Composition certifiée d'algorithmes autostabilisants silencieux †

Karine Altisen[1] et Pierre Corbineau[2]

[1] *Univ. Grenoble Alpes, CNRS, Grenoble INP ‡, Verimag, F-38000 Grenoble, France*
[2] *VERIMAG UMR 5104, Université Grenoble Alpes, France*

La composition est un outil fondamental pour élaborer des systèmes de plus en plus complexes. Dans le cadre des algorithmes distribués autostabilisants, la composition collatérale hiérarchique est un équivalent de la composition séquentielle. Nous enrichissons la bibliothèque de formalisation d'algorithmes autostabilisants PADEC de cet opérateur de composition et nous explicitons des conditions nécessaires à son utilisation. Nous démontrons la correction partielle de la composition collatérale hiérarchique et nous en prouvons la terminaison sous deux critères distincts : un critère fort pour des exécutions asynchrones et un critère plus réaliste utilisant des hypothèses d'équité faible.

Mots-clefs : Algorithmes distribués, autostabilisation, certification, preuve assistée par ordinateur, Coq

1 Introduction

La composition est un outil naturel pour simplifier la conception et la preuve d'algorithmes complexes. Notamment, la construction d'algorithmes séquentiels repose sur la composition séquentielle de sous-algorithmes : l'algorithme $\mathcal{A}_1;\mathcal{A}_2$ exécute l'algorithme $\mathcal{A}_1$ jusqu'à terminaison puis l'algorithme $\mathcal{A}_2$. En algorithmique distribuée, si l'effet recherché est le même — séquentialisation des algorithmes à l'échelle du système — sa réalisation est plus complexe. En effet, les codes des algorithmes sont répartis sur chacun des nœuds du système qui ne bénéficient que d'informations locales pour décider d'exécuter $\mathcal{A}_1$ ou $\mathcal{A}_2$.

Dans le cadre des algorithmes autostabilisants, qui offrent une solution légère pour la tolérance aux pannes [7], la composition est très utilisée ; par exemple, elle permettra de composer un algorithme de construction d'arbre couvrant avec un algorithme conçu pour des topologies en arbre [3]. Plusieurs opérateurs de composition d'algorithmes autostabilisants existent, tels que la composition équitable [8], collatérale [9], croisée [4], ou encore conditionnelle [5]. Nous nous intéressons ici à la *composition collatérale hiérarchique* [6] : c'est une variante de la composition collatérale qui déterminise l'exécution locale entre $\mathcal{A}_1$ et $\mathcal{A}_2$ (à chaque pas, un nœud n'exécutera $\mathcal{A}_2$ que s'il ne peut exécuter $\mathcal{A}_1$).

La bibliothèque PADEC [2, 1] — *Preuves d'Algorithmes Distribués En Coq* — fournit une fondation formelle rigoureuse des concepts liés à l'autostabilisation. Coq [10] est un assistant de preuve qui permet de développer semi-automatiquement des preuves certifiées (c'est-à-dire validées mécaniquement par l'outil). Nous l'utilisons dans PADEC pour modéliser des algorithmes à l'aide du modèle le plus utilisé en autostabilisation, le *modèle à états*, et prouver formellement qu'ils sont autostabilisants pour leur spécification. Dans cet article, nous formalisons l'opérateur de composition collatérale hiérarchique $\mathcal{A}_2 \circ \mathcal{A}_1$ avec une formalisation fonctionnelle adaptée à Coq et nous explicitons des jeux d'hypothèses permettant de vérifier la correction de l'opérateur. De même que la composition d'algorithmes permet de construire des algorithmes complexes à partir d'étapes plus simples, la preuve formelle de la correction de cette composition permet de prouver la correction de l'algorithme composé à partir des preuves formelles des sous-algorithmes. Bien sûr, il est alors nécessaire de faire des hypothèses supplémentaires pour s'assurer d'une composition correcte.

Dans le cadre d'algorithmes autostabilisants silencieux (c'est-à-dire qui terminent), nous fournissons la preuve de la correction partielle de $\mathcal{A}_2 \circ \mathcal{A}_1$, puis deux critères distincts pour la convergence de $\mathcal{A}_2 \circ \mathcal{A}_1$.

‡ Institute of Engineering Univ. Grenoble Alpes

† Ce travail a été financé par AGIR PADEC / ANR ESTATE

Ces critères visent à éviter une famine de l'algorithme $\mathcal{A}_1$ par l'exécution infinie et prématurée de $\mathcal{A}_2$. Le premier suppose la convergence de $\mathcal{A}_2$ quelle que soit la situation de $\mathcal{A}_1$, ce qui permet à $\mathcal{A}_2 \circ \mathcal{A}_1$ de converger sous les hypothèses d'asynchronie les plus larges (démon inéquitable) : c'est un critère fort qui s'applique peu en pratique car souvent l'étape $\mathcal{A}_1$ vise à construire une structure topologique donnée (arbre, anneau, ...) assurant la convergence de $\mathcal{A}_2$. Le second critère relâche les hypothèses sur $\mathcal{A}_2$: en supposant que $\mathcal{A}_1$ et $\mathcal{A}_2$ convergent sous hypothèse d'équité faible, on obtient que $\mathcal{A}_2 \circ \mathcal{A}_1$ converge, lui aussi sous hypothèse d'équité faible. L'équité faible impose qu'on ne retarde pas indéfiniment l'activation d'un nœud continûment activable, c'est-à-dire d'un nœud capable d'exécuter une transition à chaque pas d'exécution.

2 Autostabilisation dans la bibliothèque PADEC

Dans PADEC, nous utilisons le modèle à mémoire partagée avec atomicité composite, communément appelé *modèle à états*. Un système distribué est représenté par un réseau, composé de nœuds (ensemble $\mathcal{N}$) ayant des liens de communication vers d'autres nœuds. Ces liens sont étiquetés par des noms de canaux (ensemble $\mathcal{C}$) localement uniques. Le réseau est ainsi anonymisé : un nœud ne connaît pas les identités des nœuds voisins mais seulement les noms des canaux disponibles. Chaque nœud est équipé d'un algorithme local et d'un état — ensemble de variables valuées — noté $\mathcal{S}$; dans le modèle à état, le nœud peut aussi lire les états des nœuds qui ont des canaux vers lui (fonction ℓ de $\mathcal{C}$ vers $\mathcal{S}$).

L'algorithme local s'écrit sous la forme d'une règle de transition composée d'une condition de garde et d'un ensemble d'affectations. La condition de garde et les affectations peuvent faire référence à l'état des nœuds voisins mais seules les variables locales peuvent être affectées. Si la condition de garde d'un nœud est réalisée, on dit que le nœud est *activable*, il peut alors réaliser un *pas de calcul* atomique en exécutant les affectations. La bibliothèque PADEC permet de décrire le fonctionnement de l'algorithme $\mathcal{A}$ à l'aide d'une fonction $\texttt{run}_{\mathcal{A}}$, locale à chaque nœud, de type : $\mathcal{S} \to (\mathcal{C} \to \mathcal{S}) \to \mathcal{S}^{\perp}$ [§]. Sur un nœud p — pour lequel on note s l'état de p et $\ell(q)$, l'état de q pour tout voisin q de p — $\texttt{run}_{\mathcal{A}}(s,\ell)$ retourne la valeur $\perp$ si p est désactivé, ou un nouvel état s' si p est activable et peut transiter vers s'.

On appelle *configuration* l'ensemble des états des nœuds du réseau (fonctions de $\mathcal{N}$ vers $\mathcal{S}$). L'*exécution* d'un algorithme sur un réseau est une suite finie ou infinie de configurations. Entre deux configurations successives, au moins un nœud activable exécute un pas de calcul et change d'état. Dans PADEC, certaines variables d'un algorithme sont identifiées comme constantes et permettent d'exprimer des hypothèses sur le contexte d'exécution de l'algorithme (prédicat *Assume*). Par exemple, on peut faire l'hypothèse d'un réseau identifié (une variable identité par nœud et un prédicat exprimant l'unicité de ces identités). Un algorithme $\mathcal{A}$ est dit *autostabilisant pour la spécification S à partir d'une précondition Assume* s'il existe un ensemble L de configurations dites légitimes tel que toute exécution de $\mathcal{A}$ dont la configuration initiale satisfait *Assume* aboutit à une configuration de L après un nombre fini d'étapes, que L est stable par transition de $\mathcal{A}$ et que toute exécution initiée dans L satisfait la spécification S. Une configuration est dite *terminale* si aucun nœud n'y est activable. Nous nous intéressons ici à des algorithmes dits silencieux où toute exécution finit dans une configuration terminale. Dans ce cas, l'ensemble L est l'ensemble des configurations terminales et on doit simplement vérifier que toute configuration terminale vérifie la spécification S.

3 Composition collatérale hiérarchique

L'opérateur de composition collatérale hiérarchique compose deux algorithmes $\mathcal{A}_1$ et $\mathcal{A}_2$ opérant sur des ensembles d'états $\mathcal{S}_1$ et $\mathcal{S}_2$ pour obtenir un nouvel algorithme $\mathcal{A}_2 \circ \mathcal{A}_1$ opérant sur un ensemble d'états noté $\mathcal{S}_{1\circ 2}$. La correspondance entre ces états est établie par deux projections de $\mathcal{A}_2 \circ \mathcal{A}_1$ vers $\mathcal{A}_1$ ou $\mathcal{A}_2$ ($\mathsf{read}_1 : \mathcal{S}_{1\circ 2} \to \mathcal{S}_1$, $\mathsf{read}_2 : \mathcal{S}_{1\circ 2} \to \mathcal{S}_2$) et deux fonctions de mise à jour d'un état de $\mathcal{A}_2 \circ \mathcal{A}_1$ à partir d'un état de $\mathcal{A}_1$ ou $\mathcal{A}_2$ ($\mathsf{write}_1 : \mathcal{S}_1 \to \mathcal{S}_{1\circ 2} \to \mathcal{S}_{1\circ 2}$, $\mathsf{write}_2 : \mathcal{S}_2 \to \mathcal{S}_{1\circ 2} \to \mathcal{S}_{1\circ 2}$). On suppose que les variables constantes de $\mathcal{A}_2 \circ \mathcal{A}_1$ et leur hypothèse sont celles de $\mathcal{A}_1$ et que l'algorithme $\mathcal{A}_2$ ne peut modifier les variables calculées par l'algorithme $\mathcal{A}_1$. La fonction $\mathsf{run}_{\mathcal{A}_2\circ\mathcal{A}_1}$ de l'algorithme $\mathcal{A}_2 \circ \mathcal{A}_1$ est alors construite localement à chaque nœud p, à l'aide des fonctions $\mathsf{run}_{\mathcal{A}_1}$ et $\mathsf{run}_{\mathcal{A}_2}$ des algorithmes $\mathcal{A}_1$ et $\mathcal{A}_2$ de p :

$\mathsf{run}_{\mathcal{A}_2\circ\mathcal{A}_1}(s,\ell) :=$
$\mathsf{write}_1(s_1,s)$ `if` $\mathsf{run}_{\mathcal{A}_1}(\mathsf{read}_1(s),\ell_1) = s_1$; `else` $\mathsf{write}_2(s_2,s)$ `if` $\mathsf{run}_{\mathcal{A}_2}(\mathsf{read}_2(s),\ell_2) = s_2$; `else` $\perp$

§. L'ensemble $A^{\perp}$ est défini comme $A \cup \{\perp\}$

où $\ell_i(q) = \mathsf{read}_i(\ell(q))$ est la projection de $\ell(q)$ sur $\mathcal{A}_i$ pour tout voisin q de p, $i \in \{1,2\}$.

Le but de l'opérateur de composition est de garantir par construction la correction de l'algorithme composé à l'aide d'un théorème de composition. Un tel théorème s'énoncerait informellement ainsi : si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont autostabilisants et silencieux, alors $\mathcal{A}_2 \circ \mathcal{A}_1$ l'est aussi pour la conjonction des spécifications de $\mathcal{A}_1$ et $\mathcal{A}_2$. $\mathcal{A}_1$ sert dans la composition à valider les hypothèses permettant à $\mathcal{A}_2$ de fonctionner, c'est pourquoi nous supposons que toute configuration terminale de $\mathcal{A}_1$ vérifie la précondition de $\mathcal{A}_2$, $Assume_{\mathcal{A}_2}$. Les paragraphes suivants précisent des conditions suffisantes supplémentaires permettant de valider le théorème.

4 Correction partielle de la composition

Si, quand l'algorithme $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) est dans une configuration terminale, il satisfait sa spécification $S_{\mathcal{A}_1}$ (resp. $S_{\mathcal{A}_2}$ puisque $Assume_{\mathcal{A}_2}$ est vérifié), alors on montre facilement que l'algorithme composé $\mathcal{A}_2 \circ \mathcal{A}_1$, dans une configuration terminale, satisfait la conjonction des spécifications $S_{\mathcal{A}_1}$ et $S_{\mathcal{A}_2}$, car les projections d'une configuration terminale de $\mathcal{A}_2 \circ \mathcal{A}_1$ sur $\mathcal{A}_1$ et $\mathcal{A}_2$ sont elles aussi terminales.

5 Terminaison de la composition

La terminaison de la composition de deux algorithmes silencieux n'est pas vraie en général. Par exemple, il arrive souvent que la convergence de $\mathcal{A}_2$ requière la convergence préalable de $\mathcal{A}_1$ parce que le rôle de $\mathcal{A}_1$ est de construire une structure topologique permettant à $\mathcal{A}_2$ d'organiser la transmission d'informations sur le réseau. Dans ce cas, tant que $\mathcal{A}_1$ n'a pas convergé, $\mathcal{A}_2$ peut ne pas terminer (et empêcher ainsi de faire avancer $\mathcal{A}_1$) : il faut ajouter des hypothèses garantissant la terminaison de $\mathcal{A}_1$.

5.1 Exécutions asynchrones

Cas d'école, nous supposons la terminaison de $\mathcal{A}_1$ depuis toute configuration vérifiant $Assume_{\mathcal{A}_1}$ et nous faisons l'hypothèse que $\mathcal{A}_2$ termine pour *toute configuration initiale*. Nous matérialisons ces hypothèses en supposant que les transitions entre les configurations successives de $\mathcal{A}_1$ et $\mathcal{A}_2$ sont inclues dans deux relations bien fondées $\prec_1$ et $\prec_2$. On peut alors montrer que $\mathcal{A}_2$ va converger, ce qui provoquera l'avancée de $\mathcal{A}_1$. En projetant, configuration après configuration, une exécution de $\mathcal{A}_2 \circ \mathcal{A}_1$ sur $\mathcal{A}_1$ et sur $\mathcal{A}_2$, on obtient une suite de paires de configurations de $\mathcal{A}_1$ et $\mathcal{A}_2$. On peut observer deux types de pas entre deux paires successives : d'une part les pas qui font avancer $\mathcal{A}_1$ pour au moins un nœud — et possiblement $\mathcal{A}_2$ pour d'autres — et font ainsi décroître la configuration de $\mathcal{A}_1$ selon $\prec_1$ et d'autre part les pas qui font avancer uniquement $\mathcal{A}_2$ et font ainsi décroître la configuration de $\mathcal{A}_2$ selon $\prec_2$, la configuration de $\mathcal{A}_1$ restant inchangée. On en déduit que les paires de configurations décroissent selon le produit lexicographique formé à partir de $\prec_1$ et $\prec_2$, $\prec_1 \times_{\mathrm{LEX}} \prec_2$, qui est lui-même bien fondé.

5.2 Exécutions faiblement équitables

L'hypothèse ci-dessus est très forte et rarement réalisable en pratique. Nous avons établi la terminaison de l'algorithme composé sous des hypothèses, plus réalistes [6], de convergence sous équité faible. Précisément, une exécution d'un algorithme est dite *faiblement équitable* si tout au long de l'exécution, pour tout nœud, quand celui-ci devient activable, alors il est activé ou neutralisé ¶ en temps fini. La propriété d'équité faible est définie en Coq avec des opérateurs inductifs et co-inductifs de logique temporelle.

Pour ce deuxième cas, nous avons prouvé que si toute exécution faiblement équitable de $\mathcal{A}_1$ et de $\mathcal{A}_2$ termine, alors toute exécution faiblement équitable de $\mathcal{A}_2 \circ \mathcal{A}_1$ termine elle aussi. Pour cela, nous avons travaillé directement sur les exécutions de $\mathcal{A}_2 \circ \mathcal{A}_1$ et leurs projections sur $\mathcal{A}_1$ et $\mathcal{A}_2$: d'abord l'algorithme $\mathcal{A}_1$ va terminer (on ne s'intéresse pas à ce qu'exécute $\mathcal{A}_2$ pendant ce temps), puis une fois que $\mathcal{A}_1$ est dans une configuration terminale et donc que $Assume_{\mathcal{A}_2}$ est validé, $\mathcal{A}_2$ converge.

Projection sur $\mathcal{A}_1$. Etant donnée une exécution e de $\mathcal{A}_2 \circ \mathcal{A}_1$, la projection de e sur $\mathcal{A}_1$ (séquence des projections des configurations de e), $\pi_1(e)$, n'est pas forcément une exécution de $\mathcal{A}_1$ car certaines configurations peuvent être répétées quand les nœuds n'exécutent que $\mathcal{A}_2$. Pour obtenir une exécution de $\mathcal{A}_1$, il ne faut garder qu'une seule des configurations répétées. Nous appelons cette opération *compactage*. Construire effectivement ce compactage nécessite de s'appuyer sur l'hypothèse d'équité faible qui garantit qu'il n'y

¶. Neutralisé signifie que le nœud n'est plus activable, à cause de ses voisins qui ont changé d'état.

aura qu'un nombre fini de répétitions (et donc un nombre fini d'éléments à retirer de la séquence de configurations) entre deux pas de $\mathcal{A}_1$. Pour cela, nous avons développé des outils sur des séquences d'éléments.

D'abord, le prédicat *compactable* qualifie une séquence dont on peut prédire à chaque étape — par examen d'un préfixe fini — si elle va changer de valeur ou rester constante. Ensuite nous avons écrit un opérateur de compactage qui transforme une séquence compactable s en sa compactée $Cp(s)$ contenant la même suite de valeurs que s, mais sans répétition. La définition de Cp se fait par co-induction structurelle sur celle de *compactable*. Nous disons qu'une séquence s est une *simulation* d'une autre séquence s' si s contient la même séquence de valeurs que s' mais éventuellement avec moins de répétitions que s. Ainsi, la compactée $Cp(s)$ est une simulation de s pour toute séquence (preuve par co-induction suivant la définition).

Nous appliquons ces outils aux exécutions. Soit e une exécution faiblement équitable de $\mathcal{A}_2 \circ \mathcal{A}_1$ et soit e_1 une séquence de configurations de $\mathcal{A}_1$ telle que e_1 soit une simulation de $\pi_1(e)$. Nous avons montré qu'alors, e_1 est faiblement équitable et que, si en plus e_1 ne contient aucune répétition, e_1 est une exécution de $\mathcal{A}_1$. De plus, puisque e est faiblement équitable, $\pi_1(e)$ est compactable, ce qui garantit l'existence de sa compactée $Cp(\pi_1(e))$. En instanciant la séquence e_1 ci-dessus par $Cp(\pi_1(e))$, nous obtenons que $Cp(\pi_1(e))$ est une exécution de $\mathcal{A}_1$ faiblement équitable et donc, par hypothèse, elle termine. Ainsi, e a un suffixe d'exécution pour lequel $\mathcal{A}_1$ n'est plus jamais exécuté (configuration terminale de $\mathcal{A}_1$).

Projection sur $\mathcal{A}_2$. On ne s'intéresse à la projection de e sur $\mathcal{A}_2$ qu'une fois la configuration terminale de $\mathcal{A}_1$ atteinte (et donc $Assume_{\mathcal{A}_2}$ vérifié). Ainsi la projection $\pi_2(e)$ obtenue par simple projection des configurations est-elle une exécution faiblement équitable de $\mathcal{A}_2$ puisque $\mathcal{A}_1$ ne s'exécute plus. Par hypothèse, $\pi_2(e)$ termine ; donc e termine.

6 Conclusion

Les théorèmes prouvés permettent de montrer que la composition hiérarchique collatérale préserve l'autostabilisation dans le cadre d'exécutions faiblement équitables. Nous avons appliqué cela aux deux premières couches de l'algorithme [6]. La première couche suppose un réseau quelconque identifié et construit un arbre couvrant enraciné ; la deuxième suppose un tel arbre et calcule un ensemble k-dominant ($k \in \mathbb{N}$). Les deux algorithmes sont autostabilisants pour des exécutions faiblement équitables et notre résultat certifie que leur composition l'est aussi et construit un ensemble k-dominant à partir d'un réseau identifié.

Par la suite, nous allons travailler sur la complexité de la composition obtenue à partir des complexités des algorithmes composés : complexités exprimées en nombre de pas de calcul, puis en rondes (unité de mesure en fonction de la vitesse du processus le plus lent).

Références

[1] K. Altisen, C. Pierre, and S. Devismes. PADEC. `http://www.verimag.fr/~altisen/PADEC/`.

[2] K. Altisen, C. Pierre, and S. Devismes. A framework for certified self-stabilization. In *FORTE*, 2016.

[3] A. Arora and M. G. Gouda. Distributed reset. *IEEE Trans. Computers*, 43(9) :1026–1038, 1994.

[4] J. Beauquier, M. Gradinariu, and C. Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In *Self-Stabilizing Systems : 5th International Workshop*, 2001.

[5] A. K. Datta, S. Gurumurthy, F. Petit, and V. Villain. Self-stabilizing network orientation algorithms in arbitrary rooted networks. *Stud. Inform. Univ.*, 1(1) :1–22, 2001.

[6] A. K. Datta, L. L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Self-stabilizing small k-dominating sets. *IJNC, International Journal of Networking and Computing*, 3(1) :116–136, 2013.

[7] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.

[8] S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[9] M. Gouda and T. Herman. Adaptive programming. *IEEE Transactions on Software Engineering*, 17 :911–921, 1991.

[10] The Coq Development Team. The Coq Proof Assistant. `http://coq.inria.fr/`.