Complexité certifiée d'algorithmes autostabilisants en rondes[†]

Karine Altisen[1], Pierre Corbineau[1] et Stéphane Devismes[2]

[1] *Univ. Grenoble Alpes, CNRS, Grenoble INP - UGA, VERIMAG, F-3800 Grenoble, France*
[2] *Université de Picardie Jules Verne, MIS (UR 4290), Amiens (France)*

Cet article décrit une extension de la bibliothèque PADEC qui permet le développement de preuves certifiées en Coq d'algorithmes autostabilisants. Cette extension introduit la définition d'une mesure de complexité en temps essentielle pour ces algorithmes : la ronde. Cette définition s'accompagne d'un ensemble d'outils permettant de faciliter les preuves de complexité en rondes. L'extension contient aussi une étude de cas en validant l'usage.

Mots-clefs : Algorithmique distribuée, autostabilisation, certification, preuve assistée par ordinateur, Coq.

1 Introduction

La complexité en temps d'un algorithme est une mesure visant à représenter le temps de calcul requis par cet algorithme pour s'exécuter. En général, elle est évaluée en comptant le nombre de « transitions » effectuées par l'algorithme, c'est-à-dire le nombre d'opérations qui sont considérées comme élémentaires dans le modèle de calcul dans lequel est évaluée la complexité. Par exemple, en algorithmique séquentielle, on considère les opérations basiques, telles que la multiplication ou la division, comme élémentaires (alors que leurs implémentations ne le sont pas toujours) et donc s'exécutant en temps constant.

Dans le cadre des algorithmes autostabilisants, qui offrent une mise en œuvre légère de la tolérance aux pannes transitoires [5], la mesure de complexité en temps de référence – la *ronde* – n'est pas toujours facile à appréhender car elle n'est pas atomique dans le modèle de calcul utilisé, le modèle à états. Cette particularité n'est pas unique dans le monde des systèmes distribués ; voir par exemple l'évaluation du temps dans le modèle à passage de messages. Cependant, l'absence d'atomicité engendre une définition complexe et implique que certaines propriétés naturelles telles que la compositionalité nécessitent d'être étudiées rigoureusement.

La bibliothèque PADEC [1, 3] – *Preuves d'Algorithmes Distribués en Coq* – fournit une fondation formelle des concepts liés à l'autostabilisation. Coq [7] est un assistant de preuve qui permet de développer des preuves certifiées, c'est-à-dire validées mécaniquement par l'outil et donc plus sûres. Nous l'utilisons dans PADEC pour modéliser des algorithmes dans le modèle le plus répandu en autostabilisation, le *modèle à états*. Nous pouvons ainsi prouver formellement que ces algorithmes sont autostabilisants pour leur spécification, c'est-à-dire qu'à partir d'un état initial quelconque, l'algorithme assure que le système va atteindre en temps fini un état correct à partir duquel il va satisfaire sa spécification.

Dans ce travail, nous augmentons la librairie PADEC en y incorporant la notion de ronde. Un effort constant a été fourni pour que la formulation des définitions et résultats dans PADEC soit aussi proche que possible de celle utilisée pour les preuves sur papier. Cet effort se poursuit ici en fournissant à l'utilisateur des outils qui lui permettront de réaliser ses preuves certifiées de complexité plus aisément.

2 PADEC

La bibliothèque PADEC contient la définition formelle du modèle de calcul le plus utilisé en autostabilisation, le *modèle à états*, mais aussi des propriétés permettant de spécifier ce qu'est un algorithme

[†] Soutenu par le projet ANR SKYDATA (ANR-22-CE25-0008).

autostabilisant pour une spécification donnée ainsi que la définition de spécifications de problèmes classiques (élection, circulation de jeton, ...). Il contient des outils pour définir les hypothèses sur lesquelles travaille l'algorithme (topologies, anonymat, ...), mais aussi des résultats permettant de travailler sur des algorithmes composés ainsi qu'un certain nombre d'études de cas validant les approches. Dernièrement, nous nous sommes intéressés à la preuve de complexité en temps : il y a deux mesures de temps principales dans le modèle à états. On peut compter le nombre de *pas de calculs* réalisés dans le système, cette mesure étant atomique (*cf.* [4] pour la définition des pas de calculs dans PADEC). On peut aussi compter le nombre de *rondes* dont la définition dans PADEC fait l'objet du travail présenté ici.

Nous définissons maintenant les éléments du modèle à états et de PADEC nécessaires à la bonne compréhension des rondes. Nous renvoyons le lecteur à [1, 3] pour une présentation plus exhaustive. Dans PADEC, un système distribué est représenté par un réseau, composé de nœuds ayant des canaux de communication les liant à d'autres nœuds, leurs *voisins*. Chaque nœud est équipé d'un algorithme local et d'un état (ensemble de variables). Un nœud peut directement lire les états de ses voisins. L'algorithme du nœud s'écrit sous la forme de règles gardées, chacune étant composée d'une condition et d'un ensemble d'affectations. La condition et les affectations peuvent faire référence à l'état des nœuds voisins mais seules les variables locales du nœud peuvent être écrites. Si une condition d'une règle d'un nœud est satisfaite, le nœud est dit *activable*, il peut alors exécuter atomiquement les affectations de la règle. On appelle *configuration* l'ensemble des états des nœuds du réseau. L'*exécution* d'un algorithme dans un réseau est une suite maximale de configurations où chaque couple de configurations successives représente un *pas atomique du système* réalisé de la façon suivante. Le *démon*, un oracle modélisant l'asynchronisme, sélectionne (de façon non-déterministe) un sous-ensemble non vide de noeuds activables. Ces noeuds sont alors *activés*, c'est-à-dire que chacun d'eux exécute une règle activable. Le démon est parfois contraint par des hypothèses, d'équité par exemple, que PADEC permet aussi de modéliser. Il faut noter que lors d'un pas de calcul un nœud activable mais non activé peut à la configuration suivante devenir non activable du fait du changement d'état de certains de ses voisins. On dit alors que le nœud est *neutralisé*.

3 Complexité en rondes

3.1 Les rondes en langage naturel

La notion de ronde [‡] est dans la communauté de l'autostabilisation une notion parfaitement définie en langage naturel mais par essence, les rondes ne sont pas atomiques dans le modèle de calcul où elles sont définies, le modèle à états, et elles peuvent être infinie, ce qui, en un certain sens, ne devrait pas être pris en compte dans l'évaluation de la complexité. Dans le modèle à états, l'exécution e d'un algorithme peut être divisée en rondes de la façon suivante. Notons U l'ensemble des nœuds activables à la première configuration de e. La première ronde de e se termine à la première configuration g de e où tous les noeuds de U ont été activés ou neutralisés. Si une telle configuration n'existe pas, alors la ronde est infinie ; sinon, la seconde ronde de e est la première ronde du suffixe de e commençant par g. Une ronde peut être infinie à cause d'un effet de famine provoqué par un démon non équitable.

La ronde est donc une unité de temps et elle est utilisée pour évaluer le temps requis par une exécution e pour atteindre une certaine propriété (cette propriété, que nous noterons P dans la suite, qualifie une exécution). Si P est vraie dès le début de e, alors par convention, on dit que *e satisfait P en 0 ronde*. Sinon, *e satisfait P en i rondes* où i est l'indice de la première ronde de e contenant une configuration à partir de laquelle P est vraie.

Les résultats de complexité expriment fréquemment des bornes supérieures plutôt que des complexités exactes. Pour prendre en compte cela dans PADEC, nous avons introduit la propriété qui exprime qu'*« une exécution requiert au plus j rondes pour atteindre P »*. Cette dernière est définie comme suit. Soit e une exécution qui contient au moins j rondes, e requiert au plus j rondes pour atteindre P si e satisfait P en i rondes avec $i \leq j$. Remarquons que l'hypothèse que e contient au moins j rondes est rendue utile par le fait qu'une ronde peut être infinie (auquel cas e pourrait contenir moins de j rondes). La propriété « e requiert

‡. Traduction littérale de *Round*.

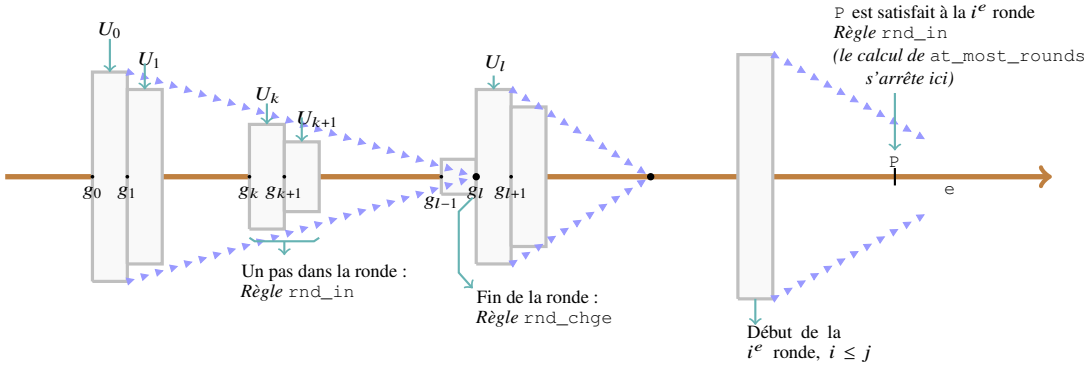

FIGURE 1 : Définition des rondes : schéma de principe pour `at_most_rounds` P j e.

au plus j rondes pour atteindre P » est ensuite naturellement étendue à toutes les exécutions possibles en la fixant à vraie pour toute exécution qui contient strictement moins de j rondes.

3.2 Les rondes dans PADEC

Nous expliquons maintenant comment cette propriété est formellement définie en PADEC où elle s'écrit : `at_most_rounds` P j e. Conformément à la définition en langage naturel, nous définissons l'ensemble U appelé *ensemble des nœuds insatisfaits*. Au début de la ronde, U est initialisé par l'ensemble de nœuds activables à la première configuration de e. Et à chaque pas, on supprime de U l'ensemble des nœuds qui ont été soit activés soit neutralisés. La ronde s'arrête quand U est vide, auquel cas on le remplit à nouveau avec les nœuds activables de la configuration courante, ce qui permet d'initialiser la ronde suivante. Notez que les ensembles de nœuds sont définis en PADEC comme des fonctions booléennes et PADEC fournit les outils permettant de les manipuler.

Le prédicat `at_most_rounds` P j e est alors défini à l'aide d'un prédicat intermédiaire qui utilise le paramètre additionnel, U, l'ensemble des nœuds insatisfaits. Voici comment est défini ce prédicat intermédiaire. La définition informelle à implémenter est la suivante : « si e contient au moins j rondes, alors e satisfait P en i rondes avec $i \leq j$ ». Bien sûr, les calculs du « si » et du « alors » ne vont se faire qu'en un seul parcours de e. Mais, puisqu'une ronde peut être infinie, ce prédicat est par essence coinductif, la coinduction étant le concept qui – en Coq en particulier – permet de manipuler des objets éventuellement infinis. Le prédicat parcourt l'exécution, pas par pas, en appliquant une des trois règles suivantes : la règle `rnd_here` détecte que P est valide et arrête l'exécution, la règle `rnd_in` s'applique à l'intérieur d'une ronde et la règle `rnd_chge` s'applique au changement de ronde.

La figure 1 illustre plus en détail le calcul du prédicat. La ligne horizontale représente l'exécution $e = g_0\, g_1 \ldots$ et les rectangles sont les ensembles $U_0, U_1, \ldots$ de nœuds insatisfaits, mis à jour à chaque pas comme expliqué ci-dessus. Au début, U_0 vaut l'ensemble des nœuds activables à la configuration g_0. Ensuite, lors des pas à l'intérieur de la ronde (voir le pas de g_k à g_{k+1}), la règle `rnd_in` s'applique de telle façon que l'ensemble de nœuds insatisfaits diminue progressivement en retirant les nœuds activés ou neutralisés lors du pas courant. À la fin de la ronde (configuration g_l), la règle `rnd_chge` s'applique alors que la mise à jour de l'ensemble des nœuds insatisfaits produit un ensemble vide : cet ensemble est immédiatement (toujours à la configuration g_l) remplit à l'aide des nœuds activables en g_l. Quand P devient vrai, pendant la ronde i, avec $i \leq j$, la règle `rnd_here` s'applique : l'évaluation du prédicat s'arrête et `at_most_rounds` P j e est satisfait.

Si on retire la partie droite de la figure, c'est-à-dire que e ne finit jamais par satisfaire P, alors la règle `rnd_here` ne peut jamais s'appliquer. Dans ce cas, l'évaluation du prédicat ne s'arrête pas, ce qui est permis par le fait que `at_most_rounds` est coinductif.

Pour valider que cette définition représente bien le décompte des rondes, un certain nombre de résultats ont été démontrés dans PADEC. En particulier, une définition fonctionnelle des rondes est aussi fournie (et qui donne heureusement les mêmes résultats !) : la fonction inductive `count_rounds P e` retourne i, le nombre exact de rondes nécessaires pour atteindre P. Bien entendu, cette fonction requiert une hypothèse supplémentaire sur le fait que e contient un suffixe qui vérifie P et que cette information est exploitable dans le calcul de la fonction.

3.3 Outils pour la preuve en rondes

Une fois définie la notion de ronde, il faut pouvoir l'utiliser pour faire des preuves de complexité d'algorithmes autostabilisants. Nous avons ainsi développé un certain nombre d'outils dans le but de faciliter l'utilisation du prédicat sur les rondes dans la preuve d'un utilisateur. Ces outils de preuve évitent à l'utilisateur de manipuler la coinduction (due à la nature coinductive du prédicat) – ce qui peut être ardu en Coq ; et ils imitent la façon dont ces preuves peuvent être conduites sur le papier. En particulier, et puisque prouver une complexité d'au plus j rondes pour atteindre P se fait souvent par induction sur j, nous proposons le schéma d'induction suivant.

Lemme : *Supposons que nous disposions d'une famille de prédicats P_n (indicés sur des naturels, n) tels que P_j implique P. Supposons aussi qu'on puisse prouver que*

1. *l'exécution e satisfait P_0 (cas de base)*
2. *tout au long de e et pour tout $i < j$, si e a un suffixe c qui satisfait P_i alors c requiert au plus une ronde supplémentaire pour atteindre P_{i+1} (pas d'induction)*

Alors, on peut déduire que e requiert au plus j rondes pour atteindre P_j et donc P. L'utilisateur d'un tel lemme n'a qu'à prouver, comme pour toute démonstration par induction, les deux hypothèses 1 et 2 pour obtenir la conclusion.

La preuve (en Coq et dans PADEC) de ce lemme n'est pas aussi triviale qu'il n'y paraît. En effet, si la preuve globale se fait simplement par induction sur le paramètre i, le passage de i à $i + 1$ se heurte au fait qu'une ronde n'est pas atomique ce qui implique que le nombre de rondes n'est pas additif de façon exacte. Nous ne pouvons expliquer la preuve par manque de place, mais voici une intuition de la difficulté. Si (a) une exécution e satisfait un prédicat P_1 en une ronde et si (b) à partir du moment où P_1 est satisfait, il faut encore une ronde pour atteindre un autre prédicat P_2, alors (c) il se peut que e satisfasse P_2 en 2 rondes mais il se peut aussi que e satisfasse P_2 en une seule ronde (évidemment e satisfait P_2 en au plus 2 rondes). Cette particularité est due au fait que, quand P_1 est satisfait, la ronde en cours n'est pas forcément terminée et peut se finir après que P_2 soit satisfait. L'hypothèse (b), par contre, recommence un calcul de ronde à partir du moment où P_1 est validé. Ainsi, la suite des ensembles de nœuds insatisfaits n'est pas la même dans l'hypothèse (b) et dans la conclusion (c). D'où, l'écart possible de résultat.

4 Conclusion

Nous avons présenté ici la dernière étape dans le développement de PADEC (accessible ici [2]). Cette étape contient l'ajout d'outils permettant de certifier les complexités en rondes. Elle représente environ 11 000 lignes de code Coq [§]. La principale difficulté rencontrée vient du fait que la ronde n'est pas atomique dans le modèle à états, ce qui la rend peu compositionnelle. Nous accompagnons la définition de ronde d'outils permettant de faciliter les preuves en ce sens qu'ils reproduisent au plus près la façon dont les preuves sont réalisées à la main et qu'ils évitent aussi à l'utilisateur de manipuler directement la notion de ronde, cette dernière étant intrinsèquement coinductive et la coinduction étant un concept parfois difficile à manipuler en Coq. Nous avons validé l'ensemble de cette extension en certifiant le temps de stabilisation en rondes d'un algorithme autostabilisant classique de la littérature : il réalise la construction d'arbre couvrant en largeur dans un réseau enraciné bidirectionnel [6]. En particulier, on peut constater que, grâce aux outils compagnons, les preuves n'utilisent aucun raisonnement par coinduction dû à la définition des rondes.

Références

[1] PADEC Framework. `http://www.verimag.fr/~altisen/PADEC/`.

[2] PADEC Round Library and Use Case. `https://doi.org/10.5281/zenodo.7513651`.

[3] K. Altisen, P. Corbineau, and S. Devismes. A framework for certified self-stabilization. *LMCS*, 13(4), 2017.

[4] K. Altisen, P. Corbineau, and S. Devismes. Certification of an exact worst-case self-stabilization time. In *ICDCN*, 2021.

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

[6] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only Read/Write atomicity. *Distributed Computing*, 7(1), 1993.

[7] The Coq Development Team. *The Coq Proof Assistant Documentation*, June 2012.

§. Mesuré par l'outil dédié `coqwc : #loc: spec = 2,698; proof = 7,892; comments = 484.`