \in R\}$*, où* $\mu((\rho)(\forall X)\ l \rightarrow r\ \textit{if}\ C))$ *dénote l'axiome d'appartenance* $(\forall X)\ r : \textit{Reachable if}\ l : \textit{Reachable} \wedge C$.

Dans la définition précédente, on remarque l'axiome d'appartenance $\{(\forall X)t_0 : \textit{Reachable}\}$ ainsi que les axiomes d'appartence $\mu(\rho)$, un par régle de réécriture ρ dans $\mathcal{R}$. L'axiome pour t_0 dit que t_0 est accessible, et l'axiome $\mu(\rho)$ "inverse" la règle ρ, afin d'exprimer le fait que, dans la réécriture, la partie droite de ρ est accessible dès lors que sa partie gauche est accessible et que sa condition est satisfaite.

Exemple 6 *Considérons le problème des lecteurs-écrivains donné dans l'exemple 3. La théorie* MEL $\mathcal{M}(\textit{ReadersWriters}, \langle 0, 0 \rangle)$ *contient les éléments suivants.*

– $K_{\mathcal{M}(\textit{ReadersWriters}\langle 0,0 \rangle)} = K_{\textit{ReadersWriters}}$.

– $\Sigma_{\mathcal{M}(\textit{ReadersWriters},\langle 0,0 \rangle)} = \Sigma_{\textit{ReadersWriters}}$.

– $S_{\mathcal{M}(\mathit{ReadersWriters},\langle 0,0\rangle)} = S_{\mathit{ReadersWriters}} \cup S'_{[State]}$,
avec $S'_{[State]} = S_{\mathit{ReadersWriters}_{[State]}} \cup \{Reachable\}$
– $E_{\mathcal{M}(\mathit{ReadersWriters},\langle 0,0\rangle)} = E_{\mathit{ReadersWriters}} \cup E'$, *où*

$$E' = \begin{cases} \langle 0,0\rangle : Reachable\ , \\ \langle 0, s(0)\rangle : Reachable \textit{ if } \langle 0,0\rangle : Reachable\ , \\ \langle R, W\rangle : Reachable \textit{ if } \langle R, s(W)\rangle : Reachable\ , \\ \langle s(R), W\rangle : Reachable \textit{ if } \langle R, W\rangle : Reachable \wedge\ W = 0, \\ \langle R, W\rangle : Reachable \textit{ if } \langle s(R), W\rangle : Reachable\ . \end{cases}$$

Théorème 1 *Soit une théorie* RL $\mathcal{R} = (K, \Sigma, S, E, R)$ *munie d'une sorte* $State \in S$ *au kind [State]* $\in K$*, et un terme* $t \in T_{\Sigma,[State]}(X)$*. Alors, pour tout terme clos* $t' \in T_{\Sigma,[State]}$*, on a l'équivalence* $\mathcal{R} \vdash\downarrow t \longrightarrow t'$ *si et seulement si* $\mathcal{M}(\mathcal{R}, t) \vdash t' : Reachable$.

La preuve est donnée en Annexe. L'idée est de mettre en relation les pas de réécriture close à la racine avec une règle donnée ρ dans la théorie RL $\mathcal{R}$, et les pas de déduction avec l'axiome d'appartenance $\mu(\rho)$ dans la théorie MEL $\mathcal{M}(\mathcal{R}, t)$. On utilise également le fait que la sorte *Reachable* est "neuve" donc elle ne peut pas influencer la déduction dans la sous-théorie MEL (Σ, E) de $\mathcal{R}$.

Observation 2 *Il est intéressant de remarquer l'importance de la réécriture* close *dans le théorème 1. En effet, on a le contre-exemple suivant pour la réécriture non close : soit la théorie de réécriture constituée du kind* Foo*, de la constante* a *du kind* Foo*, de la fonction* $f : Foo \mapsto Foo$*, et d'aucune sorte, équation, appartenance, et règle de réécriture. Soit* $t_0 = f(x)$ *le terme initial. Alors, d'après les règles de la définition 2, seul* $f(x)$ *est atteignable. Et pourtant, la théorie obtenue par la transformation* $\mathcal{M}()$ *permet de déduire* $f(a) : Reachable$*, en appliquant* Remplacement$_2$ *(définition 1) avec l'appartenance* $f(x) : Reachable$ *et la substitution* $\sigma : x \leftarrow a$*. La réécriture non-close ne permettrait donc pas le théorème 1. En revanche, pour la réécriture close, le "contre-exemple" ci-dessus n'en est plus un : on a bien que* $f(a)$ *est atteignable en utilisant la substitution* $\sigma : x \leftarrow a$ *dans la définition 3.*

Pour le corrolaire suivant, rappelons que $\vdash_{ind}$ dénote la vérité dans un modèle initial.

Corollaire 1 *Soit une théorie* RL $\mathcal{R} = (K, \Sigma, S, E, R)$ *munie d'une sorte* $State \in S$ *au kind [State]* $\in K$*, et soit un predicat d'état* $(\forall x : [State], \forall X)\varphi$*. Alors, on a l'équivalence* $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ *si et seulement si* $\mathcal{M}(R, t_0) \vdash_{ind} (\forall x : [State])(\forall X)(x :$ *Reachable* $\Rightarrow \varphi)$.

La preuve est donnée en Annexe. Elle utilise le théorème 1, ainsi que quelques résultats standard sur les modèles initiaux et, à nouveau, le fait que la sorte *Reachable* est "neuve" dans $\mathcal{M}(\mathcal{R}, t)$ et par conséquent n'influence pas la vérité dans le modèle initial de (Σ, E).

Exemple 7 *Afin de prouver* $\langle$*ReadersWriters*$, \langle 0, 0\rangle\rangle \vdash \Box mutex$ *nous allons prouver* $\mathcal{M}$*(ReadersWriters*$, \langle 0, 0\rangle) \vdash_{ind} (\forall\, x)(x :$ *Reachable* $\Longrightarrow mutex(x) = true)$*. Nous utilisons pour cela l'assistant de preuve* ITP*. L'induction sur la sorte Reachable engendre cinq sous-buts. Le premier d'entre eux représente la base de l'induction, et correspond à l'appartenance au terme intial à la sorte Reachable. Les quatres autres sous-buts représentent les pas d'induction, qui correspondent aux quatre règles de réécriture définissant la dynamique du système. Ici, tous les sous-buts sont prouvés automatiquement avec la commande* `auto` *de l'outil* ITP*, qui invoque la réécriture automatique et les procédures de décision pour l'arithmétique linéaire. Bien entendu, les preuves ne sont pas toujours aussi automatiques. Dans la section suivante nous donnons un exemple dans lequel l'invariant principal a besoin d'*invariants auxiliaires *afin de passer les étapes d'induction de la preuve avec* ITP.

4. Vérification de l'algorithme Bakery à n processus

L'algorithme Bakery à n processus est un système paramétrique, qui consiste en un nombre $n \geq 2$ de processus identiques. Chacun des processus peut être dans l'une des localités *Sleep*, *Try*, ou *Critical*, et chacun possède un ticket que lui seul peut écrire, et que tous les processus peuvent lire (cf. Fig. 2). Dans l'état initial, tous les processus sont dans *Sleep* et tous les tickets t_i valent 0.

En allant de *Sleep* à *Try*, le i-ème processus affecte son ticket à la valeur maximum des tickets plus un. La condition pour entrer en section critique - dans notre modélisation, dans la localité *Critical* - pour

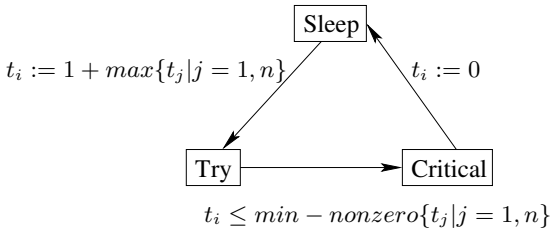

Figure 2 – Le i-ème processus dans l'algorithme Bakery à n processus

le i-éme processus, est que son ticket soit inférieur un égal au *minimum non-zéro* du multi-ensemble de tickets des processus. Le minimum non-zéro d'un multi-ensemble non vide de nombre naturels est défini par by *minNonZero(S)* $= 0$ si S contient seulement (un certain nombre de fois) l'élément 0 ; et *minNonZero*(S) = *min*$(S \setminus \{0\})$ sinon, où *min* est le minimum habituel d'un multi-ensemble non vide de nombres naturels, et $S \setminus \{0\}$ est le multi-ensemble obtenu en enlevant à S toutes les instances de 0. Enfin, lorsqu'un processus retourne dans la localité *Sleep* il remet son ticket à zéro.

La propriété d'exclusion mutuelle veut dire qu'un seul processus peut être à la fois en section critique. Pour la prouver, d'abord informellement, puis formellement avec notre approche, nous avons besoin de trois lemmes :

– les tickets des processus qui se trouvent en *Try* ou *Critical* sont *strictement positifs* ;

– les tickets des processus qui se trouvent en *Try* ou *Critical* sont *distincts deux à deux* ;

– les tickets des processus qui se trouvent en *Critical* valent le minimum non-zéro des tickets.

Le premier lemme est vrai car, lorsque les processus entrent dans *Try*, ils affectent à leur ticket une valeur strictement positive - le maximum des tickets plus un - et le fait d'entrer dans *Critical* ne change pas la valeur des tickets. Le second lemme est vrai car, en entrant dans *Try*, chaque processus affecte à son ticket une valeur *nouvelle* - le maximum des tickets plus un, et le fait d'entrer dans *Critical* ne change pas la va-

leur des tickets. Le troisième lemme est vrai car (a) la condition pour entrer dans *Critical* est d'avoir un ticket plus petit ou égal au minimum non-zéro des tickets, (b) le minimum non-zéro d'un multi-ensemble non vide de tickets non-nuls (tel que garanti par le premier lemme) est égal au minimum habituel du multi-ensemble, et (c) il n'y a pas de ticket plus petit que le minimum. Enfin, la propriété d'exclusion mutuelle est vraie car, en plus, le minimum d'un multi-ensemble non vide de valeurs distinctes deux à deux (tel que garanti par le deuxième lemme), est unique ; par conséquent, un seul processus, détenteur de cet unique ticket, peut être en section critique à un moment donné.

Spécification de l'algorithme en RL.

Nous commençons par la définition de la sorte des *états locaux*, qui sont des paires $\langle L, N\rangle$ formées d'une localité L et d'un nombre naturel N (le ticket), ainsi que des accesseurs $().loc$ et $().tic$ aux états locaux, comme suit

$$(\forall\, L, N)(\langle L, N\rangle : \mathit{LocalState}\ \textit{if}\ L : \mathit{Loc} \wedge N : \mathit{Nat})$$
$$(\forall\, L, N)\langle L, N\rangle.loc = L$$
$$(\forall\, L, N)\langle L, N\rangle.tic = N$$

Ensuite, nous définissons la sorte des états, qui sont soit des états locaux, soit une concaténation d'un état local et d'un état, séparés par ";" :

$$(\forall\, LS)(LS : \mathit{State}\ \textit{if}\ LS : \mathit{LocalState})$$
$$(\forall\, LS, ST)((LS; ST) : \mathit{State}\ \textit{if}\ LS : \mathit{LocalState} \wedge ST : \mathit{State})$$

Puis, la taille *dim()* d'un état est définie par le nombre d'états locaux qui y participent, et les accesseurs et modificateurs pour les états sont définis ainsi : $ST[i]$ rend le i-ème état local de l'état ST, et $ST\ \textit{with}[i] := LS$, rend une copie de l'état ST dans lequel le i-ème état local est remplacé par LS. Nous définissons aussi le maximum $max(ST)$ et le minimum non-zéro *minNonZero*(ST) des tickets.

Enfin, les transitions de l'algorithme sont exprimées par trois règles de réécriture, qui sont paramétrées par l'indice i du processus qui

change d'état. La règle (5) exprime la transition du processus i de $Sleep$ à Try, la règle (6), celle de Try à $Critical$, et la règle (6), celle de $Critical$ à $Sleep$.

$$(\forall\, i, ST)ST \rightarrow (ST\, with[i] := \langle Try, 1 + max(ST)\rangle) \quad \textit{if}\ i : Nat\ \wedge\ i < dim(ST)\ \wedge\ ST[i].loc = Sleep \tag{5}$$

$$(\forall\, i, ST)ST \rightarrow (ST\, with[i] := \langle Critical, ST[i].tic\rangle) \quad \textit{if}\ i : Nat\ \wedge\ i < dim(ST)\ \wedge\ ST[i].loc = Try \tag{6}$$

$$(\forall\, i, ST)ST \rightarrow (ST\, with[i] := \langle Sleep, 0\rangle) \quad \textit{if}\ i : Nat\ \wedge\ i < dim(ST)\ \wedge\ ST[i].loc = Critical \tag{7}$$

On remarque la présence de la variable supplémentaire i en partie droite de la règle et dans la condition. Sans ces variables il ne serait pas aisé de spécifier les systèmes paramétriques comme celui que nous étudions ici.

En choisissant un nombre fixe de processus (e.g., 2) et en dupliquant les règles afin de remplacer la variable i par les valeurs constantes que cette variable peut prendre, on peut exécuter le protocole, chercher des contre-exemples pour la propriété d'exclusion mutuelle (8) donnée ci dessous, etc.

Transformation de l'algorithme en MEL *et vérification.*

Afin de procéder à la vérification au moyen de notre approche, nous transformons d'abord automatiquement l'algorithme en une théorie MEL comme indiqué dans la section précédente. Les appartenances conditionnelles associés aux règles (5),(6), et (7) sont les suivantes :

$$(\forall\, i, ST)(ST\, with[i] := \langle Try, 1 + max(ST)\rangle) : Reachable \quad \textit{if}\ ST : Reachable\ \wedge\ i : Nat \wedge i < dim(ST)\ \wedge\ ST[i].loc = Sleep$$

$$(\forall\, i, ST)(ST\, with[i] := \langle Critical, ST[i].tic\rangle) : Reachable \quad \textit{if}\ ST : Reachable\ \wedge\ i : Nat \wedge i < dim(ST)\ \wedge\ ST[i].loc = Try$$

$$(\forall\, i, ST)(ST\, with[i] := \langle Sleep, 0\rangle) : Reachable\ \textit{if} \quad ST : Reachable\ \wedge\ i : Nat \wedge i < dim(ST)\ \wedge\ ST[i].loc = Critical$$

Ensuite, l'ensemble des états initiaux dans lequel se trouve l'algorithme est un nombre infini d'états locaux, ayant la localité $Sleep$ et la valeur 0 pour le ticket, et tous ces états sont accessibles :

$$
\begin{array}{l}
\mathit{Init}(0) = \langle \mathit{Sleep}, 0 \rangle \\
(\forall n)\mathit{Init}(s(n)) = \langle \mathit{Sleep}, 0 \rangle; \mathit{Init}(n) \\
(\forall n)\mathit{Init}(n) : \mathit{Reachable}\ \textit{if}\ n : \mathit{Nat}
\end{array}
$$

Puis, nous formalisons la propriété d'exclusion mutuelle comme le théorème suivant :

$$
\begin{array}{l}
(\forall\, ST, i, j)\, ST : \mathit{Reachable} \wedge i : \mathit{Nat} \wedge i < \mathit{dim}(ST) \wedge j : \mathit{Nat} \\
\wedge j < \mathit{dim}(ST) \wedge\ ST[i] = \mathit{Critical} \wedge ST[j] = \mathit{Critical} \\
\quad \Longrightarrow i = j
\end{array}
$$

Le théorème dit que dans tous les états accessibles, si deux états locaux en positions i et j sont tous les deux dans *Critical* alors $i = j$. On remarque la présence et l'utilité des variables universellement quantifiées i et j, qui apparaîssent dans la propriété en plus de la variable ST dénotant l'état.

Enfin, nous prouvons formellement la propriété d'exclusion mutuelle avec l'assistant de preuve ITP, au moyen de trois lemmes, de la même manière que la preuve informelle donnée en début de section. Le premier lemme dit que dans toutes les localités sauf *Sleep*, les tickets sont strictement positifs :

$$
\begin{array}{l}
(\forall\, ST, i)\, ST : \mathit{Reachable} \wedge i\ : \mathit{Nat} \wedge i\ <\ \mathit{dim}(ST) \\
\wedge \neg\ \mathit{sleep}(ST[i].\mathit{loc}) \\
\Longrightarrow ST[i].\mathit{tic} > 0
\end{array}
$$

Le deuxième lemme dit que dans toutes les localités sauf *Sleep* les tickets sont distincts :

$$
\begin{array}{l}
(\forall\, ST, i, j)\, ST : \mathit{Reachable} \wedge i : \mathit{Nat} \wedge i < \mathit{dim}(ST) \wedge j : \mathit{Nat} \\
\wedge j < \mathit{dim}(ST) \wedge\ i < j \wedge \neg \mathit{sleep}(ST[i].\mathit{loc}) \wedge \neg \mathit{sleep}(ST[j].\mathit{loc}) \\
\quad \Longrightarrow ST[i].\mathit{tic} \neq ST[j].\mathit{tic}
\end{array}
$$

Le troisième lemme que dans la localité *Critical*, les tickets sont inférieur ou égaux au minimum non-zéro des tickets :

$$(\forall\, ST, i)\; ST : \mathit{Reachable} \;\wedge\; i : \mathit{Nat} \wedge i < \mathit{dim}(ST)$$
$$\wedge ST[i].\mathit{loc} = \mathit{Critical}$$
$$\Longrightarrow ST[i].\mathit{tic} \leq \mathit{minNonZero}(ST)$$

Les preuves avec ITP du théorème et des trois lemmes suivent toutes le même canevas. Elles commencent par une induction sur la sorte *Reachable*, qui engendre quatre sous-buts : un pour les états initiaux, et trois pour les trois transitions du système. Le pas de base (pour les états initiaux) est résolu par la commande `auto`, qui invoque la réécriture et les procédures de décision du prouveur. Les pas inductifs sont des combinaisons de `auto`, d'analyses par cas, et de preuves d'invariants auxiliaires.

Nous avons également eu à prouver plusieurs lemmes sur *tous* les états (pas seulement sur les états accessibles, donc, des lemmes qui techniquement ne sont pas des invariants). Ces lemmes formalisent certaines relations entre les fonctions que nous avons définies, par exemple, le fait que le minimum non-zéro d'un multi-ensemble non vide de valeurs non nulles est égal au minimum du multi-ensemble, et inférieur ou égal au maximum du multi-ensemble. Les surces ITP pour cet exemple sont disponibles à l'URL `http ://www.irisa.fr/vertecs/Equipe/Rusu/itp/mutex-n`.

5. Conclusion, travaux connexes, et perspectives

L'exploration de l'ensemble des états accessibles et le *model checking* pour les systèmes finis, les abstractions pour réduire les systèmes infinis vers des systèmes finis, et la preuve interactive pour les systèmes infinis sont toutes des techniques de vérification bien connues. Pour la logique de réécriture, toutes ces approches sauf la dernière existent dans l'environnement Maude. Notre contribution a l'ambition de combler ce manque. L'approche que nous proposons est basée sur une traduction automatique d'un sous-ensemble significatif de la logique de réécriture RL (sans réécritures dans les conditions des règles, sans arguments gelés, et dont la réécriture se fait de manière close à la racine), et de propriétés d'invariance exprimées en logique équationnelle avec appartenance MEL, vers des propriétés inductives exprimées dans cette

dernière logique. Grâce à cette traduction, dont nous prouvons la correction, nous pouvons utiliser des assistants de preuve pour la logique équationnelle, tel que l'outil ITP, pour prouver des invariants sur des systèmes exprimés en logique de réécriture.

L'approche est illustrée sur l'algorithme Bakery à n processus. Les résultats sont encourageants, et ce, malgré le fait que l'assistant de preuve ITP est encore à l'état de prototype. Une trait distinctif de notre approche est qu'elle s'intègre naturellement avec les autres outils disponibles pour Maude, comme l'exploration énumérative, le *model checking*, et les abstractions équationnelles (figure 1).

L'approche proposée présente bien entendu des limites. Comme toutes les approches basées sur la preuve assistée, elle exige un utilisateur dédié et expert. D'après notre expérience, l'expertise s'obtient de manière incrémentale durant la vérification d'un exemple donné, et l'utilisateur bénéficie du retour de l'outil ITP lorsqu'il échoue à prouver un invariant donné pour cause d'information insuffiante : cette information se trouve dans le sous-but laissé non prouvé et dans le contexte dans lequel la preuve échoue. En examinant ces retours de l'outil, il est relativement aisé de poser un lemme qui, une fois prouvé, permettrait de clore la preuve du sous-but qui a engendré le problème. La difficulté principale de type d'approche réside dans le nombre de lemmes auxiliaires ; le pendant de l'explosion combinatoire en *model checking* est ici l'explosion du nombre de lemmes auxiliaires.

Travaux connexes.

Nous présentons quelques travaux connexes dans la multitude des travaux existants. L'équipe CAFEOBJ du Japan Advanced Institute of Science and Technology (JAIST) propose une approche pour prouver des invariants de systèmes dynamiques exprimés comme des *Observational Transition Systems* (OTS), les invariants étant des prédicats d'état sur les OTS. Ensuite, l'OTS et prédicat peuvent être représentés dans plusieurs formalismes (cf. Figure 3) : CAFEOBJ et Coq, pour la preuve [14, 15] ; Maude, pour la falsification d'invariants [16] ; et SMV, pour le *model checking* [17].

Le plus proche, dans les travaux de l'équipe CAFEOBJ, de ce que nous proposons est la preuve d'invariants en CAFEOBJ et leur falsifi-

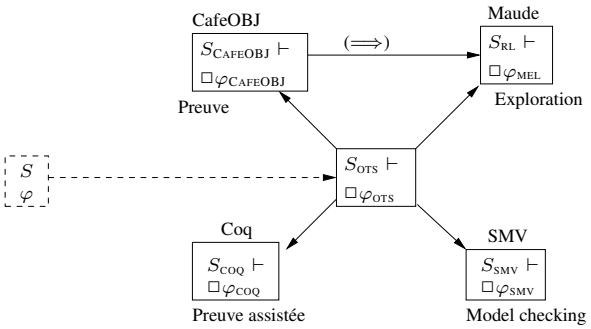

Figure 3 – *Observational Transition Systems* : représentation en CafeOBJ et en d'autres outils.

cation en Maude. Pour ce qui est de la preuve, leur approche consiste à coder l'invariance comme des prédicats sur des paires d'états, exprimant le fait que les différentes transitions du système *préservent* le prédicat d'état dont on souhaite prouver l'invariance. Ensuite, la réduction équationnelle de CAFEOBJ tenter d'établir automatiquement cette préservation.

Des opérations typiquement effectuées par un assistant de preuve, telles que le remplacement des variables universellement quantifiées par des constantes "neuves" et la décomposition en cas, sont laissées ici à l'utilisateur. Ceci constitue une source d'erreurs dans des preuves de taille importante. Pour ce qui est de la falsification d'invariants, comme cette possibilité n'existe pas en CAFEOBJ, les auteurs ont choisi de traduire les OTS en Maude et d'utiliser l'exploration énumérative. Globalement, la différence principale entre ce que nous proposons et la proposition de l'équipe CAFEOBJ réside dans le fait que nous restons dans un seul environnement et formalisme (celui de Maude, et de la logique de réecriture/équationnelle avec appartenance), ce qui nous permet de nous appuyer sur une sémantique commune dans les diverses activités de vérification (Figure 1). Ceci contraste fortement avec l'approche de l'équipe CAFEOBJ, qui utilise plusieurs outils, aux sémantiques *a priori* différentes (Figure 3). Leur approche soulève un problème de cohérence globale : pour un problème de vérification donné, obtient-on des réponses cohérentes avec l'ensemble des outils connectés ? Un dé-

but de réponse est donné dans [18] pour la correction de la traduction entre CAFEOBJ et Maude.

Dans l'article [12] les auteurs présentent un codage exhaustif de RL dans MEL. Leur codage traite toute la logique RL, y compris les aspects que nous avons choisi de laisser de coté comme les réécritures dans les conditions des règles, les arguments gelés, et la réécriture générale (pas forcément à la racine et pas forcément close). Leur codage est assez complexe[4], et leur but est d'étudier la sémantique et la théorie de la preuve de RL en terme des notions correspondantes de MEL. Notre objectif est différent : obtenir un codage d'un sous-ensemble de la logique RL, qui soit suffisant en pratique pour exprimer la dynamique de nombreux systèmes *et de leurs invariants exprimés en* MEL, et qui soit assez simple pour être utilisable dans une démarche de preuve interactive assistée. Un codage *simple* est essentiel pour que l'utilisateur retrouve, à travers le codage, les propriétés qu'il tente de prouver.

La vérification de systèmes paramétriques est en général indécidable. Par conséquent, leur vérification automatique est limitée à des sous-classes, ou est approchée. Parmi les nombreuses approches automatiques, il y a le *regular model checking* [19], où l'état du système est modélisé par un langage régulier, et la relation de transition, par un transducteur ; les *abstractions de comptage* [20], dans lesquelles seul le nombre de processus dont le contrôle se trouve dans une localité donnée est mémorisé, la vérification étant approchée par cette abstraction ; et la *génération d'invariants* observés sur des instances particulière du système, qu'on généralise et prouve sur le système paramétrique.

Les perspectives ouvertes par cette recherche sont d'utiliser effectivement la combinaison d'approches formelles existantes dans Maude, y compris la nôtre, pour vérifier une application réaliste, et montrer que l'utilisateur gagne à utiliser les différentes méthodes ensemble plutôt qu'une seule.

4. Pour l'accessibilité, leur codage comprend, *pour chaque kind* k de la théorie RL à coder, un nouveau kind k', quatre nouvelles sortes, et quatre nouvelles opérations définies par sept nouvelles équations dans la théorie MEL résultante.

Références

[1] N. Martí-Oliet and J. Meseguer. Rewriting logic : roadmap and bibliography. *TCS*, 285(2) :121-154, 2002.

[2] J. Meseguer, P. C. Ölveczky, M. O. Stehr, and C. L. Talcott. Maude as a wide-spectrum framework for formal modeling and analysis of active networks. In *DANCE*, pages 494-510. IEEE Comp. Soc., 2002.

[3] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and M. K. Sönmez. Pathway logic : Symbolic analysis of biological signaling. In *Pacific Symposium on Biocomputing*, pages 400-412, 2002.

[4] J. Meseguer and G. Rosu. The rewriting logic semantics project. *TCS*, 373(3) :213-237, 2007.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude, A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[6] P. Borovanský, C. Kirchner, H. Kirchner, P. E. Moreau, and C. Ringeissen. An overview of ELAN. *Electr. Notes Theor. Comput. Sci.*, 15, 1998.

[7] R. Diaconescu and K. Futatsugi. Logical foundations of CafeOBJ. *TCS*, 285(2) :289-318, 2002.

[8] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18-61. Springer, 1997.

[9] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. *Electr. Notes Theor. Comput. Sci.*, 71, 2002.

[10] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. In F. Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 2-16. Springer, 2003.

[11] M. Clavel, M. Palomino, and A. Riesco. Introducing the ITP tool : a tutorial. *J. Universal Computer Science*, 12(11) :1618-1650, 2006.

[12] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *TCS*, 360(1-3) :386-414, 2006.

[13] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. In *WRLA*, volume 117 of *Electronic Notes in Theoretical Computer Science*, 2004.

[14] K. Futatsugi. Verifying specifications with proof scores in CafeOBJ. In *ASE*, pages 3-10. IEEE Comp. Soc., 2006.

[15] Kazuhiro Ogata and Kokichi Futatsugi. State machines as inductive types. *IEICE Transactions*, 90-A(12) :2985-2988, 2007.

[16] W. Kong, T. Seino, K. Futatsugi, and K. Ogata. A lightweight integration of theorem proving and model checking for system verification. In *APSEC*, pages 59-66. IEEE Comp. Soc., 2005.

[17] K. Ogata, M. Nakano, M. Nakamura, and K. Futatsugi. Chocolat/SMV : a translator from CafeOBJ to SMV. In *PDCAT*, pages 416-420. IEEE Comp. Soc., 2005.

[18] M. Nakamura, W. Kong, K. Ogata, and K. Futatsugi. A specification translation from behavioral specifications to rewrite specifications. *IEICE Transactions*, 91-D(5) :1492-1503, 2008.

[19] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256(1-2) :93-112, 2001.

[20] A. Pnueli, J. Xu, and L. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 107-122. Springer, 2002.

Annexe

Théorème 1 *Soit une théorie* RL $\mathcal{R} = (K, \Sigma, S, E, R)$ *munie d'une sorte* $State \in S$ *au kind [State]* $\in K$*, et un terme* $t \in T_{\Sigma,[State]}(X)$*. Alors, pour tout terme clos* $t' \in T_{\Sigma,[State]}$*, on a l'équivalence* $\mathcal{R} \vdash\!\downarrow t \longrightarrow t'$ *si et seulement si* $\mathcal{M}(\mathcal{R}, t) \vdash t'$: *Reachable*.

Preuve. ($\Rightarrow$) Par induction sur la longueur de la réécriture close $\mathcal{R} \vdash \sigma(t) \longrightarrow t'$ correspondant à $\mathcal{R} \vdash\!\downarrow t \longrightarrow t'$ (cf. définition 3). Ici, la longueur

est le nombre d'applications de la règle de *Remplacement* dans la définition 2. Si la longueur est 0 alors $\sigma(t)$ et t' sont *égaux modulo* E (c'est à dire, $E \vdash \sigma(t) = t'$). En utilisant $(\forall X)t$: *Reachable* dans $\mathcal{M}(\mathcal{R}, t)$ on obtient, avec les règles de déduction de la logique MEL (définition 1), $\mathcal{M}(\mathcal{R}, t) \vdash \sigma(t)$: *Reachable* puis $\mathcal{M}(\mathcal{R}, t) \vdash t'$: *Reachable*.

Nous supposons l'implication vraie pour les dérivations de longueur n et la prouvons pour la longueur $n + 1$. *Modulo* l'égalité par les equations E, une dérivation de longueur $n + 1$ peut être décomposée en $\mathcal{R} \vdash \sigma(t) \longrightarrow t'' \longrightarrow t'$, pour un terme clos donné t'', tel que le dernier pas $\mathcal{R} \vdash t'' \longrightarrow t'$ soit une application du *Remplacement*, avec une règle $(\rho)\ (\forall X)l \rightarrow r$ *if* C et une substitution $\sigma' : X \mapsto T_\Sigma$.

1) alors, $t'' \equiv \sigma'(l)$, $t' \equiv \sigma'(r)$ et $E \vdash \sigma'(C) = true$, qui implique *a fortiori* $\mathcal{M}(\mathcal{R}, t) \vdash \sigma'(C) = true$. (On note par $\equiv$ l'égalité syntaxique entre deux termes).

2) par l'hypothèse d'induction, $\mathcal{M}(\mathcal{R}, t) \vdash t''$: *Reachable*, i.e., $\mathcal{M}(\mathcal{R}, t) \vdash \sigma(l)$: *Reachable*

3) donc, on peut utiliser *Replacement*$_2$ (définition 1) avec $(\mu(\rho))\ r$: *Reachable if* l : *Reachable* $\wedge\ C$ (définition 5) et σ', et en conclure $\mathcal{M}(\mathcal{R}, t) \vdash \sigma'(r)$: *Reachable*, i.e., $\mathcal{M}(\mathcal{R}, t) \vdash t'$: *Reachable*.

($\Leftarrow$) Par induction sur la longueur de la dérivation $\mathcal{M}(\mathcal{R}, t) \vdash t'$: *Reachable*. Ici, la longueur est le nombre d'applications de *Remplacement*$_2$ (définition 1) avec la sorte $s =$ *Reachable*. On remarque d'abord que $n \geq 1$, car, afin d'inférer l'appartenance à *Reachable*, on doit utiliser au moins une fois *Remplacement*$_2$ avec $s =$ *Reachable*. Le pas de base $n = 1$ implique que, nécessairement, *Remplacement*$_2$ ait été utilisée avec l'appartenance $(\forall X)t$: *Reachable* du terme initial. En effet, toutes les autres appartenances définissant *Reachable* sont récursives, e.g., elles nécessitent déjà une appartenance à *Reachable* pour être utilisées avec *Remplacement*$_2$. Par conséquent, il existe une substitution close σ telle que $t' \equiv \sigma(t)$, et ensuite on obtient trivialement $\mathcal{R} \vdash \sigma(t) \longrightarrow t'$ par *Réflexivité*.

On suppose l'implication *vraie* pour toutes les dérivations de longueur n et la prouvons pour la longueur $n + 1$. *Modulo* les équations E, une preuve de longueur $n + 1$ peut toujours être décomposée en une preuve $\mathcal{M}(R, t) \vdash t''$: *Reachable* de longueur n, pour un terme donné

$t'' \in T_\Sigma$, suivie d'une application de *Remplacement*$_2$, avec une appartenance $(\mu(\rho))\ r$: ***Reachable if*** l : ***Reachable*** $\wedge\ C$ obtenue à partir de la règle $(\rho)\ (\forall X) l \rightarrow r$ ***if*** C. Alors, il existe une substitution close σ' telle que $t' \equiv \sigma'(r)$, $t'' \equiv \sigma'(l)$, et $\mathcal{M}(R,t) \vdash \sigma'(C) = true$. Puisque la sorte ***Reachable*** est "neuve" dans $\mathcal{M}(R,t)$, elle n'apparaît pas dans la condition C, et par conséquent on a aussi $E \vdash \sigma(C) = true$, et, par ***Remplacement*** on obtient $\mathcal{R} \vdash \sigma'(l) \longrightarrow \sigma'(r)$, i.e., $\mathcal{R} \vdash t'' \longrightarrow t'$. Par hypothèse d'induction, $\mathcal{R} \vdash\!\downarrow t \longrightarrow t''$ i.e., il existe une substitution close $\sigma : X \mapsto T_\Sigma$ telle qu'on ait une dérivation $\mathcal{R} \vdash \sigma(t) \longrightarrow t''$ où toutes les applications de ***Remplacement*** se font avec des substitutions closes. Comme la dérivation $\mathcal{R} \vdash t'' \longrightarrow t'$ que nous venons d'obtenir utilise aussi une substitution close - σ' - on obtient par ***Transitivité*** $\mathcal{R} \vdash \sigma(t) \longrightarrow t'$, et par la Définition 3, on en conclut $\mathcal{R} \vdash\!\downarrow t \longrightarrow t'$. □

Corollaire 1 *Soit une théorie* RL $\mathcal{R} = (K, \Sigma, S, E, R)$ *munie d'une sorte* $State \in S$ *au kind [State]* $\in K$*, et soit un predicat d'état* $(\forall x : [State], \forall X)\varphi$. *Alors, on a l'équivalence* $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$ *si et seulement si* $\mathcal{M}(R,t_0) \vdash_{ind} (\forall x : [State])(\forall X)(x :$ ***Reachable*** $\Rightarrow \varphi)$.

Preuve. En partant de $\mathcal{M}(R,t_0) \vdash_{ind} (\forall x : [State])(x :$ ***Reachable*** $\Rightarrow (\forall X)\varphi)$ et en utilisant le fait qu'une formule universellement quantifiée est vraie dans le modèle initial d'une théorie si et seulement si toutes les instances closes de la formule sont vraies dans ce même modèle, on obtient de manière équivalente $\forall t \in T_{\Sigma,[State]}.\ \mathcal{M}(R,t_0) \vdash_{ind} (t :$ ***Reachable*** $\Rightarrow (\forall X)\varphi(t/x))$. Ensuite, en uilisant l'équivalence : $A \vdash (B \Rightarrow C)$ ssi ($A \vdash B$ implique $A \vdash C$), on obtient de manière équivalente

$$\forall t \in T_{\Sigma,[State]}. \qquad ([\mathcal{M}(R,t_0) \vdash t :\ \textit{Reachable}]\ \textit{implique}\ [\mathcal{M}(R,t_0) \vdash_{ind} (\forall X)\varphi(t/x)]) \tag{8}$$

Par le théorème 1, $\mathcal{M}(\mathcal{R},t_0) \vdash t :$ ***Reachable*** est équivalent à $\mathcal{R} \vdash\!\downarrow t_0 \longrightarrow t$; et, puisque φ ne fait pas référence à la "nouvelle" sorte ***Reachable***, $\mathcal{M}(\mathcal{R},t_0) \vdash_{ind} (\forall X)\varphi(t/x)$ si et seulement $E \vdash_{ind} (\forall X)\varphi(t/x)$. Et par conséquent, l'implication (8) s'écrit de manière équivalente $\forall t \in T_{\Sigma,[State]}.$ $\mathcal{R} \vdash\!\downarrow t_0 \longrightarrow t$ ***implique*** $E \vdash_{ind} (\forall X)\varphi(t/x)$, qui est la définition 4 de l'invariance $\langle \mathcal{R}, t_0 \rangle \vdash \Box\varphi$. □