Panorama des modèles et outils de vérification pour les systèmes probabilistes.

Julie Parreaux
julie.parreaux@ens-rennes.fr
Université de Rennes 1, ENS Rennes

Hubert Garavel *
hubert.garavel@inria.fr
INRIA, Univ. Grenoble Alpes

10 juillet 2017

Résumé

Au cours de notre stage de L3, nous avons pris connaissance des principaux outils (plus de soixante-dix) de vérification pour les systèmes probabilistes et stochastiques. Au delà de la découverte de ces outils et des travaux théoriques qui les sous-tendent, notre objectif était de regrouper l'ensemble des modèles et études de cas développés avec ces outils, d'en réaliser une collection exhaustive et ordonnée, et d'effectuer une analyse statistique afin d'en identifier les grandes orientations de ce domaine de recherche. À terme, notre travail devrait permettre de mieux comparer et connecter ces différents outils.

Mots clé analyse probabiliste ; chaîne de Markov ; modèle ; système probabiliste ; système stochastique ; vérification formelle ; vérification quantitative

Table des matières

1 Le « zoo quantitatif » 1
1.1 Les modèles théoriques . . . 2
1.2 Les outils probabilistes et stochastiques . . . 5
1.3 Formats . . . 6

2 Construction d'une collection de modèles 8

3 Classification automatique des modèles 9

4 Statistiques 11

A Statistiques en R 17

B Les études de cas 30
B.1 SLE : Synchronous Leader Election . . . 30
B.2 RDP : Randomized Dining Philosophers . . . 30
B.3 BDP : Birth–Death Process . . . 31
B.4 TQN : Tandem Queuing Network . . . 31
B.5 CPS : Cyclic Server Polling System . . . 32

*Encadrant du stage de L3

Introduction

La vérification formelle de systèmes complexes probabilistes et stochastiques nécessite l'utilisation d'outils logiciels [3, 8, 11]. On assiste, depuis quinze ans, à l'émergence d'un domaine scientifique riche, vaste et hétérogène s'appuyant sur une dizaine de modèles probabilistes et stochastiques théoriques et ayant donné naissance à une multitude d'outils académiques (on en dénombre plus de soixante-dix aujourd'hui). L'outil le plus connu du domaine est PRISM développé par les universités d'Oxford et de Birmingham).

Il existe une forte compétition entre ces différents outils concernant leur expressivité et leurs performances. Cependant, il n'existe pas de cadre standard permettant de comparer ces outils, tel que, par exemple, la compétition « Model Checking Contest [1] » qui a lieu chaque année pour comparer les outils de vérification pour les réseaux de Pétri.

Chacun de ces outils est publié avec des exemples de modèles et des études de cas. Cependant il n'existe pas de format commun à tous les outils. Les différents modèles sont souvent, malheureusement, incompatibles à cause de la non-interopérabilité des formats : le domaine d'expression de ceux-ci ne sont pas toujours compatible. Quelques passerelles existent qui permettent de convertir les modèles ayant des formats qui diffèrent uniquement de leur syntaxe.

Idéalement, on souhaiterait avoir une collection de modèles bien organisée. Une fois constituée, elle permettrait alors de faire des tests de performance entre les outils et servirait de base pour des tests de non-régression lors de l'élaboration des outils.

Nous proposons donc de récupérer un maximum de modèles existant dans le monde. PRISM a déjà une collection de modèles ; nous devons éviter de les dupliquer et récupérer des modèles d'outils moins connus afin de compléter la collection de PRISM. Idéalement, nous souhaitons stocker les modèles sous une forme commune exploitable. On pourra ainsi, dans un premier temps, en faire une analyse statistique avec un aspect masse de données permettant d'en faire une classification. Puis, dans un deuxième temps, les exploiter pour tous les outils.

Dans une première partie, nous allons présenter le domaine des outils probabilistes et stochastiques (section 1). Ensuite nous présenterons comment nous avons construit (section 2) et analysé (section 3) une collection de modèles. Nous terminerons en présentant une vérification de nos résultats ainsi que leurs applications (section 4).

1 Le « zoo quantitatif »

L'expression «zoo quantitatif» qui reflète bien la diversité et le caractère souvent incompatible des modèles et outils, figure dans un article récent de Hartmanns et Hermanns [7]. Nous allons commencer par explorer le «zoo quantitatif». Dans un premier temps, nous étudierons les modèles théoriques en jeu dans ce «zoo quantitatif» (section 1.1). Ensuite, nous présentons les différents outils quantitatifs à notre disposition (section 1.2).

1. `http://mcc.lip6.fr`

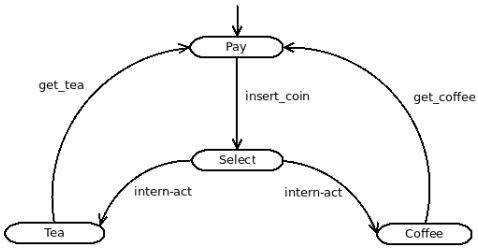


FIGURE 1 – Une distributeur de boissons modélisée par un LTS.

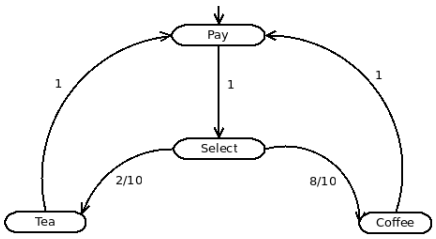


FIGURE 2 – Modélisation d'un distributeur de boissons à l'aide d'une DTMC.

1.1 Les modèles théoriques

Chacun des outils que l'on étudie est basé sur des modèles probabilistes ou stochastiques [2]. Ces modèles existent en grand nombre, nous choisissons d'en présenter seulement les plus utilisés dans les outils listés par Hartmanns et Hermanns [7]. Nous allons commencer par un modèle qui n'est ni probabiliste, ni stochastique, mais qui sert de base à tous les autres.

Les systèmes de transitions étiquetées Un système de transitions étiquetées, que l'on nommera dans la suite LTS (*Labelled Transition System*) est un modèle fondamental pour la modélisation des systèmes concurrents. Ce modèle basé sur la notion d'état et de transition exprime des propriétés fonctionnelles : il est ni probabiliste, ni stochastiques.

Un LTS peut exprimer le choix déterministe ou non-déterministe.Pour modéliser les systèmes on peut utiliser trois choix possibles : les choix déterministes, les choix probabilistes et les choix non-déterministes. Il existe une différence importante entre ces différents choix. Lorsque nous utilisons des choix déterministes, à partir du moment où on a calculé la valeur relative à ce choix, nous connaissons le futur de manière unique. Nous définirons les choix probabilistes qui sont une notion centrale plus loin dans ce rapport. Nous utilisons un choix non-déterministe lorsque nous ne voulons ou nous ne pouvons pas déterminer de manière certaine quelle transition sera franchie. Lorsque nous avons un choix entre deux transitions identiques, l'environnement ne peut pas déterminer quelle transition sera franchie.

Modéliser un distributeur de boissons avec un LTS (Figure 1) permet de modéliser les actions de l'utilisateur sur la machine. En effet, pour déguster une boisson, il fait deux actions distinctes : il insère les pièces de monnaie et il récupère sa boisson. Ce modèle ne formalisant pas le choix de l'utilisateur, on utilise un choix non-déterministe qui est une action interne à la machine. On utilise l'état *select* qui va choisir de manière non-déterministe entre du café ou du thé.

Les choix probabilistes Nous allons maintenant étudier deux modèles qui supportent les choix probabilistes. Pour un choix déterministe, introduire une probabilité n'a pas de sens. Par contre, pour un choix non-déterministe, les possibilités sont à priori équiprobables. Pour être plus précis dans la sélection de la réponse, on ajoute une notion de probabilité pour effectuer ce choix.

Une chaîne de Markov à temps discret, que l'on nommera DTMC (*Discret-time Markov*

2. Dans le «zoo quantitatif», il existe également des modèles hybrides, mais nous n'en parlerons pas dans ce préambule.

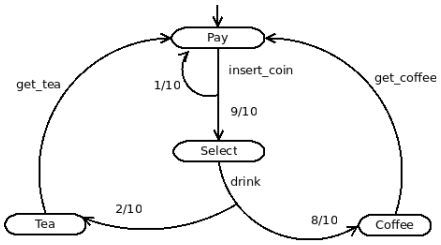


FIGURE 3 – Modélisation d'un distributeur de boissons à l'aide d'un MDP.

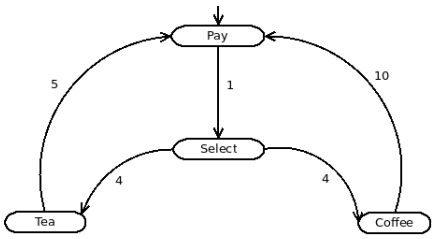


FIGURE 4 – Modélisation d'un distributeur de boissons à l'aide d'une CTMC.

Chains), contient un ensemble d'états et de transitions, chaque transition étant étiquetée par une probabilité. Contrairement aux LTSs, les DTMCs possèdent une contrainte sur le nombre de transitions sortantes d'un état. La somme des probabilités des transitions sortantes (en comptant celles permettant de rester dans l'état courant) doit être égale à 1 pour chacun des états. Nous modélisons un distributeur de boissons à l'aide d'une DTMC dans la figure 2.

Un processus de décision Markovien, que l'on notera MDP (*Markov Decision Process*) est un modèle probabiliste non-déterministe. Il est la combinaison formelle de deux modèles : un LTS pour le non-déterministe et une DTMC pour les probabilités. Un MDP contient un ensemble d'états dans lequel changer d'état revient à résoudre un test probabiliste d'un choix non-déterministe. Nous modélisons un distributeur de boissons à l'aide d'un MDP dans la figure 3. Ils sont utilisés dans le cadre d'optimisation de tâches économiques ou pour de la planification dans l'intelligence artificielle, par exemple.

L'introduction de probabilité dans les modèles permet d'étendre les propriétés que l'on souhaite vérifier, elles deviennent plus riches : quantitatives, par rapport aux propriétés vérifiées par les LTSs. Par exemple, on peut exprimer et vérifier que «tous les états sont accessibles avec une probabilité supérieure à 0.9». Dans le cadre des DTMCs, vérifier une propriété revient à résoudre un système d'équations linéaires pouvant être très volumineux, on met en place notamment des algorithmes d'approximation. La vérification des MDPs demande l'utilisation de la programmation linéaire où les variables représentent ces états. Afin de résoudre efficacement ces problèmes, on utilise des algorithmes d'approximation ceux des LTSs et des DTMCs [7].

Systèmes à temps réel mou Nous sommes maintenant capables de modéliser des systèmes possédant des choix probabilistes. Cependant, ces modèles ne sont pas capables d'exprimer les aspects liés au temps. Les modèles que nous allons présenter maintenant formalisent un temps réel «mou» qui correspond à temps statistique par exemple, le temps d'attente devant une boulangerie.

Une chaîne de Markov à temps continu, que l'on notera CTMC (*Continuous-Time Markov Chain*), est l'équivalent d'une DTMC en temps continu. Pour cela, on utilise la loi exponentielle : chaque transition d'une CTMC est étiquetée par un réel lambda tel que le temps pour franchir cette transition est déterminé par une loi : $t \mapsto exp(-\lambda \times t)$. Attention, ce λ, appelé taux, est attaché à la transition, et non à l'état de départ, car deux transitions avec des taux différents peuvent partir du même état. Les CTMCs vérifient la propriété de Markov, ce qui justifie que nous pouvons uniquement utiliser une loi exponentielle qui est la seule distribution continue sans mémoire. On modélise par une CTMC un distributeur de boissons dans la figure 4).

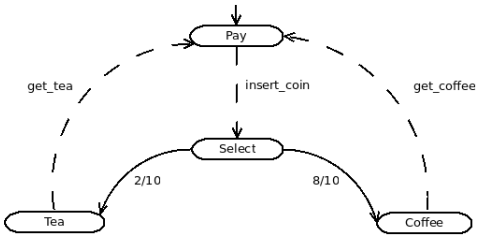


FIGURE 5 – Modélisation d'un distributeur de boissons à l'aide d'une IMC.

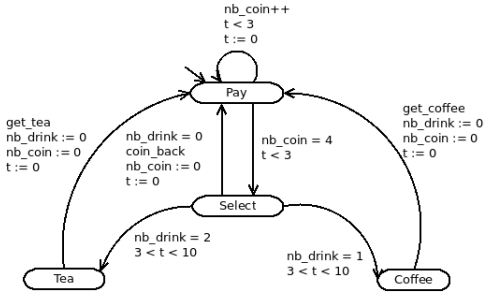


FIGURE 6 – Modélisation d'un distributeur de boissons à l'aide d'un TA étendu.

Une chaîne de Markov interactive, que l'on notera IMC (*Interactive Markov Chain*) est une combinaison d'une CTMC et d'un LTS. Les IMCs ont été introduites pour remédier aux problèmes de la composition parallèle de CTMCs qui nécessite des actions afin d'exprimer la synchronisation. Ce modèle possède donc deux types de transitions : des transitions markoviennes représentées par un taux λ (appartenant aux CTMCs) et des transitions à action (appartenant aux LTSs) comme le montre la modélisation d'un distributeur de boissons (figure 5).

Les CTMCs et IMCs sont des modèles dits «stochastiques» qui sont utilisés dans la biologie ou l'analyse de performance (CTMC) et dans les réseaux de Pétri généralisés, la conception de matériel (IMC). L'analyse des CTMCs peut se faire de manière exacte mais pour des contraintes de temps, on utilise des approximations. Pour l'analyse des IMC, deux grandes méthodes sont utilisées afin de résoudre le problème : l'analyse sur des systèmes d'état stable et transitoire et la sélection de modèle .

Systèmes à temps réel dur Dans les modèles précédents, nous avons introduit une notion de temps réel «mou». Nous allons maintenant nous intéresser à des systèmes à temps réel «dur» qui permettent de représenter l'écoulement du temps. Ils servent notamment à modéliser des situations d'urgence comme «Si je ne suis pas servi dans les trois minutes, je ne reste pas». Ils servent aussi pour modéliser la performance des différents systèmes pour lesquels on peut avoir besoin de connaître le temps d'une action, calculer des délais ou avoir une notion de temps.

Afin de répondre à ce besoin, nous allons utiliser des automates temporisés qui servent très largement à la modélisation de systèmes temps-réel. On les notera TAs (*Time automata*). On considéra qu'un automates temporisé est une extension d'un LTS avec des variables globales, appelées horloges, qui comptabilisent le temps écoulé. Elles donnent un intervalle de temps pendant lequel les transitions sont utilisables. On peut alors modéliser des délais ayant une durée déterminée ou comprise dans un intervalle, ainsi que la notion urgente (l'invariant d'état des TA peut forcer l'exécution de certaines transitions). Un exemple d'un distributeur de boissons est modélisé dans la figure 6.

Les propriétés que l'on souhaite vérifier avec ces modèles peuvent être temporelles. En vérification, on considère que les automates temporisés sont composés d'un nombre d'état indénombrable et d'une infinité de transitions sur lesquelles le temps est modélisée par une étiquette correspondant au délai de franchissement de ces transitions. En 1990 [1, 2, 7], la décidabilité du problème de vérification a été montrée même si très souvent l'ensemble des états explorés à parcourir explose.

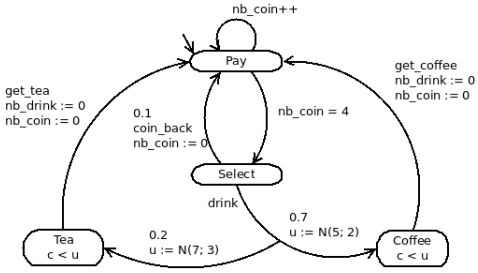


FIGURE 7 – Modélisation d'un distributeur de boissons à l'aide d'un STA.

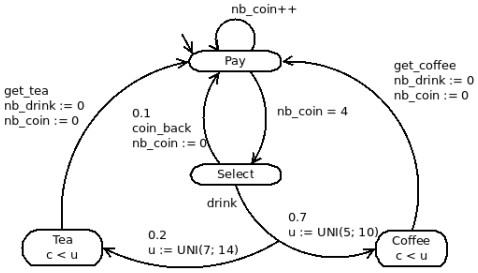


FIGURE 8 – Modélisation d'un distributeur de boissons à l'aide d'un PTA.

Avec les automates temporisés, nous avons introduit la notion de temps. Cependant, dans de tels modèles, nous n'avons pas d'aspects probabilistes. Afin de remédier à ce manque nous allons combiner les propriétés des automates temporisés avec celles des chaînes de Markov afin d'obtenir des automates temporisés probabilistes et/ou stochastiques.

Un automate temporisé probabiliste ou stochastique, que l'on notera respectivement PTA (*Probabilistic Timed Automata*) et STA (*Stochastic Timed Automata*), peut être considéré de deux manières :

— soit comme un TA avec des choix probabilistes ;

— soit comme une chaîne de Markov avec une notion de temps : des variables horloges modélisant le temps.

La différence fondamentale entre ces deux modèles est le choix de la distribution. Dans le cas d'un PTA, nous utilisons une distribution discrète comme la loi uniforme (Figure 8). Lorsqu'on utilise un STA, c'est que nous avons besoin d'une distribution continue comme la loi normale dans le cas de probabilité d'erreur (Figure 7) ou la loi exponentielle dans le cas de la mesure des intervalles de temps.

Dans ce contexte, les propriétés d'accessibilité sont souvent étudiées et vérifiées. Elles s'inscrivent dans des études pratiques de protocoles réseaux. Ceux-ci s'y prêtent bien car les probabilités permettent de modéliser les actions extérieures ou les pannes, le temps modélise la transmission des messages et leur délai ...

1.2 Les outils probabilistes et stochastiques

Nous avons recensé soixante-quatorze outils de vérifications probabilistes et stochastiques. Parmi ces outils, on en a dénombré quarante-huit supportant les aspects probabilistes dont douze sont indisponibles [6]. La liste des outils que nous avons recensés est donnée avec des informations complémentaires sur les modèles théoriques, les logiques et leur viabilité à l'adresse suivante : `cadp.inria.fr/resources/zoo`. Nous allons, cependant, présenter quelques outils dans ce rapport.

CADP[3] est un outil qui permet de vérifier des systèmes concurrents et notamment des protocoles de communication. Il a d'abord été conçu pour vérifier les problèmes décrits par les LTS spécifiés dans le langage LOTOS. Avec ces dernières versions, CADP s'étend au monde probabiliste et peut aujourd'hui supporter des CTMCs, des DTMCs et des IMCs en effectuant une analyse sur des systèmes stables et transitoires. Cependant, il n'existe pas de test de performance face à PRISM. Ces tes étant probablement freiné par

un manque de passerelles entre les formats des deux outils.

Kronos [4] est un outil pour vérifier les systèmes en temps réel. Il est le premier outil à avoir implémenté la vérification de TA de manière efficace.

MRMC [5] est un outil pour vérifier des systèmes basés sur les chaînes de Markov. Il utilise donc des CTMCs qui sont bien représentées dans le monde académique et des DTMCs. Pour vérifier les DTMCs, il implémente les algorithmes d'approximation permettant de résoudre les systèmes d'équations linéaires donnés par ces modèles.

PRISM [6] est l'outil probabiliste historique. C'est un des premiers outils à s'être imposé dans le monde académique. Supportant les DTMCs, les CTMCs, les MDPs depuis sa première version. Depuis sa quatrième version (2010), il supporte les PTAs et les STAs avec quelques restrictions sur le langage basé sur la théorie des jeux. De plus, il sert de point de repère à la communauté : les calculs de performance se font par rapport à PRISM.

SPIN [7] est un outil permettant de vérifier des logiciels *multi-thread* pour des systèmes communiquant à l'aide du langage PROMELA. Il vérifie notamment des LTSs en implémentant à la volée différentes réductions pour ne pas générer l'espace complet des états.

Storm [8] qui est un outil récent et compétitif en termes de performance (la version 1 date de 2017) modélise des systèmes à l'aide de DTMCs, de CTMCs, de MDPs et de MAs. En comparant Storm à PRISM [4], ses concepteurs ont remarqué que le meilleur moteur de Storm est plus rapide que le meilleur moteur de PRISM : aussi efficace que PRISM dans la création du modèle, il est plus rapide lors de la résolution numérique. Cependant, le support des CTMCs de Storm ne passe pas à l'échelle contrairement à PRISM.

UPPAAL [9] est un outil pour vérifier les systèmes en temps réel modélisés par des réseaux de TA. Il les implémente avec des extensions de données ou à l'aide de la notion de coût et de récompense.

Une comparaison systématique des performances des différents outils n'existe pas. Cependant, on voit apparaître des articles [4, 10] qui comparent divers outils à PRISM en termes de performance (temps et mémoire), mais également en termes de confort d'utilisation.

1.3 Formats

Il existe presque autant de formats que d'outils dans le monde académique. En effet, chacun des outils utilise leur propre format qui diffère souvent des autres de quelques détails. On peut classer les formats en deux catégories : les formats haut niveau et les formats bas niveau. La différence entre ces deux catégories est la mise en place d'une abstraction supplémentaire au-dessus de l'automate à l'aide de variables d'état et/ou de parallélisme.

Deux visions existent lorsqu'on souhaite décrire les modèles utilisés dans les outils, selon que l'on se concentre sur les états de l'automate ou sur ses transitions. Dans les deux cas, la structure des transitions est la même : elle est caractérisée par un état de départ, la probabilité correspondant à la transition et l'état d'arrivée. L'approche basée sur les états apporte plus d'informations sur ceux-ci que à l'approche basée sur les transitions qui ne différencie que les

3. `http://cadp.inria.fr`
4. http ://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos
5. http ://www.mrmc-tool.org
6. `http://www.prismmodelchecker.org`
7. `http://spinroot.com/spin/whatispin.html`
8. `http://www.stormchecker.org`
9. http ://uppaal.org

états initiaux. Utiliser des variables d'état, comme le fait PRISM, permet de se concentrer sur les états et leurs caractéristiques décrites par ces variables.

Les outils probabilistes et stochastiques permettent de vérifier des protocoles intrinsèquement concurrents comme le problème du dîner des philosophes (annexe B). Certains formats, comme le format de PRISM, formalisent ce parallélisme dans leur langage. Le modèle sera alors séquentialisé par le compilateur afin d'éliminer le parallélisme puisque ces outils ne travaillent pas directement sur le modèle concurrent. Utiliser la parallélisation permet d'obtenir des fichiers de taille raisonnable puisque toutes les transitions ne sont pas explicitées. En effet, lorsque que l'on introduit de la concurrence, l'ensemble des états et des transitions possibles augmente rapidement et la parallélisation nous permet de ne pas toutes les expliciter. Le modèle PRISM ci-dessous illustre ce point sur l'exemple du dîner des philosophes (annexe B) avec trois processus s'exécutant en parallèle.

```
module phil1
p1: [0..11];
[] p1=0 −> (p1'=1); // trying
...
[] p1=11 −> (p1'=0); // put down left and return to think
endmodule // one process

// construct further modules through renaming
module phil2 = phil1 [ p1=p2, p2=p3, p3=p1 ] endmodule // two processes
module phil3 = phil1 [ p1=p3, p2=p1, p3=p2 ] endmodule // three processes
```

Pour classer nos formats, nous utilisons deux catégories : les formats de haut niveau acceptant les variables d'état ou le parallélisme et les formats de bas niveau n'acceptant aucune de ces abstractions. Une différence essentielle entre l'utilisation d'un format de haut niveau et de bas niveau réside dans le confort de l'utilisateur. Un fichier dans un format de haut niveau est souvent compréhensible plus rapidement pour l'utilisateur et plus compacte. Ce choix n'influe pas sur l'expressivité du modèle puisque les compilateurs traduisent les modèles haut niveau en modèles bas niveau avant de les vérifier. Les fichiers PRISM et CADP ci-dessous modélisent un système de vote avec un serveur central (annexe B) en haut niveau (PRISM) ou en bas niveau (CADP).

```
module server

//State variables
s : [1..2]; // station
a : [0..1]; // action: 0=polling, 1=serving

[loop1a] (s=1)&(a=0) −> gamma : (s'=s+1);
...
[serve2] (s=2)&(a=1) −> mu  : (s'=1)&(a'=0);
endmodule
```

PRISM

```
des (0, 62754, 35471)
(2, i, 19766)
(2, i, 19637)
(27659, "prob␣0.9", 19638)
(27659, "prob␣0.1", 18605)
 ...
(27656, i, 19726)
(6, "rate␣3", 27656)
(6, "rate␣5", 27657)
(6, "rate␣2", 27658)
```

CADP

2 Construction d'une collection de modèles

Pour la vérification des modèles LTS, il existe une base de modèles appelée VLTS [10] qui est utilisée dans le monde académique ; plus de quarante articles la citent. Pour les modèles quantitatifs seule la base de PRISM [11] est disponible, mais elle ne suffit pas. En effet, pour citer un grand scientifique de la vérification quantitative : «*Currently the PRISM benchmark suite is the de facto standard used to compare probabilistic model checkers or to show the efficiency of analysis algorithms for Markov models. This is rather unfortunate. The PRISM benchmark suite is tailored to the usage of the PRISM model checker. Preferably, a much broader benchmark suite is needed to obtain a fair comparison.*» [12]. L'objectif de ce stage était de réaliser une collection des différents modèles pour les outils probabilistes et stochastiques existant dans le monde.

Notre premier travail a été de collecter tous les modèles que l'on pouvait trouver pour chacun des outils probabilistes et stochastiques existants. Pour ce faire, nous avons pris la liste d'outils effectuée par Jean-Philippe Gros dans son mémoire de master [6] et pour chacun de ses outils, nous avons cherché des exemples (modèles ou formules de logique temporelle) sur les sites web et/ou dans les publications relatifs à ces outils.

Nous avons récupéré plus de 20 000 modèles que nous avons classés par outil. Pour chacun des outils nous avons créé un dossier portant le nom de l'outil et contenant un fichier « INFO.odt » et trois sous-dossier :

Biblio contenant la bibliographie relative à l'outil et à ses modèles ;

Models contenant les modèles ;

Tool contenant l'outil si celui-ci est disponible.

Le fichier « INFO.odt » recense les URL où les modèles ont été téléchargés, les noms des différents modèles trouvés, si nous possédons l'outil et sous quelle forme nous le stockons. Dans le dossier **Models**, nous avons rangé les modèles comme nous les avions trouvés, souvent en fonction de la famille du modèle. Lorsque la description du modèle : contexte, paramètres, limites, statistiques, bibliographie, était disponible, nous avons placé ses informations dans un fichier INFO relatif à la famille du modèle. En plus de la documentation fournie, nous avons déterminé cinq caractéristiques que nous souhaitons analyser pour ces modèles. Celles-ci sont les modèles théoriques, le nombre de variables d'état, le nombre de processus s'exécutant en parallèle, le format du fichier et le (ou les) outil(s) supportant ces fichiers.

Après avoir récupérés et ordonné ces fichiers dans notre collection, nous voulons savoir combien de modèles nous avons récupéré. Le principal problème pour compter nos modèles est la présence de fichiers multiples. En effet, certains outils utilisent des modèles sur plusieurs fichiers possédant des formats différents. La solution retenue a été de compter les fichiers d'un seul format parmi ceux construisant les modèles. Afin de faciliter le calcul, nous avons utilisé le programme **guess** (voir section 3) que nous avons écrit dans le cadre de ce stage. Pour tous les formats qui ne nous servent pas au calcul ; nous avons indiqué «*auxiliary*» après le modèle. Une fois que le programme a analysé tous les fichiers, pour connaître le nombre de modèle, on garde les fichiers identifiés comme modèle desquels on supprime ceux contenant le mot clé «*auxiliary*».

10. `cadp.inria.fr/resources/vlts/`
11. `http://www.prismmodelchecker.org/casestudies/index.php`

3 Classification automatique des modèles

Nous souhaitons donner des informations sur les fichiers aux utilisateurs et ainsi répondre à la question «c'est quoi ce fichier ?». Notre objectif est d'aider un utilisateur novice qui ne connaît pas l'ensemble du panorama. Nous avons décidé de donner des informations qui ne dépendent pas du nom du fichier mais plutôt des caractéristiques du modèle que l'on retrouve dans les fichiers. À la vue du nombre de fichiers, plus de 20 000, nous n'avons pas le temps de déterminer les caractéristiques de ces fichiers un à un. Il nous faut donc un outil automatisé qui soit capable de déterminer les caractéristiques de chacun des fichiers collectés. Nous allons donc écrire un programme nommé **guess** qui nous permettra de faire cette analyse.

Fonctionnalité du programme guess Le programme **guess** permet de calculer les caractéristiques liées au fichier. On souhaiterait inscrire cette analyse dans l'esprit de la commande Unix **file** qui devine le type (texte, image, exécutable, code, ...) du fichier passé en paramètre. Dans ce but, à chacun des fichiers passés en paramètre de **guess**, nous donnons le type du fichier : un modèle, une formule logique ou un autre format (« *unknown* »). Lorsque nous sommes en présence d'un modèle, nous allons plus loin en donnant les caractéristiques du modèle.

Dans le cas où le fichier est un modèle, nous retournons ces caractéristiques sous forme de cinq champs présentés ci-dessous :

Models indique le modèle théorique utilisé. Nous sommes capables de déterminer les modèles suivants : CTMC, DTMC, HA [12], IMC, IPC [13], LTS, MA [14], MC [15], MDP, MM [16], PA [17], PEPA [18], PHA, PN [19], PTA, SMG [20], SPN [21] et TA. Pour certains fichiers comme ceux de PRISM, nous détectons des incohérences entre le modèle et le format et nous les signalons à l'aide du mot clé «*warning*» ;

Var indique le nombre de variables d'état utilisées dans le fichier. Nous retournons 0 si le fichier en contient aucune ;

Par indique le nombre de processus s'exécutant en parallèle dans le fichier. Nous retournons 1 si le fichier contient un modèle séquentiel ;

Format indique le format du fichier. C'est lui qui nous donne le type du fichier et qui nous permet de reconnaître les fichiers contenant des formules logiques ou d'autres formats ;

Tool donne la liste des outils utilisant ce fichier.

Pour chacun de ces champs, lorsque nous ne pouvons pas déterminer de réponse par manque d'informations pour conclure, nous retournons la chaîne de caractère «*unknown*».

En plus de donner ces informations, notre programme est capable de gérer des options permettant de sélectionner les informations que souhaitent l'utilisateur ou de réaliser des contrôles.

— -models affiche le modèle modélisé par le fichier.

12. Automates hybrides, *Hybrid Automata*
13. Chaîne probabiliste interactive, *Interactive Probabistic Chain*
14. Automate de Markov, *Markov Automaton*
15. Chaîne de Markov, *Markov Chain*
16. Modèle de Markov, *Markov Model*
17. Automate probabiliste, *Probabilistic Automaton*
18. Evaluation des performances des processus algébrique, *Performance Evaluation Process Algebra*
19. Réseau de Pétri, *Petri Net*
20. Jeu multijoueur stochastique *Stochastic Multiplayer Game*
21. Réseau de Pétri stochastique, *Petri Net Stochastic*

— -var affiche le nombre de variables d'état.

— -par affiche le nombre de processus s'exécutant en parallèle.

— -format affiche le format du fichier.

— -tool affiche le ou les outil(s) traitant ce fichier.

— -check-name affiche s'il existe une incohérence entre le modèle que l'on prédit et le nom du fichier.

Implémentation de guess Nous voulons réaliser rapidement un prototype fonctionnel de notre fonction **guess** retournant un résultat éventuellement non complet dans une limite satisfaisante.

Dans la recherche des caractéristiques, nous avons défini trois grandes règles permettant de chercher l'information pour la majorité des fichiers. Ces caractéristiques sont données par le format et le contenu de notre fichier. Nous n'utilisons pas son nom car nous souhaitons obtenir un script robuste face aux renommages de fichier.

Règle 1 Elle consiste à tester la présence d'un mot clé. Par exemple, un fichier contenant le mot clé « dtmc » est susceptible d'être une chaîne de Markov à temps discret encodée dans le format de PRISM. Il existe deux variantes de cette règle. Si le mot clé est unique, la commande «**grep -c**» permet de tester sa présence. Si le mot clé est dans une liste de possible alors il nous faut tester chacun des mots clés de cette liste. Dans ce cas, nous avons généralement une idée d'où ce mot clé est situé et nous effectuons un pré-traitement, avec une règle de type 3, afin de limiter la zone de recherche.

Règle 2 Elle consiste à compter le nombre de mots clés présents dans le document. Par exemple, elle permet de compter le nombre de « *endmodule* » dans un fichier PRISM symbolisant le nombre de processus parallèles. Elle est souvent associée à une règle de type 3 en pré-traitement, pour éviter de polluer le dénombrement surtout lorsque les mots clés étaient petits et localisés à un seul endroit du fichier. La règle 2 compte à l'aide de la commande «**grep -c**»| ou avec **wc** permettant de compter le nombre de lignes ou de caractères.

Règle 3 Elle permet de découper le fichier afin de gagner en précision. En effet, un inconvénient des règles précédentes est que le fichier peut contenir un commentaire dans lequel ce mot-clé figure ; pour être plus précis sur la prédiction, on peut rechercher le mot-clé à des endroits déterminés du fichier, par exemple, sur la première ligne ou dans un préambule. Lorsque la section est déterminée par des mots clés, nous utilisons alors la commande **awk** pour couper le fichier. Par exemple, dans les fichiers de PRISM, nous séparons les différents processus entre les mots clés « *module* » et « *endmodule* » afin de compter les variables d'états.

Les règles que nous appliquons sont déterminées par le format du fichier et sa documentation. En fonction du fichier, nous pouvons à l'aide d'une ou plusieurs de ces règles cibler ou élargir notre recherche.

Ces règles, qui s'appliquent à la majorité des fichiers, ne sont cependant pas utilisées dans le cadre des fichiers issus de CADP (les formats .aut et .bcg). Les formats bcg sont des binaires, on va donc utiliser la commande « **bcg_info -labels** » qui va extraire la liste des étiquettes des transitions sans doublons. Afin de connaître le modèle représenté, nous devons étudier les étiquettes. Celles-ci peuvent être caractérisées en cinq catégories, l'étiquette est de la forme :

1. "prob 0.15" ;

2. "rate 14" ;
3. "xxx ; prob 0.15" où xxx est une étiquette d'action ni probabiliste, ni stochastique ;
4. "xxx ; rate 14" où xxx est une étiquette d'action ni probabiliste, ni stochastique ;
5. xxx, c'est-à-dire étiquette d'action ni probabiliste, ni stochastique.

Ces types de transitions nous permettent de définir le modèle implémenté à l'aide de ces équivalences :

— une DTMC contient uniquement des étiquettes de type 1 ;
— une CTMC contient uniquement des étiquettes de type 2 ;
— un MDP contient des étiquettes de type 1 et 3 ou des étiquettes de type 5 ;
— une IPC contient des étiquettes de types 1 et 5 ;
— une IMC contient des étiquettes de type 2 et 5 ;
— les autres combinaisons ne sont pas reconnues.

Pour les fichiers aut, une conversion au format bcg avec la commande « **bcg_io** », avant d'utiliser la même technique que pour les bcg, nous permet d'obtenir des résultats plus précis que ceux donnés par nos règles.

Notre prototype a été implémenté en 2804 lignes de code avec les outils traditionnels d'Unix : **shell**, **awk** et de la commande **grep**. Cependant, nous avons choisi ces outils car ils étaient disponibles mais nous aurions pu utiliser R, Tcl/Tk, Perl ou encore Python.

Limites de guess Ce programme est une première version d'un catalogue de règles qui donne l'impression de l'intelligence. Mais ce ne sont que des approximations que nous calculons. Elles sont calculées sur la présence ou non de mots clés dans le langage ou/et les commentaires.

Une première limite est dans notre capacité à analyser certains fichiers. Pour certains formats, un manque de documentation ne nous permet pas de connaître la présence ou non de variable d'état et/ou de parallélisme. De plus, lorsque l'information contenant le type du modèle théorique n'est pas donnée par le langage directement (sous forme de mot clé), nous utilisons uniquement les commentaires des utilisateurs.

Une deuxième limite est dans notre analyse de la conformité des fichiers. On ne s'intéresse jamais à la correction des probabilités dans le fichier. Dans l'état actuel du programme, on est incapable de détecter un fichier dans lequel la somme des probabilités sortantes d'un des états est supérieure à 1. Pour détecter les erreurs des modèles ou simplement déterminer le modèle pour certain format, il faudrait écrire un programme en **C** ou en utilisant les outils. Mais cette écriture arrive dans un second temps dans notre projet et n'a pas eu le temps d'être traitée.

Une troisième limite est dans le traitement des fichiers multiples, certains fichiers contiennent des informations pour le modèle (comme le modèle théorique). Cependant, nous ne transmettons pas l'information aux autres fichiers, ce qui ne permet pas de vérifier toutes les incohérences. Enfin, on n'est pas capable de savoir si tous ces fichiers sont bien présents.

4 Statistiques

Pendant ce stage, nous avons réalisé une collection de modèles et analysé les différents fichiers récupérés. Ce travail va nous permettre d'obtenir des statistiques détaillées sur cette

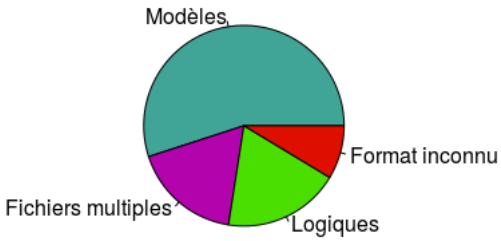


FIGURE 9 – Répartition des fichiers en fonction de leur type.

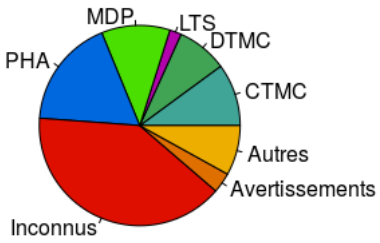


FIGURE 10 – Répartition des différents modèles théoriques.

collection de modèles. Nous avons identifié plusieurs questions intéressantes :

— Combien de modèles avons-nous récupérés au total ?

— Comment sont répartis les modèles théoriques ?

— Comment sont répartis les modèles appartenant aux différents outils ?

— Existe-t-il des familles/souches d'études de cas, par exemple, les problèmes des philosophes qui dînent, les algorithmes d'exclusion mutuelle, les protocoles "tree root", ... ?

— Pour faire suite à la remarque de J.P. Katoen mentionnée section 2 [12], on peut légitimement se demander quelle est, dans le monde, la proportion de modèles qui ne dérivent pas de la bases des exemples de l'outil PRISM, s'il existe d'autres sources de modèles et quelle est leur importance quantitative.

Afin de répondre à ces questions, nous avons utilisé l'outil **guess** pour produire des statistiques, que nous avons ensuite formatées pour qu'elles puissent être analysées avec l'outil R [22]. Nous présentons à présent les principaux enseignements de cette étude.

Nombre de modèles La première question était de savoir combien de modèles effectifs nous avions récupérés. Après analyse, sur les 21 125 fichiers récupérés, nous pouvons exclure environ 13 441 fichiers qui ne sont pas directement des modèles ni des formules de logique (ce sont, pour l'essentiel, des fichiers de type « *unknown* » contenant soit de la documentation, soit des scripts ou programmes relatifs aux modèles). Restent 7 694 fichiers qui nous intéressent pour cette étude. Nous avons classé les fichiers en quatre catégories : les modèles, les formules logiques, les fichiers auxiliaires pour modèles représentés dans un format multi-fichiers et les formats que le programme **guess** n'est pas capable de reconnaître. Le tableau ci-dessous donne la répartition de ces différents fichiers qui est également représentée à la figure 9.

Nombre de fichiers	Modèles	Fichiers multiples	Logiques	Format inconnu
7 694	4 231	1 350	1 449	664
100%	55.00%	17.55%	18.83%	8.63%

Répartition des modèles théoriques Ensuite, nous avons étudié la répartition des modèles théoriques parmi les 4 231 modèles identifiés. Le tableau suivant et la figure 10 nous donnent cette répartition.

22. https ://www.r-project.org/about.html

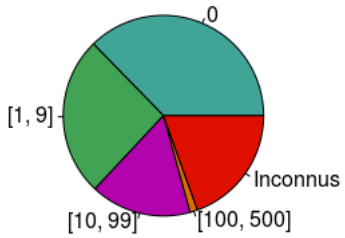


FIGURE 11 – Répartition des fichiers en fonction du nombre de variables d'état.

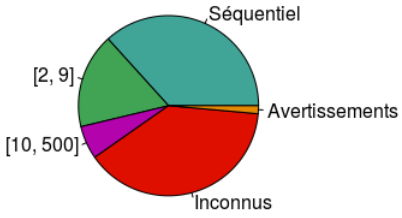


FIGURE 12 – Répartition des fichiers en fonction de la parallélisation.

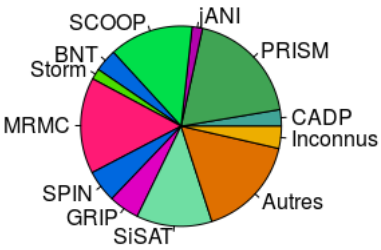


FIGURE 13 – Répartition entre les différents outils.

CTMC	DTMC	HA	IMC	IPC	LTS	MA
427	345	37	12	9	80	73
10.09%	8.15%	0.74%	0.28%	0.21%	1.98%	1.73%
MC	MDP	MM	PA	PEPA	PHA	PN
44	467	8	37	52	753	26
1.04%	11.03%	0.20%	0.87%	1.23%	17.80%	0.61%
PTA	SMG	SPN	TA	UML	Inconnu	Avertissement
9	1	8	15	4	1 686	138
0.21%	0.02%	0.20%	0.35%	0.09%	39.85%	3.26%

Les quatre modèles théoriques les plus représentés sont les PHAs (18%), puis les MDPs (11%), et les CTMCs (10%), enfin les DTMCs (8%). Le nombre important de PHAs dans notre collection est assez remarquable, alors que nous n'avons considéré que trois outils traitant des systèmes hybrides (SiSAT [23], PROHVer [24] et PHAVer [25]). On remarque aussi que les modèles théoriques vérifiés par PRISM (CTMC, DTMC et MDP) sont également bien représentés. Cependant, 40% des modèles théoriques n'ont pas pû être identifiés car **guess** donne des résultats approchés.

Variables d'état Nous avons étudié la présence des variables d'état dans chaque modèle. Nous décrivons la répartition des fichiers en fonction des variables d'état dans le tableau ci-dessous et dans la figure 12.

0	[1, 9]	[10, 99]	[100, 500]	Inconnu	Total
1 335	899	559	42	1 396	4 231
31.55%	20.81%	12.94%	1.0%	32.99%	100%

Dans notre collection, 32% des modèles ne possèdent pas de variables d'état et 35% en possédent au moins une. Par manque de documentation sur certains formats, il reste 33% des fichiers pour lesquels cette information n'est pas connue.

23. `http://www.avacs.org/tools/sisat`
24. `http://depend.cs.uni-saarland.de/tools/prohver`
25. `http://www-verimag.imag.fr/~frehse/phaver$\_$web`

Parallélisme Nous avons étudié le nombre de processus s'exécutant en parallèle dans chaque modèle. Nous décrivons la répartition des fichiers en fonction de la parallélisation dans le tableau suivant et dans la figure 12.

Séquentiel	$[2, 9]$	$[10, 500]$	Inconnu	Avertissement	Total
1 553	721	253	1 644	60	4 231
36.71%	17.04%	5.98%	38.85%	1.42%	100%

Dans notre collection, 37% des modèles sont séquentiels et 24% possèdent au moins deux processus s'exécutant en parallèle. Par manque de documentation sur certains formats, il reste 39% des fichiers pour lesquels cette information n'est pas connue.

Répartition des outils Nous décrivons la répartition des fichiers en fonction des différents outils dans le tableau suivant et dans la figure 13. Pour remplir ce tableau, nous avons compté combien de fichiers pouvait être utilisés sur chaque outil. Certains fichiers peuvent être utilisés par plusieurs outils.

APNN	Avispa	BNT	CADP	COMICS	Cosmos	DNAmaca
32	77	197	136	23	1	10
0.59%	1.69%	3.56%	2.46%	0.42%	0.02%	0.18%
FHP-Murphi	FIG	Fortuna	GreatSPN	GRIP	HSOLVER	IMCA
50	7	6	5	267	30	53
0.90%	0.13%	0.11%	0.09%	4.82%	0.54%	0.96%
IPC	jANI	Kronos	MAMOT	MARCA	Mobius	MODEST
9	91	15	5	11	11	38
0.16%	1.64%	0.27%	0.09%	0.20%	0.20%	0.69%
MRMC	MRMSolve	PASS	PAT	PEPA	PHAVer	PlasmaLab
841	7	12	4	55	181	47
15.19%	0.13%	0.22%	0.07%	0.99%	3.27%	0.85%
PMaude	PRINSYS	PRISM	ProbDivine	ProbVerus	ProHVer	PVeStA
4	91	1023	10	6	89	67
0.07%	1.64%	18.48%	0.18%	0.11%	1.61%	1.21%
PyECDAR	R	Rapture	SCOOP	SiSAT	SPIN	Storm
4	21	4	734	659	282	94
0.07%	0.38%	0.07%	13.26%	11.90%	5.09%	1.70%
TwoTowers	Inconnu	Total				
33	194	5 536				
0.60%	3.50%	100%				

Dans notre collection de modèles, les trois outils suivant se distinguent en supportant un grand nombre de modèles : PRISM avec au moins 18% des modèles, MRMC (15%) et SiSAT (11%). En revanche, pour les autres outils, il y a moins de 5% des modèles supportés.

Conclusion

La vérification de modèles appartenant au « zoo quantitatif » (c'est-à-dire, les modèles probabilistes, stochastiques, temporisés, hybrides...) demande l'utilisation d'outils supportant ces modèles. Aujourd'hui, ce domaine est en pleine expansion. Nous trouvons dans la communauté de nombreux modèles théoriques et outils divers, souvent incompatibles par manque de formats communs. Il est donc temps de chercher à organiser les connaissances et à mieux connecter les différents outils.

Pour nous familiariser avec ceux-ci, nous avons récupéré pour soixante-quatorze outils probabilistes et stochastiques plus de 21 000 fichiers de modèles. L'ensemble de ces fichiers ont été stockés de manière ordonnée par outils avant de les analyser.

Afin de les analyser, nous avons implémenté un outil prototype appelé **guess** qui fait une analyse partielle basée sur des règles de chacun des fichiers. Cette analyse permet d'obtenir le modèle théorique correspondant, le nombre de variables d'état, le nombre de processus s'exécutant en parallèle, le format et la liste des outils pouvant le traiter. Les caractéristiques données par **guess** peuvent servir à des utilisateurs non experts qui n'auraient pas une vue d'ensemble du domaine.

Nous avons utilisés **guess** afin de réaliser des statistiques. qui ont permis de confirmer que les modèles de PRISM ont une présence importante : au moins 18% des modèles peuvent être utilisé par PRISM. Mais elles ont également permis de révéler des faits peu connus comme l'existence de nombreux modèles de type PHA (18% des fichiers de modèles).

Perspectives

Notre collection de modèles appartenant au « zoo quantitatif » a été créée dans le but de faire une base de données de modèles permettait de pouvoir faire des comparaisons de performance. De plus, elle permettrait aux développeurs de ces outils d'avoir une base de données permettant d'effectuer des tests de non-régression.

À long terme, tous ces modèles pourraient être convertis dans un unique format utilisé de manière consensuelle par tous les outils. Analyser la collection et plus particulièrement les caractéristiques de ces modèles, nous permettrait de commencer à trouver quels sont les paramètres qu'il faudrait uniformiser pour obtenir ce format commun.

Nous avons commencé cette analyse avec **guess** qui est basée sur des règles et qui réalise de nombreuses approximations. En effet, dans sa version actuelle, **guess** n'est pas suffisamment « intelligent » pour inférer le modèle théorique de 40% des fichiers et 9% des fichiers restent de format inconnu. À long terme, on pourrait améliorer le programme pour qu'il soit capable de détecter toutes les informations (modèles, variables d'état, parallélisme, format et outils) pour tous les formats existants. Il pourrait également vérifier la correction des modèles théoriques en s'appuyant sur la distribution de probabilité. De plus, on peut imaginer que notre programme soit capable de déterminer la famille du modèle : dîner des philosophes, protocole d'exclusion mutuelle,... Ces extensions pour **guess** nécessiteraient d'utiliser un langage de plus haut niveau comme le langage **C** et probablement d'utiliser des algorithmes issus de l'apprentissage profond afin d'en améliorer ces prédictions.

References

[1] Rajeev Alur and David Dill. *Automata for modeling real-time systems*, pages 322–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[4] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. *CoRR*, abs/1702.04311, 2017.

[5] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, Jun 1971.

[6] Jean-Philippe Gros. A unifying framework for comparing and implementing probabilistic models, June 2017.

[7] Arnd Hartmanns and Holger Hermanns. In the quantitative automata zoo. *Sci. Comput. Program.*, 112:3–23, 2015.

[8] Boudewijn R. Haverkort, Joost-Pieter Katoen, and Kim G. Larsen. *Quantitative Verification in Practice*, pages 127–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1), 1990.

[10] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing, Third International Haifa Verification Conference, HVC 2007, Haifa, Israel, October 23-25, 2007, Proceedings*, pages 69–85, 2007.

[11] Joost-Pieter Katoen. Perspectives in probabilistic verification. In *2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 3–10, June 2008.

[12] Joost-Pieter Katoen. Private communication to Hubert Garavel, July 2017.

Remerciements

Je tiens à remercier l'ensemble des personnes qui m'ont permis de réaliser ce stage dans les meilleurs conditions.
Je remercie plus particulièrement, mon encadrant de stage, Hubert Garavel pour son accueil, le temps qu'il a passé avec moi et ses précieux conseils. Il m'a permis de confirmer mon intérêt pour la recherche.
Je remercie aussi Frédéric Lang et Lina Marsso pour leur aide y compris hors du cadre du stage.
Je remercie également les membres de l'équipe Convecs de l'INRIA Rhône-Alpes pour leur accueil et leur aide tout au long de mon stage.
Je remercie Luc Bougé pour m'avoir donné l'opportunité d'effectuer ce stage.

A Statistiques en R

Sur les fichiers que nous avons récupérés, nous avons effectués une analyse statistique. Celle-ci se base sur les résultats de notre script **guess** appliqué à l'aide de la commande ci-dessous aux fichiers qui sont potentiellement des modèles. En effet, notre script n'a pas été appliqué à la documentation, ni aux scripts permettant d'utiliser certains modèles.

```
find */Models -type f \
                    -a ! -name "*~" \
                    -a ! -name "*INFO" \
                    -a ! -name "*.pdf" \
                    -a ! -name "*.gif" \
                    -a ! -name "*.png" \
                    -a ! -name "*trace_*" \
                    -a ! -name "*.txt" \
                    -a ! -name "*.sh" \
                    -a ! -name "*Doc*" \
                    -a ! -name "*.jpg" \
                    -a ! -name "*.ps" \
                    -a ! -name "*.golden" \
                    -a ! -name "*.result" \
                    -a ! -name "*.html" \
                    -a ! -name "*README" \
                    -a ! -name "*TODO" \
                    -a ! -name "*Makefile" \
                    -a ! -name "*makefile" \
                    -a ! -name "*.gnuplot" \
                    -a ! -name "*Norm" \
                    -a ! -name "*.lib" \
                    -a ! -name "*.lotos" \
                    -a ! -name "*.xls" \
                    -a ! -name "*Readme" \
                    -a ! -name "*Makefile.FULL" \
                    -a ! -name "*Script" \
                    -a ! -name "*_list" \
                    -a ! -name "*output" \
                    -a ! -name "*OUTPUT" \
                    -a ! -name "*.readme" \
                    -a ! -name "*script" \
                    -a ! -name "*README.md" \
                    -a ! -name "*auto" \
                    -a ! -name "*run" \
                    -a ! -name "*run_" \
                    -a ! -name "*.plot" \
                    -a ! -name "*.old" \
                    -a ! -name "*clean" \
                    -a ! -name "*.ps.R" \
                    -a ! -name "*README.cmurphi" \
                    -a ! -name "*.out" \
```

```
                                -a ! -name "*.param" \
                                -a ! -name "*make-" \
                                -a ! -name "*-tests" \
                                -a ! -name "*-data" \
                                -a ! -name "*-model" \
                                -a ! -name "*-property" \
                                -a ! -name "*.rules" \
                                -a ! -name "*.bash" \
                                -a ! -name "*RELEASENOTES" \
                                -a ! -name "*.info" \
                                -a ! -name "*.tex" \
                                -a ! -name "*README.SPIN" \
                                -a ! -name "*.o" \
                                -a ! -name "*Entries" \
                                -a ! -name "*Repository" \
                                -a ! -name "*Root" \
                                -a ! -name "*.tcl" \
                                -a ! -name "*testscript*" \
                                -a ! -name "*.lot" \
                                -a ! -name "*.seq" \
                                -a ! -name "*.t" \
                                -a ! -name "*.mcl" \
                                -a ! -name "*.data.file" \
                                -a ! -name "*.awk" \
                                -a ! -name "*options" \
                                -a ! -name "*files" \
                                -a ! -name "*out_" \
                                -a ! -anme "*.class" \
                                -a ! -name "*.java" \
                                -a ! -name "*.res" \
        -exec sh ~/Scripts/guess \{\} \; |& tee @result_stat
```

Le fichier résultat ainsi obtenu a été nettoyé pour être utilisé dans R, à de l'aide de la commande suivante.

```
        cat @result_stat | grep -v "/Models/" \
                                        | grep -v "Scripts/guess" \
                                        | grep -v "unknown format" \
                                        | grep -v "grep" \
                                        | grep -v "tmp/" \
                                        | grep -v "record number" \
                                        | grep -v "awk" \
                                        | grep -v "auxiliary" \
                                        | grep -v "Logic" \
                                        | grep -v '^:' \
                                        | sed -e 's/Model: //g' \
                                        | sed -e 's/ Var: //g' \
                                        | sed -e 's/ Par: //g' \
                                        | sed -e 's/ Tool: //g' \
                                        | sed -e 's/ Format: //g' \
```

```
| sed -e 's/Part of //g' \
| sed -e 's/ model //g' \
| sed -e 's/[ \t]*//g' \
| sed -e 's/(Continuoustime)//g' \
| sed -e 's/unknow(toolsupportprismformat)/PRISM/g' \
| sed -e 's/unknow(toolsupportpepaformat)/PEPA/g' \
| sed -e 's/Areyousureoftheformat?/warning/g' \
| & tee @result_stat1
```

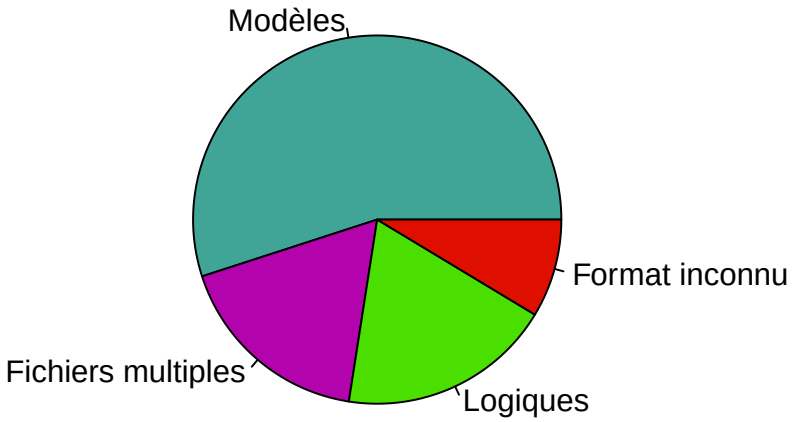
Statistiques sur la collection de modèles

```
tab <- read.table("result.txt", header=TRUE, sep= ";" )
summary(tab)
```

```
##      Model           Var            Par               Format
##  unknown:1686   unknown:1396   unknown:1644   tra         : 651
##  PHA     : 753   0       :1335   1       :1555   aig(binary): 585
##  MDP     : 467   1       : 178   4       : 168   sm          : 350
##  CTMC    : 427   8       : 132   5       : 141   nm          : 341
##  DTMC    : 345   6       : 129   3       : 125   pm          : 290
##  warning: 138   2       : 122   2       : 102   m           : 247
##  (Other): 415   (Other): 939   (Other): 496   (Other)     :1767
##                Tool
##  PRISM           :1023
##  SiSAT           : 659
##  SCOOPorMRMC     : 646
##  EitherSPINorGRIP: 236
##  BNT             : 197
##  unknown         : 194
##  (Other)         :1276
```

Nombre de modèles

```
donnee=c(4231, 1350, 1449, 664)
labeldonnee=c("Modèles", "Fichiers multiples", "Logiques", "Format inconnu")
colordonnee=c("#40A497", "#B303AD", "#4ADF00", "#DF0F00")
pie(donnee, col = colordonnee, labels = labeldonnee)
```



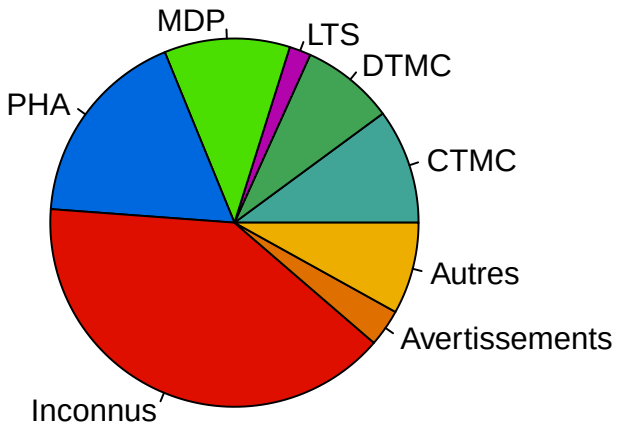
Modèles

```
summary(tab$Model)
```

```
##    CTMC    DTMC      HA     IMC     IPC     LTS      MA      MC     MDP
##     427     345      37      12       9      80      73      44     467
##      MM      PA    PEPA     PHA      PN     PTA     smg     SPN      TA
```

```
##       8      37      52     753      26       9       1       8      15
##     UML unknown warning
##       4    1686     138
```

```
model=c(427, 345, 80, 467, 747, 1686, 138, 341)
labelmod=c("CTMC", "DTMC", "LTS", "MDP", "PHA", "Inconnus", "Avertissements", "Autres")
colormod=c("#40A497", "#40A454", "#B303AD", "#4ADF00", "#0068DF", "#DF0F00", "#DF7000",
           "#EDAE00")
pie(model, col=colormod, labels = labelmod)
```



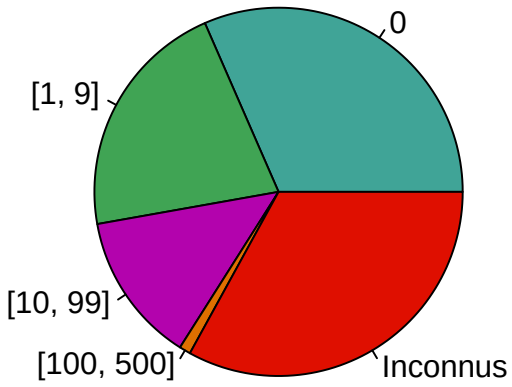
Variables d'état

```
summary(tab$Var)
```

```
##       0       1      10     102      11     110     112     118     119
##    1335     178      59       2      19       1       2       1       1
##      12     127     129      13     137      14     148      15     151
##      27       1       6      25       1      18       2      36       1
##     154      16      17      18     183     185      19     194       2
##       1      26      17      19       1       1      36       2     122
##      20      21     211      22      23      24     240      25     254
##      21       9       3      11       8      16       2      32       1
##      26     264     265      27      28     286      29       3      30
##      21       1       1       8       3       2       6      91      15
##      31      32      33     332      34      35      36      37     378
##      12      10       1       2       2       1       4       6       2
##      38      39       4      40      41      42     424      43      44
##       2       1      56       3      20       2       2       3       4
##      45      48      49       5      50      51      52      55      57
##       5       2       3      78       3       3       2       1       1
##      59       6      62      63      64       7      70      73      74
##       1     129       3       1       2      36       1       1       1
##      78       8      88       9      90      92      93      96      97
##       2     132       1      77      14       1       4       6       1
## unknown
##    1396
```

```
var=c(1335, 899, 559, 42, 1396)
labelvar=c("0", "[1, 9]", "[10, 99]", "[100, 500]", "Inconnus")
```

```
colorvar=c("#40A497", "#40A454", "#B303AD", "#DF7000", "#DF0F00")
pie(var, col=colorvar, labels = labelvar)
```

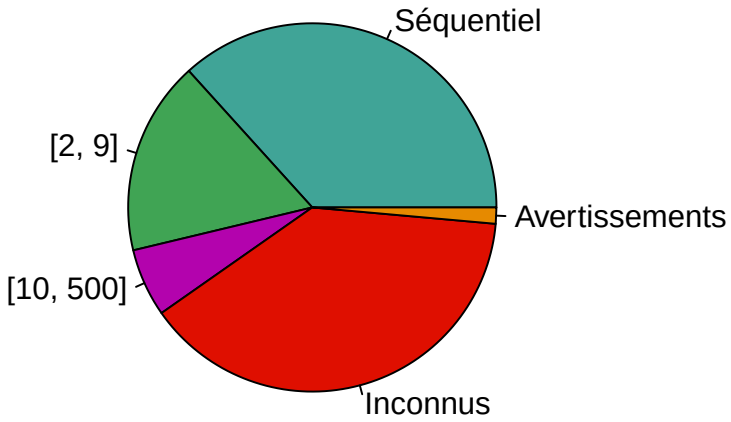


Parallélisation

```
summary(tab$Par)
```

```
##       1      10     100      11      12      13      14     147      15
##    1555      44       1      18      10      19       4       1      13
##      16      17      18      19       2      20      21      22      23
##      12      12       8      10     102       8       7       3       1
##      25       3      30     300      32      35      36       4      42
##       2     125       2       1       1       1       1     168       2
##       5      53      58       6      63      69       7       8      80
##     141       2       1      98       2       2      65      37       1
##      85       9      96 unknown Warning
##       2      43       2    1644      60
```

```
var=c(1553, 721, 253, 1644, 60)
labelvar=c("Séquentiel", "[2, 9]", "[10, 500]", "Inconnus", "Avertissements")
colorvar=c("#40A497", "#40A454", "#B303AD", "#DF0F00", "#E68A00")
pie(var, col=colorvar, labels = labelvar)
```



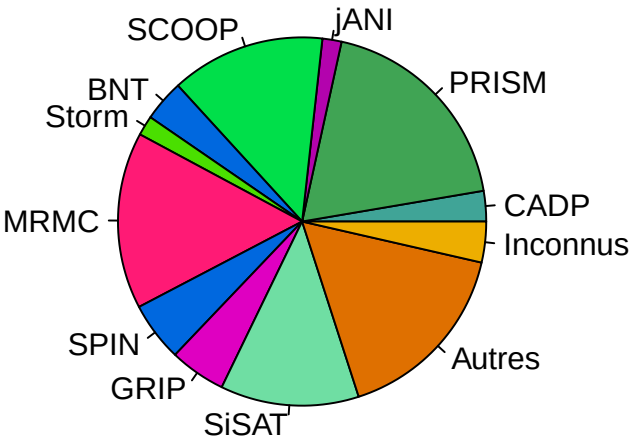
Outils

```
summary(tab$Tool)
```

```
##                          APNN                        Avispa
##                            32                            77
##                           BNT                          CADP
##                           197                           136
##                        COMICS                        Cosmos
##                            23                             1
##                       DNAmaca EitherofJANI,PRINSYSorStorm
##                            10                            91
## EitherofPEPA,MobiusorModest       EitherProHVerorPHAVer
##                             3                            87
##           eitherPVeStAorMRMC            EitherSPINorGRIP
##                            62                           236
##                   FHP-Murphi                           FIG
##                            50                             7
##                       Fortuna                       GreatSPN
##                             6                             5
##                         GRIP                        HSOLVER
##                            31                            30
##                         IMCA                           IPC
##                            53                             9
##                       Kronos                        MAMOT
##                            15                             5
##                        MARCA                        Mobius
##                            11                             9
##                       MODEST                         MRMC
##                            35                           133
##                     MRMSolve                         PASS
##                             7                            12
##                          PAT                         PEPA
##                             4                            52
##                       PHAVER                    PlasmaLab
##                             7                            47
##                       PMaude                         PRISM
##                             4                          1023
##                    ProbDivine                    ProbVerus
##                            10                             6
##                       ProHVer                       PVeStA
##                             2                             5
##                       PyECDAR                             R
##                             4                            21
##                     Rapture                          SCOOP
##                             4                            88
##                   SCOOPorMRMC                         SiSAT
##                           646                           659
##                          SPIN                         Storm
##                            46                             3
##                     TwoTowers                       unknown
##                            33                           194
```

```
tool=c(145, 1023, 91, 734, 197, 94, 841, 282, 267, 659, 891, 194)
labeltool=c("CADP", "PRISM", "jANI", "SCOOP", "BNT", "Storm", "MRMC", "SPIN",
```

```
            "GRIP", "SiSAT", "Autres", "Inconnus")
colortool=c("#40A497", "#40A454", "#B303AD", "#00DF4A", "#0068DF", "#4ADF00",
            "#ff1a75", "#0068DF", "#DF00C1", "#6FDEA3", "#DF7000", "#EDAE00",
            "#DF0F00")
pie(tool, labels = labeltool, col = colortool)
```



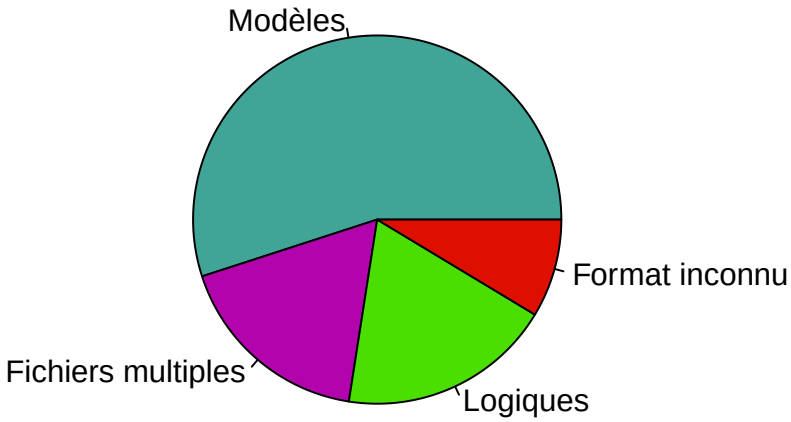
Statistiques sur la collection de modèles

```
tab <- read.table("result.txt", header=TRUE, sep= ";" )
summary(tab)
```

```
##      Model           Var            Par               Format
##  unknown:1686   unknown:1396   unknown:1644   tra         : 651
##  PHA     : 753   0       :1335   1       :1555   aig(binary): 585
##  MDP     : 467   1       : 178   4       : 168   sm          : 350
##  CTMC    : 427   8       : 132   5       : 141   nm          : 341
##  DTMC    : 345   6       : 129   3       : 125   pm          : 290
##  warning: 138   2       : 122   2       : 102   m           : 247
##  (Other): 415   (Other): 939   (Other): 496   (Other)     :1767
##                Tool
##  PRISM           :1023
##  SiSAT           : 659
##  SCOOPorMRMC     : 646
##  EitherSPINorGRIP: 236
##  BNT             : 197
##  unknown         : 194
##  (Other)         :1276
```

Nombre de modèles

```
donnee=c(4231, 1350, 1449, 664)
labeldonnee=c("Modèles", "Fichiers multiples", "Logiques", "Format inconnu")
colordonnee=c("#40A497", "#B303AD", "#4ADF00", "#DF0F00")
pie(donnee, col = colordonnee, labels = labeldonnee)
```



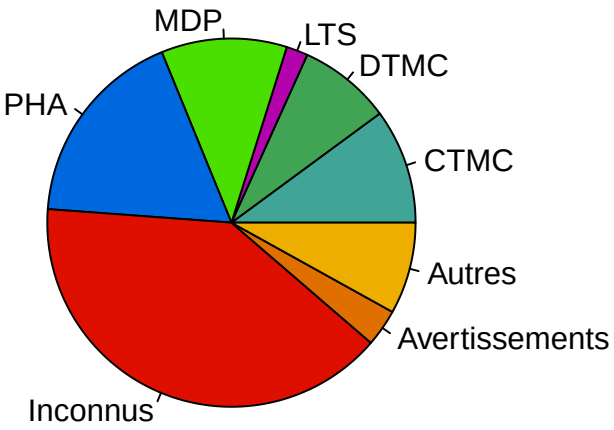
Modèles

```
summary(tab$Model)
```

```
##    CTMC    DTMC      HA     IMC     IPC     LTS      MA      MC     MDP
##     427     345      37      12       9      80      73      44     467
##      MM      PA    PEPA     PHA      PN     PTA     smg     SPN      TA
```

```
##       8      37      52     753      26       9       1       8      15
##     UML unknown warning
##       4    1686     138
```

```
model=c(427, 345, 80, 467, 747, 1686, 138, 341)
labelmod=c("CTMC", "DTMC", "LTS", "MDP", "PHA", "Inconnus", "Avertissements", "Autres")
colormod=c("#40A497", "#40A454", "#B303AD", "#4ADF00", "#0068DF", "#DF0F00", "#DF7000",
           "#EDAE00")
pie(model, col=colormod, labels = labelmod)
```



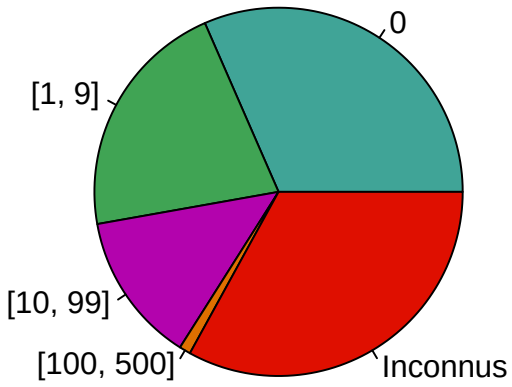
Variables d'état

```
summary(tab$Var)
```

```
##       0       1      10     102      11     110     112     118     119
##    1335     178      59       2      19       1       2       1       1
##      12     127     129      13     137      14     148      15     151
##      27       1       6      25       1      18       2      36       1
##     154      16      17      18     183     185      19     194       2
##       1      26      17      19       1       1      36       2     122
##      20      21     211      22      23      24     240      25     254
##      21       9       3      11       8      16       2      32       1
##      26     264     265      27      28     286      29       3      30
##      21       1       1       8       3       2       6      91      15
##      31      32      33     332      34      35      36      37     378
##      12      10       1       2       2       1       4       6       2
##      38      39       4      40      41      42     424      43      44
##       2       1      56       3      20       2       2       3       4
##      45      48      49       5      50      51      52      55      57
##       5       2       3      78       3       3       2       1       1
##      59       6      62      63      64       7      70      73      74
##       1     129       3       1       2      36       1       1       1
##      78       8      88       9      90      92      93      96      97
##       2     132       1      77      14       1       4       6       1
## unknown
##    1396
```

```
var=c(1335, 899, 559, 42, 1396)
labelvar=c("0", "[1, 9]", "[10, 99]", "[100, 500]", "Inconnus")
```

```
colorvar=c("#40A497", "#40A454", "#B303AD", "#DF7000", "#DF0F00")
pie(var, col=colorvar, labels = labelvar)
```

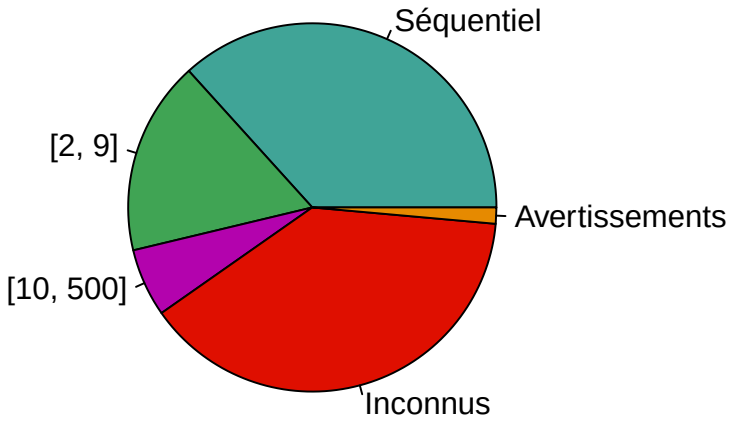


Parallélisation

```
summary(tab$Par)
```

```
##       1      10     100      11      12      13      14     147      15
##    1555      44       1      18      10      19       4       1      13
##      16      17      18      19       2      20      21      22      23
##      12      12       8      10     102       8       7       3       1
##      25       3      30     300      32      35      36       4      42
##       2     125       2       1       1       1       1     168       2
##       5      53      58       6      63      69       7       8      80
##     141       2       1      98       2       2      65      37       1
##      85       9      96 unknown Warning
##       2      43       2    1644      60
```

```
var=c(1553, 721, 253, 1644, 60)
labelvar=c("Séquentiel", "[2, 9]", "[10, 500]", "Inconnus", "Avertissements")
colorvar=c("#40A497", "#40A454", "#B303AD", "#DF0F00", "#E68A00")
pie(var, col=colorvar, labels = labelvar)
```



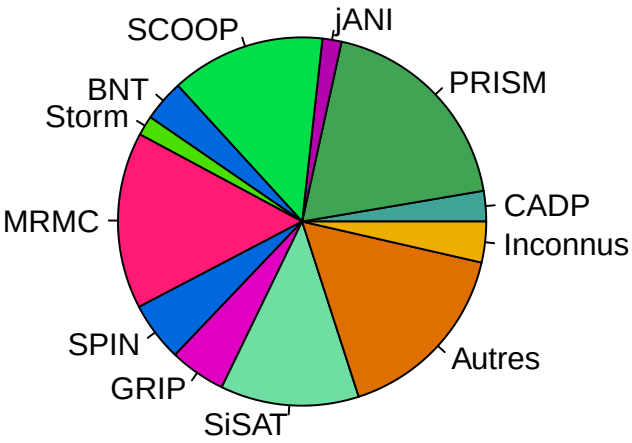
Outils

```
summary(tab$Tool)
```

```
##                          APNN                        Avispa
##                            32                            77
##                           BNT                          CADP
##                           197                           136
##                        COMICS                        Cosmos
##                            23                             1
##                       DNAmaca EitherofJANI,PRINSYSorStorm
##                            10                            91
## EitherofPEPA,MobiusorModest       EitherProHVerorPHAVer
##                             3                            87
##           eitherPVeStAorMRMC            EitherSPINorGRIP
##                            62                           236
##                    FHP-Murphi                           FIG
##                            50                             7
##                       Fortuna                       GreatSPN
##                             6                             5
##                          GRIP                        HSOLVER
##                            31                            30
##                          IMCA                           IPC
##                            53                             9
##                        Kronos                         MAMOT
##                            15                             5
##                         MARCA                        Mobius
##                            11                             9
##                        MODEST                          MRMC
##                            35                           133
##                     MRMSolve                          PASS
##                             7                            12
##                           PAT                          PEPA
##                             4                            52
##                        PHAVER                     PlasmaLab
##                             7                            47
##                        PMaude                         PRISM
##                             4                          1023
##                    ProbDivine                     ProbVerus
##                            10                             6
##                       ProHVer                        PVeStA
##                             2                             5
##                       PyECDAR                             R
##                             4                            21
##                       Rapture                         SCOOP
##                             4                            88
##                    SCOOPorMRMC                        SiSAT
##                           646                           659
##                          SPIN                         Storm
##                            46                             3
##                     TwoTowers                       unknown
##                            33                           194
```

```
tool=c(145, 1023, 91, 734, 197, 94, 841, 282, 267, 659, 891, 194)
labeltool=c("CADP", "PRISM", "jANI", "SCOOP", "BNT", "Storm", "MRMC", "SPIN",
```

```
            "GRIP", "SiSAT", "Autres", "Inconnus")
colortool=c("#40A497", "#40A454", "#B303AD", "#00DF4A", "#0068DF", "#4ADF00",
            "#ff1a75", "#0068DF", "#DF00C1", "#6FDEA3", "#DF7000", "#EDAE00",
            "#DF0F00")
pie(tool, labels = labeltool, col = colortool)
```



B Les études de cas

Aucune base commune de test n'existant, le calcul de performance est effectué sur un nombre restreint d'exemples. Dans le cadre de leur article [10] comparant les différents outils, les auteurs utilisent cinq protocoles (trois discrets et deux continus) issus des problèmes concurrents que nous allons présenter rapidement. Lors de l'élaboration de tels modèles trois caractéristiques sont à prendre en compte : le contexte du problème, le protocole que l'on souhaite tester permettant de connaître la taille du système que l'on étudie et, surtout, les propriétés que l'on souhaite vérifier.

B.1 SLE : Synchronous Leader Election

Dans l'algorithmique distribué, il arrive que l'on ait besoin d'un *leader* parmi les processus pour envoyer des messages à tout le groupe. Il faut donc le choisir : ce qui implique que le *leader* choisi sache que c'est lui qui a été choisi et que les autres connaissent le *leader*.

Dans cette étude de cas, on considère un anneau contenant n processus communiquant de manière unidirectionnelle et synchrone. Le but est d'élire un leader en communiquant à travers ce réseau. Pour ce faire, nous utiliserons le protocole fonctionnant par tour d'Itai et de Rodeh [9] suivant : à chacun des tours, on effectue les actions suivantes.

* Chacun des processus tire aléatoirement et indépendamment un identifiant dans l'ensemble suivant $\{1; ...; k\}$ où $k \in \mathbb{N}$;
* Communications des différents identifiants dans l'anneau ;
* Si au moins un des processus un identifiant unique alors celui avec le plus grand identifiant est choisi comme *leader* ;
* Sinon on recommence un nouveau round.

En jouant sur le nombre de processus (de 4 à 8) et le nombre d'identifiants (de 2 à 4), on obtient un système qui oscille entre 55 états avec 70 transitions à 458 847 états avec 524 382 transitions. Sur ces systèmes nous souhaitons vérifier les propriétés suivantes :

— un leader a finalement été élu ;
— la probabilité d'élire un leader en moins de 5 étapes est supérieur à 0.85 ;
— la probabilité d'élire un leader en moins de 40 étapes est supérieur à 0.99.

B.2 RDP : Randomized Dining Philosophers

Le dîner des philosophes est un classique de l'algorithmique distribué permettant de modéliser un système qui peut s'auto-bloquer. On considère alors une table (ronde) où n philosophes sont installés de telle sorte que chacun des philosophes est entouré d'uniquement deux baguettes. Ceux-ci n'ont que deux préoccupations dans la vie : penser et manger. Au centre de la table se tient une grande assiette de spaghetti constamment remplie. Pour pouvoir manger, quand l'envie leur en prend, ils doivent nécessairement utiliser deux baguettes (une de chaque côté). Le but est d'éviter les blocages : tous les philosophes doivent pouvoir manger un jour ou l'autre.

On considère alors le protocole de Dijkstra [5] afin de répondre à ce problème. À chaque fois qu'un philosophe a faim, il exécute les instructions suivantes.

* Il prend chacune des deux baguettes dans un ordre aléatoire ;
* S'il possède les deux baguettes : il mange ;
* Sinon il abandonne de manger (il pourra recommencer plus tard).

En variant le nombre de philosophes (de 3 à 7), on obtient un système qui peut contenir 770 états avec 2 845 transitions à un système contenant 5 454 562 états pour 44 070 594 transitions. Dans ces systèmes, on souhaite alors vérifier que le système ne va pas se bloquer et que faute de pouvoir manger, tous les philosophes meurent. On vérifie alors les propriétés suivantes :

— Un philosophe va finalement manger ;
— La probabilité qu'un philosophe mange dans les 20 prochaines étapes est supérieur à 0.9.

B.3 BDP : Birth–Death Process

Le processus BDP est largement utilisé dans l'estimation de la taille d'une population ou la taille d'une file d'attente. Lorsque l'on souhaite estimer la taille d'une population que l'on sait bornée par m, l'état *Birth-Death* des processus représentés par un nombre naturel, dénote de la taille courante de la population.

Pour répondre à cette question, on met en place une chaîne de Markov modélisant ce nombre telle que :

* Un *birth* dénote une augmentation de la population : naissance ;
* Un *death* dénote une diminution de la population : mort ;
* La probabilité d'obtenir une naissance décroît en fonction de la taille : elle vaut 0 lorsque la taille est maximale.

En faisant varier la taille maximale de la population (de 100à 100 000), nous obtenons un système qui contient au minimum 101 états avec 202 transitions, et qui au maximum contient 100 001 états avec 200 002 transitions. Sur ces systèmes on souhaite vérifier les propriétés suivantes.

— La probabilité d'obtenir un quart de la population maximale en $\frac{m}{2}$ étapes est supérieure à 0.9.
— On atteindra une population finale de 50 alors que la probabilité d'obtenir une population de 70 en 100 étapes ne sera jamais inférieure à 0.9 ;
— La population atteindra finalement sa taille maximale.

B.4 TQN : Tandem Queuing Network

Le problème TQN consiste en l'implémentation de deux files de capacité n gérant un échange de messages. Les messages arrivent sur la première file en attente d'être traités. Une fois traités, ils sont envoyés sur la deuxième file où ils seront supprimés du système. Ces échanges de messages respectent alors les propriétés suivantes.

* Les messages arrivent sur la première file avec une distribution de loi exponentielle de paramètre 4 ;
* Les messages transitent sur la deuxième file (sont traités) selon une distribution : two-phase Coxian donnant la probabilité de changer de files après chaque étape à l'aide d'une convolution entre deux lois exponentielles ;

* Les messages quittent le système selon une distribution de loi exponentielle de paramètre 4.

En jouant sur la capacité des files (de 2 à 1023) on obtient des systèmes contenant au minimum 15 états avec 23 transitions et au maximum 2 096 128 états avec 7 328 771 transitions. Sur ces systèmes on vérifie les propriétés suivantes.

* A l'équilibre, TQN est plein avec une probabilité inférieure à 0.01 ;
* TQN est plein entre $[0.5;\ 2]$ unité de temps avec une probabilité inférieure à 0.1 ;
* Si la seconde file est pleine, une erreur est finalement arrivée.

B.5 CPS : Cyclic Server Polling System

Un système cyclique de vote consiste en n stations possédant un buffer de capacité 1 et un serveur central dont son rôle est de récupérer les votes. Pendant le vote, chaque station attend l'ordre du serveur de voter et place ce vote dans son buffer. Le serveur qui va ainsi station après station récupérer leur vote présent dans le buffer. Si le buffer est vide (la station n'a pas voté), le serveur passe à la station suivante. On choisit d'étudier le système sous les aspects suivants :

* Le temps de vote est distribué selon une loi exponentielle de paramètre 200 ;
* Le temps de service est distribué selon une loi exponentielle de paramètre 1 ;
* Le taux d'arrivée des messages pour chaque station est distribué selon une loi exponentielle de paramètre $\frac{1}{n}$.

En jouant sur le nombre de serveurs (de 3 à 18), on obtient des systèmes oscillants entre 36 états avec 84 transitions et 7 077 888 états avec 69 599 232 transitions. Pour ces systèmes, on vérifie les propriétés suivantes.

— Dans un état stable, la première station attend le serveur avec une probabilité inférieure à 0.2 ;

— La probabilité que la première station soit servie dans un intervalle de temps $[40; 80]$ est inférieure à 0.99 ;

— Si la première station est occupée, la probabilité qu'elle soient servie au temps t est supérieure à 0.5 ;

— Si la première station est occupée, elle sera finalement servie.