Une Analyse Formelle en Coq d'un Algorithme Distribué Probabiliste résolvant le Problème du Rendez-Vous

Allyx Fontaine, Akka Zemmari

Une Analyse Formelle en Coq d'un Algorithme Distribué Probabiliste résolvant le Problème du Rendez-Vous

Allyx Fontaine, Akka Zemmari

Université Bordeaux 1 - LaBRI
351 cours de la Libération, 33405 Talence, France
{allyx.fontaine, akka.zemmari}@labri.fr

Résumé

Les algorithmes distribués probabilistes se formulent simplement, cependant leur analyse est complexe car il faut traiter à la fois l'aspect distribué et l'aspect probabiliste. Dans cet article, nous présentons une formalisation en *Coq* d'un algorithme distribué probabiliste résolvant le problème du rendez-vous. Cette formalisation nous permet de raisonner et de prouver des propriétés telles que la terminaison ou des calculs de probabilités.

1. Introduction

Les algorithmes distribués ont reçu une attention considérable et ont été étudiés intensivement ces dernières années. De nombreux travaux ont été faits pour étudier leur puissance de calcul et/ou pour concevoir des solutions efficaces en utilisant ces algorithmes. Les algorithmes distribués probabilistes ont été introduits pour garantir une meilleure efficacité ou pour résoudre des problèmes qui n'avaient pas de solutions déterministes. De tels résultats d'impossibilité ont été démontré en *Coq* dans la bibliothèque *Loco* [CF], il nous est alors apparu intéressant d'y intégrer aussi des algorithmes probabilistes distribués. Notre première étude concerne le problème du rendez-vous. Dans cet article, nous en faisons l'analyse en *Coq* après avoir modélisé la classe d'algorithmes sur laquelle nous voulons raisonner.

1.1. Systèmes distribués probabilistes

Un *système distribué* est une collection de nœuds autonomes - entités ayant leurs propres moyens de contrôle et de calcul - communiquant via des liens de communication [Tel00]. Ces entités peuvent être des ordinateurs ou des processeurs (entités physiques) ou simplement des processus (entités logiques). Un *algorithme distribué* est un algorithme conçu pour un système distribué.

Soit $\mathcal{T}$ une tâche à réaliser et soit $\mathcal{S}$ un système distribué à $n \geq 1$ entités. Un *algorithme distribué* pour la tâche $\mathcal{T}$ est un algorithme capable de faire collaborer tous, ou partie, des entités du système $\mathcal{S}$ afin de réaliser $\mathcal{T}$. Cet algorithme peut également être vu comme la donnée d'un algorithme local que chacune des n entités du système $\mathcal{S}$ exécute.

Un *algorithme probabiliste* est un algorithme qui utilise des tirages de type pile ou face ou des générateurs de nombres aléatoires. A la différence des algorithmes *déterministes*, le *hasard* joue un rôle fondamental dans l'exécution de tels algorithmes. Les algorithmes probabilistes permettent de fournir des solutions parfois plus "efficaces" que les solutions déterministes, ou encore des solutions pour des problèmes qui n'admettent pas de solution déterministe.

Un *algorithme probabiliste distribué* est par conséquent l'affectation d'un algorithme local probabiliste à chaque entité du système.

Dans un système distribué *synchrone*, une action (ou opération) de chaque processus se réalise en une suite de pas discrets appelée *ronde*. Dans une ronde, un processus envoie 0 ou plusieurs messages, reçoit 0 ou plusieurs messages et effectue un ensemble de calculs. Un système distribué est dit *anonyme* si on ne peut accéder aux identités des différentes entités du système. Dans la littérature (voir [Lav94]), on parle de *symétrie* : les processus jouent tous le même rôle et on ne dispose d'aucun moyen pour les distinguer.

Dans cet article, nous considérons un système distribué synchrone, communiquant par échange de messages : chaque processus envoie un message à son voisin en le déposant dans le lien correspondant à ce voisin. Nous modélisons le système par un graphe $G = (V, E)$ où V est un ensemble de sommets représentant l'ensemble des entités du système et E l'ensemble des arêtes représentant les liens de communication entre les entités. Le passage de messages est représenté par une fonction d'étiquetage sur les extrémités des arêtes.

1.2. Preuve formelle de systèmes distribués probabilistes

En général, les algorithmes distribués probabilistes s'expriment de façon concise. Par contre, leur analyse reste délicate et complexe, ce qui rend difficile la preuve de leur correction. Nous pensons que les preuves obtenues et vérifiées par un outil auquel on peut se fier garantit leur correction et le calcul de leur distribution.

Nous avons pour objectif d'intégrer des algorithmes distribués probabilistes dans l'environnement de preuve en construction *Loco* [CF], jusqu'à présent spécialisé dans les preuves formelles de systèmes déterministes de calculs locaux.

Notre modèle des systèmes distribués repose sur la théorie des graphes. *Coq* n'ayant pas de bibliothèque standard pour les graphes, nous nous sommes intéressés aux alternatives. La bibliothèque *ssreflect* [GMT08] est une extension qui a été utilisée pour réaliser la formalisation du théorème des quatre couleurs. Les outils pour raisonner sur les graphes en tant qu'un ensemble fini de sommets et d'une relation d'adjacence qui y sont développés ont facilité la rédaction des preuves, en les allégeant via la réflection ou les systèmes de réécriture mis en place.

A la modélisation que nous avons faites des systèmes distribués, vient s'ajouter l'aspect probabiliste. Les algorithmes probabilistes ont été peu formalisés. Cependant, des outils pour raisonner sur de tels algorithmes ont été développés dans la bibliothèque *Alea* par P. Audebaud et C. Paulin [APM09]. C'est pourquoi, nous avons eu recours à cette bibliothèque suffisamment complète pour nos raisonnements.

1.3. Le problème du rendez-vous

Dans un réseau de processeurs communiquant par échange de messages en mode synchrone, l'émetteur et le récepteur doivent être tous les deux prêts à communiquer. Un rendez-vous (ou poignée de main) permet de réaliser des communications exclusives entre des paires de sommets voisins. Une solution à ce problème est une brique de base pour la mise en œuvre de systèmes de réécriture impliquant des règles sur les arêtes (voir [MS97]) ou encore, pour la réalisation (après un nombre d'itérations suffisant) d'un couplage maximal du graphe représentant le réseau. L'objectif de ce papier est de montrer comment raisonner sur une solution distribuée probabiliste du problème du rendez-vous décrite dans [MSZ03]. Par la suite, nous parlerons des algorithmes, résultats, analyse relatifs à cet article dans les termes “algorithme papier”, “résultats papier” ou “analyse papier”.

1.4. Travaux reliés

1.4.1. Algorithmes distribués

Dans [Cho95] Ching-Tsun Chou montre la correction d'algorithmes distribués grâce à l'assistant de preuve *HOL*. Les algorithmes distribués sont modélisés par des systèmes de transitions d'étiquetage et les spécifications sont exprimées en termes de logique temporelle.

D. Cansell et D. Méry [CM07, Can] décrivent les systèmes de calculs locaux à partir de leur spécifications formelles par des étapes successives de raffinement par le biais du formalisme `Event-B` [Abr10]. Une tâche est représentée par une machine abstraite, chaque raffinement est prouvé avec l'aide de la plate-forme `Rodin` [Dep].

P. Castéran et V. Filou ont développé une bibliothèque [CF] sur les graphes étiquetés et les systèmes de réétiquetages de graphes. Elle permet à l'utilisateur de spécifier des tâches et de prouver la correction de systèmes de réétiquetage par rapport à leur tâches ainsi que des résultats d'impossibilité.

1.4.2. Algorithmes probabilistes

Dans [Hur02], J. Hurd représente un programme probabiliste comme étant une monade d'état où l'état est une suite infinie de booléens. Cette interprétation sous la forme d'une fonction de type $\beta \to \mathbb{B}^\infty \to (\beta \times \mathbb{B}^\infty)$ a été implantée dans l'assistant de preuve *HOL*. Le programme prend en entrée une suite infinie de booléens tirés uniformément au hasard pour faire le calcul et retourner le résultat ainsi que le reste de la suite de booléens (la suite privée des booléens consommés).

Une autre interprétation est de considérer les programmes probabilistes comme des transformateurs de mesures. P. Audebaud et C. Paulin [APM09] utilisent une transformation monadique pour représenter les distributions de probabilité ce qui permet d'estimer la probabilité qu'un programme vérifie une certaine propriété. Ils ont développé *Alea*, une bibliothèque pour raisonner sur les programmes probabilistes. *Alea* est utilisée dans des applications qui nécessitent des outils probabilistes telles que CertyCript/EasyCrypt [BCG12] qui aide à construire et vérifier des preuves dans le domaine de la cryptographie.

1.4.3. Algorithmes distribués probabilistes

Plusieurs approches prennent en compte le double paradigme des systèmes distribués probabilistes : l'aspect probabiliste et le non-déterminisme dû au temps de réponse variant d'un processeur à l'autre. Elles requièrent des modèles avec des choix non-déterministes entre plusieurs distributions de probabilités. Ces choix peuvent être faits par un ordonnanceur ou un adversaire. Des modèles équivalents existent en suivant cette idée : automates probabilistes [PS95], des processus décisionnels de Markov [Der70]. Pour spécifier des propriétés d'algorithmes distribués probabilistes, on peut utiliser la logique temporelle avec des opérateurs probabilistes et un seuil. Le *Model Checking* est un outil utilisé pour assurer la correction des systèmes, mais utilisé avec des probabilités, cela mène à une explosion de la complexité en espace. Il existe des méthodes pour réduire cette explosion. Une analyse qualitative des algorithmes distribués probabilistes est faisable grâce au model checker PRISM[KNP02]. G. Norman fait un résumé de ces approches dans [Nor04].

Dans notre modélisation, nous considérons que l'algorithme opère par rondes, en appliquant un algorithme sur des parties disjointes du graphe. Cela enlève le non-déterminisme. Selon nos connaissances, il n'y a pas de développement utilisant un assistant de preuve qui s'intéresse aux algorithmes distribués probabilistes.

1.5. Notre contribution

Nous nous intéressons à l'obtention de preuves formelles d'algorithmes distribués, y compris les algorithmes probabilistes. Pour ce faire, nous utilisons l'assistant de preuves *Coq*, la bibliothèque *Alea* et le plugin *ssreflect*.

Dans cet article, nous décrivons une analyse formelle d'un algorithme distribué probabiliste qui résout le problème du rendez-vous. Cet algorithme a été introduit et étudié dans [MSZ03]. Les auteurs y analysent la probabilité d'avoir au moins un rendez-vous dans le réseau.

Nos travaux modélisent les algorithmes distribués probabilistes puis développent des outils en *Coq* pour les analyser. Cela nous mène à prouver formellement des résultats extraits de [MSZ03]. La formalisation en *Coq* de ces résultats a fait ressortir des obligations de preuve n'apparaissant pas explicitement dans l'analyse papier.

Remarque : Les lemmes et théorèmes énoncés dans cet article sont étiquetés de leur nom dans le développement en *Coq* disponible à l'adresse [CFZ].

Selon nous, c'est la première fois qu'un travail s'intéressant à la modélisation formelle et à la preuve d'algorithmes distribués probabilistes et plus particulièrement au problème du rendez-vous est réalisé.

1.6. Organisation de l'article

Cet article est organisé comme suit : après avoir introduit notre modélisation des systèmes distribués dans la Section 2, la Section 3 présente un algorithme résolvant le rendez-vous comme il a été décrit dans [MSZ03]. Dans les Sections 4 et 5, nous donnons une définition formelle de cet algorithme de façon à rendre possible son analyse en *Coq*. Enfin, la structure principale de l'analyse en *Coq* est présentée en Section 6.

2. Le modèle

Notre modèle est le modèle synchrone par passage de messages [Tel00]. Un système distribué est représenté par un graphe simple connexe $G = (V, E)$ où les sommets (l'ensemble V) correspondent aux processus et les arêtes (l'ensemble E) aux liens de communications.

Le nombre de sommets de V est appelée taille du graphe et est notée n, et le nombre d'arêtes dans E est noté m. L'arête qui lie deux sommets adjacents v et w est notée $\{v, w\}$. On note $d(v)$ le degré d'un sommet v dans le graphe G. Nous sommes amenés à manipuler des ensembles de sommets ou arêtes. Étant donné une fonction d'énumération d'ensemble fini, `enum`, chaque fois que nous avons besoin d'une énumération explicite, d'un ensemble S par exemple, nous utilisons les notations `(enum` S`)` ou encore $s_1 \dots s_n$ où s_i est un élément de S.

Chaque processus peut distinguer les différentes arêtes incidentes, *i.e.*, pour chaque sommet $u \in V$, il existe une bijection entre les voisins de u dans G et $[1..d(u)]$. Les nombres associés par chaque sommet à ses voisins sont appelés numéro de ports.

Nous définissons un port comme le couple d'un sommet et d'un entier afin de représenter le lien par lequel un sommet dépose son message. Ainsi, si v envoie un message à son $i^{\text{ème}}$ voisin, il va envoyer son message par le port (v, i). Supposons que w est le $i^{\text{ème}}$ voisin de v, un port peut être aussi sous la forme d'un couple de sommets, ici (v, w). Par exemple, dans la Figure 1(c), v_5 est le troisième voisin de v_2 et v_2 est le voisin numéro 0 de v_5 ; le port (v_2, v_5) peut aussi être représenté par le couple $(v_2, 3)$, de même le port (v_5, v_2) peut être représenté par le couple $(v_5, 0)$. Notons que les deux représentations sont utiles. La première est utile pour assurer l'anonymat et ainsi écrire des interactions locales au sujet d'un sommet et de son voisinage, pour définir par exemple l'algorithme local. La seconde est

utilisée pour exprimer avec une vue globale du graphe une situation entre deux sommets v et w.

Chaque processus v dans le réseau représente une entité qui est capable d'exécuter des pas de calculs, en envoyant des messages par ses ports et en en recevant par les ports utilisés par ses voisins. Nous considérons des systèmes synchrones : une action de chaque processus se réalise en une suite de pas discrets appelée *ronde*. Dans une ronde, un processus envoie un message, reçoit un message de chacun de ses voisins et effectue un ensemble de calculs locaux. Les messages envoyés sont reçus pendant la même ronde, c'est-à-dire, si p envoie un message à q à la $i^{\text{ème}}$ ronde, alors le message est reçu par q pendant la $i^{\text{ème}}$ ronde de q.

La communication entre les processus se fait par échange de messages : chaque processus envoie un message à son voisin en le déposant dans le lien correspondant à ce voisin.

Le réseau G est anonyme : des identités uniques ne sont pas disponibles pour différencier les processus lors de l'exécution. Nous ne supposons pas donnée des connaissances globales du réseau comme sa taille ou une borne supérieure de sa taille. Les processus n'ont pas besoin d'informations relatives à la position des sommets ou à leur distance dans le graphe. Cependant, un processus sait distinguer ses voisins grâce aux numéros de ports.

Remarque : L'hypothèse de l'anonymat est souvent considérée lorsque les processus ne peuvent divulguer leur identité pendant l'exécution, et ce pour des raisons liées à leur vie prouvée ou à des problèmes de politique de sécurité. En plus, chaque processus peut être intégré dans un réseau à grande échelle ce qui rend infaisable l'unicité des identifiants.

Dans ce modèle, un algorithme distribué est donné par un algorithme local que chaque processus doit exécuter, donc les processus qui ont le même degré ont le même algorithme. Un algorithme local consiste en une séquence de calculs locaux entrecoupés avec des instructions d'envoi et de réception de messages.

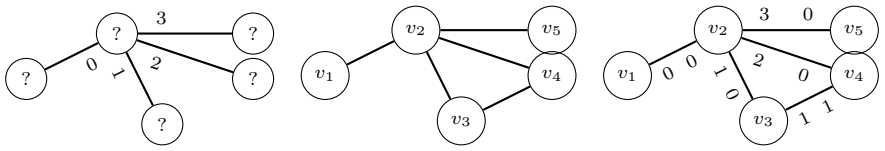
Un algorithme probabiliste est un algorithme qui fait des choix aléatoires basés sur une certaine distribution de probabilité.

Un algorithme distribué probabiliste est une collection d'algorithmes probabilistes locaux ; comme notre réseau est anonyme, si deux processeurs ont le même degré, leurs algorithmes locaux probabilistes sont identiques et ont la même distribution de probabilité.

Nous différencions la vue locale (l'exécution) de la vue globale (l'analyse). Lors de l'analyse de l'algorithme, des propriétés sur le graphe, dans sa globalité, seront étudiées, pour cela nous avons besoin de distinguer les sommets. Ainsi une vue globale du graphe consiste à mettre des identifiants sur les sommets. Un exemple de graphe se trouve à la Figure 1(b). Le système étant anonyme, un sommet n'a accès ni à son identifiant ni aux identifiants des autres sommets. Un sommet exécute un algorithme local avec des informations locales. La Figure 1(a) nous donne la vue locale d'un sommet à l'initialisation, un sommet ne peut distinguer que ses liens, il peut donc les numéroter. Enfin pour lier le local au global, il nous faudra passer d'une vue à l'autre. La Figure 1(c) est un exemple de numérotation de lien en correspondance avec les identifiants des sommets.

Nous modélisons l'échange des messages, d'un point de vue global, par une fonction d'étiquetage des ports sur le graphe G. La Figure 2 représente une ronde d'un algorithme où des sommets envoient des booléens à leurs voisins. Pour chaque port (v, w), la valeur envoyée par v pour w est affichée sous une flèche pointant de v vers w. Par exemple, nous pouvons voir, en mettant en parallèle la numérotation des ports de la Figure 1(c) que le sommet v_2 envoie **1** à son troisième voisin. Son troisième voisin correspond à v_5 d'un point de vue global.

Définition 1. *(Zone d'influence d'un sommet v)*
La zone d'influence *d'un sommet v est l'ensemble des ports qu'il peut modifier, c'est-à-dire les ports qui lui sont liés (du type* $(v, ?)$*). En d'autres termes, il s'agit de la zone de réétiquetage d'un sommet.*



(a) Vue locale d'un sommet (b) Représentation d'un réseau ayant 5 processeurs (c) Représentation d'un réseau avec une énumération locale possible du voisinage

FIGURE 1 – Modèle des algorithmes distribués par passage de messages

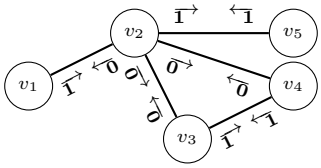


FIGURE 2 – Exemple d'une ronde où les messages sont des booléens

Considérons σ une fonction d'étiquetage qui associe à chaque port son état. Par la suite, l'état d'un port sera de type booléen. Le type de telles fonctions d'étiquetage sera appelé par la suite `State`. Nous notons simplement $\sigma(v, i)$ l'état stocké sur le port (v, i). Nous notons $\sigma[(v, i) \leftarrow s]$ la fonction d'étiquetage dont les états sont identiques à ceux associés à σ sauf sur le port (v, i) associé à s.

3. Le problème du rendez-vous

Le but du rendez-vous est de créer des liens de communication exclusifs entre des voisins dans un réseau. Par la suite, nous étudierons un algorithme distribué probabiliste pour le problème du rendez-vous dans un graphe G décrit et analysé dans [MSZ03].

3.1. L'algorithme papier

L'algorithme local, `hs_local`, appliqué à chaque sommet v est le suivant : v choisit uniformément au hasard un voisin, l'informe qu'il l'a choisi et informe les autres voisins qu'ils n'ont pas été choisis.

```
Algorithme hs
   Chaque sommet v applique simultanément l'algorithme local : (hs_local v)

Algorithme (hs_local v)
   v choisit uniformément au hasard l'un de ses voisins, le i^ème
   v envoie 0 à tous ses voisins sauf au i^ème à qui il envoie 1
```

FIGURE 3 – L'algorithme papier

La notion de rendez-vous s'exprime avec une vue globale sur le graphe entre deux sommets voisins, après l'exécution de l'algorithme global `hs`. Il y a un rendez-vous entre deux sommets voisins v et w si le choix de v correspond à w et celui de w à v.

Définition 2. *(Rendez-vous entre deux sommets)*

Supposons que, après une exécution de **hs**, *v ait choisi son $i^{ème}$ voisin et que w ait choisi son $j^{ème}$ voisin, il y a un* rendez-vous *entre v et w si le $i^{ème}$ voisin de v est w et le $j^{ème}$ voisin de w est v.*

La Figure 2 illustre le résultat d'une exécution possible de `hs`. Deux rendez-vous apparaissent : v_3 est associé à v_4 et v_2 à v_5.

3.2. Principaux résultats

Nous faisons l'analyse de cet algorithme qui se déroule en une ronde. Notons que la phase de réception des messages n'intervient pas ici.

Définition 3. *Nous notons $\mathcal{HS}(e)$ l'évènement "il y a un rendez-vous sur l'arête e".*

Nous définissons $\mathcal{H}(e)$ comme étant la fonction caractéristique de $\mathcal{HS}(e)$, c'est-à-dire un booléen valant **1** *s'il y a un rendez-vous sur e et* **0** *sinon.*

Le but de notre modélisation est de prouver formellement en *Coq* le résultat suivant [MSZ03] :

Théorème 1. *La probabilité $\mathbb{P}(\exists\ e \in E,\ \mathcal{H}(e))$ d'avoir au moins un rendez-vous après une exécution de* **hs** *est supérieure ou égale à la constante $(1 - e^{-1/2})$.*

Remarque : A la fin d'une ronde, la probabilité d'avoir au moins un rendez-vous est supérieure à $1 - e^{-1/2} \approx 0.39$. Par conséquent, si nous nommons par T le nombre de fois qu'il faut répéter `hs` pour avoir au moins un rendez-vous dans le réseau, alors T est une variable aléatoire géométrique de paramètre $p \geq 0.39$, d'où son espérance $1/p \leq 2.54$. On en déduit que pour obtenir un rendez-vous dans un graphe, 3 itérations de l'algorithme `hs` sont en moyenne suffisantes.

4. L'algorithme du rendez-vous comme un programme $\mathcal{R}ml$

L'algorithme `hs` opère par changement local et utilise des primitives probabilistes. Pour en faire une analyse rigoureuse en utilisant un assistant de preuve tel que *Coq*, nous avons besoin d'exprimer cet algorithme dans un langage fonctionnel permettant de faire une analyse probabiliste.

Notre volonté d'intégrer l'aspect probabiliste dans l'environnement *Loco* [CFM09, CF11] nous a poussé à utiliser *Alea* [APM09], une bibliothèque conçue en *Coq* par P. Audebaud et C. Paulin pour formellement analyser les algorithmes probabilistes.

Dans leur article [APM09], les auteurs définissent un langage basique pour exprimer simplement les constructions probabilistes, appelé $\mathcal{R}ml$, qui est un langage fonctionnel basé sur *ML* [MTH90]. Les auteurs y prouvent la validité de la transformation d'expressions de $\mathcal{R}ml$ à *Coq*. Nous utilisons également ce langage pour décrire intuitivement nos algorithmes. Cependant, pour exprimer l'algorithme du rendez-vous en $\mathcal{R}ml$, nous avons besoin d'étendre ce langage en autorisant l'utilisation de types de données discrets : des listes et des fonctions finies.

4.1. $\mathcal{R}ml$

$\mathcal{R}ml$ est un langage basique contenant des constructions semblables à *ML* (variables, constantes, conditionnelles, liaisons locales, applications, abstractions typées, déclarations de fonctions et de fonctions récursives), mais aussi des primitives probabilistes comme `random` ou `flip`. La primitive `flip` retourne 0 ou 1 avec probabilité 1/2 et `random(`n`)` retourne un entier entre 0 et n avec probabilité $\frac{1}{n+1}$. Des appels de ces fonctions probabilistes sont indépendants les uns des autres (si les arguments passés ne sont pas liés entre eux).

Les types des expressions sont restreints à des types simples : les entiers (`nat`), les booléens (`bool`), les réels entre 0 et 1 ($[0,1]$), et les types fonctionnels $\tau_1 \to \tau_2$.

4.2. Extension de $\mathcal{R}ml$

Nous avons vu que notre modèle nécessite la structure d'un graphe et la structure de fonctions d'étiquetage sur un graphe. Nous avons aussi vu que nous manipulerons des ensembles (sommets ou arêtes). Une énumération d'un ensemble est une séquence d'éléments de cet ensemble sans répétition, il peut être représenté par une liste.

Ainsi, nous étendons le système de type avec des listes (`list`, `enum`) et avec des fonctions finies (`State`). Les notations relatives à la fonction d'étiquetage de type `State` introduites dans la Section 2 font partie de cette extension de $\mathcal{R}ml$.

4.3. L'algorithme du rendez-vous en $\mathcal{R}ml$

L'algorithme papier est écrit dans un modèle par passage de messages. Nous choisissons de représenter le passage de messages par un étiquetage des ports : une fonction finie qui associe un booléen à chaque port. Un étiquetage donne l'état du graphe, nous nommons le type d'un tel étiquetage `State`. L'algorithme du rendez-vous peut être simplement exprimé comme une transformation d'états.

Soit $G = (V, E)$ un graphe et σ un étiquetage de port de G. La Figure 4 présente la traduction de l'algorithme papier en $\mathcal{R}ml$. La fonction globale `Fhs` consiste à exécuter une ronde `Fround` dans laquelle l'algorithme local `Fhs_local` est appliqué sur la liste des sommets du graphe. L'algorithme local `Fhs_local` appliqué à un sommet consiste à modifier la zone d'influence de v selon ses choix dans la fonction d'étiquetage. La fonction `write_ports` est utilisée par un sommet v pour envoyer son choix à son voisinage. Ainsi, l'état généré, `(write_ports` v k σ`)`, est obtenu à partir de σ en modifiant la valeur des ports liés à v à $\mathbf{0}$, sauf pour le port (v, k) mis à $\mathbf{1}$.

```
(* write_ports : V → nat → State → State *)
     let write_ports v k σ =
       let rec aux i σ'=
         if i > d(v) then σ'
         else if (i=k) then aux (i+1) (σ'[(v,i) ←1])
         else aux (i+1) (σ'[(v,i) ←0])
       in aux 0 σ

(* Fhs_local : V → State → State *)
     let Fhs_local v σ =
       let k = random (d v)-1 in
         write_ports v k σ

(* Fround : list V → State → State *)
     let rec Fround sV σ =
       if (null sV) then σ
       else let r = Fround (tail sV) σ in
         Fhs_local (head sV) r

(* Fhs : graph →State → State *)
     let Fhs (V,E) σ =
       Fround (enum V) σ
```

FIGURE 4 – L'algorithme en $\mathcal{R}ml$

5. Cohérence de l'algorithme du rendez-vous en $\mathcal{R}ml$

5.1. Aspect distribué

Dans notre cas, nous considérons une seule ronde, une seule étape de calcul pour chaque sommet. On ne prend pas en compte la phase de réception de messages des voisins. L'algorithme simule donc l'envoi de message en mettant à jour pour chacun des sommets l'état σ.

La fonction simulant l'envoi de messages d'un sommet à son voisinage, `write_ports`, est déterministe et opère seulement sur les ports liés à v (zone d'influence de v) :

$$\forall w\ j,\ v \neq w \Rightarrow (\texttt{write\_ports}\ v\ k\ \sigma)(w, j) = \sigma(w, j)$$

Comme les zones d'influence des sommets sont deux à deux disjointes (les réétiquetages ne se chevauchent pas), deux applications de cette fonction, chacune sur un sommet différent, commutent. Il est équivalent d'appliquer cette fonction d'abord à un sommet v puis à un sommet w ou vice-versa.

Lemme 1. *(labelling.update_Pcomm)*
Si $v \neq w$ *alors*

$$(\textit{write\_ports}\ v\ k\ (\textit{write\_ports}\ w\ j\ \sigma)) = (\textit{write\_ports}\ w\ j\ (\textit{write\_ports}\ v\ k\ \sigma))$$

La fonction `Fhs` dépend formellement de l'implantion de la fonction `enum`. `Fhs` décrit d'une façon séquentielle, l'application d'une fonction locale simultanément sur tous les sommets. En effet, notre système est distribué, cela implique que plusieurs sommets peuvent réétiqueter leur voisinage en même temps. Cependant, afin de pouvoir raisonner sur l'algorithme, nous souhaitons le séquentialiser. Pour ce faire, nous montrons que l'ordre dans lequel les sommets exécutent l'algorithme local n'importe pas. Cette propriété est assurée grâce à la commutativité de la fonction `write_ports`. Cela assure, les zones d'influence étant disjointes deux à deux, que le résultat sera toujours le même que celui obtenu si les sommets avaient exécuté cet algorithme en même temps.

Lemme 2. *(FGlobalCommute3)*
Soit lv *une liste de sommets de* G. *Soit* lv' *une permutation de* lv, *nous avons :*

$$\textit{Fround}\ lv\ \sigma = \textit{Fround}\ lv'\ \sigma$$

Nous pouvons remarquer que chaque exécution de `Fhs` nécessite n appels de `random`. Les appels de `random` sont indépendants les uns des autres.

5.2. Aspect probabiliste

Différentes évaluations d'une expression probabiliste e mène à différentes valeurs. L'expression probabiliste e représente donc un ensemble de valeurs. Afin de pouvoir raisonner sur de telles expressions dans un langage fonctionnel, une solution consiste à étudier la distribution de cette expression.

Dans *Alea*, une expression probabiliste $(e : \tau)$ est vue comme une distribution de type $(\tau \to [0, 1]) \to [0, 1]$. Ce type sera nommé par la suite $M\tau$. Nous utilisons la notation $[e]$ pour représenter la mesure associée à l'expression e.

Supposons connue la distribution $[e]$ d'une expression probabiliste $(e : \tau)$ et soit f une fonction de type $([0, 1] \to [0, 1])$, $[e]f$ représente la somme $\sum_{\omega \in \tau}([e]\omega) f(\omega)$.

Définition 4. *(Probabilité)*

Considérons une propriété Q, *dont la fonction caractéristique est notée* $\mathbb{1}_Q$. *La probabilité pour le résultat de l'expression* e *de satisfaire* Q *est représentée, à l'aide de la fonction caractéristique de* Q, *par* $[e]\mathbb{1}_Q$.

Dans *Alea*, les auteurs utilisent une interprétation monadique de $\mathcal{R}ml$ afin de pouvoir raisonner en *Coq* sur les algorithmes probabilistes. Deux opérateurs sont introduits :

- `unit:` $\tau \to M\tau$
  $=$ `fun` $(x : \tau) \Rightarrow$ `fun` $(f : \tau \to [0,1]) \Rightarrow f\ x$
- `bind:` $M\tau \to (\tau \to M\sigma) \to M\sigma$
  `=fun` $(\mu : M\tau) \Rightarrow fun(M : \tau \to M\sigma) \Rightarrow$
  `fun` $(f : \sigma \to [0,1]) \Rightarrow \mu($`fun`$(x : \tau) \Rightarrow M\ x\ f)$

Ci-dessous sont présentés quelques exemples de traductions de $\mathcal{R}ml$ sous forme de distribution (pour plus de détails, voir [APM09]) :

- `[random]` n `: Mnat`
  `=fun` $(f$`:nat` $\to$ `[0,1])` $\Rightarrow \sum_{i=0}^{n} \frac{1}{1+n}(f\ i)$
- $[v] : \mathrm{M}\beta$ v variable ou constant
  `unit` v
- $[$`let` $x = a$ `in` $b]$
  `bind` $[a]$ `(fun` $x \Rightarrow$ $[b]$`)`

Les fonctions présentées en Section 5 sont traduites de façon à ce que leur résultat soit une distribution. Comme la fonction `write_ports` n'utilise pas de primitive probabiliste, la traduction ne la modifie pas.

```
(* Mhs_local : V → State → MState *)
     let Mhs_local v σ =
       bind Mrandom (d v)-1
         fun k ⇒ unit (write_ports v k σ)

(* Mround : list V → State → MState *)
     let rec Mround sV σ
       if (null sV) then unit σ
       else bind Mround (tail sV) σ
         fun r ⇒ Mhs_local (head sV) r

(* Mhs : graph → State → MState *)
     let Mhs (V,E) σ=
       Mround (enum V) σ
```

FIGURE 5 – L'algorithme sous forme de distribution

La plupart des preuves en *Coq* sont basées sur des transformations du programme telles que :

Lemme 3. *(Munit_simpl [APM09]).* $\forall\ (P : \tau)\ (f : \tau \to [0,1]),\ [P]f = (f\ P)$

Lemme 4. *(Mlet_simpl [APM09])*
$\forall\ (P\ Q : \tau)\ (f : \tau \to [0,1]),\ [$`let` $x = P$ `in` $(Q\ x)]f = [P](\mathbf{fun}\ x \Rightarrow [Q\ x]f)$

6. Preuve formelle en *Coq* de l'algorithme

Nous étudions la probabilité de succès, c'est-à-dire la probabilité d'obtenir au moins un rendez-vous dans un graphe $G = (V, E)$ après une exécution de notre algorithme. Nous voulons prouver le Théorème 1 : cette probabilité est supérieure à $(1 - e^{-1/2})$.

Ce résultat a été prouvé en Coq. Le lecteur est invité à se rapporter à [CFZ] pour les détails des preuve en *Coq*.

Théorème 2. *(handshake.Fhs_ deg).* $\forall\ G\ \sigma,\ [Fhs\ G\ \sigma]\ (\exists\ e,\ \mathcal{H}(e)) \geq 1 - e^{-1/2}$

Par la suite, nous utiliserons $\mathbb{P}$ comme une abréviation de la distribution $[Fhs\ G\ \sigma]$. Nous énonçons dans un premier temps les techniques de preuves utilisées pour arriver à ce résultat, puis détaillons les étapes essentielles de la preuve du Théorème 1.

6.1. Techniques de preuve

La fonction `Fround` opère séquentiellement sur un ensemble de sommets, plus précisément, c'est une application de la fonction locale `Fhs_local` sur une séquence de sommets : `Fround` $(v_1 :: v_2 :: \ldots :: v_n)$ revient à exécuter `Fhs_local` v_1 `(Fhs_local` v_2 `(..Fhs_local` $v_n\sigma$`)..)`.

La fonction `Fhs_local` est locale, c'est-à-dire qu'elle affecte seulement un sommet et ses ports relatifs. De plus, elle contient des primitives probabilistes dont les appels sont indépendants les uns des autres.

Nous avons développé les techniques de preuves suivantes : commutativité, composition, résultat local transformé en global, probabilité non nulle.

6.1.1. Commutativité

Tous les sommets effectuent les mêmes tâches en simultané. Leurs actions affectent leurs ports. Les zones d'influence des sommets sont disjointes deux à deux. Par conséquent, il n'y a pas de différence s'ils exécutent leurs tâches les uns après les autres. Pour simuler un algorithme distribué, nous pouvons le décrire comme un algorithme local appliqué à chaque sommet l'un après l'autre. Pour garantir l'aspect distribué, nous avons besoin de prouver que l'ordre n'est pas important.

Nous avons prouvé (voir Lemme 2) que pour toutes séquences s_1 et s_2 telles que s_2 est une permutation de s_1, `(Fround` s_1`)` a la même sortie que `(Fround` s_2`)`.

Ainsi, si l'on considère l'étiquetage σ et la séquence $(v :: s)$ où v est un sommet et s une suite de sommet où v n'apparaît pas, cela revient au même d'incorporer le résultat de la fonction locale appliquée à v dans `Fround` $s\ \sigma$ que d'incorporer le résultat de `Fround` $s\ \sigma$ dans le résultat de la fonction locale appliquer à v. Plus formellement :

Lemme 5. *(rdaTool.FGlobalcons2)* $\forall\ s\ \sigma$,

```
Fround s σ =
  if (null s) then σ
    else let l = Fhs_local (head s) σ in
      Fround (tail s) l
```

La preuve de ce résultat repose sur la discrétisation de la mesure de la fonction locale `Fhs_local`, c'est-à-dire sa réécriture en une somme finie. La mesure `[Fhs_local]` est discrétisable tout comme celle de `[random]` (voir la définition de `[random]` en Section 5).

6.1.2. Composition

Une façon de prouver des propriétés sur `Fround` est de procéder par récurrence sur la séquence de sommets. Par exemple, nous avons prouvé que la fonction termine avec probabilité 1 :

Lemme 6. *(handshake.Fhs_ total).* $\forall\ \sigma\ s$, `[Fround` $s\ \sigma$`]𝟙 = 1`

Démonstration. Par récurrence sur s. Supposons que la propriété est vérifiée pour s', nous montrons qu'elle est vérifiée pour $s = (v :: s')$, c'est-à-dire :

$$\forall\ \sigma\ s'\ v,\ \texttt{[Fround}\ v :: s'\ \sigma\texttt{]}\mathbb{1} = 1$$

En utilisant le Lemme 5, cette expression devient :

$$\forall\ \sigma\ s'\ v,\ \texttt{[let x = (Fhs\_local } v\ \sigma\texttt{) in (Fround } s'\ \texttt{x)]}\mathbb{1} = 1$$

La transformation du Lemme 4 nous donne :

$$\forall\ \sigma\ s'\ v,\ \texttt{[Fhs\_local } v\ \sigma\texttt{] (fun x} \Rightarrow \texttt{ [Fround } s'\ \texttt{x]}\mathbb{1}\texttt{)} = 1$$

Or, par hypothèse, `(Fround` $s'\ \sigma$`)` termine. Comme la définition de la fonction indicatrice $\mathbb{1}$ est `(fun` $x \Rightarrow 1$), il nous faut montrer que :

$$\forall\ \sigma\ v,\ \texttt{[Fhs\_local } v\ \sigma\texttt{]}\mathbb{1} = 1$$

Ce qui veut dire que `Fhs_local` termine. Or `Fhs_local` est définie à partir de `random` et `random` termine. D'où le résultat. □

Une technique générale ressort de cette preuve. L'expression peut se décomposer en la mesure pour un sommet et en celle sur le reste. Ainsi, si nous voulons prouver une propriété au sujet d'un sommet v, nous pouvons utiliser cette technique. Prenons pour exemple $P(v, w)$ comme étant la propriété *"v choisit w"*. On note sV la liste des sommets du graphe.

Lemme 7. *(handshake.Fhs_ degv_ global).* $\forall\ G\ \sigma\ \{v, w\}$, `[Fround` $sV\ \sigma$`]`$\mathbb{1}_{P(v,w)} = 1/d(v)$.

Démonstration. Nous avons prouvé que l'ordre n'est pas important (Lemme 2). Notons $(sV \backslash v)$ la sous-liste de sV où l'élément v a été retiré. Nous pouvons écrire :

$$\forall\ G\ \sigma\ \{v, w\},\ \texttt{[Fround (}v :: \texttt{(}sV \backslash v\texttt{))}\ \sigma\texttt{]}\mathbb{1}_{P(v,w)} = 1/d(v).$$

Nous pouvons ensuite appliquer la technique de composition. □

6.1.3. Résultat local transformé en résultat global

Sous certaines hypothèses, il est possible de reporter un résultat local en un résultat global :

Lemme 8. *(rdaTool.FLocalGlobal)*
Soit v un sommet et $x \in [0, 1]$, si :
- *`Fhs_local` est discrétisable,*
- *`Fhs_local` termine,*
- *les zones d'influence de chaque sommet sont disjointes deux à deux,*
- *l'évènement local El se retrouve dans le global Eg,*

alors :

$$[\texttt{Fhs\_local } v\ \sigma]\mathbb{1}_{Elv} = x \Rightarrow [\texttt{Fhs } v\ \sigma]\mathbb{1}_{Egv} = x$$

Comme la fonction `Fhs_local` est discrétisable et termine, et comme chaque sommet peut seulement changer ses ports, il est possible de reporter quelques résultats locaux vers des globaux comme celui-ci :
- Localement : la probabilité pour un sommet v de choisir son $i^{\text{ème}}$ voisin est $1/d(v)$
- Globalement : la probabilité pour v de choisir w est $1/d(v)$.

En effet, le fait pour un sommet de choisir localement un de ses ports est exprimé globalement par le fait qu'il choisisse un de ses voisins. Ainsi, nous avons la même probabilité dans les deux cas.

6.1.4. Probabilité non nulle

La probabilité qu'un évènement se produise dans un algorithme probabiliste est non nulle s'il existe une exécution possible de l'algorithme dans laquelle l'évènement est vérifié. Ainsi, pour montrer qu'une probabilité est non nulle, il nous faut exhiber un témoin.

Lemme 9. *(my_ alea.proba_ not_ null)*
Considérons un algorithme probabiliste A. Soit t un témoin et E un évènement,

$$\textit{si } [A]\mathbb{1}_{.=t} > 0 \quad \textit{et} \quad (E\ t) > 0 \textit{ alors } [A]\mathbb{1}_{(E\ t)} > 0$$

Nous utilisons ce résultat générique pour prouver l'un des lemmes de la section suivante, le Lemme 11.

6.2. Preuve du Théorème 1

La preuve du Théorème 1 a été réalisée en *Coq*. Nous détaillons ci-dessous les étapes essentielles de la preuve.

Démonstration. **Preuve du Théorème 1**
Nous avons :

$$\mathbb{P}(\exists\ e \in E,\ \mathcal{H}(e)) \quad = 1 - \mathbb{P}(\textstyle\prod_{j=1}^{m}\ \overline{\mathcal{H}(e_j)})$$

D'autre part :

$$\begin{aligned}
\mathbb{P}(\textstyle\prod_{j=1}^{m}\ \overline{\mathcal{H}(e_j)}) \quad &= \textstyle\prod_{i=1}^{m}(1 - \mathbb{P}(\mathcal{H}(e_i)|\ \prod_{j=1}^{i-1}\overline{\mathcal{H}(e_j)})) \\
&\leq \textstyle\prod_{i=1}^{m}(1 - \mathbb{P}(\mathcal{H}(e_i))) && \text{(cf. Lemme 10)} \\
&\leq \textstyle\prod_{i=1}^{m}(1 - \frac{\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j))}{m}) \\
&\leq (1 - \frac{\frac{1}{2}}{m})^m && (*) \\
&\leq e^{-\frac{1}{2}}
\end{aligned}$$

Remarque : (*) Cette étape consiste à prouver : $\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j)) \geq \frac{1}{2}$.

Et $\sum_{j=1}^{m}\mathbb{P}(\mathcal{H}(e_j)) = \sum_{j=1}^{m}\frac{1}{d(e_j^1)*d(e_j^2)}$ où $e_j = \{e_j^1, e_j^2\}$.

Le résultat suivant : $\sum_{j=1}^{m}\frac{1}{d(e_j^1)*d(e_j^2)} \geq \frac{1}{2}$ nous mène à la conclusion. Ce dernier résultat résultat reste à prouver dans notre développement. C'est un résultat classique de la théorie des graphes, c'est pourquoi nous avons choisi de remettre sa preuve à plus tard, préférant nous intéresser à l'aspect probabiliste. □

Le Lemme 10 est utilisé pour la seconde étape décrite plus haut.

Lemme 10. *(my_ alea.Mcond_ prodConjBound)*
Soit $\delta(e, e')$ le booléen valant 1 si les arêtes e et e' ne sont pas adjacentes et 0 sinon.
Si les hypothèses suivantes sont vraies :

1. $\forall i \in 1..m,\ \mathbb{P}(\prod_{j=i+1}^{m}\overline{\mathcal{H}(e_j)}) \neq 0$,
2. $\forall i \in 1..m,\ \mathcal{HS}(e_i)$ *et* $\bigwedge_{j=i+1|\delta(e_i,e_j)}^{m}\overline{\mathcal{HS}(e_j)}$ *sont indépendants,*
3. $\forall i \in 1..m, \mathcal{H}(e_i) * \prod_{j=i+1|\delta(e_i,e_j)}^{m}(\overline{\mathcal{H}(e_j)}) = \mathcal{H}(e_i)$

alors pour tout i dans $1..m$ et toute arête e,

$$\mathbb{P}(\mathcal{H}(e)) \leq \mathbb{P}(\mathcal{H}(e)|\prod_{j=i+1}^{m}\overline{\mathcal{H}(e_j)})$$

Démonstration. La preuve de ce lemme repose sur une partition de E en deux sous-ensembles : le premier est composé des arêtes adjacentes à e et le second est composé de celles qui ne le sont pas :

$$\text{soit } A = \prod_{j=i+1|\delta(e_i,e_j)}^{m} \overline{\mathcal{H}(e_j)} \text{ et } B = \prod_{j=i+1|\sim\delta(e_i,e_j)}^{m} \overline{\mathcal{H}(e_j)}.$$

Nous pouvons écrire grâce à l'hypothèse 1. l'expression : $\frac{\mathbb{P}(B)}{\mathbb{P}(A*B)}$. Ensuite, nous avons montré que :

$$1 \leq \frac{\mathbb{P}(B)}{\mathbb{P}(A*B)}$$

L'hypothèse 2. nous mène à :

$$\mathbb{P}(\mathcal{H}(e)) \leq \frac{\mathbb{P}(\mathcal{H}(e)*B)}{\mathbb{P}(A*B)}$$

Finalement, l'hypothèse 3. nous donne le résultat :

$$\mathbb{P}(\mathcal{H}(e)) \leq \frac{\mathbb{P}(\mathcal{H}(e)*A*B)}{\mathbb{P}(A*B)} = \mathbb{P}(\mathcal{H}(e)|\prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)})$$

□

Remarque : L'hypothèse 1. est nécessaire pour éviter une division par zéro. Elle n'était pas mentionnée dans la preuve papier et fut exigée comme obligation de preuve par *Coq* lors du développement.

Lemme 11. *(handshake.hs1)*
Pour tout sous-ensemble $S = \{e_{i+1}, \ldots, e_m\}$ d'arêtes, la probabilité qu'aucun rendez-vous ne se produise dans S n'est pas nulle, c'est-à-dire :

$$\forall i,\ \mathbb{P}(\prod_{j=i+1}^{m} \overline{\mathcal{H}(e_j)}) \neq 0$$

Démonstration. Considérons l'ensemble d'arêtes $E = e_1, \ldots, e_m$.

Pour prouver que cette probabilité n'est pas nulle, nous mettons en évidence un témoin qui est une exécution possible de l'algorithme et dans lequel il n'y a pas de rendez-vous sur les arêtes $e_{i+1}, \ldots, e_m$ (Lemme 9).

A partir de la théorie des graphes [Ros], nous pouvons déduire qu'il est toujours possible de construire une fonction de parenté représentant un arbre enraciné de n'importe quel graphe connexe $G = (V, E)$ tel que la racine soit une extrémité de l'arête e_1. Ce fait relatif à la théorie des graphes est actuellement admis dans notre développement.

Nous supposons donc l'existence de cette fonction de parenté dans notre développement. Nous en extrayons une fonction totale où la racine est associée à l'autre extrémité de l'arête e_1. Cet étiquetage peut être obtenu par notre algorithme. De plus, il assure qu'il n'y aura pas de rendez-vous dans le graphe à part peut-être en e_1 ce qui n'est pas un problème car nous considérons seulement les arêtes e_{i+1}, .., e_m. C'est pourquoi nous avons besoin dans l'hypothèse d'avoir au moins une arête. □

7. Conclusion

Nous avons modélisé les algorithmes distribués probabilistes par la donnée d'un algorithme local probabiliste et d'un réseau représenté par un graphe dont les ports sont étiquetés. Les sommets

représentent les processeurs, les arêtes les liens de communication et l'étiquetage des ports représente les échanges de messages entre les sommets. Cette modélisation nous permet de raisonner en *Coq* sur des algorithmes distribués. L'aspect probabiliste est traité grâce à la bibliothèque *Alea*. Notre première étude concerne le problème du rendez-vous, dont nous avons fait l'analyse et prouvé le résultat principal : le calcul de la probabilité d'avoir au moins un rendez-vous.

Plusieurs algorithmes distribués sont construits sur le même modèle que l'algorithme sur le rendez-vous. Considérons le modèle suivant : les sommets communiquent avec leur voisin par passage de messages et opèrent en rondes. Une ronde est composée pour un sommet de la réception de messages de son voisinage, d'un calcul avec ces informations locales, puis de l'envoi d'informations locales à son voisinage. Dans notre étude particulière, la phase de réception n'intervient pas. De telles fonctions peuvent être modélisées à l'aide d'une fonction globale qui applique sur chacun de ses sommets un algorithme local.

Plusieurs résultats génériques ont été obtenus en *Coq* avec ce modèle et ont été appliqués au problème du rendez-vous : la commutativité, la composition, la transformation d'un résultat local en global, la preuve que la probabilité de certains évènements est non nulle. Le développement de la bibliothèque permettant de raisonner sur ce modèle d'algorithmes distribués se trouve sur la page web [CFZ].

Nos travaux futurs sont orientés vers l'étude du problème du couplage maximal, qui consiste à itérer l'algorithme du rendez-vous de telle façon à obtenir un couplage maximal. Nous souhaitons à terme pouvoir raisonner non seulement sur des algorithmes qui se déroulent en une seule ronde, mais aussi sur ceux qui en ont plusieurs, voire ceux qui ne se terminent pas.

Remerciements

Les auteurs tiennent à remercier Pierre Castéran pour son co-encadrement des travaux présentés dans ce papier.

Ils remercient également Christine Paulin et David Baelde pour les différentes discussions autour de la bibliothèque *Alea*, et Assia Mahboubi pour son aide sur l'utilisation de *ssreflect*.

Références

[Abr10] J.R. Abrial. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[APM09] P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in coq. *Sci. Comput. Program.*, 74(8) :568–589, 2009.

[BCG12] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, and S. Z. Béguelin. Computer-aided cryptographic proofs. In *ITP*, pages 11–27, 2012.

[Can] D. Cansell. A development in Event-B of a distributed algorithm for building a spanning tree. Personnal communication.

[CF] P. Castéran and V. Filou. Loco : A Coq Library on Local Computation Systems. http ://www.labri.fr/∼casteran/Loco.

[CF11] P. Castéran and V. Filou. Tasks, types and tactics for local computation systems. *Studia Informatica Universalis*, 9.1 :39–86, 2011.

[CFM09] P. Castéran, V. Filou, and M. Mosbah. Certifying Distributed Algorithms by Embedding Local Computation Systems in the Coq Proof Assistant. In *Proceedings of Symbolic Computation in Software Science (SCSS 2009)*, Tunisie, 2009.

[CFZ] P. Castéran, A. Fontaine, and A. Zemmari. Rda : A Coq Library on Randomised Distributed Algorithms. http ://www.labri.fr/∼fontaine/RDA.

[Cho95] C.-T. Chou. Mechanical verification of distributed algorithms in higher-order logic. *The Computer Journal*, 38(2) :152–161, 1995.

[CM07] D. Cansell and D. Méry. *Logics of Specification Languages*, chapter The Event-B Modelling Method : Concepts and Case Studies, pages 47–152. Springer Verlag, 2007.

[Dep] Deploy European Community Project, www.event-b.org. *Event-B and the Rodin Platform.*

[Der70] C. Derman. *Finite state Markovian decision processes.* Mathematics in science and engineering. Academic Press, 1970.

[GMT08] G. Gonthier, A. Mahboubi, and E. Tassi. A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA, 2008.

[Hur02] J. Hurd. *Formal Verification of Probabilistic Algorithms.* PhD thesis, University of Cambridge, 2002.

[KNP02] M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism : Probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.

[Lav94] C. Lavault. Orientation of distributed networks : Graph- and group- theoretic modelling. In *SIROCCO*, pages 49–70, 1994.

[MS97] Y. Métivier and E. Sopena. Graph relabelling systems : A general overview. *Computers and Artificial Intelligence*, 16(2), 1997.

[MSZ03] Y. Métivier, N. Saheb, and A. Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.

[MTH90] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* Computational Models of Cognition and Perception Series. MIT Press, 1990.

[Nor04] G. Norman. Analysing randomized distributed algorithms. In *Validation of Stochastic Systems*, pages 384–418, 2004.

[PS95] A. Pogosyants and R. Segala. Formal verification of timed properties for randomized distributed algorithms. In *PODC*, pages 174–183, 1995.

[Ros] K. H. Rosen. Handbook of discrete and combinatorial mathematics.

[Tel00] G. Tel. *Introduction to distributed algorithms.* Cambridge University Press, 2000.