Rare event simulation for statistical model checking

Cyrille Jegourel

ANNÉE 2014

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Cyrille Jegourel

préparée à l'unité de recherche Estasys - INRIA
Institut National de Recherche en Informatique et en Automatique
ISTIC

Intitulé de la thèse
Rare Event Simulation for Statistical Model Checking

Thèse soutenue à Rennes le 19 novembre 2014

devant le jury composé de :

Pieter-Tjerk DE BOER
Professeur à l'Université de Twente / rapporteur

David PARKER
Professeur à l'Université de Birmingham / rapporteur

Jean-Marc JÉZÉQUEL
Professeur à l'Université de Rennes 1 / examinateur

Heinz KOEPPL
Professeur à l'Université de Darmstadt / examinateur

Sylvain PEYRONNET
Professeur à l'Université de Caen Basse-Normandie / examinateur

Axel LEGAY
Chargé de recherche à Inria Rennes / directeur de thèse

Remerciements

Bien qu'une thèse ne comporte qu'un seul auteur, en aucun cas, elle n'est le produit d'un seul individu. Les idées qu'elle contient, les discussions préalables, le travail d'écriture et tous les à-côtés sont des composantes plus ou moins collectives et inaliénables du manuscrit final. Je profite donc de ces lignes pour remercier les personnes qui m'ont accompagné tout au long de cette étape.

En premier lieu, je remercie mes deux rapporteurs de thèse, David Parker et Pieter-Tjerk De Boer d'avoir pris le temps de lire mes travaux, de les comprendre et de les corriger. Je les remercie également d'avoir accepté de faire partie de mon jury ainsi que Jean-Marc Jézéquel, Heinz Koeppl et Sylvain Peyronnet.

Je remercie bien entendu Axel Legay d'avoir encadré ma thèse et de m'avoir tendu la main alors que j'étais sans-emploi depuis trop longtemps, à mon grand dam. Je lui sais gré de toutes les opportunités de rencontres et de voyages qu'il m'a offertes et de m'avoir constamment poussé et encouragé tout au long de ce travail. Je remercie également Sean Sedwards avec qui j'ai travaillé ces quatre dernières années, sur les thématiques liées aux événements rares. Ce fut un plaisir sincère d'échanger avec Sean, aussi bien sur nos recherches que sur le reste. Son expérience, sa maturité et son pragmatisme sont un exemple dont je compte bien m'inspirer par la suite.

J'en profite également pour témoigner de ma reconnaissance envers les autres chercheurs avec qui j'ai interagi durant ma thèse, à commencer par Radu Grosu qui m'a accueilli à Vienne en novembre 2013, Scott Smolka à Stony Brook en avril 2013, Saddek Bensalem à Grenoble à l'aube de ma thèse et Kim G. Larsen à Aalborg dans les jours qui viennent.

Bien entendu, je remercie les membres des différentes équipes dont j'ai eu la chance de faire partie antérieurement : S4, DistribCom et Triskell. Je remercie aussi l'équipe Vertecs pour toutes les pauses-cafés que j'ai partagées avec elle lors de mes deux premières années ainsi que la tour des Maths dans son ensemble pour les repas du midi. Je ne peux malheureusement pas énumérer tous les noms, tant de générations de personnes sympathiques s'y étant succédé. Je remercie mes collègues d'Estasys pour leur soutien et leurs conseils et je laisse à chacun d'entre eux le soin de s'attribuer son propre mérite.

Je me dois également de remercier l'équipe ASPI, Teddy Furon et Fida El Haje Hussein. J'ai eu le privilège de travailler au sein de ce groupe en 2008-2009. Le chapitre 5 de ma thèse est d'ailleurs directement inspiré des travaux de Frédéric

Cérou et Arnaud Guyader. J'en profite également pour déclarer que Arnaud fut également l'un des meilleurs professeurs, en plus d'être l'un des plus sympathiques, que j'ai pu avoir durant mon cursus universitaire.

Enfin, une thèse n'est pas seulement un processus de trois ans ; elle est l'aboutissement d'un long chemin parsemé de rencontres marquantes. Je profite donc de cette page pour remercier les personnes qui m'ont apporté tour à tour éveil, savoir et bonheur. Tout d'abord, je souhaite remercier toute ma famille et en particulier mes parents qui m'ont apporté un soutien affectif, financier et moral sans faille aucune. Je remercie M. Courtois, Yannis et Stanislas pour tout ce qu'ils m'ont apporté au cours de mon adolescence et de mes premières années d'étude.

Et car les amis sont une denrée rare, évocatrice de moments de bonheur partagés, de rigolades et de discussions passionnées, je remercie l'ensemble de mes copains de Rennes 2, en particulier Marine, Élodie, Amandine et Jeff, les joueurs du Cercle Paul Bert emmené par le guerrier Monroy, les joueurs du RMPT, Nicolas, les amis de Rennes 1, en particulier Yoann, Maëlle, Karine, Basile, Jean-Louis, Catherine, Tiffany, Gaël, No, Baptiste, Camton et j'en oublie sans doute.

Pour conclure, je tiens à remercier le plus chaleureusement possible celle qui partage ma vie depuis plus de cinq ans et qui trouve encore le moyen de me soutenir dans tous les moments, à savoir mon épouse Nora.

Contents

List of Figures

Preamble

The results presented in this thesis are based on the following publications and submissions:

- Cyrille Jegourel, Axel Legay, Sean Sedwards: Command-based Importance Sampling for Statistical Model Checking. *Submitted in Theoretical Computer Science in 2014*.
- Kenan Kalajdzic, Cyrille Jegourel, Ezio Bartocci, Axel Legay, Scott Smolka, Radu Grosu: Model Checking as Control: Feedback Control for Statistical Model Checking of Cyber-Physical Systems. *Submitted in TACAS, 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, London, United Kingdom, 2015.
- Cyrille Jegourel, Axel Legay, Sean Sedwards: An Effective Heuristic for Adaptive Importance Splitting in Statistical Model Checking. *In ISoLA, 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Limni, Corfu, Greece, 2014.
- Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jegourel, Axel Legay, Ayoub Nouri: Statistical Model Checking QoS Properties of Systems with SBIP. *In Software Tools for Technology Transfer*, 2014.
- Cyrille Jegourel, Axel Legay, Sean Sedwards: Importance Splitting for Statistical Model Checking Rare Properties. *In CAV, 25th International Conference on Computer Aided Verification*, Saint-Petersburg, Russia, 2013.
- Cyrille Jegourel, Axel Legay, Sean Sedwards: Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. *In CAV, 24th International Conference on Computer Aided Verification*, Berkeley, California, USA, 2012.
- Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jegourel, Axel Legay, Ayoub Nouri: Statistical Model Checking QoS Properties of Systems with SBIP. *In ISoLA, 5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Amirandes, Heraclion, Greece, 2012.

- Cyrille Jegourel, Axel Legay, Sean Sedwards: A Platform for High Performance Statistical Model Checking - PLASMA. *In TACAS, 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Tallinn, Estonia, 2012.

Other works by the same author:

- Uli Fahrenberg, Fabrizio Biondi, Kevin Corre, Cyrille Jegourel, Simon Kongshøj, Axel Legay: Measuring Global Similarity between Texts. *In SLSP, 2nd International Conference on Statistical Language and Speech Processing*, Grenoble, France, 2014.

Other works by the same author prior the thesis:

- Frédéric Cérou, Teddy Furon, Arnaud Guyader, Cyrille Jegourel: Estimating the probability of false alarm for a zero-bit watermarking technique. *In DSP, 16th International Conference on Digital Signal Processing*, IEEE, Fira, Santorini, Greece, 2009.

Résumé de la thèse : Simulation d'événements rares en Model Checking statistique

0.1 Introduction

0.1.1 Contexte

Les technologies de l'information et de la communication ont pris une place considérable dans nos sociétés ces dernières décennies. Elles pénètrent tous les espaces, publics et privés, de notre quotidien. La plupart des maisons, bureaux, moyens de transport, entreprises et administrations disposent d'équipements électroniques plus ou moins modernes, d'une connexion internet, de téléphone ou d'une télévision. L'avènement de l'internet et des réseaux sociaux ont transformé les rapports humains, économiques et industriels. Par voie de conséquence, les entreprises utilisent ou développent des technologies de pointe de plus en plus complexes pour offrir toujours plus de solutions, de précision et de vitesse dans l'exécution des tâches demandées.

Ces *systèmes d'ingénierie*, de plus en plus composites et implexes, impliquent des interactions fortes entre divers programmes ou logiciels informatiques, composants électroniques, etc. Or, s'il était possible dans le passé pour une entreprise d'encadrer au sein d'une même structure les activités de conception de systèmes embarqués de la spécification jusqu'à l'implémentation, c'est aujourd'hui chose impossible du fait de l'explosion croissante de complexité induite par le besoin de compatibilité entre éléments d'un système. Ces composantes sont ainsi développées en général par différentes équipes qui travaillent indépendamment les unes des autres mais qui s'accordent éventuellement sur la spécification des composants qu'elles devront utiliser et partager. Par ailleurs, certains de ces systèmes, dits *critiques*, ont pour objet la réalisation de tâches dont l'échec peut avoir des répercussions économiques, humaines ou environnementales dramatiques, par exemple en aéronautique ou en télécommunication. Cette complexité croissante a entraîné en conséquence une sérieuse augmentation de bogues ou de comportements non-désirés. Ainsi, entre 1985 et 1987, au moins cinq personnes décédèrent des suites d'une overdose de radi-

ation imputée à des défaillances de sécurité et des erreurs logicielles d'un appareil de radiothérapie, le Therac-25 [95]. En 1996, une erreur de conversion d'un entier 16-bit rn virgule flottante 64-bit provoquait l'explosion de la fusée Ariane-5, seulement quelques secondes après son décollage [96]. En août 2003, une situation critique de concurrence entre différents agents du réseau électrique nord-américain provoquait en quelques heures la panne de 256 centrales électriques [3].

Si les conséquences sociétales ne sont pas vraiment quantifiables, les répercussions économiques, elles, ont été évaluées plusieurs fois. En 2002, une étude commandée par l'institut national des normes et de la technologie du département du commerce étasunien estimait la perte annuelle due aux erreurs et bogues logiciels pour l'économie américaine à 59 milliards de dollars [67]. Plus récemment, une étude de l'université de Cambridge, publiée en 2012, avançait le chiffre de 312 milliards de dollars de perte pour l'économie mondiale[1]. La fiabilité et la compréhension du comportement de ces systèmes informatiques est donc devenue un enjeu clé dans le processus de conception du système.

0.1.2 Méthodes formelles

À l'heure actuelle, il n'existe pas de méthode universelle et automatisable permettant de trouver et corriger tous les bogues d'un logiciel ou d'un matériel électronique. Les techniques de vérification ont pour objectif de réduire significativement leur fréquence. Ainsi, elles sont utilisées dans le *design* des systèmes dans le but de garantir que ce système satisfasse certaines propriétés. Ces propriétés caractérisent donc ce que le système doit être en mesure de faire et de ne pas faire et un système qui satisfait l'ensemble des propriétés spécifiées est dit "correct".

La méthode la plus couramment utilisée pour vérifier la conformité d'un système est le *test* [26][48]. Une fois que le système est construit, on étudie un ensemble de cas de tests qui correspond à un ensemble d'entrées du système à vérifier, suivi éventuellement d'une séquence d'actions à exécuter, qui conduit à des valeurs de sortie attendues *a priori* du système. Un comportement non désiré ou un bogue a lieu lorsqu'une valeur de sortie diffère de la sortie spécifiée attendue. Les techniques de *test* ont montré une efficacité certaine dans la recherche de bogues dans de nombreux problèmes industriels. Néanmoins, en général, un ensemble fini de cas de tests ne permet pas toujours de couvrir tous les scénarios possibles et des erreurs peuvent rester non détectées.

Les méthodes formelles sont des techniques qui permettent de raisonner rigoureusement à l'aide de logique mathématique sur des programmes informatiques (*software systems*) ou du matériel électronique (*hardware systems*) afin de prouver leur validité par rapport à une certaine spécification. Elles sont communément appelées vérification *model-based* lorsqu'elles consistent en une modélisation du comportement d'un

[1] http://www.jbs.cam.ac.uk/media/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/

système dans un langage mathématique rigoureux et cohérent. On applique sur ces modèles des algorithmes qui explorent systématiquement tous les états du modèle. Ces méthodes permettent ainsi d'assurer une correction totale du système.

Le formalisme le plus courant utilisé pour ces techniques est celui des *systèmes de transition*, c'est-à-dire des systèmes constitués d'états et de transitions qui caractérisent les changements de l'état du système. Une suite de changements d'état du système est appelé un *chemin*. Les conditions nécessaires que doit satisfaire le comportement du système sont spécifiées dans un *langage formel* comme étant des conditions nécessaires sur les chemins. On parle souvent d'automate pour caractériser un système muni d'états initiaux et terminaux et dont les transitions sont étiquetées par une lettre d'un alphabet. À l'origine, seules des analyses qualitatives étaient réalisées. Les questions posées étaient alors du type : "le système satisfait-il cet ensemble de propriétés ?" ou bien "Ce mot est-il accepté par l'*automate* ?". Plus récemment, de nombreuses extensions de ces modèles ont été proposées et impliquent des analyses quantitatives. Ainsi,

- nous distinguons les automates pondérés (*weighted automata*) [33] qui peuvent être vus comme des automates auxquels des poids ont été rajoutés sur les transitions. Ces poids peuvent par exemple modéliser des coûts de transition entre états. En pratique, ces modèles sont souvent utilisés pour modéliser des systèmes pour lesquels on cherche à optimiser une fonction économique d'utilité.

- Nous pouvons aussi mentionner les automates temporisés [2][16] et les automates hybrides [59][90] qui sont des automates auxquels on a rajouté un ensemble fini d'horloges qui activent et désactivent certaines transitions en fonction du temps écoulé. Ces automates sont souvent utilisés pour modéliser divers systèmes dynamiques ou des systèmes temps réel [1].

- Une troisième classe de systèmes quantitatifs est caractérisée par les systèmes stochastiques. Cette classe permet entre autres de modéliser des programmes faisant intervenir des *thread* concurrents [55][126]. Les temps de séjour dans les états et/ou les passages d'un état à un autre sont soumis à des lois de probabilité.

Ces classes, extrêmement riches et non exclusives les unes des autres, sont de nos jours utilisées dans de nombreux problèmes industriels. Les systèmes probabilistes ont par ailleurs des champs d'application très variés que ce soit dans le domaine des files d'attentes, de l'analyse de fiabilité et de performance, des systèmes biologiques, des sciences sociales, de la recherche opérationnelle ou de la théorie du contrôle. Dans la suite de cette thèse, nous travaillerons essentiellement sur des systèmes probabilistes.

0.1.3 Model checking et logique temporelle

Selon le type de propriétés considérées, différentes logiques et modèles peuvent être utilisés. Prenons les deux exemples suivants : "en pilotage automatique, l'avion ne s'écrase jamais" et "le logiciel de surveillance automatique de fraudes n'accuse jamais à tort plus d'un utilisateur sur un million". Ces deux propriétés revêtent un caractère temporel. Par ailleurs, la seconde revêt également un aspect quantitatif : on s'intéresse plus à la probabilité qu'une personne soit accusée à tort qu'à savoir si une personne peut être accusée à tort ou non. Les logiques temporelles ont été développées pour exprimer de telles propriétés. Ainsi, dans les années 70, Pnueli [103], entre autres, proposa l'utilisation de la logique temporelle (linéaire) pour spécifier des programmes contenant divers processus concurrents. De nombreux travaux démontrèrent par la suite que cette logique était idéale pour exprimer des concepts comme l'exclusion mutuelle, l'absence d'interblocage (*deadlock*) ou encore l'absence de famine.

En vérification formelle, la logique LTL (*Linear Temporal Logic*) introduite par Pnueli est utilisée pour exprimer les propriétés de sûreté ("rien de mal n'arrivera") et de vivacité ("quelque chose de bien se produit inéluctablement"). Le caractère temporel de LTL peut être construit à partir des deux opérateurs modaux **X** (*next*) et U (*until*). La logique CTL (*Computational Tree Logic*) [34], initialement utilisée dans le *model checking* (voir paragraphe suivant) est une logique temporelle dont la structure du temps se présente comme dans un arbre généalogique. À partir d'un état de l'arbre, un chemin ne peut emprunter qu'un seul embranchement descendant. Cette logique est équipée d'opérateurs de quantification **A**, "pour tout chemin", et **E**, "il existe un chemin". Ces deux logiques, bien que très proches en pratique, ne sont pas équivalentes au sens où il est possible d'exprimer des propriétés dans l'une sans qu'elles le soient dans l'autre. La logique CTL* [43] permet cependant de combiner l'expressivité de ces deux logiques au détriment de la décidabilité.

Le *Model Cheking* [4] [34] [123] qui a valu à ses inventeurs E. Clarke, A. Emerson et J. Sifakis le prix Turing en 2007, est une technique automatique de vérification *model-based* qui a été utilisée de nombreuses fois dans des systèmes des technologies de l'information et de la communication (TIC) pour la détection de bugs ou de comportements non désirés. Ainsi, pour ne citer qu'un exemple, 5 erreurs non détectées au préalable furent découvertes par cette approche dans un module de contrôleur du véhicule spatial Deep Space 1 de la NASA [4].

Le problème que résout le *model checking* est le suivant :

Soit M un système de transition et ϕ une formule exprimée en *logique temporelle*. Trouver tous les états s de M tels qu'ils satisfassent ϕ (noté $M, s \models \phi$).

Les algorithmes de *model checking* fonctionnent ainsi. Un préprocesseur construit à partir d'un programme ou d'un circuit un système de transition ; puis un

contrôleur prend en entrée ce système et une propriété temporelle et vérifie si la propriété est vraie ou fausse. L'un des avantages du *model checking* est qu'il ne nécessite pas la construction d'une preuve de validité de la propriété ce qui rend cette approche particulièrement facile et rapide d'utilisation comparée à d'autres méthodes qui font intervenir des vérificateurs de preuve. Dans le cas où la propriété n'est pas satisfaite, un contre-exemple est exhibé. Ainsi, la mise en place de la vérification nécessite l'utilisation de trois ingrédients : une abstraction mathématique du système à vérifier, appelée *modèle*, des propriétés exprimées dans un langage logique et des algorithmes développés pour vérifier que l'abstraction satisfasse les propriétés données. Les algorithmes de *model cheking* réalisent une exploration exhaustive de l'espace d'états du système. Ainsi, chaque chemin est examiné et une conclusion peut être établie sur l'ensemble des chemins à l'égard de la propriété.

Le *model checking* apporte une garantie absolue de correction, du type "le système ne plante jamais". En pratique, cette notion est trop rigide et irréaliste à garantir. De nombreux systèmes, évoqués ci-dessus, sont par nature stochastiques et on cherche dans ce cas à garantir des propriétés du type "le système ne plante pas avec 99% de chance". Les aspects probabilistes sont essentiels pour :

- l'évaluation de performance de modèle. De tels systèmes possèdent en général dans leur description une information probabiliste du type délai moyen de transmission, taux d'échec d'un processus, durée de vie moyenne d'un composant électronique, etc.

- les algorithmes probabilistes. Certains algorithmes distribués comme le dîner des philosophes de Rabin ou l'élection de leader utilisent un processus de pile ou face pour éviter de manière imparable des situations de blocage.

- la modélisation de systèmes imprévisibles faisant intervenir du non-déterminisme parfois résolu de manière probabiliste.

Le *model checking* quantitatif répond à ce problème en équipant notamment les logiques temporelles d'un opérateur de probabilité. Ainsi, les propriétés à vérifier peuvent être de nature qualitative ou quantitative. Les propriétés qualitatives sont typiquement des propriétés qui doivent être garanties avec probabilité 1 comme "quelque chose de bien va se produire" ou au contraire avec probabilité 0 car ne doivent jamais avoir lieu. Les propriétés dites de *reachability* comme "cet état est-il traversé infiniment souvent ?", "cet événement se produit-il toujours ?" sont donc qualitatives. Les propriétés quantitatives sont des propriétés sur lesquelles des contraintes de probabilité ont été ajoutées sur un événement, par exemple, "la probabilité qu'un leader soit choisi en moins de 10 tours est supérieure à 0.95".

Au fil des années, de nombreux outils de *model checking* ont été développés et utilisés avec succès contribuant ainsi à la popularité de la technique : BLAST [60] adapté pour des programmes C, Java PathFinder [56] utilisé pour la vérification de programmes Java, SPIN [64], Prism [86], UPPAAL [91], Verisoft [52]. Néanmoins,

dans la plupart des applications réelles, la taille de l'espace d'états augmente exponentiellement avec le nombre de composants interagissants et la vérification devient rapidement trop difficile (ou longue) à résoudre. Ce problème est connu sous le nom de problème de l'explosion de l'espace d'états. Plusieurs méthodes ont été proposées pour combattre ce problème.

- La première méthode, introduite par Ken McMillan, [97] est celle du *model checking* symbolique. Selon cette méthode, la relation de transition est représentée de manière canonique sous forme d'un diagramme de décision binaire (DDB). Une telle représentation est souvent très compacte en comparaison d'une représentation explicite des états, bien qu'il existe des exemples pathologiques. Les algorithmes de *model checking* peuvent être construits de manière à manipuler directement les représentations BDD du système, améliorant ainsi les performances de temps et d'espace.

- Une autre possibilité est le recours à l'interprétation abstraite popularisée par Patrick et Radhia Cousot [36]. Dans cette théorie, on cherche à recueillir de l'information sur la sémantique du programme (sa structure de contrôle par exemple), sans avoir à le traiter complètement et ce, à l'aide de treillis mathématiques. Cette méthode conduit à une simplification du modèle qui se doit d'être saine au sens où une propriété vérifiée sur le modèle simplifié doit être vraie sur le modèle initial.

- Enfin, les méthodes de réduction d'ordre partiel [51] ont pour but d'identifier et réduire des entrelacs de processus concurrents indépendants ; l'idée étant que si, au regard de la propriété à laquelle on s'intéresse, exécuter "a" puis "b" ou "b" puis "a" ne change rien à l'analyse, autant éviter une redondance de l'analyse.

Dans certaines classes de problème, il est envisageable de considérer ces méthodes de réduction et de réduire la taille du modèle pour rendre l'analyse plus facile. Elles ont souvent été implémentées dans les *model checkers* cités ci-dessus et ont produit des résultats prometteurs. Cependant, elles peuvent parfois être difficiles à mettre en oeuvre car elles nécessitent des calculs intermédiaires pour identifier les classes d'équivalence sur les états ou les transitions et simplifier les analyses. Par ailleurs, la réduction peut malgré tout conduire à un modèle encore trop large.

0.1.4 Model checking statistique

Le problème de l'explosion de l'espace d'états a suscité l'intérêt pour le *model checking* statistique [134]. Cette technique nécessite un modèle exécutable du système pour estimer, à partir d'un nombre de simulations indépendantes, la probabilité qu'une propriété soit vérifiée. L'idée principale est de déduire si le système satisfait ou non une propriété en observant un ensemble de traces d'exécutions du système à

l'aide d'un moniteur [57] [10] et d'utiliser les tests d'hypothèse pour déduire que les traces fournissent une preuve statistique de la satisfaction ou de la violation de la spécification [139]. Contrairement aux approches numériques, l'approche statistique n'offre pas nécessairement un résultat correct. Néanmoins, il est possible de borner la probabilité de se tromper. Par ailleurs, cette technique consomme beaucoup moins de ressources en mémoire que les méthodes numériques. En principe, seule la valeur de l'état courant est stockée durant la phase de vérification. D'ailleurs, ce problème de stockage de données en mémoire a pour effet que certains systèmes ne peuvent être étudiés qu'à l'aide de simulations [136]. Initialement, les propriétés utilisée en *model checking* statistique étaient exprimées à l'aide de la logique PCTL à temps borné [139]. Mais cette technique peut désormais gérer des propriétés comportant des opérateurs "until" non bornés [119]. Le *model checking* statistique peut également être utilisé pour la vérification de systèmes de boîte noire [118] [133]. Enfin, les algorithmes de *model checking* statistique sont facilement parallélisables ce qui permet le passage à l'échelle de systèmes très larges.

Parmi les premières plateformes de *model checking* statistique, on peut citer notamment APMC, YMER et VESTA qui ont d'ailleurs été utilisés sur des systèmes industriels [120] [138]. Quelques *model checkers* numériques reconnus, tels que PRISM ou UPPAAL, ont également inclus un *model checker* statistique afin de traiter des modèles plus larges. Actuellement, certaines plateformes dédiées au *model checking* statistique telles que PLASMA [70] intègrent des algorithmes avancés permettant de traiter les problèmes liés aux événements rares et au non-déterminisme.

L'approche statistique a néanmoins quelques désavantages par rapport à l'approche numérique. Tout d'abord, les garanties quant à la correction de la réponse donnée par les algorithmes restent probabilistes et non exactes. De plus, l'approche statistique ne fonctionne que sur des systèmes probabilistes qui ne contiennent pas de non déterminisme. Enfin, la taille de l'échantillon grandit largement dès lors que la précision requise quant à la réponse du *model checker* devient élévée.

0.1.5 Deux défis en model checking statistique

Dans cette thèse, nous considérons deux problèmes auxquels le *model checking* statistique doit faire face et tentons d'y apporter des solutions :

- le problème inhérent aux systèmes hétérogènes qui introduit complexité et non-déterminisme dans l'analyse,
- le problème des événements rares qui impliquent par voie de conséquence l'augmentation du nombre de simulations.

Système composite

La plupart des logiciels ou appareils électroniques récents ont besoin de partager des informations entre différents systèmes. On appelle ces systèmes dont les composants

sont développés par différents fournisseurs des systèmes hétérogènes. De nos jours, la grande majorité des systèmes embarqués sont des systèmes hétérogènes. En plus de techniques d'analyse efficaces, modéliser l'expressivité d'un système hétérogène dans un formalisme muni d'une sémantique correcte est essentiel pour le développement *model-based* des systèmes embarqués. En effet, la compatibilité et l'interopérabilité de ses composants est une condition sine qua non à son bon fonctionnement.

Dans le cas des systèmes hétérogènes, la complexité croît avec le nombre de composants et leurs interactions ce qui limite le *model checking* statistique car le nombre de simulations ou leur longueur doivent être considérablement augmentés afin de garder des performances statistiques acceptables. Par ailleurs, l'augmentation des interactions entre composants introduit en général du non-déterminisme qui rend le *model checking* statistique inopérant tel quel.

Événements rares

L'autre problème majeur du *model checking* statistique réside dans la vérification ou l'estimation de propriétés qui se produisent rarement. En effet, pour pouvoir estimer (correctement) la probabilité d'un événement rare, encore faut-il l'avoir observé, et ce suffisamment souvent. Ainsi, en théorie, il faut en moyenne deux million d'essais pour observer une seule fois un événement qui se produit une fois sur deux millions.

Un autre problème vient de la variance relative de l'estimateur usuel de Montecarlo. Plus l'espérance théorique de la variable de Bernoulli est petite, plus sa variance tend vers cette moyenne. Cela implique que si le nombre n de simulations n'est pas assez grand, deux jeux de n expériences conduisent potentiellement à deux estimateurs de l'espérance très différents ; la fiabilité des résultats n'est plus garantie. Pour compenser ces problèmes de faible occurrence et de variance relative élevée, l'algorithme de Montecarlo n'offre pas d'autre alternative que d'augmenter en conséquence le nombre de simulations.

0.2 Contributions et plan de thèse

0.2.1 Plan de thèse

Les contributions de cette thèse peuvent être divisés en deux axes principaux qui sont présentés plus en détail dans les sections 0.2.2, 0.2.3 et 0.2.4. Au préalable, un aspect important de ce travail a été de formaliser le plus possible les notions utilisées et surtout de développer des algorithmes nécessitant un faible travail d'implémentation afin que les solutions proposées soient les plus générales possibles. Le chapitre 2 rappelle formellement certains éléments mathématiques et logiques induits par l'utilisation du *model checking* statistique. En particulier, puisque cette approche implique d'exécuter un modèle mathématique, il est nécessaire dans un premier temps de définir comment sont simulées les exécutions du système et le cadre prob-

abiliste sur l'espace des traces d'exécution permettant l'analyse. Nous rappelons pour cela les notions de théorie de la mesure qui permettent de définir les processus stochastiques à événements discrets et en particulier les chaînes de Markov, majoritairement utilisées ici. Nous redonnons également la sémantique de la logique temporelle utilisée, PBLTL. Enfin, nous présentons les principaux algorithmes utilisés en *model checking* statistique et qui sont traditionnellement implémentés dans la plupart des *model checker* statistiques.

Dans le chapitre 3, nous présentons des contributions originales pour le formalisme des systèmes composites dans le langage BIP. Nous en proposons une extension stochastique, SBIP, sémantiquement cohérente, qui permet le recours à l'abstraction stochastique de composants et d'éliminer le non-déterminisme. Ce double effet a pour avantage de réduire la taille du système initial en le remplaçant par un système dont la sémantique est purement stochastique sur lequel les algorithmes de *model checking* statistique sont définis.

La deuxième partie de cette thèse est consacrée à la vérification de propriétés rares dans le cadre du *model checking* statistique. Dans le chapitre 4, nous avons proposé le recours à un algorithme original d'échantillonnage préférentiel pour les modèles dont le comportement est décrit à travers un ensemble de commandes. Enfin, dans le chapitre 5, nous avons introduit les méthodes multi-niveaux pour la vérification de propriétés rares et nous avons justifié et mis en place l'utilisation d'un algorithme multi-niveau optimal. Ces deux méthodes poursuivent le même objectif de réduire la variance de l'estimateur et le nombre de simulations. Néanmoins, elles sont fondamentalement différentes, la première attaquant le problème au travers du modèle et la seconde au travers des propriétés.

0.2.2 SBIP : une extension stochastique pour la vérification statistique de systèmes composites

Le langage BIP (Behaviour, Interaction, Priority) est un langage de programmation orienté composant qui permet de représenter rigoureusement le design de systèmes embarqués. Il est associé à une chaîne d'outils développés par le laboratoire Verimag[2] permettant la compilation et l'analyse de ces systèmes embarqués. Le langage BIP permet donc de construire des abstractions de système d'ingénierie complexe sous forme d'un ensemble de composants atomiques interagissant selon des règles bien précises. Le comportement d'un composant est modélisé sous forme d'un système de transition symbolisant des actions de mise à jour des variables internes dès lors que l'une d'entre elles est réalisée. Les interactions entre composants atomiques ou groupes de composants distincts caractérisent les synchronisations entre leurs actions tandis que les priorités servent à modéliser le non-déterminisme lorsque différentes interactions sont possibles ou à définir une politique d'ordonnancement des interactions dans le système.

[2] www-verimag.imag.fr

Afin d'améliorer l'efficacité du processus de vérification, au lieu de réaliser directement une analyse sur un système entier, [6] analyse chaque composante séparément afin d'en tirer une abstraction stochastique qui représente les interactions entre les applications qui s'exécutent sur le système et qui partagent des ressources de communication et de calcul. Cette abstraction permet d'identifier et regrouper des objets communs et de simplifier la manipulation d'un grand nombre de composants par des lois de probabilité.

Dans le chapitre 3, nous présentons SBIP, une extension stochastique du formalisme BIP et de sa suite d'outils. Tandis que BIP est utilisé pour modéliser des composants dont le comportement est intrinsèquement déterministe ou non déterministe, SBIP permet de rajouter de l'incertitude dans leur comportement ou d'utiliser l'abstraction définie dans [6]. De plus, il peut être utile d'ajouter à un composant atomique des transitions caractérisant des défaillances techniques dans des études de sûreté afin d'étudier quelles sont les répercussions sur le système global. Par ailleurs, certains composants peuvent être par nature stochastiques comme un dé ou une roulette russe. Nous produisons la syntaxe et la sémantique étendue de SBIP et prouvons que la sémantique du système complet est décrite par une chaîne de Markov en montrant que le non-déterminisme issu des interactions du système est automatiquement éliminé par BIP. Ainsi, l'extension SBIP permet de produire des traces d'exécution de manière aléatoire sur un système markovien et ainsi d'utiliser le model checking statistique pour quantifier des propriétés BLTL. Le *checker* de SBIP a ensuite été équipé des algorithmes usuels de *model checking* statistique (SSP, SPRT, borne de Chernoff...) rappelés dans ce chapitre.

SBIP a été utilisé sur deux cas d'étude issus de l'industrie. Le premier cas concerne un protocole de synchronisation d'horloges qui fonctionnent en parallèle dans un système de communication. Le système est composé d'une horloge maître qui synchronisent régulièrement les autres horloges dites esclaves à travers des canaux de communication. Les composants modélisent les délais aléatoires de communication sur le réseau. Dans un premier temps, la propriété à garantir est que l'écart entre toutes les horloges soient bornés par une valeur déterminée expérimentalement. Puis, en diminuant cette borne, on cherche à évaluer le nombre moyen d'échecs de synchronisation par simulation.

Le second cas d'étude concerne une installation multimédia qui transmet en temps réel une vidéo encodée en MPEG2. On définit la gigue (*jitter*) comme la différence de délai de transmission entre des paquets choisis dans un même flux de paquets, sans prendre en compte les paquets éventuellement perdus. Dans certaines applications interactives en temps réel, la gigue peut fluctuer de manière conséquente. Par conséquent, les transmissions de voix ou de vidéos ont besoin de réseaux où la qualité de service est assurée de manière à offrir un canal de transmission de haute qualité. Ainsi, on peut supprimer les effets indésirables de la gigue en plaçant une mémoire tampon du côté du récepteur. Ce tampon de gigue provoque un délai détectable (*playout delay*) au début de la restitution du flux. Réduire ce

délai augmente la probabilité de recevoir une vidéo ou un son partiellement dégradé puisque les images ou les paquets sont consommés plus tôt. L'étude cherche à trouver un compromis entre la qualité de retransmission et un délai de tampon acceptable pour l'utilisateur.

0.2.3 L'échantillonage préférentiel pour les propriétés rares

Dans les chapitres 4 et 5, on s'intéresse à la problématique soulevée par les propriétés rarement observées lors d'une exécution arbitraire du système. La vérification de propriétés pose problème car le nombre de simulations requis pour atteindre des niveaux de confiance et de précision acceptables augmentent de manière quadratique avec la petitesse de la probabilité. Dans cette thèse, nous considérons deux angles d'attaque différents pour résoudre ce problème. Le premier, traité dans le chapitre 4, consiste à étudier le modèle et à en changer quelques caractéristiques afin de provoquer plus souvent l'occurrence de l'événement rare. Le second angle, abordé dans la prochaine section, consiste à étudier la propriété à vérifier pour construire une séquence de propriétés *emboîtées* plus simples à vérifier et à définir la probabilité de l'événement rare comme le produit des probabilités des propriétés intermédiaires.

L'échantillonnage préférentiel consiste à remplacer la mesure de probabilité μ du système par une mesure η *plus favorable* absolument continue par rapport à μ. Ensuite, à chaque trace produite *gagnante*, on associe un poids appelé ratio de vraisemblance qui est défini comme le ratio de la probabilité du chemin sous μ par sa probabilité sous η. À chaque trace *perdante*, un poids nul est associé. L'estimateur d'échantillonnage préférentiel est la moyenne des poids et présente la caractéristique d'être sans biais. Par ailleurs, dans le cas où la mesure de probabilité provoque beaucoup plus souvent l'occurrence de l'événement rare, il est vraisemblable que la variance de l'estimateur soit plus faible que la variance de l'estimateur de Monte-carlo. Ainsi, l'échantillonnage préférentiel peut être utilisé pour réduire le nombre de simulations. En fait, il existe même une mesure de probabilité qui définit un estimateur à variance nulle ce qui impliquerait qu'une seule simulation est nécessaire pour déterminer la probabilité d'intérêt. Malheureusement, connaître cette mesure optimale revient à connaître a priori la probabilité d'intérêt. Par conséquent, le challenge devient alors de trouver un changement de mesure efficace (au sens de la réduction de variance) et simple à implémenter afin de garder les avantages du *model checking* statistique.

Il existe dans la littérature scientifique de nombreux procédés qui utilisent un échantillonnage préférentiel. Malheureusement, ils sont parfois complexes à mettre en place car impliquent une grande connaissance du système à traiter ou tout simplement nécessitent des modifications des paramètres du système sur lequel l'utilisateur n'a pas forcément le plein contrôle.

Pour être efficace, il faut que la propriété soit observée plus souvent grâce à la nouvelle mesure et que la distribution des chemins qui satisfont la propriété soit la

plus proche possible dans les deux systèmes, original et nouveau, à un facteur de normalisation près. Ces deux propriétés sont satisfaites lorsque la mesure optimale est utilisée. L'algorithme d'entropie croisée de Rubinstein est un algorithme itératif qui permet de choisir parmi une famille paramétrique de distributions la mesure de probabilité (paramétrée) qui minimise la divergence de Kullback-Leibler entre cette mesure et la mesure parfaite. Ainsi, il permet de choisir une mesure candidate qui est *le plus proche possible* de la mesure parfaite, ce qui permet de penser que la variance de l'estimateur est ainsi minimisée. Bien entendu, plus la famille de distributions est riche, plus il est vraisemblable de trouver une mesure candidate dont les performances sont acceptables. Cependant, minimiser la divergence de Kullback-Leibler est en pratique ardu car l'équation d'optimisation admet rarement une solution générale. Dans le contexte des chaînes de Markov à temps discret sur un espace fini d'états, la plus grande famille paramétrique que l'on peut considérer est l'ensemble des matrices de transition définies sur cet espace d'états. Le paramètre à optimiser est donc de la taille de l'espace des éléments non nuls de la matrice initiale, ce qui implique de stocker entièrement cette matrice. Ridder a décrit un algorithme utilisant cette matrice pour vérifier des propriétés de type $\phi\, U\, \psi$ avec ϕ et ψ atomiques et a montré que la précision obtenue était la meilleure possible. Néanmoins, sur un système extrêmement large, le problème de stockage de la matrice limite le recours à cette solution.

Dans ce chapitre, nous considérons des systèmes stochastiques à événements discrets qui décrivent le comportement d'un modèle sous forme d'un ensemble de commandes probabilistes $C_k = (g_k, f_k, h_k)$ où la garde g_k est un prédicat sur les variables du modèle, f_k une fonction des variables du système vers les réels positifs et h_k une fonction de mise à jour des variables du système. Chaque mise à jour décrit une transition qui ne peut être prise que si la garde est vraie. Lorsque plusieurs gardes sont vraies, une mise à jour h_k est choisie avec une probabilité égale au taux f_k divisée par la somme des taux des commandes dont la garde est vraie. Les commandes gouvernent un ensemble de transitions et non pas une transition individuelle entre deux états du système. Ainsi, les modèles sont décrits d'une manière beaucoup plus compacte et sont de facto en pratique plus lisibles.

Notre contribution est d'avoir proposé pour ce type de systèmes un algorithme d'entropie croisée qui permet de sélectionner une *bonne* mesure de probabilité pour réaliser un estimateur d'échantillonage préférentiel. À chaque commande k, nous assignons une valeur λ_k (par défaut égale à 1). La probabilité de prendre une transition possible k est $\lambda_k f_k$ divisé par le produit scalaire des taux des commandes dont la garde est vraie et du sous-vecteur correspondant de λ. La famille de mesures que l'on considère est donc paramétrée par ce vecteur λ auquel on ajoute une contrainte linéaire (la somme de ses éléments reste constante après chaque itération). Nous montrons que si l'algorithme converge, il converge vers un paramètre λ optimal et unique. Cet algorithme offre en général des performances moins bonnes que l'algorithme de Ridder du simple fait que notre famille de distributions est stricte-

ment incluse dans la sienne. Néanmoins, nous offrons des pistes pour améliorer ses performances et comment initier l'algorithme. Nous discutons également des problèmes usuels rencontrés lors d'une simulation par échantillonnage préférentiel, notamment en ce qui concerne les intervalles de confiance induits par cette technique et le type de propriétés BLTL pour lesquelles il est envisageable de trouver une distribution parfaite.

0.2.4 Méthode multi-niveaux pour les propriétés rares

Dans le chapitre 5, nous cherchons à vérifier la propriété globale sans avoir à modifier le modèle en utilisant les méthodes multi-niveaux. Soit A un événement (rare) dont on cherche à connaître la probabilité et $A_0 \supset A_1 \supset \cdots \supset A_{n-1} \supset A_n = A$ une suite emboîtées d'événements. Le théorème de Bayes et l'emboîtement des événements permettent de réécrire la probabilité $Pr(A)$ de la manière suivante :

$$Pr(A_n) = Pr(A_n \cap A_{n-1}) = Pr(A_n \mid A_{n-1}) Pr(A_{n-1}) \tag{1}$$

Puis, par itération et en posant $Pr(A_0 \mid A_{-1}) = Pr(A_0)$,

$$Pr(A) = \prod_{k=0}^{n} Pr(A_k \mid A_{k-1}) \tag{2}$$

Par construction, ces probabilités conditionnelles sont supérieures à $Pr(A)$. L'idée des méthodes multi-niveaux est donc d'estimer chacune de ces probabilités conditionnelles séparément mais pas indépendamment et ce, par des techniques de *rebranchage*. Dans le problème original introduit dans [75], la probabilité que A se réalise correspondait à la probabilité qu'un neutron traverse un seuil l. Ainsi, les événements A_k avec $0 \leq k \leq n$ correspondaient à des seuils intermédiaires qui devaient nécessairement être traversés si l'on voulait observer A. Il existe plusieurs façons de conduire des simulations multi-niveaux mais l'idée générale est toujours la même. En supposant que $Pr(A_0)$ est connu ou que les traces commencent à partir d'un état initial unique, on commence par estimer $Pr(A_1 \mid A_0)$ à l'aide de simulations. Celles-ci sont arrêtées dès lors qu'elles ont satisfait une condition d'arrêt (ici, atteindre A_1 ou "mourir"). L'estimateur de $Pr(A_1 \mid A_0)$ est simplement donné par le nombre de traces ayant atteint A_1 divisé par le nombre de traces exécutées. Les traces qui n'atteignent pas A_1 sont détruites et remplacées par une trace ayant atteint A_1. Ensuite, la même procédure est répétée pour estimer $Pr(A_2 \mid A_1)$, les traces étant lancées de où elles ont été arrêtées ou rebranchées. Par conséquent, cette procédure permet d'éviter une réduction progressive des simulations partant d'un niveau intermédiaire.

Dans cette thèse, nous avons adapté cette technique pour la transposer dans le cadre du *model checking* statistique. Mais, dans notre cas, contrairement à un problème physique où les variables peuvent naturellement caractériser une notion de

distance ou de quantité, les variables sont souvent booléennes ou bien ne permettent pas toujours d'identifier facilement des "niveaux" comme décrits précédemment. Néanmoins, en général, les propriétés temporelles à vérifier sont exprimées sous forme d'un automate fini et il est parfois possible d'identifier un ensemble d'états intermédiaires que l'automate doit nécessairement visiter afin d'atteindre un état terminal correspondant à la satisfaction de la propriété. Prenons l'exemple classique du problème du dîner des philosophes. Lorsque le philosophe part de l'état où il pense, ses mains sont vides et il doit obligatoirement s'emparer d'une première baguette puis d'une deuxième baguette avant de pouvoir accéder au bol de riz communal et ainsi satisfaire sa faim. Atteindre ces états intermédiaires est équivalent à avoir satisfait une propriété intermédiaire. Il faut donc construire une séquence de propriétés dont chacune est une restriction de la précédente :

$$\phi = \phi_n \Leftarrow \phi_{n-1} \Leftarrow \cdots \Leftarrow \phi_1 \Leftarrow \phi_0 \equiv \top \tag{3}$$

Nous donnons dans cette thèse un ensemble de clés pour décomposer une propriété le plus finement possible en décomposant simultanément les propositions atomiques quand elles contiennent une notion naturelle de niveau et certains opérateurs temporels.

Afin d'implémenter cette idée dans un *model checker*, nous définissons une fonction score qui assigne à chaque trace satisfaisant une propriété une valeur réelle croissante, disons une valeur entière entre $\{0, \cdots, n\}$ pour simplifier. Ainsi, si deux traces ont un score différent, la trace ayant un plus haut score satisfait plus de propriétés intermédiaires. Une trace avec un score maximal est une trace qui satisfait la propriété globale. Cette fonction peut être vue comme une généralisation de la fonction de Bernoulli utilisée dans la simulation de Montecarlo qui ne prend que les deux valeurs possibles 0 ou 1. Idéalement, pour que l'algorithme soit le plus efficace possible, à nombre de simulations et de niveaux fixés, il faudrait que les probabilités conditionnelles soient toutes égales pour minimiser la variance de l'estimateur. Or, cela dépend essentiellement de la décomposition de la formule. Par ailleurs, même si la granularité de la fonction score est fine au point qu'il existe un ensemble de seuils tels que les probabilités conditionnelles soient (presque) égales, le connaître est un problème nettement plus ardu. Nous présentons dans cette thèse différents algorithmes qui présentent avantages et inconvénients par rapport aux autres. Le premier s'appuie sur une fonction score qui associe simplement aux propriétés $(\phi_k)_{0 \leq k \leq n}$ les entiers correspondant à leur indice. Le nombre n de niveaux à franchir est fixé ainsi que le nombre N de simulations pour estimer chaque probabilité conditionnelle. Les autres algorithmes s'appuient sur une fonction score heuristique. En effet, un grand nombre de traces satisfont le même nombre de sous-propriétés. Le but de la fonction score heuristique est de discriminer "intelligemment" deux traces ayant satisfait le même nombre de propriétés intermédiaires et par la même occasion d'augmenter la granularité de la fonction score. La fonction score idéale devrait être ordonnée de la même manière que la fonction $Pr(\varphi \mid \omega)$ mais connaître cette fonction pour tout ω du

système revient à connaître la probabilité que l'on cherche à estimer. Nous justifions donc le recours à des heuristiques simples qui permettent néanmoins de raffiner considérablement la fonction score. Reprenons l'exemple des philosophes et supposons que la propriété d'intérêt soit la suivante : "Le philosophe mange avant que le système ait emprunté 30 transitions". Un score plus élevé sera accordé à une trace ω telle qu'au bout de 5 transitions, le philosophe a pour la première fois deux baguettes en main, plutôt qu'à celles où il les tient en main pour la première fois au bout de 28 transitions. Augmenter le nombre de niveaux possibles permet d'augmenter la valeur moyenne des probabilités conditionnelles et, à nombre de niveau fixé, de se donner plus de chance de trouver un ensemble de seuils avec une probabilité conditionnelle presque égale. Ces deux phénomènes combinés permettent de réduire la variance de l'estimateur. Nous présentons enfin deux algorithmes, qui choisissent les niveaux à-la-volée de façon optimale, c'est-à-dire tels que les probabilités conditionnelles soient égales. Les arguments en entrée deviennent donc une probabilité conditionnelle fixe entre chaque niveau et un nombre de simulations à prendre en compte entre chaque niveau. Le dernier algorithme proposé est une optimisation de cette variante qui consiste à choisir une probabilité conditionnelle maximale à chaque fois. Nous avons comparé empiriquement ces différents algorithmes sur un exemple de dîner de 150 philosophes.

Dans la partie d'ouverture de ce chapitre, nous enrichissons la classe de systèmes pour lesquels un algorithme de simulation multi-niveaux peut être utilisé en l'étendant à des systèmes décrits par une chaîne de Markov cachée.

Chapter 1

Summary of the thesis

1.1 Introduction

1.1.1 Context

Information and Communication Technologies have taken a considerable place in our society in recent decades. They penetrate all areas, public and private, of our daily life. Most houses, offices, means of transport, companies and administrations are equipped with more or less modern electronic devices, have an Internet connection, a telephone or a television. The advent of the Internet and the social networks have transformed human, economic and industrial relations. Consequently, companies use or develop increasingly complex advanced technologies to offer ever more solutions, accuracy and speed in the execution of required tasks.

These complex engineering systems imply strong interactions between various computer programs, electronic components, etc. However, if it was possible in the past for a company to supervise within the same structure the design activities of embedded systems from the specification to the implementation, it is nowadays impossible because of the growing explosion of complexity induced by the need for compatibility of system elements. These components are thus developed in general by various teams who work independently from each other but who possibly agree in the component specification that they will have to use and share. In addition, some of these systems, known as critical, aim to perform tasks of which failure can have dramatic economic, human or environmental repercussions, for example in aeronautics or telecommunications. This growing complexity consequently involves a serious increase in bugs or non-desired behaviours. For example, between 1985 and 1987, at least five people died from a massive overdose of radiation due to safety failures and software errors of a radiotherapy machine, Therac-25 [95]. In 1996, an error of conversion of a 64-bit floating point into a 16-bit integer value caused the crash of Ariane-5, only a few seconds after its launch [96]. In August 2003, a critical situation of concurrency between various parts of the North-American electrical network caused in a few hours the breakdown of 256 power plants [3].

While societal consequences are not really quantifiable, the economic repercussions have been evaluated several times. In 2002, a study conducted by the National Institute of Standards and Technology, an agency of the United States Department of Commerce, reported that software bugs cost the U.S. economy 59 billion dollars annually [67]. More recently, a study of the University of Cambridge, published in 2012, gave the figure of 312-billion-dollar loss for the worldwide economy[1]. Reliability and understanding of the behaviour of these computer systems has thus become a key challenge in the design process of systems.

1.1.2 Formal Methods

There does not exist at the moment any universal and automatic method to find and correct all the bugs of software or electronic devices. The verification techniques aim to significantly reduce their frequency. They are used in system design in order to guarantee that a system satisfies some properties. These properties prescribe what the system must be able to do or not do and a system which satisfies all the specified properties is said to be "correct".

The most common technique to check the correctness of a system is testing [26] [48]. Once the system is built, one studies some tests which correspond to a set of inputs of the system to check, possibly followed by a sequence of executions, which lead to expected values of system outputs. A non-desired behaviour or a bug occurs whenever an actual output differs from the specified output. Testing has shown a certain effectiveness in the search of bugs in many industrial problems. Nevertheless, in general, a finite set of test case does not cover all the possible scenarios and errors can remain not detected.

Formal methods are verification techniques reasoning rigorously using mathematical logic on computer programs or electronic devices in order to check their correctness with respect to a software or hardware specification. They are commonly called model-based techniques whenever they consist of modelling the system behaviour in a rigorous and consistent mathematical language. One applies to these models algorithms which systematically explore all the states of the model. These methods thus ensure a full correctness of the system.

The most basic formalism for models is that of a transition system, that consists of states and transitions that characterise changes of the state of the system. A path is a sequence of state changes. Requirements of the system behaviour are specified, through a formal language, as necessary conditions on paths. We talk about automata whenever the system has initial and terminal states and the transitions are labelled by a letter of an alphabet. Originally, only qualitative analysis was considered. The standard questions were of the following type: "does the system satisfy this set of properties?" or "is this word accepted by the automaton?". More

[1] http://www.jbs.cam.ac.uk/media/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/

recently, extensions of these models were proposed and imply quantitative analysis. Hence,

- We distinguish weighted automata [33] which may be seen as automata on which weights have been added over the transitions. These weights may model for example the cost involved whenever executing a transition. As an example, these models are useful to model systems for which one desires to optimise a utility function in economics.
- We also mention timed automata [2][16] and hybrid systems [59] [90] which are automata to which one adds a finite set of clocks that enable or disable some transitions according to time. These automata model various dynamic and real-time systems [1].
- A third class of quantitative systems is characterized by stochasticity. This class allows inter alia modelling programs with concurrent threads [55][126]. Holding times in states and/or next-state transitions are subject to probability distributions.

These wide classes are nowadays used in many industrial problems. In the rest of this thesis, we will mainly work on probabilistic systems. Probabilistic systems have various domains of application in queueing theory, reliability and performance analysis, biological systems, social sciences, operation research and control theory.

1.1.3 Model checking and temporal logic

Depending on the type of properties, various logics and models can be used. Let us consider the two following examples: "in autopilot mode, the plane never crashes" and "the automatic fraud detection system does not accuse wrongly more than one user in a million". Both properties take on a temporal aspect. In addition, the second takes on a quantitative aspect: the point is not to tell if a user is wrongly accused or not but to know the probability that the system wrongly accuses a user. Temporal logics were developed to express such properties. Thus, in 1977, Pnueli [103], amongst others, proposed to use (linear) temporal logic to specify programs containing several concurrent processes. It has been proved thereafter that this logic was ideal to express concepts like mutual exclusion, absence of deadlock or starvation.

In formal verification, Linear Temporal Logic (LTL) introduced by Pnueli is used to express safety ("nothing bad will occur") and liveness properties ("something good will eventually happen"). The temporal aspect of LTL is built from two modal operators $\mathbf{X}$, "next", sometimes denoted $\bigcirc$, and $\mathcal{U}$ (until). Computational Tree Logic (CTL) [34], initially used in *model checking* (see next paragraph) is a temporal logic in which the time structure may be regarded as in a (descending) family tree. Starting from a state of the tree, a path can take only one downward branching.

This logic is equipped with quantification operators **A**, "for any path", and **E**, "there exists a path". These logics are not equivalent since it is possible to express properties in one without them being expressed in the other. CTL* logic [43] however combines the expressivity of these two logics at the cost of decidability.

Model Checking [4][34][123] for which its inventors E. Clarke, A. Emerson and J. Sifakis were awarded the Turing prize in 2007, is an automatic technique of model-based verification that has been used many times in information and communication technology for bug and non-desired behaviour detection. Just to name one example, 5 previously undiscovered errors were identified by this approach in an execution module of the Deep Space 1 spacecraft NASA controller [4].

The model checking problem can be summarised in the following way:

Let M be a transition system and ϕ a formula expressed in temporal logical. Find all states s of M such that they satisfy ϕ (denoted $M, s \models \phi$).

Model checking works as follows. From a program or a circuit, a preprocessor builds a transition system; a controller then takes as an input this system and a temporal property and checks if the property is true or false. An advantage of model checking is that it does not require a correctness proof of the property, which makes the approach particularly easy and fast compared to other methods that use a proof checker. If the property is not satisfied, a counterexample is displayed. So, model checking set-up requires three ingredients: (i) a mathematical abstraction of the system to check, called a model, (ii) properties expressed in a logical language and (iii) algorithms developed to check that the abstraction satisfies the given properties. Model checking algorithms carry out an exhaustive exploration of the system state space but generally avoid checking paths. Nevertheless, a conclusion is established on the path space with regard to the property.

Model checking gives an absolute guarantee of correctness such that "the system never crashes". In practice, this concept is too rigid and unrealistic to guarantee. Many systems, evoked above, are by nature stochastic and in this case, one seeks to guarantee properties of type: "the system does not crash with 99% of certainty". Probabilistic aspects are essential for:

- Model performance evaluation. In general, such systems have in their description probabilistic information of the mean transmission delay, time failure rate of a process, average lifetime of an electronic component, etc.

- Probabilistic algorithms. Some distributed algorithms like Rabin's randomised dining philosophers or leader election protocol use a two-sided-coin process to certainly avoid deadlock situations.

- Non-determinism properties of systems are sometimes solved in a probabilistic way.

Quantitative model checking handles this problem by equipping the temporal logics with a probability operator. The properties can be of qualitative or quantitative nature. Qualitative properties are properties which must be guaranteed with probability 1 to always occur or dually with probability 0 because they must never happen. Properties such as "is this state visited infinitely often?", "does this event always occur?" are thus qualitative. Quantitative properties are properties on which probabilistic constraints are added to an event. For example, "the probability that a leader is elected in less than 10 rounds is higher than 0.95".

Over the years, many model checkers have contributed to the popularity of the technique: BLAST [60] for C programs, PathFinder [56] for Java program verification, SPIN [64], PRISM [86], UPPAAL [91], Verisoft [52]. Nevertheless, in most real applications, the size of the state space grows exponentially with respect to the number of components and the verification quickly becomes too difficult (or long) to solve. This is called the state (space) explosion problem. Several methods have been proposed to overcome this problem.

- One method, introduced by Ken McMillan, [97] is that of symbolic model checking. According to this method, the transition relation is canonically represented in the form of a binary decision diagram (BDD). Such a representation is often very compact in comparison to an explicit representation of states, although it is proved that there can be pathological examples. Model checking algorithms may be constructed that directly manipulate the BDD representation of the system, gaining performance in both space and time.

- Another possibility is the recourse to abstract interpretation popularised by Patrick and Radhia Cousot [36]. In this theory, one seeks to collect pertinent information on the program semantics (its control flow for example), without having to treat it completely, using mathematical lattices. This method aims to get a simplified sound model: a property, checked true on the simplified model, must be true on the original model.

- Lastly, partial order reduction methods [51] identify and reduce interleavings of independent concurrent processes; the idea being that if, with respect to the property of interest, executing "A" then "B" or "B" then "A" does not impact the analysis, one can avoid some analysis redundancy.

Abstraction and symmetry reduction may make certain classes of systems tractable. These techniques are often implemented in standard model checkers and produce promising results. However, their implementation may be difficult because they require intermediate calculations to identify the classes of equivalence on the states or the transitions. In addition, in spite of the reduction, the reduced model can still be very large.

1.1.4 Statistical Model Checking

In the context of Continuous-Time Markovian systems (CTMC), the verification of time bounded properties relies on transient analysis. Efficient numerical techniques, such as uniformisation [73], are known for several decades and are the core of probabilistic model checkers. However, the stationary distribution vector of the CTMC requires solving a system of linear equations or a system of differential equations.

The state space explosion problem has prompted the development of statistical model checking [134, 61]. This technique employs an executable model of the system to estimate the probability of a property from a number of independent simulations. The key idea is to deduce whether or not the system satisfies the property by observing some of its executions with a monitoring procedure [57] [10] and use hypothesis testing to infer whether the samples provide a statistical evidence for the satisfaction or violation of the specification [139]. In contrast to a numerical approach (NMC), a simulation-based solution does not guarantee a correct result. However, it is possible to bound the probability of making an error using Chernoff-Hoeffding's inequality [61]. Simulation-based methods are known to be far less memory than numerical ones and are sometimes the only option [136]. In principle, only the current state value is stored during the verification phase. Moreover, the verification time for NMC increases exponentially with the state space size. This implies that statistical approaches are also often less time consuming especially if no very high accuracy is required and if the formula time bound is not too large [136]. Initially, the properties were expressed in bounded time PCTL logic [139] but statistical model checking has been extended to handle unbounded "until" properties [119]. Moreover, the technique may be used for checking black-box probabilistic systems [118] [133]. Finally, the statistical model checking algorithms are easily parallelisable, which is useful for the scalability of large systems.

Early statistical model checking platforms include APMC, YMER and VESTA, with the latter two having been applied to industrial systems [120] [138]. Well-established numerical model checkers, such as PRISM and UPPAAL, are now also including statistical model checking engines to cope with larger models. Current high performance platforms dedicated to statistical model checking, such as PLASMA [70], incorporate sophisticated algorithms to handle rare events and non-determinism.

A statistical approach has nevertheless a few drawbacks. First of all, the guarantees of correctness given by the algorithms remain probabilistic and non-exact. Moreover, the statistical approach only works on stochastic systems without non-determinism. Lastly, high confidence and accuracy substantially increase the number of observations.

1.1.5 Two challenges in statistical model checking

In this thesis, we consider two problems that statistical model checking must cope with and try to provide solutions:

- heterogeneous systems naturally introduce complexity and non-determinism into the analysis,
- rare properties pose problems because they are difficult to observe, and so to quantify, though often highly relevant to system performance (e.g., system failure is usually required to be rare).

Composite System

Most recent software or electronic devices need to share information with other devices. One calls a system whose components are developed by various suppliers a heterogeneous system. Nowadays, most embedded systems are heterogeneous systems. In addition to effective analysis techniques, modelling the expressivity of a heterogeneous system in a semantically correct formalism is essential for the model-based development of embedded systems. Indeed, the compatibility and the interoperability of its components are a sine qua non condition for performance.

In the case of heterogeneous systems, the number of components and their interactions are limiting factors on the number and length of simulations that can be conducted and hence on the accuracy of the statistical estimates. Moreover the increase of component interactions adds non-determinism that makes statistical model checking challenging.

Rare Events

The other major challenge of statistical model checking lies in the estimate of properties which seldom occur. Indeed, estimating (correctly) the probability of a rare event implies observing it sufficiently often. In theory, one needs in average two million tests to see only once an event that occurs once on two million.

Another problem comes from the relative variance of the standard Monte Carlo estimator. As the mean of the Bernoulli variable tends to zero, its variance tends to its average. That implies that if the number of simulations n is not large enough, two samples of n experiments give potentially two very different estimations of the mean; the reliability of the results is not guaranteed any more. To compensate the weak occurrence and relative variance problems, the Monte Carlo simulation only suggests to consequently increase the number of simulations.

1.2 Contributions and Outline

The contributions of this thesis are of two types that are presented in more detail in sections 1.2.2, 1.2.3 and 1.2.4. As a preliminary, an important aspect of this work was to formalize the introduced concepts and to develop algorithms requiring a low implementation effort so that the suggested solutions remain the most feasible in practice.

1.2.1 Outline

Chapter 2 formally points out certain mathematical elements and logics induced by the use of statistical model checking. In particular, since this approach implies to carry out a mathematical model, it is worth defining (i) how system executions are drawn and (ii) the probabilistic framework of the execution trace space adapted for the analysis. We briefly recall concepts of measure theory in order to introduce discrete-event stochastic processes and, more especially, Markov chains. We also give the semantics of the PBLTL temporal logic. Lastly, we present the main algorithms of statistical model checking, normally implemented in statistical model checkers.

In chapter 3, we present original contributions for the formalism of composite systems in BIP language. We propose SBIP, a stochastic extension and define its semantics. SBIP allows the recourse to the stochastic abstraction of components and eliminate the non-determinism. This double effect has the advantage of reducing the size of the initial system by replacing it by a system whose semantics is purely stochastic, a necessary requirement for standard statistical model checking algorithms to be applicable.

The second part of this thesis is devoted to the verification of rare properties in statistical model checking. In chapter 4, we present a state-of-the-art *importance sampling* algorithm for models described by a set of guarded commands. Lastly, in chapter 5, we motivate the use of *importance splitting* for statistical model checking and set up an optimal splitting algorithm. Both methods pursue a common goal to reduce the variance of the estimator and the number of simulations. Nevertheless, they are fundamentally different, the first tackling the problem through the model and the second through the properties.

1.2.2 SBIP: a stochastic extension for statistical verification of composite systems

The BIP formalism is a component-based framework supporting rigorous design of embedded systems. BIP is supported by an extensible toolset developed by the Verimag research center[2] which includes tools for checking correctness, for model transformations and for code generation. BIP is currently equipped with a series of runtime verification and simulation engines. BIP allows the construction of complex, hierarchically structured models from atomic components characterized by their behaviour and their interfaces. Such components are transition systems enriched with variables. Transitions are used to move from a source to a destination location. Each time a transition is taken, component variables may be assigned new values. Atomic components are composed by layered application of interactions and priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used both to select amongst possible

[2] www-verimag.imag.fr

interactions and to steer system evolution e.g. to express scheduling policies.

In order to improve the verification process, instead of performing a system analysis, [6] separately analyses each component so as to draw its stochastic abstraction that represents the interactions between executing applications of the system which share communication and computation resources. This abstraction allows to find and gather common objects and to simplify the handling of a large number of components by probability distributions.

In chapter 3, we present SBIP, a stochastic extension of the BIP formalism and toolset. While BIP is used to model components for which behaviour is intrinsically deterministic or nondeterministic, SBIP permits to add uncertainty in their behaviour or to make use of the abstraction defined in [6]. Moreover, it can be useful to include in an atomic component transitions characterising execution platform assumptions or faults in safety analysis to study the effects on the entire system. Finally, some components are inherently stochastic like a die or a Russian roulette. We give the syntax and the extended semantics of SBIP and prove that the semantics of the entire system describes a Markov chain by showing that the non-determinism resulting from the interactions is automatically eliminated by BIP. This extension allows us to produce execution traces of the designed system in a random manner on a Markovian system and thus to use standard statistical model checking algorithms (SSP, SPRT, Chernoff bound,...) to quantify BLTL properties.

We use SBIP on two industrial case studies [17]. The first case study concerns a clock synchronization protocol running within a distributed heterogeneous communication system (HCS). This model is composed by a "master" clock which synchronises all the "slave" clocks through stochastic communication channels. These components model communication delays over the network. Initially, the property to guarantee is that the difference between all the clocks is bounded by some value. Then, by decreasing this bound, one seeks to evaluate the average failure per execution.

The second case study relates to a multimedia video player set-up. In multimedia literature, it has been shown that some quality degradation is tolerable when playing MPEG2-coded video. In fact, a loss under two consecutive frames within a second can be accepted. One defines the jitter as the difference in end-to-end one-way delay between selected packets in a flow with any lost packets being ignored. In some interactive real-time applications, the jitter can fluctuate in a consequent way. As a result, video and sound transmissions require high quality of service networks. One can overcome the undesirable jitter effects with buffers. This jitter buffers causes a playout delay at the beginning of the flow restitution. If one starts to reduce the playout delay, the playout buffer fill level decreases, which induces some probability of failure since the player starts to consume the frames sooner. The goal of the analysis is to enable a designer to choose a trade-off amount of quality degradation that reduces the buffer size and does not imply a big playout delay.

1.2.3 Importance Sampling for Rare Properties

In chapters 4 and 5, we focus on problems caused by properties rarely observed during an arbitrary execution of the system. The verification of such properties poses problems because the number of simulations necessary to get acceptable confidence and accuracy increases quadratically with rarity. In this thesis, we consider two angles of attack to solve this problem. The first, importance sampling, addressed in chapter 4, consists of studying the model and changing some of its parameters in order to more often provoke the rare event occurrence. The second angle, described in the next section, consists of studying the property to verify to build a nested sequence of properties, that are easier to check, and define the probability of the rare event as the product of the probabilities of these intermediate properties.

Importance sampling [77] consists of replacing probability measure μ of the system by a more favorable probability measure η, absolutely continuous with μ. Then, one assigns to each successful path a weight, called likelihood ratio, defined as the ratio of the probability of the path under μ by its probability under η. A zero weight is assigned to a failed path. The importance sampling estimator is the average of the weights and is unbiased. In addition, if the probability measure causes more often the rare event occurrence, it is likely that the variance of the estimator is lower than the variance of the Monte Carlo estimator. So, importance sampling can be used to reduce the number of simulations. In fact, there exists a notional measure that defines a zero-variance estimator. It would imply that only one simulation is necessary to determine the probability of interest. Unfortunately, knowing this perfect measure returns means knowing a priori the probability of interest. Consequently, the challenge then becomes to find an effective change of measure (so that reduces the variance), simple enough to implement in order to keep the advantages of statistical model checking.

A lot of procedures exploiting importance sampling have been proposed in the scientific literature [121, 74]. Unfortunately, they are often intricate to set up because they require a great knowledge of the system or more simply require modifications of system parameters over which a user does not have necessary control.

To be effective, the property must be more often observed with the new measure and the distribution of the successful paths (with respect to the property) must be the closest possible in both systems, original and new, up to a factor of normalisation. Both conditions are satisfied whenever the perfect measure is used. Rubinstein's cross-entropy minimising framework is an iterative procedure that finds a probability measure that minimises the Kullback-Leibler divergence with respect to the perfect measure, among a parametric family of distributions [117]. Hence, it allows to choose a candidate measure that is the closest possible to perfect measure, which is a good heuristic with regard to variance minimization. Of course, the bigger the family is, the more likely it is to find an efficient candidate measure. However, minimizing the Kullback-Leibler divergence is in practice difficult because the optimization equation

rarely has a closed-form solution. In the case of discrete-time Markov chains over a finite state space, the richer set of parametric importance sampling distributions is the set of all transition probability matrices over that state space. The size of the parameter to optimise is thus equal to the number of non-zero elements in the original matrix, which implies to entirely store this matrix. In [112], Ridder introduced an algorithm using this matrix to check properties of type $\phi\, U\, \psi$, with ϕ and ψ atomic, and showed that the obtained accuracy was the best possible [113]. The cost of considering all transitions makes Ridder's algorithm of little practical value, however, given that numerical algorithms have the same requirement but do not incur the cost of simulation.

In this chapter, we consider stochastic discrete-event systems such that the model behaviour is described with a set of probabilistic commands $C_k = (g_k, f_k, h_k)$ where the guard g_k is a predicate over system variables, f_k a function of the system variables to the non-negative reals and h_k an update function of the variables. Each update describes a transition which can be taken only if the guard is true. Whenever several guards are true, an update h_k is selected with a probability equal to the rate f_k divided by the sum of the command rates with true guard. The commands control a set of transitions and not an individual probability transition between two states of the system. Thus, the models are described in a much more compact way and are in practice more convenient.

Our contribution is to propose for command systems an alternative cross-entropy algorithm to carry out an importance sampling estimator. We assign to each command k a value λ_k (by default equal to 1) and define λ as the vector of parameters λ_k. The probability of executing enabled transition k is $\lambda_k f_k$ divided by the scalar product of the command rates with true guard and the corresponding sub-vector of λ. The considered set of measures is thus parametrised by this vector λ to which we add a linear constraint (the sum of its elements remains constant after each iteration). We show that if the algorithm converges, it converges to an optimal and unique parameter λ. This algorithm offers in general lower performance than Ridder's algorithm because we consider a restricted set of distributions. However, we give tracks to improve its performances and how to initiate the algorithm. We also discuss some standard problems encountered with an importance sampling simulation, in particular the lack of associated confidence intervals, and the type of BLTL properties for which there exists a perfect distribution.

1.2.4 Important Splitting for Rare Properties

In chapter 5, we check the overall property without modifying the model using importance splitting methods [75, 29]. Let A be a (rare) event that one seeks to know the probability and $A_0 \supset A_1 \supset \cdots \supset A_{n-1} \supset A_n = A$ a nested sequence of events. Bayes' theorem and the event nestedness allows us to rewrite probability

$Pr(A)$ as follows:

$$Pr(A_n) = Pr(A_n \cap A_{n-1}) = Pr(A_n \mid A_{n-1})Pr(A_{n-1}) \tag{1.1}$$

Then, by iteration and with $Pr(A_0 \mid A_{-1}) = Pr(A_0)$,

$$Pr(A) = \prod_{k=0}^{n} Pr(A_k \mid A_{k-1}) \tag{1.2}$$

By construction, these conditional probabilities are greater than $Pr(A)$. The main idea of the splitting methods is thus to estimate each conditional probabilities separately (but not independently) using sampling/branching techniques. In the (earliest) application of importance splitting of [75], the probability of event A occurrence corresponded to the probability that neutrons would pass through certain shielding materials and reach a threshold l. The events A_k with $0 \leq k \leq n$ corresponded to intermediate thresholds that must be necessarily crossed in order to observe A. There have been many different implementations of this idea, but a generalised procedure is as follows.

Assuming a set of increasing levels is defined as above, a number of simulations are generated, starting from a distribution of initial states that correspond to reaching the current level. The procedure starts by estimating $Pr(A_1 \mid A_0)$, where the distribution of initial states for A_0 is usually given (often a single state). Simulations are stopped as soon as they reach the next level; the final states becoming the empirical distribution of initial states for the next level. Simulations that do not reach the next level (or reach some other stopping criterion) are discarded. The estimator of $Pr(A_1 \mid A_0)$ is estimated by the number of simulation traces that reach A_1, divided by the total number of traces started from A_0. Simulations that reached the next level are continued from where they stopped. To avoid a progressive reduction of the number of simulations, the generated distribution of initial states is sampled to provide additional initial states for new simulations, thus replacing those that were discarded.

In this thesis, we apply this idea to statistical model checking. But, in our case, contrary to physical systems where distances and quantities may provide a natural notion of level that can be finely divided, variables may be Boolean and temporal properties may not contain an obvious notion of level. Nevertheless, in model checking, temporal properties are expressed in the form of a finite-state machine and it is sometimes possible to identify a set of intermediate states that the automaton must necessarily visit in order to reach a terminal state equivalent to the property satisfaction.

Let us consider the classical model checking dining philosophers problem. Philosophers think and occasionally wish to eat from a communal bowl. From a think state, a philosopher must independently pick up one and then a second chopstick before eating. To reach these intermediate states is equivalent to have satisfied an intermediate property. It is thus necessary to define a set of levels based on a sequence

of temporal properties that have the logical characteristic:

$$\phi = \phi_n \Leftarrow \phi_{n-1} \Leftarrow \cdots \Leftarrow \phi_1 \Leftarrow \phi_0 \equiv \top \tag{1.3}$$

We give in this thesis keys to decompose a temporal logic property the most finely possible by decomposing atomic propositions when they contain a natural notion of level and simultaneously some temporal operators.

In order to implement this idea in a statistical model checker, we define a score function assigning higher values to paths which more nearly satisfy the overall property. For more convenience, assume for now that these values are integers between $\{0, \cdots, n\}$. If two traces have a different score, the trace having a higher score satisfies more intermediate properties. A trace with a maximum score is a trace which satisfies the overall property. Standard statistical model checking can be seen as a degenerate case of splitting, in the sense that computing $Pr(\omega \models \phi)$ is equivalent to compute $Pr(S(\omega) \geq 1)$ where S is the Bernoulli distributed model checking function only taking values 0 or 1. Ideally, given a fixed number of simulations and levels, it is desirable to choose levels that make the conditional probabilities all the same to minimise the relative variance of the final estimate. However, that primarily depends on the decomposition of the formula. In addition, even if the granularity of the score function is so fine that there exists a set of thresholds such that the conditional probabilities are (almost) equal, knowing these is a more difficult problem. In this thesis we present several importance splitting algorithms for statistical model checking, which have various advantages and disadvantages. The first is based on a score function that simply associates to properties $(\phi_k)_{0 \leq k \leq n}$ their index integer. The number n of levels to be crossed is fixed as well as the number N of simulations to estimate each conditional probability. The other algorithms are based on a heuristic score function. Indeed, the logical levels may be too coarse; a large number of traces satisfy the same sub-properties. The goal of the heuristic function score is to discriminate in a clever way two traces having satisfied the same intermediate properties and to increase the granularity of the simple score function. The ideal score function should be ordered as function $Pr(\varphi \mid \omega)$ but knowing this function for all path ω is equivalent to know the probability that one seeks to estimate. We thus justify the recourse to simple heuristics that nevertheless refine considerably the score function. Let's return to the dining philosopher example and assume that the property of interest is the following: "The philosopher will eat before the system executes 30 transitions". A higher score will be assigned to a path such that the philosopher holds two chopsticks for the first time after 5 transitions than to a path such that the philosopher holds two chopsticks for the first time after 28 transitions. To increase the number of levels increases the mean value of conditional probabilities and, at fixed number of levels, to give more chance to find a set of thresholds evenly spaced in terms of probability. Both conditions reduce the estimator variance. We eventually present two algorithms in which levels are chosen on-the-fly in an optimal way, i.e. such as the conditional probabilities are equal. The input

arguments thus become a fixed conditional probability between each level and as usual a number of simulations to be taken into account between each level. The last proposed algorithm proposed is an optimization that consists of choosing the maximum conditional probability at each iteration. We empirically compared these various algorithms on a 150-dining-philosophers problem.

In an opening section of this chapter, we extend the class of systems for which a splitting algorithm may be used by considering a system described by an hidden Markov model.

Chapter 2

Background about Statistical Model Checking

This chapter introduces various notions and fixes notations that will be used throughout this thesis. In order to introduce statistical model checking, three points need to be clarified:

- Which systems do we use and how do we simulate traces?
- In which logic do we express properties?
- Which statistical algorithms do we use?

In the first section, we recall some notions of measure theory to define stochastic discrete-event systems and introduce Markov chains. The second section is a brief introduction of probabilistic bounded linear temporal logic. We pose in this section the notations concerning properties and their semantics. In the third section, we only focus on a few notions ubiquitous in statistical model checking like Monte Carlo estimation, confidence intervals and Chernoff bound.

2.1 Stochastic Discrete-Event System

We present the class of systems on which statistical model checking may be used. For this purpose, we introduce stochastic process which is a "process whose evolution we can analyse successfully in terms of probability" [41] [42].

2.1.1 Measure theory

We first recall a few notions of measure theory necessary to formally define a stochastic process.

Definition 1 (Sigma-algebra) *Let Ω be a set. A sigma-algebra on Ω is a non-empty collection $\mathcal{F}$ of subsets of Ω closed under complement and countable union operations, ie:*

- $\mathcal{F} \neq \emptyset$
- $\forall A \in \mathcal{F},\ A^c \in \mathcal{F}$
- $\forall n \in \mathbb{N},\ A_n \in \mathcal{F} \Rightarrow \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}$

Sigma-algebras are a key element of measure and probability theory axiomatized by Andrey Kolmogorov [82]. As we only consider in this thesis real random variables, the implicit sigma-algebra is the Borel set on $\mathbb{R}$, denoted $\mathcal{B}(\mathbb{R})$, which is the sigma-algebra generated by open intervals of $\mathbb{R}$.

Definition 2 (Measurable space) *A measurable space is a couple $(\Omega, \mathcal{E})$ with Ω a set and $\mathcal{E}$ a sigma-algebra over Ω.*

Definition 3 (Non-negative Measure) *Let $(\Omega, \mathcal{E})$ be a measurable space. A non-negative measure μ is a function defined on $\mathcal{E}$ to $\mathbb{R}$ such that:*

- $\mu(\emptyset) = 0$,
- *for all $E \in \mathcal{E}$, $\mu(E) \geq 0$,*
- *for all countable collections $\{E_i\}_{i\in\mathbb{N}}$ of pairwise disjoint sets in $\mathcal{E}$,*

$$\mu\left(\bigcup_{i\in\mathbb{N}} E_i\right) = \sum_{i\in\mathbb{N}} \mu(E_i) \tag{2.1}$$

Definition 4 (Absolute continuity of measures) *Let f and f' be two non-negative measures over the same measurable space $(\Omega, \mathcal{A})$. f is said absolutely continuous with f' if $f(A) = 0$ for every set A for which $f'(A) = 0$.*

A non-negative measure Pr such that $Pr(\Omega) = 1$ is called a probability measure. A measure is so a function which assigns a quantitative value (a distance, a probability, ...) to each subset of a universe Ω.

Definition 5 (Probability space) *A probability space is a triplet $(\Omega, \mathcal{E}, Pr)$ with $(\Omega, \mathcal{E})$ a measurable space and Pr a probability measure. Ω is called the sample space and an element of $\mathcal{E}$ an event.*

2.1.2 Stochastic process

Now equipped with a sigma-algebra and a probability measure, we can formally define random variables:

Definition 6 (Random variable) *Let $(\Omega, \mathcal{E}, Pr)$ be a probability space and $(F, \mathcal{F})$ a measurable space. A random variable X is a function from Ω to F such that the reciprocal sigma-algebra by X of the sigma-algebra $\mathcal{F}$ is included in $\mathcal{E}$, ie:*

$$\forall B \in \mathcal{F},\ X^{-1}(B) \in \mathcal{E} \tag{2.2}$$

As we manipulate variables evolving randomly with time, it is necessary to generalize the notion of random variables which leads to the following definitions:

Definition 7 (Stochastic process) *Let $(\Omega, \mathcal{E}, Pr)$ be a probability space, $(S, \mathcal{F})$ be a measurable space and T be a totally ordered set. A stochastic process $\mathcal{X} = \{X_t \mid t \in T\}$ is a family of random variables defined on $(\Omega, \mathcal{E}, Pr)$, with each random variable X_t having range S. S is called the state space.*

A particular class of stochastic process is the class of stochastic discrete-event system that is "a stochastic process that can be thought of as occupying a single state for a non-zero duration of time before an event causes an instantaneous state transition to occur." [134]

The non zero duration of time guarantees that the change state caused by the triggering of an event is discrete. The index set T represents time and can be an instant of $\mathbb{R}^+$, a date or a point at some instant. When only the order of events may matter along a timeline, by convenience, we use $T = \mathbb{N}$ and $k \in T$ must be interpreted as the index of an observation.

Definition 8 (Trajectory) *Let $\omega \in \Omega$. A trajectory $X(\omega)$ from T to S is the application: $t \longmapsto X_t(\omega)$.*

Concretely, a trajectory is a set of observations of the random variables $X_t \in \mathcal{X}$ where X_t is the random variable representing the chance of observing the stochastic process $\mathcal{X}$ at time t. We assume that only a single state can be occupied at time t. In a simulation-based approach, a trajectory is a set of observations recorded and indexed by T. Moreover, we assume that we do not record an infinite number of events.

It can be represented as a finite sequence $\omega = (s_0, t_0)(s_1, t_1)\cdots$ with $s_i \in S$ and $t_i \in T^*$. s_0 is called the initial state of the system. At time $\sum_{k=0}^{i-1} t_i$, the system enters into state s_i and stays in this state for t_i time units. If time is not relevant with respect to a property, we simply denote $\omega = s_0 s_1 \cdots$. We define the length of a trajectory $|\omega|$ as the number of recorded transitions.
A prefix $\omega_{\leq\tau}$ of a trajectory $\omega = (s_0, t_0)(s_1, t_1)\cdots$ is a sequence

$$\omega_{\leq\tau} = (s_0, t_0)\cdots(s_{k-1}, t_{k-1})(s_k, t_k)$$

such that $\tau = \sum_{i=0}^{k} t_i$ and $k \leq |\omega|$. We denote $\tau_j = \sum_{i=0}^{j} t_i$ and $Path(\omega_{\leq \tau})$ the set of trajectories with common prefix $\omega_{\leq \tau}$.

The idea of statistical model checking is to quantify by estimation the probability of logical properties based on a sample of trajectories. For that purpose, it is necessary to define a probability measure over sets of trajectories for a stochastic discrete event system.

Proposition 1 *If S and T are measurable, the set of trajectories of a stochastic discrete event system is measurable.*

The probability measure on sets of trajectories for a stochastic discrete-event system can be defined and built recursively using a holding time distribution with probability density function $f(.; \omega_{\leq \tau})$ and a next-state distribution $p(.; \omega_{\leq \tau}, t)$.

This definition of the probability measure μ for a stochastic discrete-event system provides, up to the complexity of implementation of f and p in a simulation engine, a convenient way to sample trajectories which are the main elements of statistical model checking. For any stochastic discrete-event system, given an initial state s_0, we first sample with respect to $f(.; s_0)$ a holding time t_0 and then a new state s_1 with respect to $p(.; (s_0, t_0), \tau_0)$. At the next step, a holding time t_1 is chosen with respect to $f(.; (s_0, t_0)s_1)$ and a next state s_2 with respect to $p(.; (s_0, t_0)(s_1, t_1), \tau_1)$, etc...

2.1.3 Markov and semi-Markov process

We next recall the definition of a common stochastic discrete-event system: Markov process.

Definition 9 (Filtration) *Let $(\Omega, \mathcal{F}, Pr)$ be a probability space and T a totally ordered index set. A filtration $\{\mathcal{F}_t \mid t \in T\}$ is a weakly increasing collection of sigma-algebras on Ω bounded above by $\mathcal{F}$.*

Given a stochastic process $\mathcal{X} = \{X_t \mid t \in T\}$, the natural filtration induced by this process is the filtration where $\mathcal{F}_t$ is generated by all values of X_s up to time $s = t$, ie: $\langle \{X_s^{-1}(A) \mid s \leq t,\ A \in S\} \rangle$.

Definition 10 (Markov property) *Let $(\Omega, \mathcal{F}, Pr)$ be a probability space, T a totally ordered index set and $(S, \mathcal{E})$ be a measurable space. An S-valued stochastic process $\mathcal{X} = \{X_t \mid t \in T\}$ is said to possess the Markov property with respect to its natural filtration if, for each $A \in \mathcal{E}$ and each $(s, t) \in T^2$ with $s < t$, $Pr(X_t \in A \mid \mathcal{F}_s) = Pr(X_t \in A \mid X_s)$.*

Definition 11 (Markov process) *A Markov process is a stochastic process which satisfies the Markov property with respect to its natural filtration.*

In practise, it means that the process is Markovian if and only if the next state of the system depends on the current state and does not depend on the previous states. If the behaviour of the future states are also not dependent of the time of observation, the Markov process is said to be *time homogeneous*. In this case, for all prefix $\omega_{\leq\tau} = (s_0, t_0) \cdots (s_k, t_k)$,

$$\mu(Path(\omega_{\leq\tau})) = \mu(Path((s_k, 0))) \tag{2.3}$$

which is equivalent to $f(t_k + t; \omega_{\leq\tau}) = f(t; s_k)$ and $p(\cdot; \omega_{\leq\tau}, t) = p(\cdot; s_k)$

A semi-Markov process is a stochastic process in which the next state is dependent on the current state (and not the previous) and on the time spent in this state. In this case, for all prefix $\omega_{\leq\tau} = (s_0, t_0) \cdots (s_k, t_k)$,

$$\mu(Path(\omega_{\leq\tau})) = \mu(Path((s_k, t_k))) \tag{2.4}$$

2.1.4 Markov chains

Markov processes are also called in the literature Markov chains. We next give a more explicit definition of two kinds of Markov chains: discrete-time and continuous-time Markov chains.

Definition 12 (Discrete-time Markov Chain (DTMC)) *An S-valued discrete-time Markov chain is a (discrete-time) Markov process where state space S is non-empty and countable.*

Formally, given an initial distribution $\mu_0 : S \to [0, 1]$ such that $\sum_{s\in S} \mu_0(s) = 1$, a S-valued Markov process $\mathcal{X} = \{X_k\}_{k\in\mathbb{N}}$ and $P : S \times S \to [0, 1]$ the transition probability function such that:

$$\forall s \in S, \quad \sum_{s'\in S} P(s, s') = 1. \tag{2.5}$$

$\mathcal{X}$ is a discrete-time Markov chain if for all $k \in \mathbb{N}$ and $(s_0, \cdots, s_{k+1}) \in S^{k+2}$:

$$P(X_{k+1} = s_{k+1} \mid X_k = s_k, X_{k-1} = s_{k-1}, \cdots, X_0 = s_0) = P(X_{k+1} = s_{k+1} \mid X_k = s_k) \tag{2.6}$$

In the case where the Markov chain is time-homogeneous, the previous property takes the following form for all $k \in \mathbb{N}$ and $(s, s') \in S^2$:

$$P(X_{k+1} = s' \mid X_k = s) = P(X_1 = s' \mid X_0 = s) \tag{2.7}$$

It follows that a discrete-time Markov chain has for each state a geometric holding time distribution. However, we usually do not need the holding time distribution to simulate a discrete-time Markov chain; the next state distribution is only required. In the rest of thesis, the Markov chains will be considered as time-homogeneous.

In general, if S is finite, the function P is represented as a matrix of size $|S|^2$ where $P(s_i, s_j)$ (also denoted p_{ij} when no confusion is possible) is the probability that the stochastic process occupies state s_j at time $k+1$ knowing that it occupies state s_i at time k.

Time in DTMC proceeds in discrete time. They model accurate systems of time units (for example, a clock ticking in an electronic device) or may be used when transition times are not an issue with respect to the property to check.

Definition 13 (Continuous-time Markov Chain) *An S-valued continuous-time Markov chain is a discrete-time Markov chain, equipped with an exit-rate function $r : S \to \mathbb{R}^+$, in which residence time in state s is exponentially distributed with rate $r(s)$.*

Transition rate $Q(s, s')$ from s to s' is given by the relation: $Q(s, s') = P(s, s')r(s)$. Then, the probability to take an outgoing transition from s in $[0, t]$ is $1 - e^{-r(s)t}$ and the probability to move from s to s' in $[0, t]$ is $P(s, s')(1 - e^{-r(s)t})$.

In automata theory, we deal with labeled transition systems. For this purpose, let $\mathbb{B}$ be a set of atomic propositions and $\Sigma = 2^{\mathbb{B}}$.

Definition 14 (Labeled (discrete-time) Markov Chain) *A Labeled Markov Chain (LMC) $\mathcal{S}$ is a tuple $\langle S, Act, \iota, \pi, L^M \rangle$ where,*

- *S is a finite set of states,*
- *Act is a finite set of actions,*
- *$\iota : S \to [0, 1]$ the initial states distribution such that $\sum_{s \in S} \iota(s) = 1$,*
- *$\pi : S \times Act \times S \to [0, 1]$ the probability transition function such that for each $s \in S$ and $a \in Act$, $\sum_{s' \in S} \pi(s, a, s') = 1$, and*
- *$L^M : S \to \Sigma$ a state labeling function.*

A labeled Markov chain is deterministic (DLMC) iff:

- $\exists s_0 \in S$ such that $\iota(s_0) = 1$, and
- $\forall s \in S$ and $a \in Act$, there exists at most one $s' \in S$ such that $\pi(s, a, s') > 0$.

We write $\pi(s_i, a, s_j) = \pi_{ij}$, the transition from s_i to s_j as $s_i \xrightarrow{a, \pi_{ij}} s_j$ for $s_i, s_j \in S, \pi_{ij} \in [0, 1]$ and $a \in Act$.

2.2 Probabilistic Bounded Linear Time Logic

Temporal logic has been introduced in the late 1950s by Arthur Prior [104] and can be interpreted as a system of rules for representing and reasoning on propositions expressing a temporality, for example: "I never lie", "I will eventually lie", "I will lie until I get elected". The work of Rescher and Urquhart [111] pioneered temporal logic in the context of program verification and a complete formalization known as *Linear Temporal Logical* has been proposed by Pnueli in 1977 [103] for specifying properties of systems with temporal operators **F** and **G** (respectively readable as follows: *eventually* and *always*). In what follows, they will be respectively denoted $\Diamond$ and $\Box$ for notation convenience.

In this thesis, properties are specified with Probabilistic Bounded Linear Temporal Logic (PBLTL) which is a formalism for describing stochastic temporal properties. These formulas are interpreted by a monitor over a *model* of system producing traces $\omega = (s_0, t_0)(s_1, t_1)\cdots$.

2.2.1 BLTL semantics

We first recap Bounded Linear Temporal Logic and then define its probabilistic extension.

Let the set of state variables SV be a finite set of real-valued variables of a stochastic model $\mathcal{M}$.

A Boolean predicate over SV is a formula of the form: $x \sim v$ with $x \in SV$, $\sim \in \{<, >, =\}$ and $v \in \mathbb{R}$. A LTL formula is built on a finite set of Boolean predicates over SV using Boolean connectives and temporal operators. The syntax of this logic is given by the following grammar:

$$\phi \ ::= \ x \sim v \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid \neg\phi \mid \bigcirc\phi \mid \phi \mathcal{U} \phi$$

Then, temporal operators *eventually* and *always* are defined as follows:

- $\Diamond\phi = \mathbf{True}\,\mathcal{U}\phi$
- $\Box\phi = \neg\Diamond\neg\phi$

We define the semantics of this logic with respect to the executions of the system. $\omega \models \phi$ means that execution ω satisfies property ϕ and $(\omega, j) \models \phi$ means that the trace starting at a position $j \geq 0$ in ω satisfies ϕ. We denote the value of state variable x in ω at step j by $V(\omega, j, x)$. For a state formula:

$$(\omega, j) \models x \sim v \quad \text{iff} \quad V(\omega, j, x) \sim v$$

For the Boolean connectives:

$$\begin{array}{lll} (\omega, j) \models \phi_1 \vee \phi_2 & \text{iff} & (\omega, j) \models \phi_1 \text{ or } (\omega, j) \models \phi_2 \\ (\omega, j) \models \phi_1 \wedge \phi_2 & \text{iff} & (\omega, j) \models \phi_1 \text{ and } (\omega, j) \models \phi_2 \end{array}$$

For the temporal operators:

$$\begin{array}{lll}(\omega,j)\models\bigcirc\phi & \text{iff} & (\omega,j+1)\models\phi\\ (\omega,j)\models\Box\phi & \text{iff} & \forall i\geq j,\ (\omega,i)\models\phi\\ (\omega,j)\models\Diamond\phi & \text{iff} & \exists i\geq j,\ (\omega,i)\models\phi\\ (\omega,j)\models\phi_1\mathcal{U}\phi_2 & \text{iff} & \exists k\geq j,\ (\omega,k)\models\phi_2 \text{ and } \forall i,\ j\leq i<k,\ (\omega,i)\models\phi_1\end{array}$$

In statistical model checking, traces must be of finite duration in order to simulate a sample and we consequently add bounds on the temporal operators.

The semantics of bounded temporal operators is so:

$$\begin{array}{lll}(\omega,j)\models\Box^{\leq t}\phi & \text{iff} & \forall i,\ j\leq i\leq t,\ (\omega,i)\models\phi\\ (\omega,j)\models\Diamond^{\leq t}\phi & \text{iff} & \exists i,\ j\leq i\leq t,\ (\omega,i)\models\phi\\ (\omega,j)\models\phi_1\mathcal{U}^{\leq t}\phi_2 & \text{iff} & \exists k,\ j\leq k\leq t,\ (\omega,k)\models\phi_2 \text{ and } \forall i,\ j\leq i<k,\ (\omega,i)\models\phi_1\end{array}$$

2.2.2 PBLTL semantics

Definition 15 (PBLTL property) *A Probabilistic Bounded LTL property is a formula of the form $Pr_{\sim\gamma}\phi$ where Pr is a probabilistic operator, $\sim\in\{<,=,>\}$, $\gamma\in[0,1]$ is a probability and ϕ is a BLTL formula.*

The system $\mathcal{S}$ satisfies a PBLTL property $Pr_{\sim\gamma}\phi$, denoted $\mathcal{S}\models Pr_{\sim\gamma}\phi$, if and only if an arbitrary execution of the system satisfies BLTL property ϕ with probability $\sim\gamma$. It has been proved that this problem is well-defined in [137, 142], since each $\omega\models\phi$ is decidable on a finite prefix of ω and since, due to the existence of a probability measure over the traces of $\mathcal{S}$, the set of all (non-zero) executions of $\mathcal{S}$ that satisfy a given BLTL formula is measurable [137].

It implies, given a stochastic discrete-event system $\mathcal{S}$ and its underlying probability measure μ, that the probability to satisfy a BLTL formula ϕ exists and is given by $\mu\{\omega\in\Omega\mid\omega\models\phi\}$.

2.3 Statistics and Statistical Model Checking

Two theorems are widely used in statistical model checking: the strong law of large numbers and the central limit theorem.

2.3.1 Recall and notations

Whenever it exists, we denote $\mu=E[X]$ the expected value or the mean of random variable X. By definition, for discrete probability measure with (countable but not necessarily finite) range $\{x_1,x_2,\cdots\}$,

$$E[X]=\sum_{k=1}^{+\infty}x_k Pr(X=x_k) \tag{2.8}$$

and for continuous measure f over Ω,

$$E[X] = \int_{\Omega} xdf \tag{2.9}$$

Whenever it exists, we denote $\sigma^2 = V(X)$ the variance of X. By definition, the variance equals $E[(X-\mu)^2]$ and we call standard deviation the square root σ of the variance.

The mean value measures the location of a random variable while the standard deviation measures its spread.

In this thesis, we will use extensively concepts based on the following notions and theorems.

Theorem 1 (Strong Law of large numbers) *Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of independent and identically distributed (iid) random variables such that $E[|X_0|] < +\infty$ with mean value $E[X]$. We have:*

$$Pr\left(\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i = E[X]\right) = 1 \tag{2.10}$$

Theorem 2 ((Lindeberg-Lévy) Central Limit Theorem) *Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of independent and identically distributed (iid) random variables with mean value $E[X_n] = \mu < +\infty$ and variance $V(X_n) = \sigma^2 < +\infty$ and denote:*

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{2.11}$$

and

$$Z_n = \sqrt{n}\frac{\bar{X}_n - \mu}{\sigma} \tag{2.12}$$

Z_n converges in distribution to the standard gaussian distribution $Z \sim \mathcal{N}(0,1)$.

Moreover, the Berry-Esseen theorem claims that the speed of convergence is at least of order $1/\sqrt{n}$ if $E[|X|^3] < +\infty$.

Definition 16 (Confidence interval) *Let $\mathcal{X} = (X_n)_{n\in\mathbb{N}}$ be a random sample from a probability distribution X with statistical parameters γ. A confidence interval for the parameter γ, with confidence level or confidence coefficient $1-\alpha$, is an interval with random endpoints $(u(\mathcal{X}), v(\mathcal{X}))$, determined by the pair of random variables $u(\mathcal{X})$ and $v(\mathcal{X})$, with the property: $Pr(\gamma \in [u(\mathcal{X}), v(\mathcal{X})]) = 1-\alpha$*

Gaussian confidence interval for mean estimation of X is based on the central limit theorem and the property becomes:

$$Pr\left(\bar{x}_n - z_\alpha\frac{\sigma}{\sqrt{n}} < \gamma < \bar{x}_n + z_\alpha\frac{\sigma}{\sqrt{n}}\right) \geq 1-\alpha \tag{2.13}$$

with z_α the Gaussian quantile of $1 - \alpha/2$.

The confidence interval is a random variable which may depend on other parameters of X. If these parameters are unknown, one usually uses approximate confidence intervals. The symbol $=$ in the previous property is then replaced by $\simeq$ or $\geq$. As in general the standard deviation of X is unknown, an approximation of σ is used in the Gaussian confidence interval.

Moreover, if X follows a Bernoulli distribution, its variance is smaller than $1/4$ and an exact conservative interval may be built, based on the Chernoff bound:

Theorem 3 (Chernoff bound) *Let $(X_k)_{1 \leq k \leq n}$ be a sequence of independent Bernoulli distribution with same mean parameter γ. For all $\epsilon \geq 0$,*

$$Pr\left(|\gamma - \frac{1}{n}\sum_{k=1}^{n} X_k| \geq \epsilon\right) \leq 2e^{-2\epsilon^2 n} \tag{2.14}$$

This bound is useful in practise because allows to compute, given a confidence level and a precision level (half of the length of a Chernoff confidence interval), a minimal n such that equation 2.14 holds.

2.3.2 Statistical Model Checking

Consider a Markov Chain $\mathcal{S}$ and a BLTL property ϕ. *Statistical model checking* refers to a series of simulation-based techniques that can be used to answer two questions:

1. **Qualitative :** Is the probability for $\mathcal{S}$ to satisfy ϕ greater or equal to a certain threshold θ?

2. **Quantitative :** What is the probability for $\mathcal{S}$ to satisfy ϕ ?

The behaviour of the model with respect to property ϕ is modeled by a Bernoulli random variable Z of parameter γ. Given a path ω, such a variable can only return 2 values: 1 if $\omega \models \phi$ and 0 otherwise. In probabilistic words,

$$Pr[Z(\omega) = 1] = \gamma \quad \text{and} \quad Pr[Z(\omega) = 0] = 1 - \gamma \tag{2.15}$$

This Bernoulli random variable has a mean value $E[Z] = \gamma$ and a variance $V(Z) = \gamma(1 - \gamma)$.

Monte Carlo simulation

A sample of Z is obtained by running simulations of the model and by checking property ϕ on the resulting traces.

In our context, variable Z_i is associated with one simulation of the system. The outcome for Z_i, denoted z_i, is 1 if the i^{th} simulation satisfies ϕ and 0 otherwise. Traces are generated independently, so by the stong law of large numbers,

$$\frac{1}{n}\sum_{i=1}^{n} z_i \xrightarrow{n\to+\infty} \gamma \tag{2.16}$$

It means that we can approximate γ by taking the average of a finite number of realizations of Z. Given a sequence of n independent random variables Z_i of distribution Z, the Monte Carlo estimator $\tilde{\gamma}_n = \frac{1}{n}\sum_{i=1}^{n} Z_i$ converges to γ. Moreover, the estimator is unbiased and its variance decreases to zero when n tends to the infinity:

$$E\left[\tilde{\gamma}_n\right] = \gamma \quad \text{and} \quad V(\tilde{\gamma}_n) = \frac{V(Z)}{n} = \frac{\gamma(1-\gamma)}{n} \tag{2.17}$$

Then, we use the central limit theorem or the Chernoff bound to construct a confidence interval.

Qualitative Answer using Statistical Model Checking

The main approaches [134] [118] proposed to answer the qualitative question are based on *hypothesis testing*. Let $\gamma = Pr(\omega \models \phi)$, to determine whether $\gamma \geq \theta$, we can test $H : \gamma \geq \theta$ against $K : \gamma < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error.

The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error) is less or equal to α (respectively, β).

A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [134] [130] for details). A solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : \gamma \geq p_0$ against $H_1 : \gamma \leq p_1$. We now very briefly sketch an hypothesis testing algorithm that is called the *sequential probability ratio test (SPRT in short)* [130].

In SPRT, one has to choose two values A and B $(A > B)$ that ensure that the strength (α, β) of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(Z_i = z_i \mid \gamma = p_1)}{Pr(Z_i = z_i \mid \gamma = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}, \tag{2.18}$$

where $d_m = \sum_{i=1}^{m} z_i$. The idea behind the test is to:

$$(1) \text{ accept } H_0 \text{ if } \frac{p_{1m}}{p_{0m}} \geq A \quad \text{or } (2) \text{ accept } H_1 \text{ if } \frac{p_{1m}}{p_{0m}} \leq B \tag{2.19}$$

The SPRT algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied; the algorithm terminates with probability 1 [130]. This has the advantage of minimizing the number of simulations. In his thesis [134], Younes proposed a logarithmic based algorithm SPRT that given p_0, p_1, α and β implements the sequential ratio testing procedure. When one has to test $\theta \geq 1$ or $\theta \geq 0$, it is better to use *Single Sampling Plan* (SSP) (see [134][92][118] for details) that is another hypothesis testing algorithm whose number of simulations is pre-computed in advance. In general, this number is higher than the one needed by SPRT, but is it known to be optimal for the above mentioned values. More details about hypothesis testing algorithms and a comparison between SSP and SPRT can be found in [92].

Quantitative Answer using Statistical Model Checking

In [61, 89] Peyronnet et al. propose an estimation procedure to compute the probability γ for $\mathcal{S}$ to satisfy ϕ. Given a *precision* δ, Peyronnet's procedure, which we call PESTIMATION, computes a value $\tilde{\gamma}$ for γ such that:

$$|\tilde{\gamma} - \gamma| \leq \delta \quad \text{with } \textit{confidence } 1 - \alpha. \tag{2.20}$$

The procedure is based on the *Chernoff-Hoeffding bound* [62]. Let $Z_1 \dots Z_m$ be m discrete random variables with a Bernoulli distribution of parameter γ associated with m simulations of the system. Recall that the outcome for each of the Z_i, denoted z_i, is 1 if the simulation satisfies ϕ and 0 otherwise. Let $\tilde{\gamma}_m = (\sum_{i=1}^{m} b_i)/m$, then Chernoff-Hoeffding bound [62] gives $Pr(|\tilde{\gamma}_m - \gamma| > \delta) < 2e^{-2m\delta^2}$. As a consequence, if we take

$$m \geq \frac{1}{2\delta^2} \log(\frac{2}{\alpha}), \tag{2.21}$$

then

$$Pr(|\tilde{\gamma}_m - \gamma| \leq \delta) \geq 1 - \alpha \tag{2.22}$$

Observe that if the value $\tilde{\gamma}_m$ returned by PESTIMATION is such that $\tilde{\gamma}_m \geq \theta - \delta$, then $\mathcal{S} \models Pr_{\geq\theta}(\phi)$ with confidence $1 - \alpha$.

Playing with Statistical Model Checking Algorithms

The efficiency of the above algorithms is characterized by the number of simulations needed to obtain an answer. This number may change from execution to execution and can only be estimated (see [134] for an explanation). However, some generalities are known. For the qualitative case, it is known that, except for some situations, SPRT is always faster than SSP. PESTIMATION can also be used to solve the qualitative problem, but it is always slower than SSP [134]. If θ is unknown, then a good strategy is to estimate it using PESTIMATION with a low confidence and then validate the result with SPRT and a strong confidence.

Chapter 3

SBIP, a stochastic formalism for component-based systems

3.1 Introduction

Expressive modelling formalisms with sound semantical basis and efficient analysis techniques are essential for successful model-based development of embedded systems. While expressivity is needed for mastering heterogeneity and complexity, sound and rigorous models are mandatory to establish and reason meaningfully about system correctness and performance at design time.

The BIP (Behaviour-Interaction-Priority) formalism [9] is an example of a highly expressive, component-based framework with rigorous semantical basis. BIP allows the construction of complex, hierarchically structured models from atomic components characterized by their behaviour and their interfaces. Such components are transition systems enriched with variables. Transitions are used to move from a source to a destination location. Each time a transition is taken, component variables may be assigned new values, possibly computed by C functions. Atomic components are composed by layered application of interactions and priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used both to select amongst possible interactions and to steer system evolution e.g. to express scheduling policies.

BIP is supported by an extensible toolset [**?**] which includes tools for checking correctness, for model transformations and for code generation. Correctness can be either formally proven using invariants and abstractions, or tested using simulation. For the latter case, simulation is driven by a specific middleware, the BIP engine, which allows to generate and explore execution traces corresponding to BIP models. Model transformations allow to realize static optimizations as well as special transformations towards distributed implementation of models. Finally, code generation targets both simulation and implementation models, for different platforms and operating systems support (e.g., distributed, multi-threaded, real-time, etc.). The tool

has been applied to a wide range of academic case studies as well as to industrial applications [19].

BIP is currently equipped with a series of runtime verification [44] and simulation engines. While those facilities allow us to reason on a given execution, they cannot be used to assess the overall correctness of the entire system. This chapter presents SBIP, a stochastic extension of the BIP formalism and toolset. Adding stochastic aspects permits to model uncertainty in the design e.g., by including faults or execution platform assumptions. Moreover, it allows to enhance the simulation engine of BIP with statistical inference algorithms in order to reason on properties in a quantitative manner. Stochastic BIP relies on two key features.The first is a stochastic extension of the syntax and the semantics of the BIP formalism. This extension allows us to specify stochastic aspects of individual components and to produce execution traces of the designed system in a random manner.

The second feature is a Statistical Model Checking (SMC) [118, 134, 80, 105, 20, 141, 81] engine (SBIP) that, given a randomly sampled finite set of executions/simulations of the system, can decide with some confidence whether the system satisfies a given property. The decision is taken through either a Monte Carlo (that estimates the probability) [53], or an hypothesis testing algorithm [134, 118] (that compares the probability to a threshold). To guarantee termination of each simulation, these properties must be evaluated on bounded executions. Nevertheless, SMC has been recently extended to cover unbounded properties. Extension such as those introduced in [135, 119, 81, 105] rely on an interleaving of estimation of probabilistic operator or a non stochastic exploration of the state space–two techniques known to be costly. In our work, we consider systems with finite life, hence bounded properties, expressed in Bounded Linear Temporal Logic (BLTL) are sufficient. Observe that the techniques in [135, 119, 81, 105] can be easily implemented in SBIP.

As it relies on sampling executions of a unique distribution, SMC can only be applied to pure stochastic systems i.e., systems without non-determinism. The problem is that most component-based design approaches exhibit non-determinism due to interleaving semantics, usually adopted for parallel execution of components and their interactions. SBIP allows to specify systems with both non-deterministic and stochastic aspects. However, the semantics of such systems will be purely stochastic, as explained hereafter. Syntactically, we add stochastic behaviour to atomic components in BIP by randomizing individual transitions. Indeed, it suffices to randomize the assignments of variables, which can be practically done in the C functions used on transition. Hence, from the user point of view, dealing with SBIP is as easy as dealing with BIP.

We illustrate SBIP on several case studies that cannot be handled with existing model checkers for stochastic systems [87, 70]. The presentation restricts to the analysis of a clock synchronization protocol [6] and an MPEG decoder [107]. Other examples can be found in [7].

Structure of the chapter. Section 3.2 presents some background on BIP. The stochastic extension for BIP and its associated semantics are introduced in Section 3.3. Section 3.4 describes the statistical model checking procedure as well as its implementation in SBIP. In section 3.5 we describe practical utilization of the SBIP tool. Finally, Sections 3.6 and 3.7 present experiments and conclusion, respectively.

3.2 BIP

The BIP framework, introduced in [9], supports a methodology for building systems from *atomic components*. It uses *connectors*, to specify possible interactions between components, and *priorities*, to select amongst possible interactions.

3.2.1 Atomic Component

Atomic components are finite-state automata that are extended with variables and ports. Variables are used to store local data. Ports are action names, and may be associated with variables. They are used for interaction with other components. States denote control locations at which the components await for interaction. A transition is a step, labeled by a port, from a control location to another. It has associated a guard and an action that are, respectively, a Boolean condition and a computation defined on local variables. In BIP, data and their related computation are written in C. Formally:

Definition 17 (Atomic Component in BIP) *An atomic component is a transition system extended with data* $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$*, where:*

- (L, P, T) *is a transition system, with* $L = \{l_1, l_2, \ldots, l_k\}$ *a set of control locations,* P *a set of ports, and* $T \subseteq L \times P \times L$ *a set of transitions,*
- $X = \{x_1, \ldots, x_n\}$ *is a set of variables over domains* $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ *and for each* $\tau \in T$ *respectively,* $g_\tau(X)$ *is a guard, a predicate on* X*, and* $X' = f_\tau(X)$ *is a deterministic update relation, a predicate defining* X' *(next) from* X *(current) state variables.*

For a given valuation of variables, a transition can be executed if the guard evaluates to true and some *interaction* involving the port is enabled. The execution is an atomic sequence of two micro-steps: 1) execution of the interaction involving the port, which is a synchronization between several components, with possible exchange of data, followed by 2) execution of internal computation associated with the transition. Formally:

Definition 18 (Semantics of atomic component) *The semantics of an atomic component* $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$ *is a transition system* (Q, P, T_0) *such that*

- $Q = L \times \mathbf{X}$ *where* $\mathbf{X}$ *denotes the set of valuations* v_X *of variables in* X.
- T_0 *is the set including transitions of the form* $((l, v_X),\ p, (l', v'_X))$ *such that* $g_\tau(v_X) \wedge v'_X = f_\tau(v_X)$ *for some* $\tau = (l, p, l') \in T$. *As usual, if* $((l, v_X), p, (l', v'_X)) \in T_0$, *we write* $(l, v_X) \xrightarrow{p} (l', v'_X)$.

3.2.2 Composite Components

Composite components are defined by assembling sub-components (atomic or composite) using *connectors*. Connectors relate ports from different sub-components. They represent sets of interactions, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the connector provides the guard and the data transfer, that are, respectively, an enabling condition and an exchange of data across the ports involved in the interaction. Formally:

For a model built from a set of components B_1, $B_2, \ldots$, B_n, where atomic component $B_i = (L_i, P_i, T_i, X_i, \{g_\tau\}_{\tau \in T_i}, \{f_\tau\}_{\tau \in T_i})$, we assume that their respective sets of ports and variables are pairwise disjoint, i.e. for any two $i \neq j$ in $\{1 \ldots n\}$, we require that $P_i \cap P_j = \emptyset$ and $X_i \cap X_j = \emptyset$. Thus, we define the set $P = \bigcup_{i=1}^{n} P_i$ of all ports in the model as well as the set $X = \bigcup_{i=1}^{n} X_i$ of all variables.

Definition 19 (Interaction) *An interaction* a *is a triple* (P_a, G_a, F_a) *where* $P_a \subseteq P$ *is a set of ports,* G_a *is a guard, and* F_a *is a data transfer function. We restrict* P_a *so that it contains at most one port of each component, therefore we denote* $P_a = \{p_i\}_{i \in I}$ *with* $p_i \in P_i$ *and* $I \subseteq \{1 \ldots n\}$. G_a *and* F_a *are defined on the variables available on the interacting ports* $\bigcup_{p \in a} X_p$.

Given a set of interactions γ, the composition of the components following γ is the component $B = \gamma(B_1, \ldots, B_n) = (L, \gamma, \mathcal{T}, X, \{g_\tau\}_{\tau \in \mathcal{T}}, \{f_\tau\}_{\tau \in \mathcal{T}})$, where $(L, \gamma, \mathcal{T})$ is the transition system such that $L = L_1 \times \ldots \times L_n$ and $\mathcal{T} \subseteq L \times \gamma \times L$ contains transitions of the form $\tau = ((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, p_i, l'_i) \in T_i\}_{i \in I}$ such that $\{p_i\}_{i \in I} = a \in \gamma$ and $l'_j = l_j$ if $j \notin I$. The resulting set of variables is $X = \bigcup_{i=1}^{n} X_i$, and for a transition τ resulting from the synchronization of a set of transitions $\{\tau_i\}_{i \in I}$, the associated guard (resp. update relation) is the conjunction of the individual guards (resp. update relations) involved in the transition.

3.2.3 Priorities

Finally, *priorities* provide a means to coordinate the execution of interactions within a BIP system. They are used to specify scheduling or similar arbitration policies between simultaneously enabled interactions. More concretely, priorities are rules, each consisting of an ordered pair of interactions associated with a condition. When

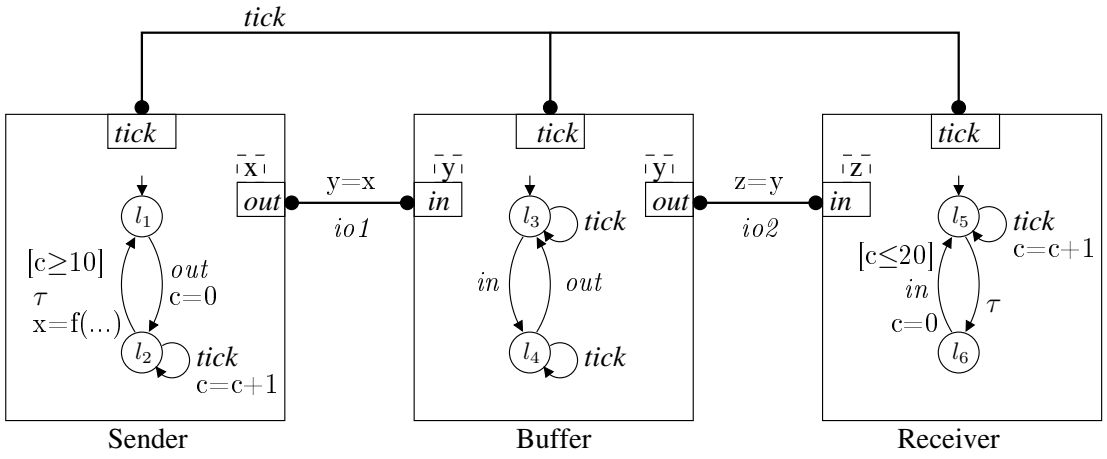


Figure 3.1: BIP example: Sender-Buffer-Receiver system.

the condition holds and both interactions of the corresponding pair are enabled, only the one with the highest priority can be executed. Non-determinism appears when several interactions are enabled. In the following, when we introduce probabilistic variables, we will thus have to make sure that non-determinism is resolved in order to produce a purely stochastic semantics.

Example 1 *Figure 3.1 shows a graphical representation of an example model in BIP. It consists of atomic components Sender, Buffer and Receiver. The behavior of the Sender is described as a transition system with control locations l_1 and l_2. It communicates through ports tick and out. Port out exports the variable x. Components Sender, Buffer and Receiver are composed by two binary connectors io1, io2 and a ternary connector tick. tick represents a rendezvous synchronization between the tick ports of the respective components. io1 represents an interaction with data transfer from the port out of Sender to the port in of Buffer. As a result of the data transfer associated with io1, the value of variable x of Sender is assigned to the variables y of the Buffer.*

3.2.4 Synchronization

BIP can model various types of synchronization. Using less expressive frameworks e.g. based on a single composition operator, often leads to intractable models. For instance, BIP directly encompasses multiparty interaction between components. Modeling multiparty interaction in frameworks supporting only point-to-point interaction e.g. function call or binary synchronization, requires the use of protocols. This can lead to overly complex models with complicated coordination structure. Similarly, priorities in BIP allow to express scheduling policies or general arbitration mechanisms between interactions in a declarative way. The use of scheduler components and explicit coordination between components may also obscure the overall design. The use of multiparty interactions and priorities confers a highly

expressive power. This has been not only formally proven e.g., in [23] but also practically illustrated on the modeling of several complex case studies [8, 6, 7].

Finally, it is worth noticing that the clear separation between architecture (interactions and priorities) and behavior (automata) in BIP allows compositional and incremental analysis. This is advantageously exploited by tools like D-Finder [18] which separately analyzes behavior of atomic components and extracts interaction invariants characterizing architectural constraints.

3.3 SBIP: A Stochastic Extension for BIP

The stochastic extension of BIP allows:

- to specify stochastic aspects of individual components
- and to provide a purely stochastic semantics for the parallel composition of components through interactions and priorities.

3.3.1 Syntax for Stochastic Atomic Components

Syntactically, we add stochastic behavior to atomic components in BIP by allowing the definition of probabilistic variables. Probabilistic variables x^P are attached to given distributions μ_{x^P} implemented as C functions. These variables can then be updated on transition using the attached distribution. The semantics on transitions is thus fully stochastic. We first define atomic components and interaction between them in SBIP, and then define the corresponding stochastic semantics.

Definition 20 (Atomic Component in SBIP) *An atomic component in SBIP is a transition system extended with data $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$, where $L, P, T, \{g_\tau\}_{\tau\in T}$ are defined as in Definition 17, and*

- *$X = X_D \cup X_P$, with $X_D = \{x_1, \ldots, x_n\}$ the set of deterministic variables and $X_P = \{x_1^P, \ldots, x_m^P\}$ the set of probabilistic variables.*
- *For each $\tau \in T$, the update function $X' = f_\tau(X)$ is a pair $(X'_D = f_\tau^D(X), R_\tau)$ where $X'_D = f_\tau^D(X)$ is an update relation for deterministic variables and $R_\tau \subseteq X_P$ is the set of probabilistic variables that will be updated using their attached distributions. Remark that the current value of the probabilistic variables can be used in the update of deterministic variables.*

In the following, given a valuation v_X of all the variables in X, we will denote by v_Y the projection of v_X onto a subset of variables $Y \subseteq X$. When clear from the context, we will denote by v_y the valuation of variable $y \in X$ in v_X.

Some transitions in the associated semantics are thus probabilistic. As an example, consider an atomic component B with a transition τ that goes from a location l

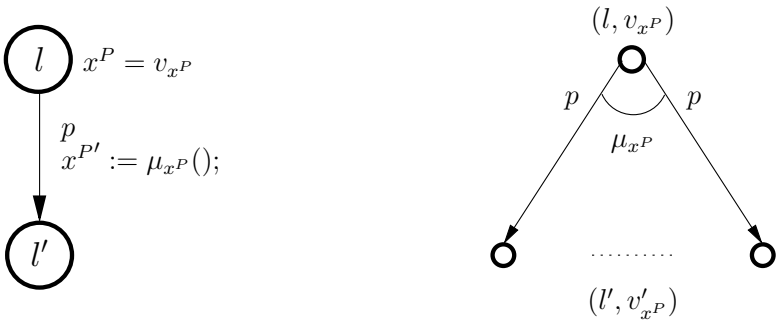


Figure 3.2: Example of an abstract component B and its semantics in SBIP.

to a location l' using port p and updates a probabilistic variable x^P with the distribution μ_{x^P} over the domain $\mathbf{x^P}$. In the associated semantics, assuming the initial value of x^P is v_{x^P}, there will be several transitions from state (l, v_{x^P}) to states (l', v'_{x^P}) for all $v'_{x^P} \in \mathbf{x^P}$. According to the definition of probabilistic variables, the probability of taking transition $(l, v_{x^P}) \xrightarrow{p} (l', v'_{x^P})$ will then be $\mu_{x^P}(v'_{x^P})$. This example is illustrated in Figure 3.2. When several probabilistic variables are updated, the resulting distribution on transitions will be the product of the distributions associated to each variable. These distributions are fixed from the declaration of the variables, and are considered to be independent. The syntactic definitions of interactions and composition are adapted from BIP in the same manner. For the sake of simplicity, we restrict data transfer functions on interactions to be deterministic.

Remark 1 *We write a transition in SBIP as* $l_i \xrightarrow[f]{p,g} l_j$, *where* $l_i, l_j \in L, p \in P, g \in \{g_t\}_{t \in T}$ *and* $f \in \{f_t\}_{t \in T}$.

3.3.2 Stochastic Semantics for Atomic Components

Adapting the semantics of an atomic component in BIP as presented in Definition 18 to atomic components with probabilistic variables leads to transition systems that combine both stochastic and non-deterministic aspects. Indeed, even if atomic transitions are either purely deterministic or purely stochastic, several transitions can be enabled in a given system state. In this case, the choice between these potential transitions is non-deterministic. In order to produce a purely stochastic semantics for components defined in SBIP, we resolve any non deterministic choice left after applying the priorities by applying uniform distributions. Remark that other distributions could be used to resolve this non-determinism and that using uniform distributions is the default choice we made. In the future, we will allow users to specify a different way of resolving non-determinism.

Consider a component $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$ in SBIP. Given a state (l, v_X) in $L \times \mathbf{X}$, we denote by $\mathsf{Enabled}(l, v_X)$ the set of transitions in T that are

enabled in state (l, v_X), i.e. transitions $\tau = (l, p, l') \in T$ such that $g_\tau(v_X)$ is satisfied. Since priorities only intervene at the level of interactions, the semantics of a single component does not take them into account. Remark that the set $\mathsf{Enabled}(l, v_X)$ may have a cardinal greater than 1. This is the only source of non-determinism in the component. In the semantics of B, instead of non-deterministically choosing between transitions in $\mathsf{Enabled}(l, v_X)$, we will choose probabilistically using a uniform distribution. Formally:

Definition 21 (Semantics of a single component in SBIP) *The semantics of* $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$ *in SBIP is a probabilistic transition system* (Q, P, T_0) *such that* $Q = L \times \mathbf{X}$ *and* T_0 *is the set of probabilistic transitions of the form* $((l, v_X), p, (l', v'_X))$ *for some* $\tau = (l, p, l') \in \mathsf{Enabled}(l, v_X)$ *such that* $v'_{X_D} = f^D_\tau(v_X)$, *and for all* $y \in X_P \setminus R_\tau$, $v'_y = v_y$.

In a state (l, v_X), *the probability of taking a transition* $(l, v_X) \xrightarrow{p} (l', v'_X)$ *is the following:*

$$\frac{1}{|\mathsf{Enabled}(l, v_X)|} \left[\sum_{\substack{\{\tau \in \mathsf{Enabled}(l, v_X) \\ s.t. \ \ \tau = (l,p,l')\}}} \left(\prod_{y \in R_\tau} \mu_y(v'_y) \right) \right].$$

The probability of taking transition $(l, v_X) \xrightarrow{p} (l', v'_X)$ is computed as follows. For each transition $\tau = (l, p, l') \in \mathsf{Enabled}(l, v_X)$ such that $v'_{X_D} = f^D_\tau(v_X)$ and for each $y \in X_P \setminus R_\tau$, $v'_y = v_y$, the probability of reaching state (l', v'_X) is $\prod_{y \in R_\tau} \mu_y(v'_y)$. Since there may be several such transitions, we take the sum of their probabilities and normalize by multiplying with $\frac{1}{|\mathsf{Enabled}(l, v_X)|}$.

3.3.3 Stochastic Semantics for Composing Components

When considering a system with n components in SBIP

$$B_i = (L_i, P_i, T_i, X_i, \{g_\tau\}_{\tau \in T_i}, \{f_\tau\}_{\tau \in T_i}) \tag{3.1}$$

and a set of interactions γ, the construction of the product component

$$B = \gamma(B_1, \ldots, B_n) \tag{3.2}$$

is defined as in BIP. The resulting semantics is given by Definition 21 above, where $\mathsf{Enabled}(l, v_X)$ now represents the set of *interactions* enabled in global state (l, v_X) that are maximal with respect to priorities. By construction, it follows that the semantics of any (composite) component in SBIP is purely stochastic.

Example 2 *Consider SBIP components* B_1 *and* B_2 *given in Figures 3.3a and 3.3b.* B_1 *has a single probabilistic variable* x^P_1, *to which is attached distribution* μ_1 *and a single transition from location* l^1_1 *to location* l^1_2 *using port* p_1, *where* x^P_1 *is updated.*

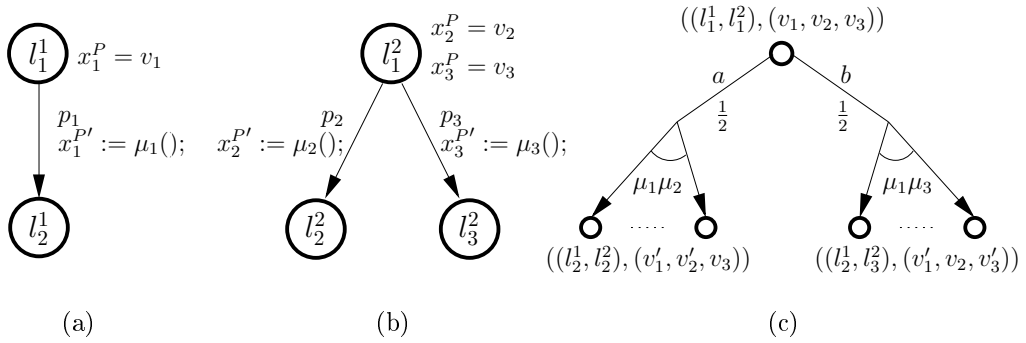


Figure 3.3: Illustration of the purely stochastic semantics of composition in SBIP.

In location l_1^1, the variable x_1^P is assumed to have value v_1. B_2 has two probabilistic variables x_2^P and x_3^P, to which are attached distributions μ_2 and μ_3 respectively. B_2 admits two transitions: a transition from location l_1^2 to location l_2^2 using port p_2, where x_2^P is updated, and a transition from location l_1^2 to location l_3^2 using port p_3, where x_3^P is updated. In location l_1^2, the variables x_2^P and x_3^P are assumed to have values v_2 and v_3 respectively. Let $\gamma = \{a = \{p_1, p_2\}, b = \{p_1, p_3\}\}$ be a set of interactions such that interactions a and b have the same priority. The semantics of the composition $\gamma(B_1, B_2)$ is given in Figure 3.3c. In state $((l_1^1, l_1^2), (v_1, v_2, v_3))$ of the composition, the non-determinism is resolved between interactions a and b, choosing one of them with probability $1/2$. After choosing the interaction, the corresponding transition is taken, updating the corresponding probabilistic variables with the associated distributions. Remark that this gives rise to a single purely stochastic transition. As an example, the probability of going to state $((l_2^1, l_2^2), (v_1', v_2', v_3))$ with interaction a is $1/2 \cdot \mu_1(v_1') \cdot \mu_2(v_2')$, while the probability of going to state $((l_2^1, l_3^2), (v_1', v_2, v_3'))$ with interaction b is $1/2 \cdot \mu_1(v_1') \cdot \mu_3(v_3')$.

An execution π of a BIP model is a sequence of states that can be generated from an initial state by following a sequence of (probabilistic) transitions. From the above, one easily sees that the semantics of any SBIP (composite) system has the structure of a discrete Markov chain. Consequently, one can define a probability measure μ on its set of executions in the usual way [102].

3.3.4 DTMC Modeling in SBIP

In the previous section, we saw that the semantics of an SBIP model is purely stochastic and is equivalent to a Discrete Markov Chain. In this section we provide an operational semantics that deals with Markov chains to SBIP model transformation. The transformation rules may be then used in model engineering to build from a system described directly in the formalism of DTMC a SBIP model.

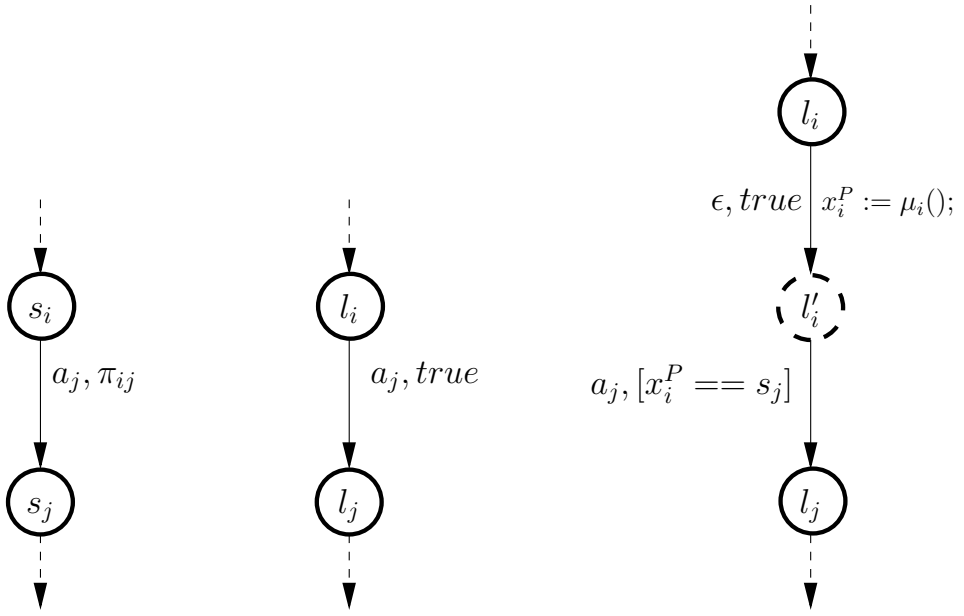


Figure 3.4: Illustration of the transformation from DTMC to SBIP model.

Definition 22 *Let ϵ be the empty action. Given a DTMC $M = \langle S, Act, \iota, \pi, L^M \rangle$, we define the transformation from M to a stochastic BIP model $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$ as follows:*

- $L = \{l_i \text{ for each } s_i \in S\} \cup \{l_i' \text{ for each } s_i \in S \mid \exists \text{ unique } a \in Act \text{ s.t. } \pi(s_i, a, s_j) = 1\}$,
- $P = Act \cup \{\epsilon\}$,
- $T \subseteq L^M \times P \times L^M$,
- $X = \{x_i^P \text{ for each } s_i \in S \mid \mu(x_i^P = s_j) = \pi_{ij}\}$, *and*

$$\frac{s_i \xrightarrow{a_j, \pi_{ij} > 0} s_j}{l_i \xrightarrow[x_i^P := \mu_i()]{\epsilon, true} l_i',\ l_i' \xrightarrow{a_j, [x_i^P == s_j]} l_j}, \ if\ \pi_{ij} < 1 \tag{3.3}$$

$$\frac{s_i \xrightarrow{a_j, \pi_{ij} > 0} s_j}{l_i \xrightarrow{a_j, true} l_j}, \ if\ \pi_{ij} = 1 \tag{3.4}$$

Intuitively, the transformation states that for a given Markov Chain M, each transition $s_i \xrightarrow{a, \pi_{ij}} s_j$ that has a probability $\pi_{ij} < 1$, is associated, in the corresponding SBIP model, with two transitions. The first is $l_i \xrightarrow[x_i^P := \mu_i()]{\epsilon, true} l_i'$ that is a probabilistic step based on the related distribution which is directly obtained from the DTMC

Next State ($\mathbf{x_1^P}$ domain)	Probability ($\mu_1(x_1^P := s_i)$)
s_1	1/6
s_2	1/6
s_3	2/3

Table 3.1: Probability distribution in state s_1.

(the one that characterize the next state weights from the state s_i), while the second is $l'_i \xrightarrow{a_j,[x_i^P==s_j]} l_j$ which stand for a next location choice as shown in Figure 3.4 and rule (3.3) of Definition 22. Another case is also presented in this definition where the transition probability $\pi_{ij} = 1$, the Markov Chain transition is then associated with a unique SBIP transition $l_i \xrightarrow{a_j,true} l_j$ as specified by rule (3.4) of the same definition. Note that, in the first case, the first transition correspond to a sampling operation over possible next locations ($x_i^P := \mu_i()$) (since there are more than one possible transition with different probabilities in the DTMC) and that the second transition uses BIP guards to select the next location with respect to the chosen value.

Example 3 *Figure 3.5 shows the DTMC Model of a simple sending protocol. Initially, the protocol tries to send which leads to state* s_1*. From that state, the process could try again with probability* 1/6*, fail with probability* 1/6*, or succeed with probability* 2/3*. In case of fail, the protocol is restarted through the init action. The probabilities* 1 *on the transitions try, init and success are omitted.*

The corresponding SBIP model is shown in Figure 3.6. It consists in one SBIP component where the probabilistic variable x_1^P *that models the next state distribution from* s_1 *is described in table 3.1.*

Remark that the try transition from state s_0 *in the DLMC in Figure 3.5 is preserved as it is in the SBIP component in Figure 3.6 as well as init and success transitions from state* s_3*. In fact, since their probabilities are equal to* 1*, the rule* (3.4) *of Definition 22 is applied. For the transitions fail, success, and try from state* s_1 *in the DLMC, they are transformed using the rule* (3.3) *since their probabilities are smaller that* 1 *which gives an additional sampling step from* l_1 *to* l'_1 *in the SBIP component that uses the* X_{s_1} *distribution.*

3.4 SMC for SBIP

Any statistical model checking of Markov Chains and BLTL properties requires to implement two routines: 1. a runtime verification procedure to decide whether a finite execution satisfies a BLTL formula, and 2. one or many SMC algorithms as described earlier. In this section, we first present the SMC capabilities and the architecture of SBIP. Then, we describe the implemented runtime verification procedure.

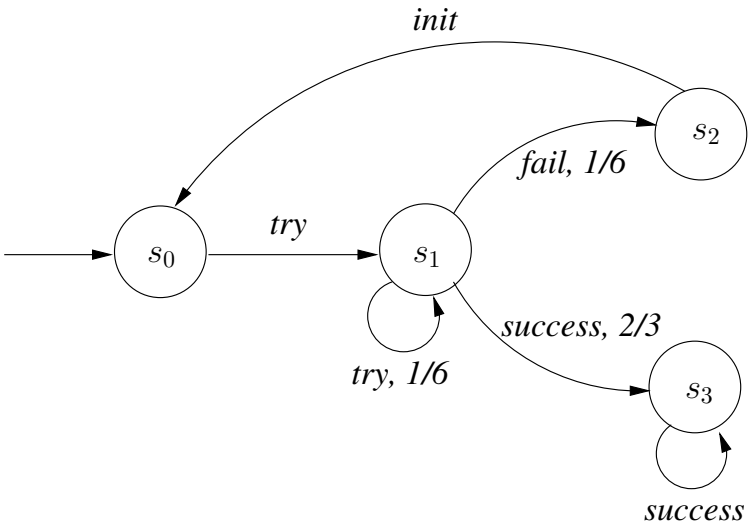


Figure 3.5: A DLMC for a sending protocol example.

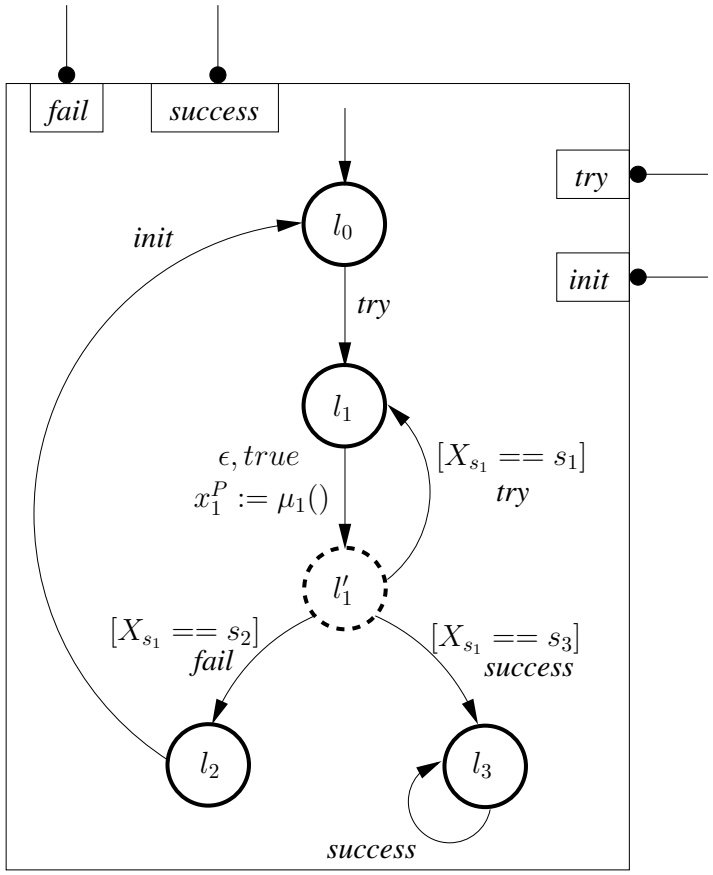


Figure 3.6: Corresponding SBIP model for the sending protocol example.

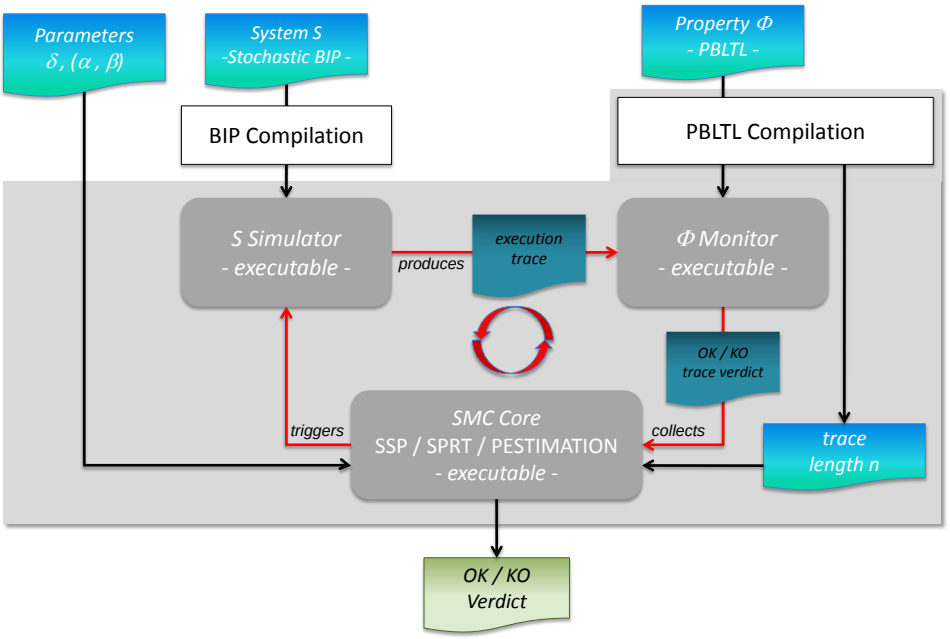


Figure 3.7: SBIP tool architecture and work flow.

3.4.1 Tool Architecture

The SBIP tool [100] implements the statistical algorithms described in section 2.3, namely, *SSP*, *SPRT*, and *PESTIMATION* for stochastic BIP systems. Figure 3.7 shows the tool architecture and execution flow. SBIP takes as inputs a stochastic system written in the BIP language, a PBLTL property, and a series of confidence parameters needed by the statistical test. First, the tool generates an executable model and builds a monitor for the property under verification. Afterward, it iteratively triggers the stochastic BIP engine to generate random execution traces (sampling) which are checked with respect to the input property using the monitor. This procedure is repeated until a decision can be taken by the SMC core. As our approach relies on SMC and consider bounded LTL formulas, we are guaranteed that the procedure will eventually terminate.

3.4.2 Monitoring and Runtime Verification

Monitoring

For applying statistical model checking on stochastic systems it is mandatory to be able to evaluate the BLTL property under consideration on system execution traces. Indeed, this monitoring operation shall generate binary observations $x_i = \{0, 1\}$ (single trace verdict) which are requested by the statistical algorithms to provide a global verdict that concerns the whole system (all traces verdict). In theory, monitoring consists to check if some word (labeling the current execution trace) belongs to the language generated by some automaton encoding the property. Actually, there

exists an important research literature about the efficient transformation from LTL to Buchi [47, 132] or alternating [127] automata. Some of these works cover bounded LTL [46, 49]. Nonetheless, despite these important theoretical results, it seems that no efficient method to transform BLTL to finite automata is yet established nor implemented.

To avoid this technical difficulty, in the current SBIP implementation, we restricted syntactically BLTL to a fragment where the temporal operators cannot be nested. This simplification restricts the definition to a finite number of automata patterns that covers all property classes. Moreover, this fragment has been expressive enough to cover all properties of interest in practical applications. Furthermore, it is always possible to enrich this set with additional patterns, as needed.

Runtime Verification

Runtime Verification (RV) [57, 44, 115] refers to a series of techniques whose main objective is to instrument the specification of a system (code, etc.) in order to observe and potentially refute complex properties at execution. The main issue of the runtime verification approach is, however, that it does not permit to assess the overall correctness of the entire system but only to identify potential errors.

In order to support runtime verification, the BIP framework allows for addition of observer components that enable to observe specific events of the system and/or to (partially) encode the evaluation of requirements (if they are otherwise difficult to express using BLTL). It is important to mention that such observers can be added to a BIP system in a totally non-intrusive way, that is, they run in parallel to the system components and only interact loosely with them, through specific connectors. A detailed presentation of the approach for construction and insertion of observers in BIP systems can be found in [45].

3.5 How to Use SBIP

In this section we show how to practically use the SBIP tool [100] to model a stochastic system and to verify it using statistical model checking techniques.

3.5.1 Modeling in SBIP Language

The first step to use SBIP is to formally model the system to verify using the stochastic BIP formalism. Syntactically, using stochastic BIP is same as using BIP language [**?**] since the extension concerns essentially the semantics level and also because BIP is able to use external C++ code that is a strong way to extend it. Nevertheless, SBIP provides an additional library that should be used jointly with BIP and which provides probabilistic and tracing functionality to build an SBIP compatible model.

In the following we give an example of an SBIP component that uses the aforementioned functionality. We illustrate on the sending protocol component in Figure 3.5.

```
/* Declares an atomic BIP component */
atomic type sending_protocol

  /* Declares a probabilistic variable */
  data int Xs1
  /* Declares a probabilistic distribution */
  data distribution_t dist_1
  /* Declares an integer variable */
  data int success
  ...

  /* Declares and exports ports:
  init, try, fail, success */
  export port Port init
  export port Port try
  export port Port fail
  export port Port success
  /* Declares an internal BIP port */
  port Port epsilon
  ...

  /* Declares BIP locations */
  place l0, l1, l1', l2, l3

  /* Initialization */
  initial to l0 do {
    /* Init dist_1 from empirical dist. */
    dist_1 = init_distribution(``dist_1.txt'');
    /* update success flag and trace it */
    success = 0;
    trace_i(``sending_protocol.success'', success);
  }

  /* Transition from l0 to l1 */
  on try from l0 to l1
  on epsilon from l1 to l1' do {
      /* Updates Xs1 wrt. dist_1 */
      Xs1 = select(dist_1);
  }
```

```
  /* Transition from l1' to l1 */
  on try from l1' to l1 provided (Xs1 == s1)
  /* Transition from l1' to l3 */
  on success from l1' to l3 provided (Xs1 == s3) do {
    /* update success flag and trace it*/
    success = 1;
    trace_i(``sending_protocol.success'', success);
  }
  /* Transition from l1' to l2 */
  on fail from l1' to l2 provided (Xs1 == s2) do {
    /* update success flag and trace it*/
    success_flag = 0;
    trace_i(``sending_protocol.success'', success);
  }
  /* self loop on l3 */
  on success from l3 to l3
  /* Transition from l2 to l0 */
  on init from l2 to l0 do {
    /* update success flag and trace it*/
    success_flag = 0;
    trace_i(``sending_protocol.success'', success);
  }

end
```

The code above, describes the SBIP sending protocol model that uses some of the provided functionality in SBIP. For instance, the *distribution_ t* predefined type is used to define a probabilistic distribution which is initialized, in this case, using *init_ distribution()* function. This one optionally takes as input a text file that contains an empirical distribution. The declared distribution can be then used to update probabilistic variables (declared as classical BIP variables) using the *select()* function that returns a value with respect to its weight in the input distribution parameter. Similar functions could be also used to sample from standard probabilistic distributions such as *Uniform*, *Normal*, *Exponential*, etc. For instance, *Uniform* sampling could be done by just specifying the bounds of the interval to consider and without any initialization. For example, the call $select(125, 500)$ returns uniformly selected values in the interval $[125, 500]$.

Remark 2 *The choice of using text files to describe empirical distributions, is made for practical reasons. Such files are usually automatically generated through system simulation.*

Another functionality shown in this code is variables tracing which is mandatory to do trace monitoring. SBIP provides several tracing procedures with respect to

variables type: *trace_ i()* for Integer, *trace_ b()* for Boolean, *trace_ d()* for Double, and *trace_f()* for Float. Those functions take as parameters a string that specifies the component name and the variable name, in addition to the variable value. In the code sample above, the variable of interest that is, subject to verification, is *success* (note that this step of code annotation with tracing functions should be done when a property to check is fixed that is, to identify the variables to trace). This variable is of type Integer, hence the function call

```
trace_i(‘‘sending_protocol.success’’, success)
```

is used.

3.5.2 Properties Specification in SBIP

Whenever, the stochastic BIP model is built, the next step is to specify the property to be checked. As mentioned before, in the case of SBIP, this should be done in PBLTL syntax which is defined with respect to the following grammar:

$$\Phi ::= P \geqslant \theta[\Psi] \mid P =?[\Psi]$$
$$\Psi ::= \varphi\ U\{i\}\ \varphi \mid (\ G\{i\} \mid F\{i\}\ )\ \varphi \mid N\ \varphi$$
$$\varphi ::= true \mid false \mid \omega \mid \varphi\ (\ \wedge \mid \vee\ )\ \varphi$$
$$\omega ::= \upsilon \mid !\,\upsilon \mid \varepsilon\ (\ > \mid < \mid \geq \mid \leq \mid = \mid \neq\ )\ \varepsilon$$
$$\varepsilon ::= \upsilon \mid K \mid \varepsilon\ (\ + \mid - \mid \times \mid / \mid \%\ )\ \varepsilon \mid F(\upsilon, \cdots, \upsilon)$$

In this grammar, θ is a probability threshold, U, G, F, N are respectively *Until*, *Always*, *Eventually*, and *Next* temporal operators, i is an integer bound on the mentioned operators, υ is a state variable, K is an integer constant, and F denotes predefined functions.

Note that it is possible through this syntax to either ask for a probability estimation using $P =?$ operator or to check if the property probability respects some bound θ using $P \geqslant \theta$ operator. For example, given the SBIP model of the sending protocol above, a requirement to check could be that the probability to send always succeed is greater than a fixed threshold $\theta = 0.9$, which is formulated in PBLTL as follow:

$$P \geq 0.9[G\{1000\}(sending\_protocol.success)]$$

It is also possible to ask what is the probability that the send action eventually fails which is specified in PBLTL as follow:

$$P =?[F\{1000\}(!sending\_protocol.success)]$$

3.5.3 Statistical Model Checking with SBIP

Once the stochastic BIP model and the corresponding PBLTL properties are ready, SBIP could be used as follows to probabilistically check if the specified property

hold on the system under consideration.

To use SBIP tool, the first step to do is to download it from the Web page on `http://www-verimag.imag.fr/Statistical-Model-Checking.html`. In addition, you should download and correctly set up the BIP tool (SBIP works with the old and the new BIP version) to be able to build stochastic BIP models as shown above.

When downloaded and extracted, the obtained tool directory is structured as follow:

- **lib**\ directory which hold tool libraries/dependencies,
- **bin**\ directory that contains tool binaries,
- **examples**\ directory that contains some stochastic BIP examples,
- **setup.sh** file that should be used to install the tool, and finally,
- **README** file that explains the tool usage.

To set up the tool environment, go under the tool root directory and type the command below:

```
$ source setup.sh
```

Henceforth, it is possible to statistical model check stochastic systems built as BIP models using the following command prototype:

```
$ sbip [-htest|-pestim] [Formula] [Delta] [Alpha]
[Beta] [-bip1|-bip2] [Executable]
```

where [-htest | -pestim] are options to specify hypothesis testing or probability estimation as statistical test, [Formula] is a PBLTL formula to check, [Delta], [Alpha] and [Beta] defines the level of confidence of the statistical tests, [-bip1 | -bip2] are options to specify which BIP language version will be used, and finally, [Executable] is the BIP binary of the system to verify. For example, to verify the quantitative property $P >= 0.8[G\{10000\}(sending\_protocol.success)]$ with confidence 10^{-5} on the sending sending protocol model, it is possible to use the following command:

```
$ sbip -htest \
```

```
P >= 0.8[G{1000}(sending_protocol.success)]\
```

```
0.05 0.00001 0.00001 -bip2 sending_protocol
```

The command line specifies that the hypothesis testing technique is used as statistical test (with 0.8 as a threshold), and that the probability to make errors (typeI and typeII) is bounded to 10^{-5} with an indifference region of 5.10^{-2}.

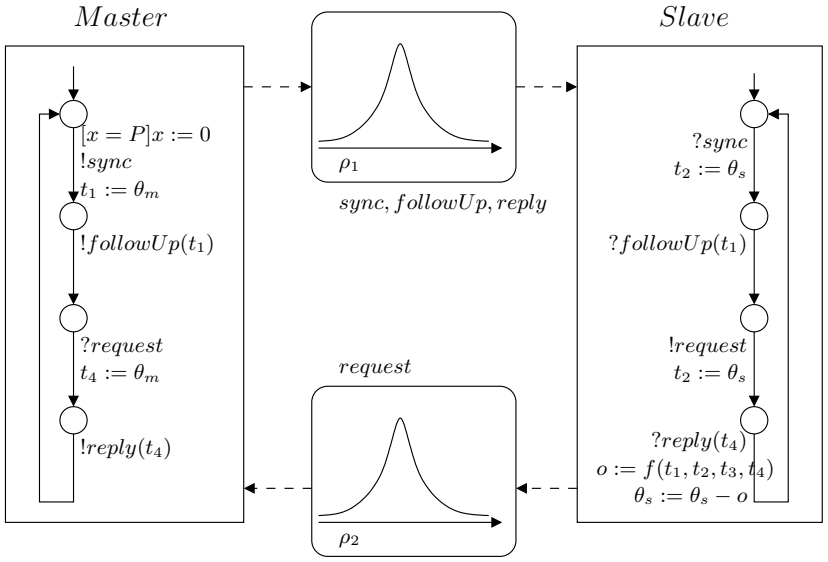


Figure 3.8: PTP stochastic model.

3.6 Case Studies

While still at prototype level, SBIP has been already applied to several case studies coming from serious industrial applications.

3.6.1 Accuracy of Clock Synchronization Protocol IEEE.1588

Model Description

The case study concerns a clock synchronization protocol running within a distributed heterogeneous communication system (HCS) [6]. This protocol allows to synchronize the clocks of various devices with the one of a designated server. It is important that this synchronization occurs properly, i.e., that the difference between the clock of the server and the one of any device is bounded by a small constant.

To verify such property, we build the stochastic model depicted in Figure 3.8. This model is composed by two deterministic components namely *Master*, and *Slave* and two communication channels. In the PTP model, the time of the master process is represented by the clock variable θ_m. This is considered the reference time and is used to synchronize the time of the slave clock, represented by the clock variable θ_s. The synchronization works by messages exchange between the server and a slave device. Each one of them saves the time of message reception $(t_i)_{i=1,4}$ with respect to its local clock. Finally, the slave device computes the offset between its time and the master time and updates its clock accordingly. Communication channels have been modeled using stochastic components. These components model communication delays over the network using empirical distributions obtained by simulating a detailed HCS model.

Precision	10^{-1}		10^{-2}		10^{-3}	
$1-$Confidence	10^{-5}	10^{-10}	10^{-5}	10^{-10}	10^{-5}	10^{-10}
PESTIM	4883 $17s$	9488 $34s$	488243 $29m$	948760 $56m$	48824291 $> 3h$	94875993 $> 3h$
SSP	1604 $10s$	3579 $22s$	161986 $13m$	368633 $36m$	16949867 $> 3h$	32792577 $> 3h$
SPRT	316 $2s$	1176 $7s$	12211 $53s$	22870 $1m38s$	148264 $11m$	311368 $31m$

Table 3.2: Number of simulations / Amount of time required for PESTIMATION, SSP and SPRT.

The accuracy of the synchronization is defined by the absolute value of the difference between the master and slave clocks $|\theta_m - \theta_s|$, during the lifetime of the system we consider (in this case, 1000 steps). Our aim is to verify the satisfaction of the bounded LTL formula $P =?[G\{1000\}(abs(Master.\theta_m - Slave.\theta_s) \leq \Delta)]$ for arbitrary fixed non-negative Δ.

Experiments and results

Two types of experiments are conducted. The first one is concerned with the bounded accuracy property ϕ. In the second one, we study average failure per execution for a given bound.

Property 1: Synchronization. To estimate the best accuracy bound, we have computed, for each device, the probability for synchronization to occur properly for values of Δ between $10\mu s$ and $120\mu s$. Figure 3.9 gives the results of the probability of satisfying the bounded accuracy property ϕ as a function of the bound Δ. In this protocol, the devices are connected to the server using network access controllers (NAC). For simplicity, devices are addressed (i, j), where i is the address of the NAC and j is the address of the device. The figure shows that the smallest bound which ensures synchronization for any device is $105\mu s$ (for Device $(3, 0)$). However, devices $(0, 3)$ and $(3, 3)$ already satisfy the property ϕ with probability 1 for $\Delta = 60\mu s$. For this experiments, we have used SPRT and SSP jointly with PESTIMATION for a higher degree of confidence. The results, which are presented in Table 3.2 for Device $(0, 0)$, show that SPRT is faster than SSP and PESTIMATION.

Property 2: Average failure. In the second experiment, we try to quantify the average and worst number of failures in synchronization that occur *per simulation* when working with smaller bounds. Our goal is to study the possibility of using such bounds. For a given simulation, the *proportion of failures* is obtained by dividing the number of failures by the number of rounds of PTP. We will now estimate,

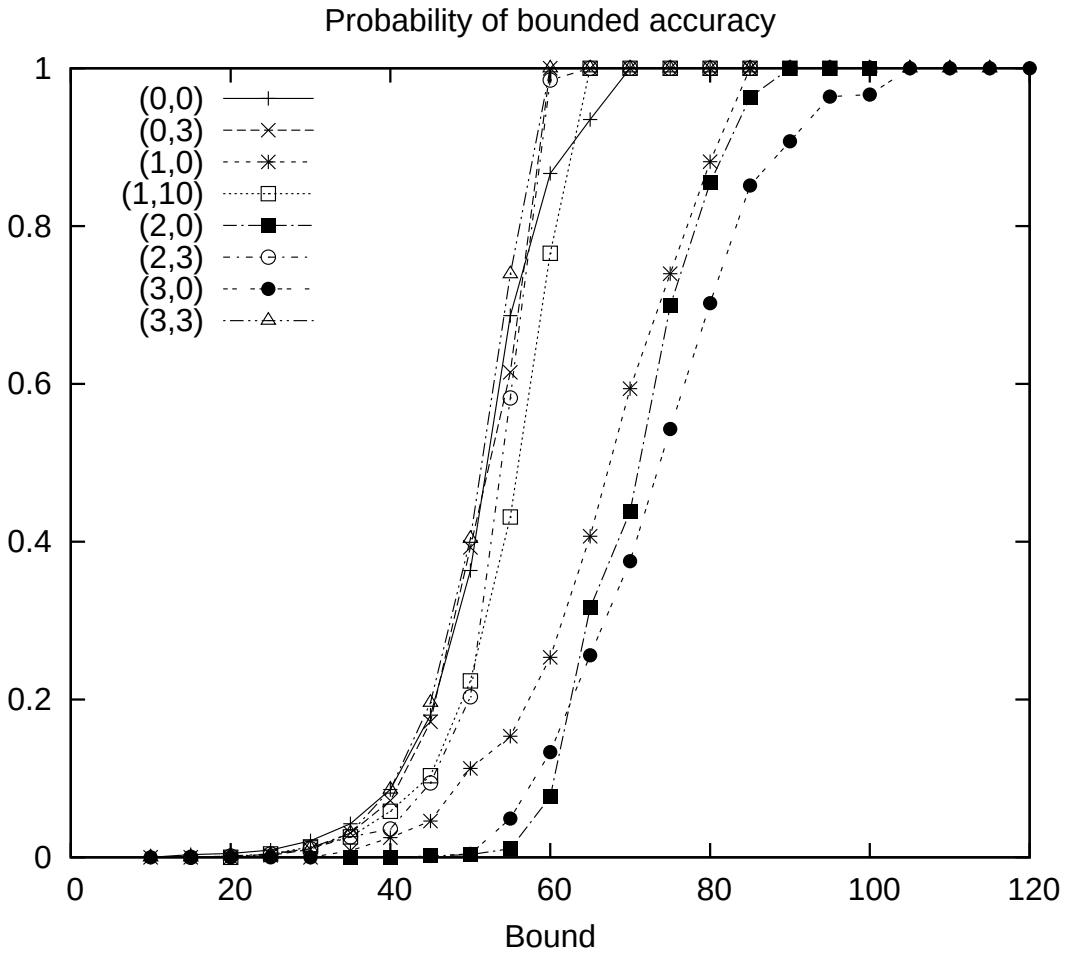


Figure 3.9: Probability of satisfying bounded accuracy property as functions of the bound Δ.

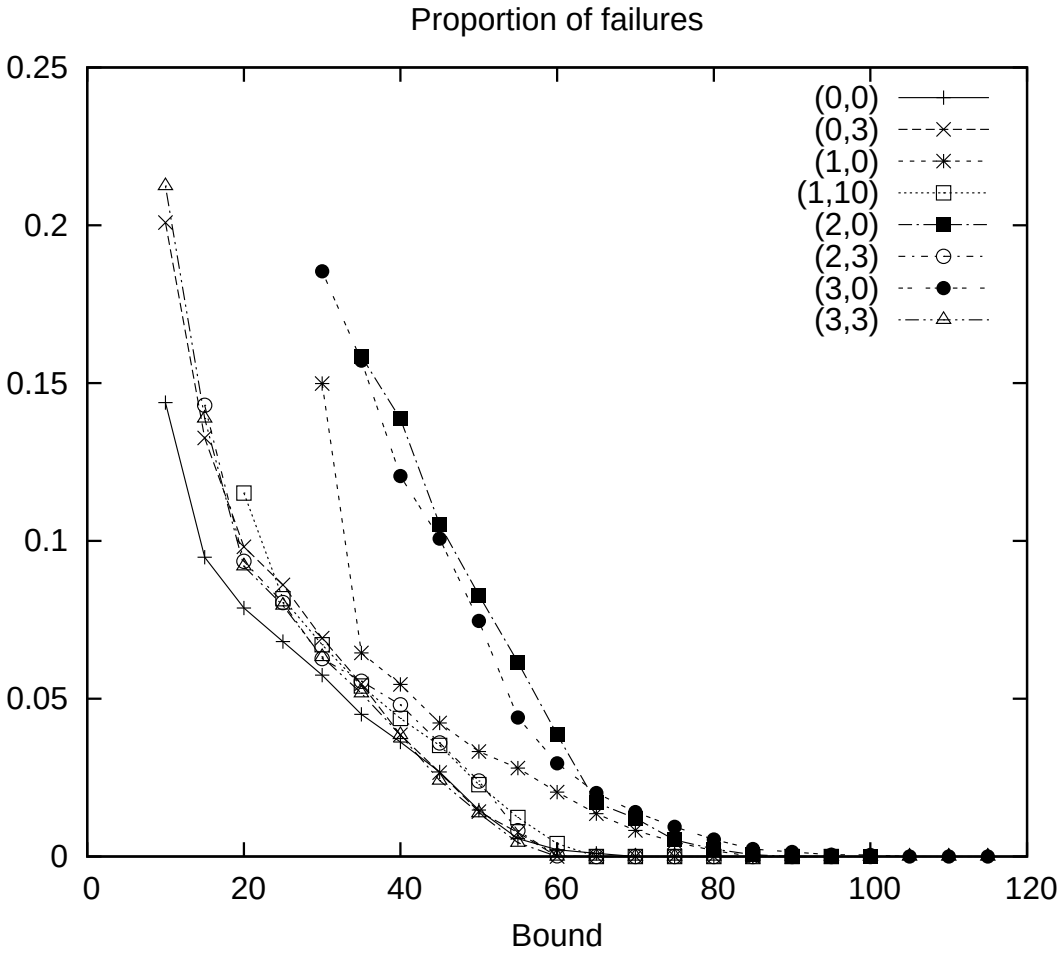


Figure 3.10: Average proportion of failures as functions of the bound Δ.

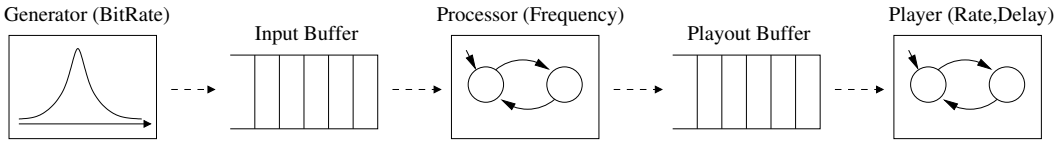


Figure 3.11: MPEG2 player stochastic model.

for a simulation of 1000 steps (66 rounds of the PTP), the average value for this proportion. To this purpose, we have measured for each device this proportion on 1199 simulations with a different synchronization bounds Δ between $10\mu s$ and $120\mu s$. Figures 3.10 gives the average proportion of failure as a function of the bound.

3.6.2 Playout Buffer Underflow in MPEG2 Player

In multimedia literature [131], it has been shown that some quality degradation is tolerable when playing MPEG2-coded video. In fact, a loss under two consecutive frames within a second can be accepted. In this study, we want to check an MPEG2 player implementation with respect to the aforementioned QoS property, in addition to buffer size reduction [107].

Model Description

We illustrate the multimedia player set-up that has been modeled using the stochastic BIP framework. The designed model captures the stochastic system aspects that are, the macro-blocks arrival time to the input buffer and the their processing time.

The stochastic system model is shown in Figure 3.11. It consists of three functional components namely *Generator*, *Processor*, and *Player*. In addition to these, the buffers between the above functional components are modeled by explicit *buffer* components, namely *Input buffer* and *Playout buffer*. The transfer of the macro-blocks between the functional blocks and the buffers are described using interactions. All the functional components are timed, and the simulated time is modeled by the *tick* connector, which provides global synchronization between them.

The *Generator* is a stochastic component which models macro-blocks production based on three probabilistic distribution in a frame-type fashion as shown in Figure 3.12. It generates an MPEG2-coded stream with respect to a fixed Group-of-Pictures (GOP) pattern [83, 84] and simulates the arrival time of macro-blocks to the *input buffer*.

The *Processor* reads them sequentially, decodes them and write them to the *Playout buffer*. The *Player* starts to read macro-blocks from the *Playout buffer* after a defined initial delay namely *Playout Delay*. Once this delay ends, the consumption is performed periodically with respect to a fixed consumption rate. Each period, the

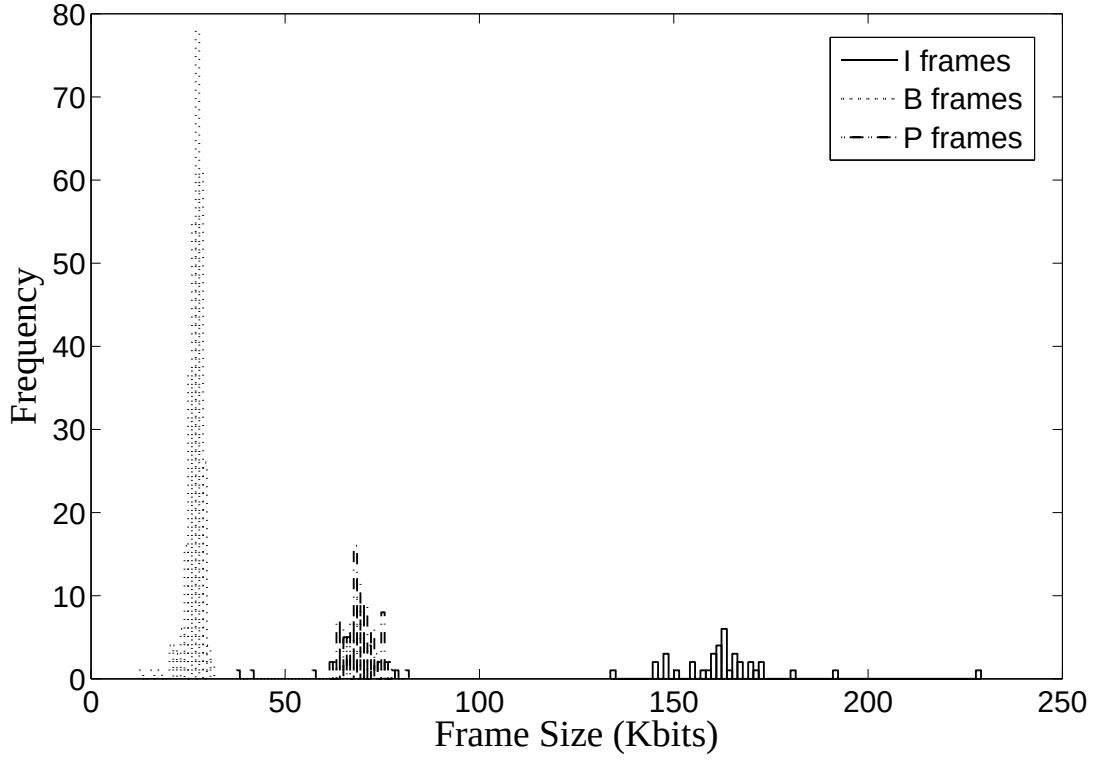


Figure 3.12: Frequency distribution of I, P, and B frames in an MPEG2 video.

Player sends a request of N macro-blocks to the *Playout buffer*, where $N = 1$ the first time. Then it gets a response of M macro-blocks, where $0 \leq M \leq N$. An underflow happens when $M < N$. In this case, the next request N will be $(N - M) + 1$. That is, the player will try to read all the missed macro-blocks.

Experiments and results

To check the described model with respect to the desired QoS property, we used the SBIP tool. The PBLTL specification of the QoS property to check is:

$$P = ?[G\{1500000\}(!Observer.fail)], \tag{3.5}$$

where *fail* denotes a failure state condition corresponding to the underflow of two consecutive frames within a second. The *fail* state is represented in an *Observer* BIP component which captures the failure condition by monitoring the *Player* frame consumption.

Figure 3.13 shows a bench of results for the `mobile.m2v` open source video. In this figure, the x-axis represents the probability of failure (a loss of two consecutive frames within a second) and the y-axis illustrates the playout buffer fill level. In addition, it shows, in the top, the playout delay evolution. We can see first, that for a high playout delay, the playout buffer is highly filled and hence that the probability of underflow is null. If we start reducing the playout delay, the playout buffer fill

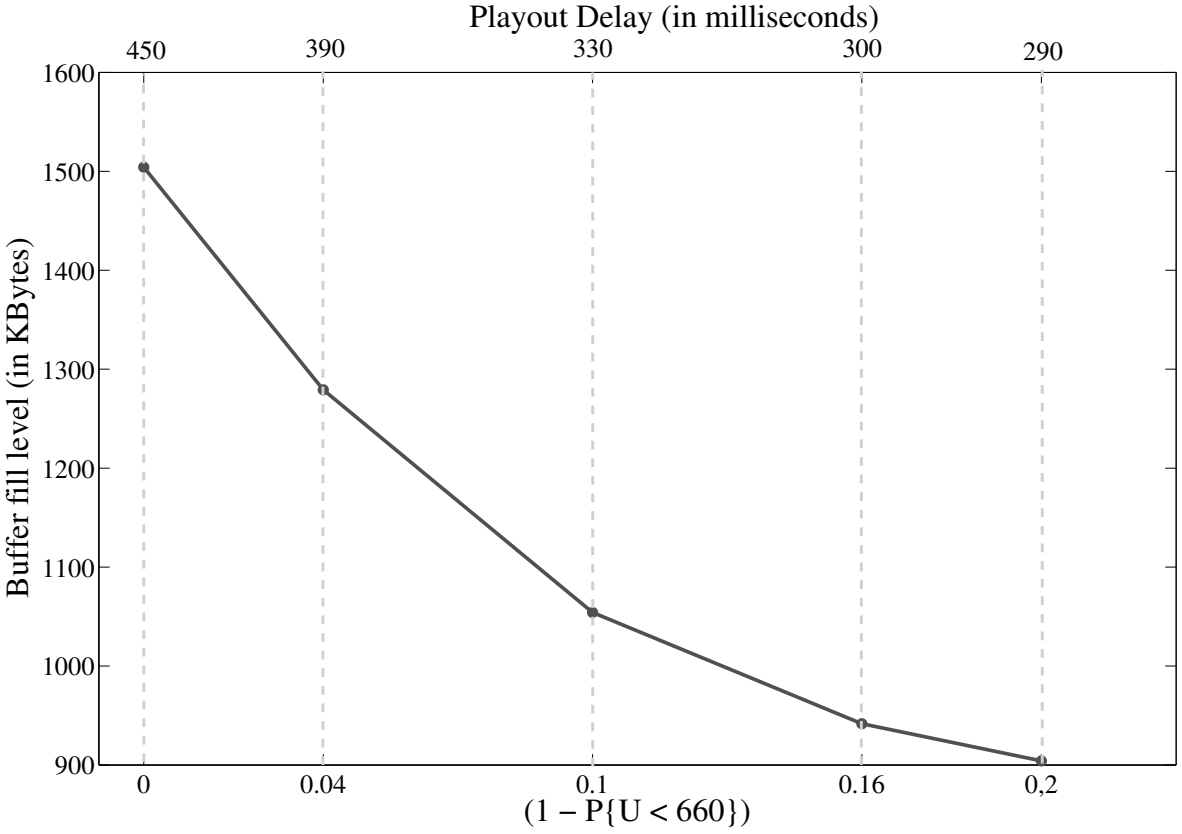


Figure 3.13: Playout buffer fill level as function of playout delay and probability of property failure for `mobile.m2v` video.

level decreases, which induces some probability of failure since the player starts to consume the frames sooner. The goal of the analysis is to enable designer to choose a trade-off amount of quality degradation that reduces the buffer size and does not imply a big playout delay.

3.7 Conclusion and Related Work

Stochastic systems can also be analyzed with a pure probabilistic model checking approach. While there is no clear winner, SMC is often more efficient in terms of memory and time consumption [68]. The above experiments are out of scope of probabilistic model checking. Also, there are properties such as clock drift in Clock Synchronization Protocols (see [6]) that could not have been analyzed with a pure formal approach. The PRISM toolset [87] also incorporates a statistical model checking engine. However, it can only be applied to those systems whose individual components are purely stochastic. Moreover, probability distributions are described in a very simple and restrictive language, while we can use the full fledged C to describe complex distributions. Nevertheless, we have observed that PRISM can be faster than our tool on various case studies such as those where the same process is repeated a certain number of times. A comparison between PRISM and SBIP is beyond the scope of this thesis. Solutions to considerably enhance the efficiency of SMC in particular cases have recently been developed [69, 71], but have not yet been implemented in SBIP. In a recent work [24], it has been proposed to use partial order to solve non-determinism when applying SMC (which rarely works). Another approach [21] consists to automatically synthesize distributed scheduling that accounts for concrete implementation information to solve non-determinism. In SBIP, the order is directly given in the design through priorities specified by the user.

We shall continue the development by implementing new heuristics to speed up simulation and to reduce their number as well as techniques to support unbounded properties. We shall also implement an extension of the stochastic abstraction principle from [6] that allows to compute automatically a small stochastic abstraction from a huge concrete system.

Chapter 4

Command-based Importance Sampling for Rare Properties

4.1 Introduction

Since statistical model checking relies on multiple independent simulations, it may be efficiently divided on parallel computer architectures, such as grids, clusters, clouds and general purpose computing on graphics processors. Despite this, complex models often require a large number of simulation steps and rare properties require a large number of simulations. Hence, while statistical model checking may make a verification task feasible, it may nevertheless be computationally intense. In particular, rare properties pose a particular problem for simulation-based approaches, since they are not only difficult to observe (by definition) but it is difficult to bound [58]. Two approaches are used in rare event simulation to provide statistical results: Importance Splitting and Importance Sampling.

Although the term 'rare event' is ubiquitous in the literature, here we specifically consider rare *properties* of paths, defined in temporal logic. This extends the common notion of rarity from states to paths. States are rare if the probability of reaching them from the initial state is small. Paths are rare if the probability of executing their sequence of states is unlikely – whether or not the states themselves are rare. Rare properties are therefore more general than rare states, however the distinction does not significantly alter the mathematical derivation of our algorithm. It can nevertheless affect the applicability of importance sampling. In particular, it is possible to construct pathological systems and properties for which there is no good importance sampling distribution using the states and transitions of the original system. This point is explored in Section 4.6.

Our goal is to estimate the probability of a property by simulation and bound the error of our estimation. When the property is not rare there are standard bounding formulae (e.g., the Chernoff bound [101]) that relate absolute error, confidence and the required number of simulations to achieve them, *independent* of the probability

of the property. As the property becomes rarer, however, absolute error ceases to be useful and it is necessary to consider relative error, defined as the standard deviation of the estimate divided by its expectation. With Monte Carlo simulation relative error is unbounded with increasing rarity [116], but it is possible to bound the error by means of importance sampling [121, 58].

Importance sampling is a technique that can improve the efficiency of simulating rare events and has been receiving considerable interest of late in the field of statistical model checking (e.g., [35, 5]). It works by simulating under an (importance sampling) distribution that makes a property more likely to be seen and then uses the results to calculate the probability under the original distribution by compensating for the differences. The concept arose from work on the 'Monte Carlo method' [98] in the Manhattan project during the 1940s and was originally used to quantify the performance of materials and solve otherwise intractable analytical problems with limited computer power (see, e.g., [75]). For importance sampling to be effective it is necessary to define a 'good' importance sampling distribution:

- (i) The property of interest must be seen frequently in simulations,
- (ii) The distribution of the paths that satisfy the property in the importance sampling distribution must be as close as possible to the distribution of the same paths in the original distribution (up to a normalising factor).

The term 'zero variance' is often used in the literature to describe an optimal importance sampling distribution, referring to the fact that with an optimum importance sampling distribution all simulated paths satisfy the property and the estimator has zero variance. It is important to note, however, that a sub-optimal distribution may meet requirement (i) without necessarily meeting requirement (ii). Failure to consider (ii) can result in gross errors and overestimates of confidence (e.g. a distribution that simulates just one path that satisfies the given property). The algorithm we present in Section 4.4 addresses both (i) and (ii).

4.1.1 Related work

This last decade, several articles presented Importance Sampling as an efficient technique to address the rare-event problem in Statistical Model Checking. One of the key issues is to find a good distribution to bias the system. In [35], the authors present the Cross-Entropy method as an efficient algorithm to adress this problem and apply it on a cyber-physical system. They report well-known results about this technique introduced by Rubinstein in [117]. In [110], the authors consider a benchmark of typed components. Every component is either operational or failing. When a component is failing, it can be repaired with high probability whereas when it is operational the probability of failure is very low. The system is considered "globally failing" if some components of different types are failing at the same time. The property to check is, "starting from the operational state, reach a global failure

specification within a given amount of time". The authors heuristically construct an importance sampling distribution based on the most likely paths (to a global failure). Nevertheless, even if reduction variance or asymptotical optimality have been obtained for importance sampling in several contexts, theoretical results don't provide any reliable confidence interval for the estimated probability since the distribution of the likelihood ratio is unknown. In most cases, the central limit theorem is applied by the use of an approximation of the unknown variance of the estimate. This approach can lead to wrong results if there is no guarantee of controlling this variance. In [5], the authors set up a framework using coupling theory in order to guarantee the variance reduction and provide a reliable confidence interval. The necessary assumptions for achieving this goal and the use of coupling theory however restricts the class of systems on which the method is available.

In [35] the authors present a specific application of the cross-entropy method to a simple continuous time failure model. The system comprises independent components that fail at times that are exponentially distributed. By considering the first simultaneous failure of all components (a rare event), the authors are able to use a standard closed form solution to find an importance sampling distribution that increases the occurrence of the rare event. Although the notions of temporal logic and statistical model checking are introduced, they effectively play no part because the technique is not generalisable to other properties or systems.

In [5] the authors attempt to address the important challenge of bounding the error of estimates when using importance sampling with statistical model checking (we discuss this open challenge in Section 4.7). The work proposes some interesting ideas, however it does not actually provide any practical solutions. The problem considered is a rare property in a system that is intractable to numerical methods. The basic idea is to perform numerical analysis on a reduced (abstracted) model of the system, in order to infer importance sampling parameters for the full model that will allow statistical confidence to be specified. The authors assume the existence of a suitable property-dependent abstraction function that maps states in the full model to states in the abstracted model, in such a way that all abstracted traces that satisfy the property have probability greater than or equal to the traces they abstract. No algorithmic means of generating such a function is provided—the 'coupling' mentioned in the title is only a way to verify that an existing function is correct—and this is generally non-trivial. Since the abstraction function is also specific to a particular property, we believe these ideas do not yet have a practical application.

4.1.2 Contribution

In what follows we consider discrete space Markov models and present a simple algorithm to find an optimal set of importance sampling parameters, using the concept of minimum cross-entropy (min C-E) [85, 122]. In [112] the author proposes a min

C-E algorithm that is asymptotically optimal but requires the storage of an entire transition matrix. This effectively negates the advantage of simulation. In our case, the parametrisation arises naturally from the syntactic description of the model and thus constitutes a low dimensional vector in comparison to the state space of the model. We show that this parametrisation has a unique optimum and demonstrate its effectiveness on reliability and (bio)chemical models. We describe the advantages and potential pitfalls of our approach and highlight areas for future research.

4.1.3 Specification of the model

In order to use the algorithm in Section 4.4, the behaviour of the models must be describable by a set of commands $C_k = (g_k, \eta_k, h_k)$ defined as follows:

- The guard g_k is a predicate over all the variables in the model.
- The function η_k is a function from the set of variables of the system to $\mathbb{R}^+ \setminus \{0\}$.
- The function h_k is an update function of the variables of the system.

Each update describes a transition which can be taken only if the guard is true. A transition is specified by assigning new values to the variables of the system. When several guards are true, an update h_k is chosen with probability equal to the rate η_k divided by the sum of rates of all transitions such that their guard is true.

Remark 3 *Note that each command governs a set of transitions and not necessarily a single individual transition. The models are thus described in a much compact and convenient way.*

Example 4 *The language used for describing an individual SBIP component, as well as the Prism language [86], are examples of language in which the models are described through a set of commands.*

4.2 Monte Carlo Integration and Importance Sampling

Statistical model checking is based on the concept of Monte Carlo integration [114, Ch. 3]. Given a random variable X, with sample space Ω and probability measure f, the expectation of a function $z(X)$ can be expressed as

$$\mathrm{E}[z(X)] = \int_{\Omega} z(\omega)\,\mathrm{d}f(\omega). \tag{4.1}$$

Monte Carlo integration works by drawing N samples $\omega_i \sim f$, $i \in \{1, \ldots, N\}$, to estimate $\mathrm{E}[z(X)]$ according to

$$\mathrm{E}[z(X)] \approx \frac{1}{N} \sum_{i=1}^{N} z(\omega_i). \tag{4.2}$$

Figure 4.1a illustrates how (4.2) works. The outer square denotes the space of all traces Ω, the leaf shape denotes the set of traces that satisfy ϕ. The red dots are uniformly sampled at random from Ω, such that the fraction of samples falling within the leaf is an approximation of the probability that the system will satisfy ϕ. With increasing N, the right hand side of (4.2) is guaranteed to converge to the left hand side by the law of large numbers. In the context of statistical model checking, the function z takes values in $\{1, 0\}$, indicating whether a simulation trace ω_i satisfies property ϕ or does not, respectively. Equation (4.2) thus estimates the probability that a system will satisfy probability ϕ. In the specific context of statistical model checking, we denote this particular function z by $\mathbb{1}(\omega_i \models \phi)$, to emphasise its characteristics.

Let Ω be a sample space of paths, with f a probability measure over Ω and $z(\omega) \in \{0, 1\}$ a function indicating whether a path ω satisfies some property ϕ. In the present context, z is defined by a formula of an arbitrary temporal logic over execution traces. The probability γ that ϕ occurs in a path is then given by

$$\gamma = \int_{\Omega} z(\omega) \, \mathrm{d}f(\omega) \tag{4.3}$$

and the standard Monte Carlo estimator of γ is given by

$$\tilde{\gamma} = \frac{1}{N_{\mathrm{MC}}} \sum_{i=1}^{N_{\mathrm{MC}}} z(\omega_i) \tag{4.4}$$

N_{MC} denotes the number of simulations used by the Monte Carlo estimator and ω_i is sampled according to f. Note that $z(\omega_i)$ is effectively the realisation of a Bernoulli random variable with parameter γ. Hence $\mathrm{Var}(\tilde{\gamma}) = \gamma(1 - \gamma)$ and for $\gamma \to 0$, $\mathrm{Var}(\tilde{\gamma}) \approx \gamma$.

Let f' be another probability measure over Ω, absolutely continuous with zf, then in virtue of Radon-Nikodym theorem [99], (4.3) can be written

$$\gamma = \int_{\Omega} z(\omega) \frac{\mathrm{d}f(\omega)}{\mathrm{d}f'(\omega)} \, \mathrm{d}f'(\omega) \tag{4.5}$$

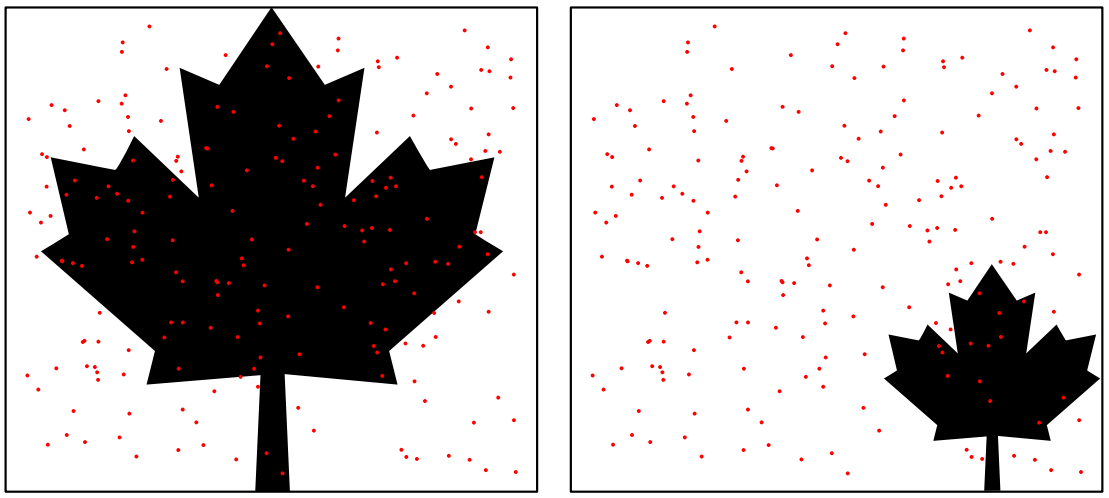
$L = \frac{\mathrm{d}f(\omega)}{\mathrm{d}f'(\omega)}$ is the *likelihood ratio* function, so

$$\gamma = \int_{\Omega} L(\omega) z(\omega) \, \mathrm{d}f'(\omega) \tag{4.6}$$

We can thus estimate γ by simulating under f' and compensating by L:

$$\tilde{\gamma}_{N_{IS}} = \frac{1}{N_{\mathrm{IS}}} \sum_{i=1}^{N_{\mathrm{IS}}} L(\omega_i) z(\omega_i) \tag{4.7}$$

N_{IS} denotes the number of simulations used by the importance sampling estimator. The goal of importance sampling is to reduce the variance of the rare event and



(a) (b)

Figure 4.1: Monte Carlo integration.

so achieve a narrower confidence interval than the Monte Carlo estimator, resulting in $N_{\text{IS}} \ll N_{\text{MC}}$. In general, the importance sampling distribution f' is chosen to produce the rare property more frequently, but this is not the only criterion. The optimal importance sampling distribution, denoted f^* and defined as f conditioned on the rare event, produces only traces satisfying the rare property and satisfies:

$$\mathrm{d}f^* = \frac{z\mathrm{d}f}{\gamma} \tag{4.8}$$

This leads to the term 'zero variance estimator' with respect to Lz. Indeed, under f^*, every path ω has a likelihood ratio equal to $\gamma/z(\omega)$ when $f(\omega) > 0$ and so:

$$Var(\tilde{\gamma}_{N_{IS}}) = \frac{1}{N_{IS}} Var_{f^*}(L(X)z(X)) \tag{4.9}$$

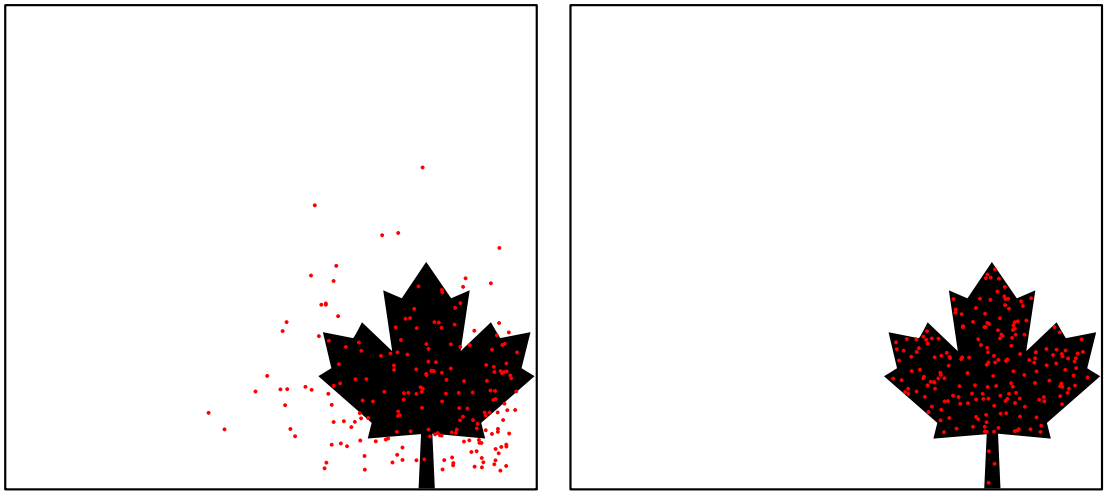
$$= \frac{1}{N_{IS}} Var_{f^*}(\gamma) \tag{4.10}$$

$$= 0 \tag{4.11}$$

Figure 4.1b illustrates the situation when a property is rare. Fewer samples fall within the leaf and, moreover, the coverage of the leaf is apparently less uniform than in Fig. (4.1a). Unbiased convergence is still guaranteed with increasing N, but the variance of the estimate is higher.

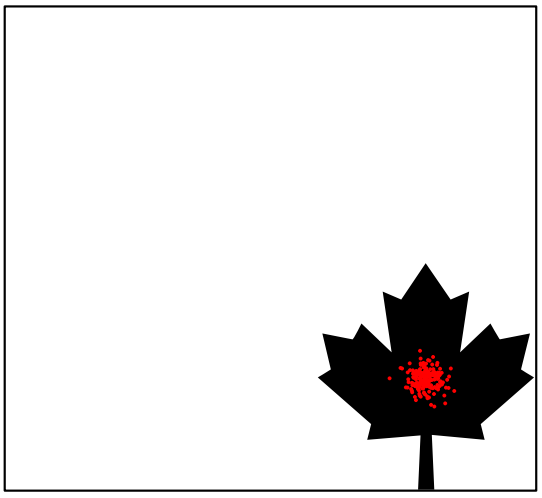
Figure 4.2a illustrates the basic notion of importance sampling. The sampling distribution is weighted in such a way that most of the samples fall within the leaf. The fraction of samples falling within the leaf is no longer an approximation of the probability we seek, but knowing the values of the weights it is possible to compensate and gain an unbiased estimate.

Figure 4.2b illustrates the notion of a perfect importance sampling distribution. All the samples fall within the leaf and the coverage is uniform. In practice, it



(a) (b)

Figure 4.2: Importance Sampling integration.



(a)

Figure 4.3: A wrong Importance Sampling integration

is usually only possible to observe the percentage of successful simulations and not possible to judge how uniformly the distribution covers the target area. The percentage of success does not necessarily indicate the quality of the importance sampling distribution. For example, the distribution illustrated in Fig. 4.3a produces 100% success but is focused on a small percentage of the target area. This distribution will produce a severe underestimate of the true probability.

4.2.1 Importance Sampling for Command Systems

Importance sampling schemes fall into two broad categories: *state dependent tilting* and *state independent tilting* [37]. State dependent tilting refers to importance sampling distributions that individually weight ('tilt') every transition probability in the system. State independent tilting refers to importance sampling distributions that change classes of transition probabilities, independent of state. The former offers greater precision but is infeasible for large models. The latter is more tractable but may not produce good importance sampling distributions. Our approach may be seen as *parametrised tilting*, that potentially affects all transitions differently, but does so according to a set of parameters.

In the context of statistical model checking, the function f usually arises from the specifications of a model described in some relatively high level language. Such models do not, in general, explicitly specify the probabilities of individual transitions, but do so implicitly by parametrised functions over the states. We therefore consider a class of models that can be described by guarded commands [40] extended with stochastic rates. Our parametrisation is a vector of strictly positive values $\lambda \in (\mathbb{R}^+)^n$ that multiply the stochastic rates and thus maintain the absolutely continuous property between distributions. Note that this class includes both discrete and continuous time Markov chains and that in the latter case our mathematical treatment works with the embedded discrete time process.

In what follows we are therefore interested in parametrised distributions and write $f(\cdot, \lambda)$, where $\lambda = \{\lambda_1, \ldots, \lambda_n\}$ is a vector of parameters, and distinguish different probability measures by their parameters. In particular, μ is the original vector of the model and $f(\cdot, \mu)$ is therefore the original measure. We can thus rewrite (4.6) as

$$\gamma = \int_\Omega L(\omega) z(\omega) \, \mathrm{d}f(\omega, \lambda), \tag{4.12}$$

where $L(\omega) = \mathrm{d}f(\omega, \mu)/\mathrm{d}f(\omega, \lambda)$. We can also rewrite (4.8) as

$$\mathrm{d}f^* = \frac{z \, \mathrm{d}f(\cdot, \mu)}{\gamma} \tag{4.13}$$

and write for the optimal parametrised measure $f(\cdot, \lambda^*)$. We define the optimum parametrised probability measure as the measure that minimises the *cross-entropy* [85] between $f(\cdot, \lambda)$ and f^* for a given parametrisation and note that, in general, $f^* \neq f(\cdot, \lambda^*)$.

4.3 The Cross-Entropy Method

Cross-entropy [85] (alternatively *relative entropy* or Kullback-Leibler divergence) has been shown to be a uniquely correct directed measure of distance between distributions [122]. With regard to the present context, it has also been shown to be useful in finding optimum distributions for importance sampling [117, 37, 112].

Given two probability measures f and f' over the same probability space Ω, the cross-entropy from f to f' is given by

$$\mathrm{CE}(f, f') = \int_\Omega \log \frac{\mathrm{d}f(\omega)}{\mathrm{d}f'(\omega)} \, \mathrm{d}f(\omega) \tag{4.14}$$

$$= \int_\Omega \log \mathrm{d}f(\omega) - \log \mathrm{d}f'(\omega) \, \mathrm{d}f(\omega) \tag{4.15}$$

$$= \mathrm{H}(f) - \int_\Omega \log \mathrm{d}f'(\omega) \, \mathrm{d}f(\omega) \tag{4.16}$$

where $\mathrm{H}(f)$ is the entropy of f. To find λ^* we minimise:

$$\min_{f(.,\lambda)} \mathrm{CE}(f^*, f(., \lambda)), \tag{4.17}$$

noting that $\mathrm{H}(f(\omega, \mu))$ is independent of λ:

$$\lambda^* = \arg\max_\lambda \int_\Omega z(\omega) \log \mathrm{d}f(\omega, \lambda) \, \mathrm{d}f(\omega, \mu) \tag{4.18}$$

Estimating λ^* directly using (4.18) is hard, so we re-write it using importance sampling measure $f(\cdot, \lambda')$ and likelihood ratio function $L(\omega) = \mathrm{d}f(\omega, \mu)/\mathrm{d}f(\omega, \lambda')$:

$$\lambda^* = \arg\max_\lambda \int_\Omega z(\omega) L(\omega) \log \mathrm{d}f(\omega, \lambda) \, \mathrm{d}f(\omega, \lambda') \tag{4.19}$$

Using (4.19) we can construct an unbiased importance sampling estimator of λ^* and use it as the basis of an iterative process to obtain successively better estimates:

$$\tilde{\lambda}^* = \lambda^{(j+1)} = \arg\max_\lambda \sum_{i=1}^{N} z(\omega_i^{(j)}) L^{(j)}(\omega_i^{(j)}) \log \mathrm{d}f(\omega_i^{(j)}, \lambda) \tag{4.20}$$

N is the number of simulation runs on the iterations, $\lambda^{(j)}$ is the j^{th} set of estimated parameters, $L^{(j)}(\omega) = f(\omega, \mu)/f(\omega, \lambda^{(j)})$ is the j^{th} likelihood ratio function, $\omega_i^{(j)}$ is the i^{th} path generated using $f(\cdot, \lambda^{(j)})$ and $f(\omega_i^{(j)}, \lambda)$ is the probability of path $\omega_i^{(j)}$ under the distribution $f(\cdot, \lambda^{(j)})$.

4.4 Command-based Cross-Entropy Algorithm

Let P be the original matrix of transition of the stochastic system $\mathcal{S}$. In [112], the author considers the class $\mathcal{Q}$ of all the stochastic matrices absolutely continuous

with P such that the transition graphs are similar up to the rates over the edges. He finds the 'best' matrix with respect to the Kullback-Leibler divergence by means of the Cross-entropy algorithm. The parameter λ is here the vector of all the non-zero individual transition probabilities of P. The parameter is then updated until convergence of the algorithm. This algorithm is optimal in the sense that the author considers the biggest possible family of parametric distributions. However it implies that the full matrix has to be stored which makes difficult the use of the technique for very large systems. A question is: is it possible to consider a subclass of $\mathcal{Q}$, generated by a vector of lower dimension, such that the 'best' distribution in this subclass gives decent results?

Moreover, we consider here stochastic systems generated by commands. It could be that for any engineering cause we are not able to reason directly on the underlying matrix of transition. In this case, using an alternative algorithm is of prime interest.

We consider a system of n guarded commands with vector of rate functions $\eta = (\eta_1, \ldots, \eta_n)$ and corresponding vector of parameters $\lambda = (\lambda_1, \ldots, \lambda_n)$. We thus define n classes of transitions. In any given state x_s reached after s transitions, the probability that command $k \in \{1 \ldots n\}$ is chosen is given by

$$\frac{\lambda_k \eta_k(x_s)}{\langle \eta(x_s), \lambda \rangle}$$

where η is parametrised by x_s to emphasise its state dependence and the notation $\langle \cdot, \cdot \rangle$ denotes a scalar product. For the purposes of simulation we consider a space of finite paths $\omega \in \Omega$. Let $U_k(\omega)$ be the number of transitions of type k occurring in ω. Let $\bigsqcup_{k=1}^{n} J_k(\omega) = \{0, \cdots, |\omega| - 1\}$ the disjoint union of sets such that each $J_k(\omega)$ contains the indices of states in which a type-k transition occurred in path ω. We therefore have

$$\mathrm{d}f(\omega, \lambda) = \prod_k^n \left((\lambda_k)^{U_k(\omega)} \prod_{s \in J_k(\omega)} \frac{\eta_k(x_s)}{\langle \eta(x_s), \lambda \rangle} \right)$$

The likelihood ratios are thus of the form

$$L^{(j)}(\omega) = \prod_k^n \left(\left(\frac{\mu_k}{\lambda_k^{(j)}} \right)^{U_k(\omega)} \prod_{s \in J_k(\omega)} \frac{\langle \eta(x_s), \lambda^{(j)} \rangle}{\langle \eta(x_s), \mu \rangle} \right)$$

We define $\eta_k^{(i)}(x_s)$ and $\eta^{(i)}(x_s)$ as the respective values of η_k and η functions in state x_s of the i^{th} trace. We substitute the previous expressions in the cross-entropy estimator (4.20) and for compactness substitute $z_i = z(\omega_i)$, $J_k^{(i)} = J_k(\omega_i)$, $u_i(k) = U_k(\omega_i)$ and

$l_i = L^{(j)}(\omega_i)$ to get

$$\arg\max_{\lambda} \sum_{i=1}^{N} l_i z_i \log \prod_{k}^{n} \left(\lambda_k^{u_i(k)} \prod_{s \in J_k^{(i)}} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda \rangle} \right) \tag{4.21}$$

$$= \arg\max_{\lambda} \sum_{i=1}^{N} \sum_{k}^{n} l_i z_i \left(u_i(k)\ \log(\lambda_k) + \sum_{s \in J_k^{(i)}} \log(\eta_k^{(i)}(x_s)) - \sum_{s \in J_k^{(i)}} \log(\langle \eta^{(i)}(x_s), \lambda \rangle) \right)$$

We denote $F(\lambda)$ the second member to maximise in the previous equality. We partially differentiate with respect to λ_k and get the non-linear system

$$\frac{\partial F}{\partial \lambda_k}(\lambda) = 0 \Leftrightarrow \sum_{i=1}^{N} l_i z_i \left(\frac{u_i(k)}{\lambda_k} - \sum_{s=1}^{|\omega_i|} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda \rangle} \right) = 0 \tag{4.22}$$

where $|\omega_i|$ is the length of the path ω_i.

Theorem 4 *A solution of* (4.22) *is almost surely a unique maximum, up to a normalising scalar.*

Proof *Consider:*

$$F_i(\lambda) = \sum_{k=1}^{n} \left(u_i(k)\ \log(\lambda_k) + \sum_{s \in J_k^{(i)}} \log(\eta_k^{(i)}(x_s)) - \sum_{s \in J_k^{(i)}} \log(\langle \eta^{(i)}(x_s), \lambda \rangle) \right)$$

and $F_{i,k}(\lambda)$ each element of this sum. Thus, note that $F_i(\lambda) = \sum_{k=1}^{n} F_{i,k}(\lambda)$ and $F(\lambda) = \sum_{i=1}^{N} l_i z_i F_i(\lambda)$. For sake of simplicity, we sometimes omit index i in the notations. Using a standard result, it is sufficient to show that the Hessian matrix H of F in λ is negative semi-definite.

The Hessian matrix H_i of F_i in λ is of the following form with $v_k^{(s)} = \frac{\eta_k(x_s)}{\langle \eta(x_s), \lambda \rangle}$ and $v_k = (v_k^{(s)})_{1 \leq s \leq |\omega|}$:

$$H_i = G - D$$

where $G = (g_{kk'})_{1 \leq k,k' \leq n}$ is the following Gram matrix

$$g_{kk'} = \langle v_k, v_{k'} \rangle$$

and D is a diagonal matrix such that

$$d_{kk} = \frac{U_k(\omega)}{\lambda_k^2}.$$

Remark that $U_k(|\omega|)$ is the number of times a transition of type k has been choven over $|\omega|$ transitions. In average, $U_k(\omega)$ is equal to the sum of probabilities of choosing

transition k in each state x_s. Thus, in average, $d_{kk} = \frac{1}{\lambda_k}\sum_{s=1}^{|\omega|} v_k^{(s)}$. We write $\mathbf{1}_{|\omega|} = (1,\ldots,1)$ for the vector of $|\omega|$ elements 1, hence

$$d_{kk} = \frac{1}{\lambda_k}\langle v_k, \mathbf{1}_{|\omega|}\rangle.$$

Furthermore, $\forall s, \sum_{k=1}^{n} \lambda_k v_k^{(s)} = 1$. So, $\sum_{k'=1}^{n} \lambda_{k'} v_{k'} = \mathbf{1}_{|\omega|}$. Finally,

$$d_{kk} = \sum_{k'=1}^{n} \frac{\lambda_{k'}}{\lambda_k}\langle v_k, v_{k'}\rangle.$$

Let x be a non-zero vector of $\mathbb{R}^n$. To prove the theorem we need to show that $-x^t H_i x \geq 0$.

$$\begin{aligned} -x^t H_i x &= x^t D x - x^t G x \\ &= \sum_{k,k'} \frac{\lambda_{k'}}{\lambda_k}\langle v_k, v_{k'}\rangle x_k^2 - \sum_{k,k'}\langle v_k, v_{k'}\rangle x_k x_{k'} \\ &= \sum_{k<k'} \left(\left[\frac{\lambda_{k'}}{\lambda_k} x_k^2 + \frac{\lambda_k}{\lambda_{k'}} x_{k'}^2 - 2 x_k x_{k'} \right] \langle v_k, v_{k'}\rangle \right) \\ &= \sum_{k<k'} \left(\sqrt{\frac{\lambda_{k'}}{\lambda_k}} x_k - \sqrt{\frac{\lambda_k}{\lambda_{k'}}} x_{k'} \right)^2 \langle v_k, v_{k'}\rangle \\ &\geq 0 \end{aligned} \tag{4.23}$$

The Hessian matrix H of F is of the general form

$$H = \sum_{i=1}^{N} l_i z_i H_i$$

which is a positively weighted sum of non-positive matrices.

Moreover, for all $\lambda \in \mathbb{R}^{+n}$,

$$\left(x^t H x = 0\right) \Leftrightarrow \left(\forall k \forall k' > k,\ x_k \neq 0 \wedge \frac{\lambda_{k'}}{\lambda_k} = \frac{x_{k'}}{x_k} \right) \Leftrightarrow (\exists r \in \mathbb{R}^*,\ x = r\lambda) \tag{4.24}$$

This is because for all $\lambda \in \mathbb{R}^n$, $F(\lambda) = F(r\lambda)$ for all $r \in \mathbb{R}^*$. Geometrically, it means that the function is flat along a line generated by a vector λ. If λ was a solution of (4.22) then $r\lambda, r \in \mathbb{R}^+$, would also be a solution.

Assume now that there exists λ and μ two non-collinear vectors, solutions of (4.22). By concavity of F, these two vectors are global maximum of F and it implies that F is a constant over the cone generated by vectors λ and μ. In particular, function F would be constant along the line segment $\alpha\lambda + (1-\alpha)\mu$ with $\alpha \in [0,1]$. Let $y \in \mathbb{R}^n$ the direction vector of the line containing this segment and ν an element in the interior of this segment.

Denoting $H(\nu)$ the Hessian of F at point ν, $y^t H(\nu) y = 0$. But y is not collinear to vector ν, that contradicts hypothesis (4.24).

A solution λ^* of (4.22) is thus a unique maximum up to a linear constraint over its norm.

The fact that there is a unique optimum makes it conceivable to find λ^* using standard optimising techniques such as Newton and quasi-Newton methods. To do so would require introducing a suitable normalising constraint in order to force the Hessian to be negative definite. In the case of the cross-entropy algorithm of [112], this constraint is inherent because it works at the level of individual transition probabilities that sum to 1 in each state. We note here that in the case that our parameters apply to individual transitions, such that one parameter corresponds to exactly one transition, (4.27) may be transformed to Equation (9) of [112] by constraining in every visited state x, $\langle \eta(x), \lambda \rangle = 1$. Equation (9) of [112] has been shown in [113] to converge to f^*, implying that under these circumstances $f(\cdot, \lambda^*) = f^*$ and that it may be possible to improve our parametrised importance sampling distribution by increasing the number of parameters.

Equation (4.22) leads to the following expression for λ_k:

$$\lambda_k = \frac{\sum_{i=1}^{N} l_i z_i u_i(k)}{\sum_{i=1}^{N} l_i z_i \sum_{s=1}^{|\omega_i|} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda \rangle}} \tag{4.25}$$

In this form the expression is not useful because the right hand side is dependent on λ_k in the scalar product. Hence, in contrast to update formulae based on unbiased estimators, as given by (4.20) and in [112, 37], we construct an iterative process based on a biased estimator, but having a fixed point that is the optimum:

$$\lambda_k^{(j+1)} = \frac{\sum_{i=1}^{N} l_i z_i u_i(k)}{\sum_{i=1}^{N} l_i z_i \sum_{s=1}^{|\omega_i|} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda^{(j)} \rangle}}. \tag{4.26}$$

Equation (4.26) is the basis of our algorithm and can be seen as an implementation of (4.25) that uses the previous estimate of λ in the scalar product. As a result, in contrast to previous applications of the cross-entropy method, (4.26) converges by reducing the distance between successive distributions, rather than by explicitly reducing the distance from the optimum.

4.4.1 Smoothing

It is conceivable that certain guarded commands play no part in traces that satisfy the property, in which case (4.26) would make the corresponding parameter zero with no adverse effects. It is also conceivable that an important command is not seen on a particular iteration, but making its parameter zero would prevent it being seen on any subsequent iteration. To avoid this it is necessary to adopt a 'smoothing' strategy [112] that reduces the significance of an unseen command without setting it to zero. Smoothing therefore acts to preserve important but as yet unseen parameters. It is of increasing importance as the parametrisation gets closer to the level of individual transition probabilities, since only a tiny proportion of possible transitions are usually seen on any simulation run. Typical strategies include adding

a small fraction of the original parameters, or a fraction of the parameters from the previous iteration, to the new parameter estimate. With smoothing parameter $\alpha \in]0,1[$, these two strategies can be summarised as follows:

- Weighting with the original parameters:

$$\lambda_k^{(j+1)} = \alpha\mu_k + (1-\alpha)\frac{\sum_{i=1}^{N} l_i z_i u_i(k)}{\sum_{i=1}^{N} l_i z_i \sum_{s=1}^{|\omega_i|} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda^{(j)} \rangle}} \tag{4.27}$$

- Weighting with the previous parameters:

$$\lambda_k^{(j+1)} = \alpha\lambda_k^{(j)} + (1-\alpha)\frac{\sum_{i=1}^{N} l_i z_i u_i(k)}{\sum_{i=1}^{N} l_i z_i \sum_{s=1}^{|\omega_i|} \frac{\eta_k^{(i)}(x_s)}{\langle \eta^{(i)}(x_s), \lambda^{(j)} \rangle}} \tag{4.28}$$

We have found that our parametrisation is often insensitive to smoothing strategy because each parameter typically governs many transitions and most parameters are affected by each run. The smoothing strategy adopted in the case studies described below is to divide the parameter of unseen commands by two (a compromise between speed of convergence and safety). The effects of this can be seen clearly in Figure 4.10. Whatever the strategy, since the parameters are unconstrained it is advisable to normalise them after each iteration (i.e., $\sum_k \lambda_k = \text{const.}$), in order to quantify convergence.

4.4.2 Convergence

To show that our algorithm converges, we prove convergence of (4.26), under the standard assumption of sufficient simulations per iteration. We first recall that Theorem 4 proves that there is a unique optimum (λ^*) of (4.22), which is therefore the unique solution of (4.25). By inspection and comparison with (4.25), we see that any fixed point of (4.26) is also a solution of (4.25). Since (4.25) has a unique solution, (4.26) has a unique fixed point that is the optimum. □

The inclusion of smoothing in the algorithm is a practical measure to prevent parameters being rejected prematurely when using finite numbers of simulations. Smoothing may have the undesirable side effect of slowing convergence and, when using (4.27), may prevent the algorithm from reaching the theoretical optimum. E.g., if the optimal value of λ_k is ≈ 0, (4.27) will nevertheless set $\lambda_l = \alpha\mu_k$. In practice, however, the smoothing strategy is chosen to avoid problems and have insignificant effect on the final distribution.

Given an adequate initial distribution and a sufficient number of successful traces from the first iteration, (4.27) and (4.28) should provide a better set of parameters. In practice we have found that a single successful trace is often sufficient to initiate convergence. This is in part due to the existence of a unique optimum and partly to the fact that each parameter generally governs a large number of semantically-linked

transitions. The expected behaviour is that on successive iterations the number of traces that satisfy the property increases, however it is important to note that the algorithm minimises the cross-entropy and that the number of traces that satisfy the property is merely emergent of that. As has been noted, in general $f(\cdot, \lambda^*) \neq f^*$, hence it is likely that fewer than 100% of traces will satisfy the property when simulating under $f(\cdot, \lambda^*)$. One consequence of this is that an initial set of parameters may produce more traces that satisfy the property than the final set (see, e.g., Figs. (4.4) and 4.8).

Once the parameters have converged it is then possible to perform a final set of simulations to estimate the probability of the rare property. The usual assumption is that $N \ll N_{\text{IS}} \ll N_{\text{MC}}$, however it is often the case that parameters converge fast, so it is expedient to use some of the simulation runs generated during the course of the optimisation as part of the final estimation.

Rare event simulation process To summarise the whole process of rare event simulation, we first run Algorithm 1. One has to initialise algorithm 1 with a vector of parameters $\lambda^{(0)}$ supposedly more favourable with respect to property ϕ.

- The first 'while' loop (line 2) corresponds to the cross-entropy iterations. This loop stops when vector of parameters $\lambda^{(j)}$ converges. A convergence criteria can be satisfied, for example, whenever $\max_{0 \leq k \neq l \leq 2} \|\lambda^{(j-k)} - \lambda^{(j-l)}\| \leq \epsilon$
- At line 12, the second 'while' loop is the path generator. Likelihood ratio l_i is updated on-the-fly.
- At line 16, each time a transition of type k is taken, the corresponding coordinate of u_i is incremented by 1.
- Line 23 corresponds to the normalisation of $\lambda^{(j)}$.
- At line 24, parameter $\lambda^{(j)}$ is smoothed by a strategy described in 4.4.1. The resulting parameter is used to generate the new samples.

We then run Algorithm 2 for γ estimation.

4.4.3 Initial Distribution

Algorithm (1) requires an initial simulation distribution ($f(\cdot, \lambda^{(0)})$) that produces at least a few traces that satisfy the property using N_0 simulation runs. Finding $f(\cdot, \lambda^{(0)})$ for an arbitrary model may seem to be an equivalently difficult problem to estimating γ, but this is not in general the case. When a property (e.g., failure of the system) is semantically linked to an explicit feature of the model (e.g, a command for component failure), good initial parameters may be found relatively easily by heuristic methods such as failure biasing [121]. Here are a few strategies to choose an initial parameter:

Algorithm 1: Cross-Entropy Algorithm for Parametrised Commands

Data: Let μ be the original parameter, $\lambda^{(0)}$ the initial parameter and N the number of paths per iteration.

$j = 0$
while $\neg cond$ **do**
  $A = \vec{0}$
  $B = 0$
  $S = 0$
  for $i \in \{1, \ldots, N\}$ **do**
    $\omega_i = x_0$
    $l_i = 1$
    $\vec{u_i} = \vec{0}$
    $S = 0$
    $s = 1$
    while $\neg stop$ **do**
      generate x_s under measure $f(., \lambda^{(j)})$
      $\omega_i = x_0 \cdots x_s$
      $l_i \leftarrow l_i \times \frac{\mu(x_{s-1}, x_s)\langle \eta^i(x_{s-1}), \lambda^{(j)} \rangle}{\lambda(x_{s-1}, x_s)\langle \eta^i(x_{s-1}), \lambda^{(j)} \rangle}$
      update $\vec{u_i}$
      $S = S + \frac{\eta_k^i(x_s)}{\langle \eta^i(x_s), \lambda^{(j)} \rangle}$
      $s = s + 1$
    $z_i = \mathbb{1}(\omega_i \models \phi)$
    $A \leftarrow A + l_i z_i \vec{u_i}$
    $B \leftarrow B + l_i z_i S$
    $\lambda_k^{(j+1)} = \frac{A_k}{B}$
    $\lambda^{(j+1)} \leftarrow \frac{\lambda^{(j+1)}}{\|\lambda^{(j+1)}\|}$
    smoothing of $\lambda^{(j+1)}$
  $j = j + 1$
$\lambda^* = \lambda^{(j-1)}$

Algorithm 2: Importance sampling by $f(.,\lambda^*)$

Data: Let μ be the original parameter, λ^* the parameter computed previously and N_{IS} the number of paths

for $i \in \{1,\ldots,N_{IS}\}$ **do**
- $\omega_i = x_0$
- $l_i = 1$
- $s = 1$
- **while** $\neg stop$ **do**
  - generate x_s under measure $f(.,\lambda^*)$
  - $\omega_i = x_0 \cdots x_s$
  - $l_i \leftarrow l_i \times \frac{\mu(x_{s-1},x_s)\langle \eta^i(x_{s-1}),\lambda^{(j)}\rangle}{\lambda(x_{s-1},x_s)\langle \eta^i(x_{s-1}),\lambda^{(j)}\rangle}$
  - $s = s + 1$
- $z_i = \mathbb{1}(\omega_i \models \phi)$

$\tilde{\gamma} = \frac{1}{N_{IS}} \sum_{i=1}^{N_{IS}} z_i l_i$

- If the system depends on a vector of parameters with large variance, take a uniform vector.
- Simulate the system once with n initial points chosen randomly. Then, use the barycenter of the points that produce a success.
- Simulate the sytem m times with n different points chosen randomly. Then, choose the barycenter of the points that produce at least one success or choose the closest successful point from the original in terms of Euclidean distance.

Alternatively, if the model and property are similar to a previous combination for which parameters were found, those parameters are likely to provide a good initial estimate. Increasing the parameters associated to obviously small rates may help (along the lines of failure biasing), however the rareness of a property expressed in temporal logic may not be related to low transition probabilities.

An important consideration is that the rareness of the property in trace space does not imply that good parameters are rare in parameter space. Consequently, a random search of parameter space often requires many orders of magnitude fewer attempts to find an example of the rare property than the expected number under the original distribution (i.e., $1/\gamma$). This phenomenon is the basis of the algorithmic approaches to finding initial distributions given below.

Figure 4.4 illustrates the parameter space of the chemical model described in Section 4.5.1. Although the majority of parameters, including those which generate the original distribution, fall into a region where the probability of satisfying the property is near zero, a significant region of the parameter space ($\approx 37\%$) gives near 100% success. A narrow strip between these two regions (indicated by a grey line

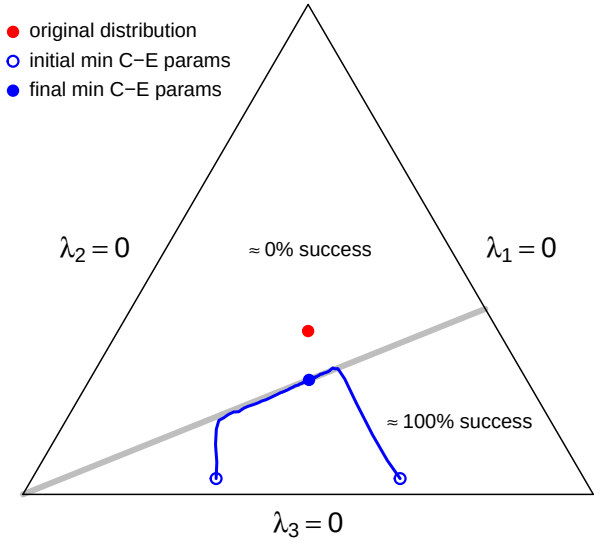


Figure 4.4: Parameter simplex of three parameter chemical model.

in Fig. 4.4) contains parameters with intermediate levels of success, among which is the unique vector of parameters for minimum cross-entropy. The figure also shows how two different initial parameter vectors converge to the optimum.

4.5 Case Studies

The following examples are included to illustrate the performance of our algorithm and parametrisation. The first is an example of a chemical system, often used to motivate stochastic simulation, while the second is a standard repair model. In both cases, initial distributions were found by the heuristic of performing single simulations using parameters drawn uniformly from parameter space and using the first set of parameters that produce a path satisfying the property. For the chosen examples fewer than 500 attempts were necessary; a value less than N and considerably less than $1/\gamma$, the expected number of simulations necessary to see a single successful trace. All simulations were performed using a prototype of our statistical model checker, PLASMA [70] and all the PLASMA or PRISM models are available online[1][2].

[1] http://people.irisa.fr/Cyrille.Jegourel/models.html
[2] https://project.inria.fr/plasma-lab/documentation/examples/

4.5.1 Chemical network

Following the success of the human genome project, with vast repositories of biological pathway data available online, there is an increasing expectation that formal methods can be applied to biological systems. The network of chemical reactions given below is abstract but typical of biochemical systems and demonstrates the potential of statistical model checking to handle the enormous state spaces of biological models. In particular, we demonstrate the efficacy of our algorithm by applying it to quantify two rare dynamical properties of the system.

We consider a well stirred chemically reacting system comprising five reactants (molecules of type A, B, C, D and E), a dimerisation reaction and two decay reactions. We denote the instantaneous number of molecules of A, B, C, D and E by state variables A, B, C, D and E, respectively. The reactions are modelled by three guarded commands having importance sampling parameters λ_1, λ_2 and λ_3, respectively:

$$(A > 0 \wedge B > 0, \lambda_1 \times A \times B, A \leftarrow A - 1; B \leftarrow B - 1; C \leftarrow C + 1) \quad (4.29)$$

$$(C > 0, \lambda_2 \times C, C \leftarrow C - 1; D \leftarrow D + 1) \quad (4.30)$$

$$(D > 0, \lambda_3 \times D, D \leftarrow D - 1; E \leftarrow E + 1) \quad (4.31)$$

Under the assumption that the molecules move randomly and that elastic collisions significantly outnumber unreactive, inelastic collisions, the system may be simulated using mass action kinetics as a continuous time Markov chain [50]. The semantics of (4.29) is that if a molecule of type A encounters a molecule of type B they will combine to form a molecule of type C after a delay drawn from an exponential distribution with mean $\lambda_1 \times A \times B$. The decay reactions have the semantics that a molecule of type C (D) spontaneously decays to a molecule of type D (E) after a delay drawn from an exponential distribution with mean $\lambda_2 \times C$ ($\lambda_3 \times D$). A typical simulation run is illustrated in Fig. 5.2, where the x-axis is steps rather than time to aid clarity. A and B combine rapidly to form C which peaks before decaying slowly to D. The production of D also peaks, while E rises monotonically. With an initial vector of molecules $(10^3, 10^3, 0, 0, 0)$, corresponding to variables (A, B, C, D, E), the state space comprises approximately 1.6×10^8 states and 4.8×10^8 transitions. Although extremely simple in the context of typical biological systems, the model is intractable to numerical analysis. By inspection, we can infer that it is possible for the numbers of molecules of C and D to reach the initial number of A and B molecules (i.e., 1000) and that this is unlikely. To find out exactly how unlikely we consider the probabilities of the following rare properties defined in linear temporal logic: (i) $\Diamond^{3000} C \geq x, x \in \{970, 975, 980, 985, 990, 995\}$ and (ii) $\Diamond^{3000} D \geq y, y \in \{460, 465, 470, 475, 480, 485\}$. The results are plotted in Figure 4.6.

Having found an initial set of parameters by the means described in Section (4.4.3), the Algorithm (1) was iterated 20 times using $N = 1000$. Despite the large state space, this value of N was found to be sufficient to produce reliable

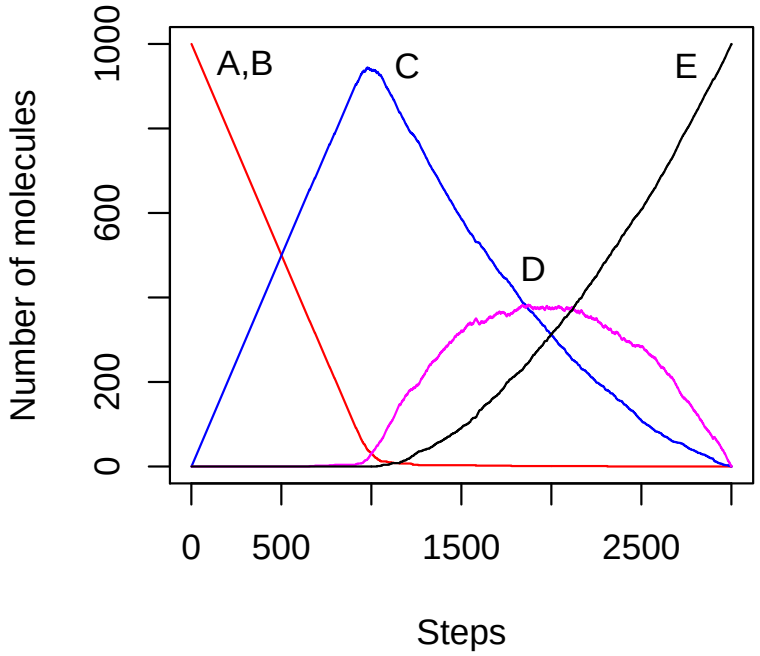


Figure 4.5: A typical stochastic simulation trace of reactions (4.29-4.31).

results. The convergence of parameters for the property $\Diamond^{3000} D \geq 470$ can be seen in Figure 4.7. Figure 4.8 illustrates that the number of paths satisfying a property can actually decrease as the quality of the distribution improves. Figure 4.9 illustrates the convergence of the estimate and sample variance using the importance sampling parameters generated during the course of running the algorithm. The initial set of parameters appear to give a very low variance, however this is clearly erroneous with respect to subsequent values. Noting that the variance of standard Monte Carlo simulation of rare events gives a variance approximately equal to the probability and assuming that the sample variance is close to the true variance, Figure 4.9 suggests that we have made a variance reduction of approximately 10^7.

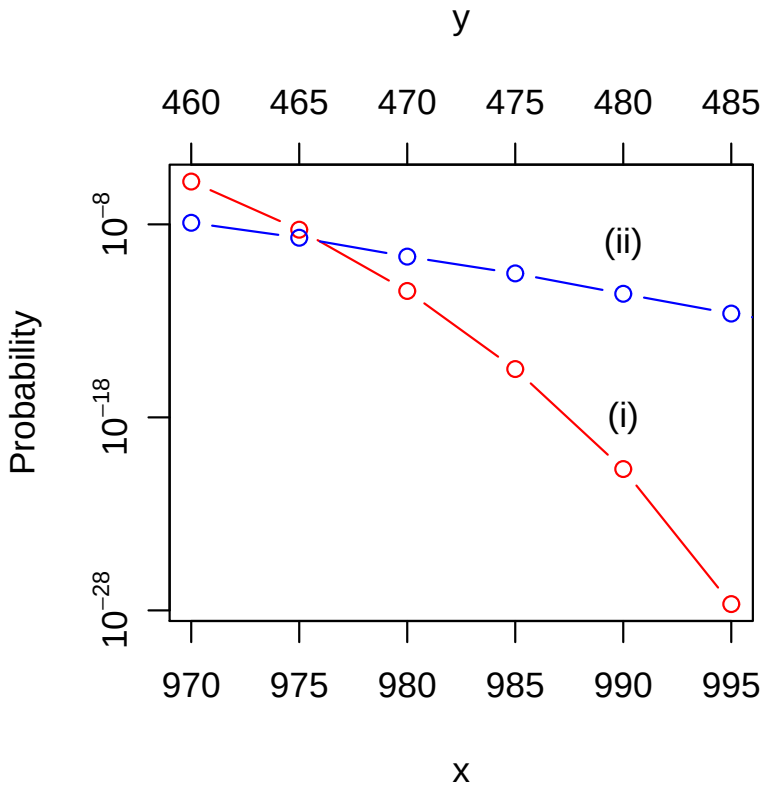


Figure 4.6: (i) $\mathrm{P}[\Diamond^{3000} C \geq x]$ (ii) $\mathrm{P}[\Diamond^{3000} D \geq y]$

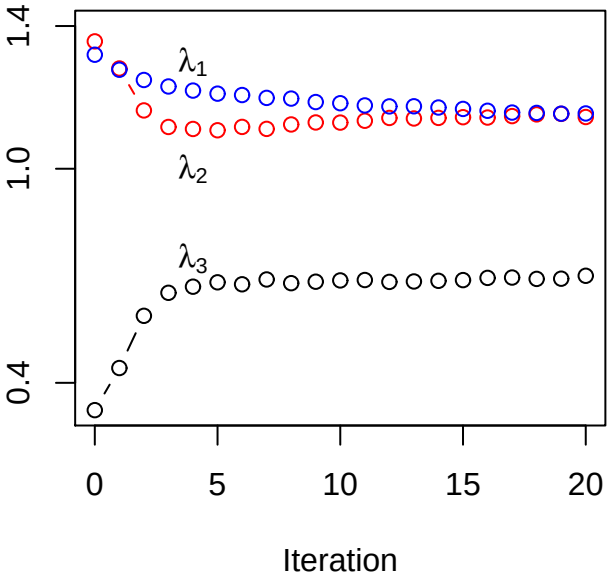


Figure 4.7: Convergence of parameters for $\Diamond^{3000} D \geq 470$ in the chemical model using $N = 1000$.

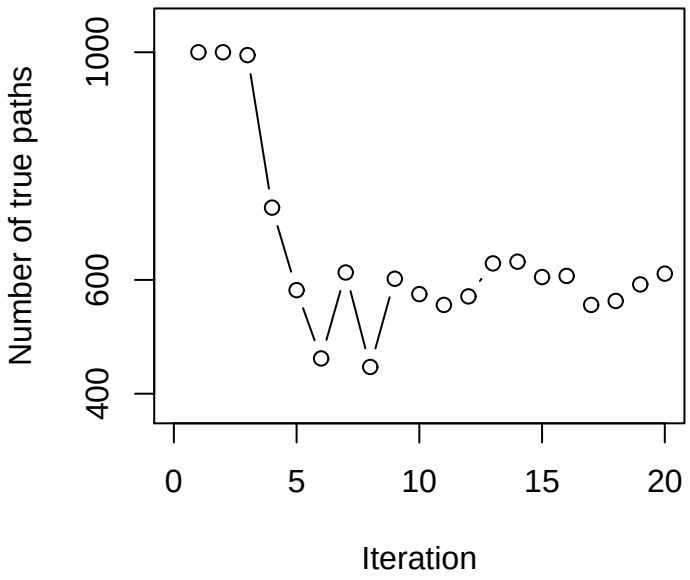


Figure 4.8: Convergence of number of paths satisfying $\Diamond^{3000}\, D \geq 470$ in the chemical model using $N = 1000$.

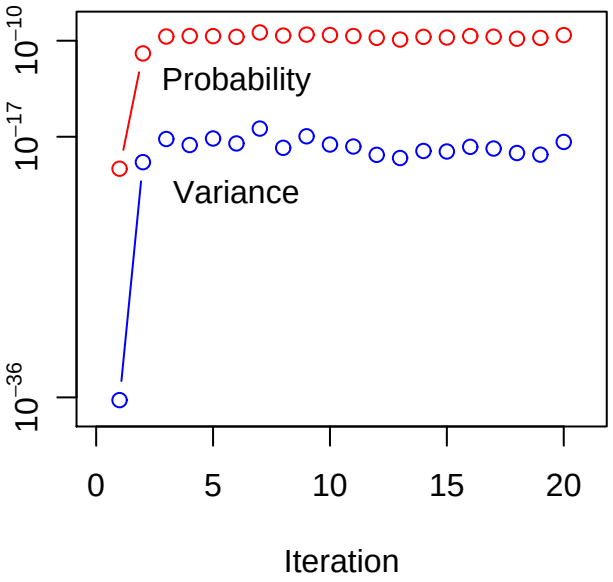


Figure 4.9: Convergence of probability and sample variance for $\Diamond^{3000}\, D \geq 470$ in the chemical model using $N = 1000$.

4.5.2 Repair model

The need to certify system reliability often motivates the use of formal methods and thus reliability models are studied extensively in the literature. The following example is taken from [112] and features a moderately large state space of 40,320 states that can be investigated using numerical methods to corroborate our results.

A repair model with balanced rates The system is modelled as a continuous time Markov chain and comprises six types of subsystems $(1, \ldots, 6)$ containing, respectively, $(5, 4, 6, 3, 7, 5)$ components that may fail independently. The system's evolution begins with no failures and with various probabilistic rates the components fail and are repaired. The failure rates are $(2.5\epsilon, \epsilon, 5\epsilon, 3\epsilon, \epsilon, 5\epsilon)$, $\epsilon = 0.001$, and the repair rates are $(1.0, 1.5, 1.0, 2.0, 1.0, 1.5)$, respectively. Each subsystem type is modelled by two guarded commands: one for failure and one for repair. We thus define twelve parameters λ_k for the twelve corresponding commands. In the original system, they are all set to 1. The property under investigation is the probability of a complete failure of a subsystem (i.e., the failure of all components of one type), given an initial condition of no failures. This can be expressed in temporal logic as $\mathrm{P}[\mathit{init} \wedge \bigcirc(\neg \mathit{init}\, U^{1000}\, \mathit{failure})]$.

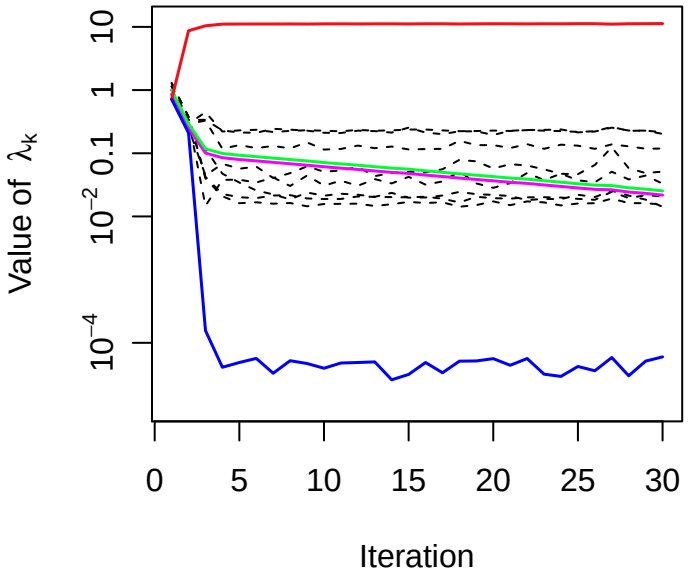


Figure 4.10: Convergence of parameters and effect of smoothing (green and magenta lines) in repair model using $N = 10000$.

Figure 4.10 shows the convergence of parameters and highlights the effects of the adopted smoothing strategy. While most parameters converge to stable values, the parameters denoted by green and magenta lines (corresponding to repair of components of types 5 and 6, respectively) are continually attenuated by the smoothing factor (0.95 in this case). Their commands are not seen in successful traces, sug-

gesting that they are less important than the other parameters with respect to the property. Most of the other parameters lie in the approximate range 0.01 to 0.25, but the parameters denoted by red and blue lines (corresponding to the failure and repair, respectively, of components of type 4) are significantly outside. It is clear that increasing the failure rate of components of type 4 is critical to the property. The fact that repair transitions are generally made less likely by the algorithm agrees with the intuition that we are interested in direct paths to failure. The fact that they are not necessarily made zero reinforces the point that the algorithm seeks to consider *all* paths to failure, including those that have intermediate repairs.

Figure 4.11 plots the number of paths satisfying $\bigcirc(\neg \mathit{init}\, U^{1000}\, \mathit{failure})$ and suggests that for this model the parametrised distribution is close to the optimum. Figure 4.12 plots the estimated probability and sample variance during the course of the algorithm and superimposes the true probability calculated by PRISM [3]. The long term average agrees well with the true value (an error of -1.7%, based on an average excluding the first two estimates), justifying our use of the sample variance as an indication of the efficacy of the algorithm: our importance sampling parameters provide a variance reduction of more than 10^5.

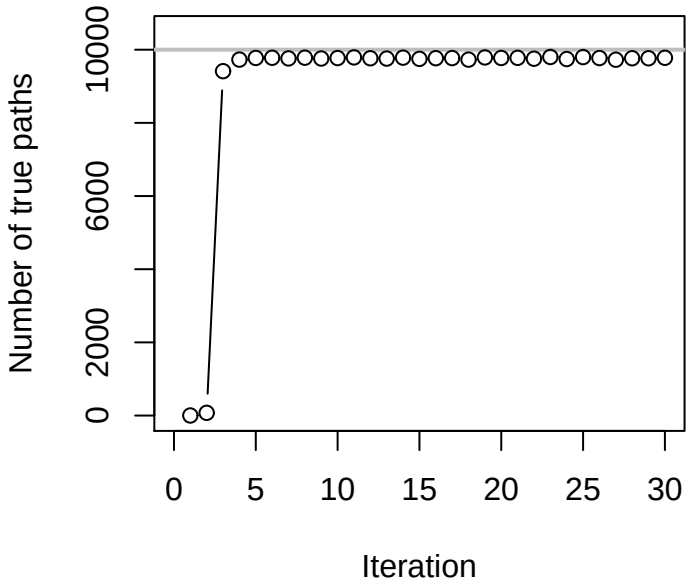


Figure 4.11: Convergence of number of paths satisfying $\bigcirc(\neg \mathit{init}\, U^{1000}\, \mathit{failure})$ in the repair model using $N = 10000$.

[3] http://www.prismmodelchecker.org/

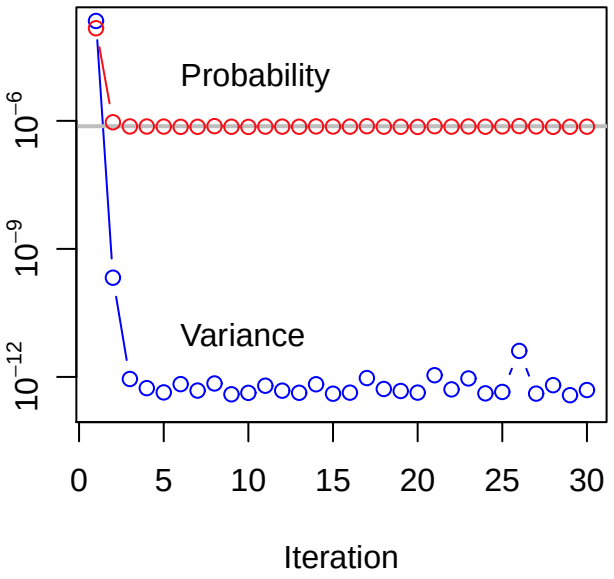


Figure 4.12: Convergence of estimated probability and sample variance for repair model using $N = 10000$. True probability shown as horizontal line.

A repair model with unbalanced rates We present an other repair model taken from [112]. This example features a smaller state space of 125 states but it has two difficulties, group repair and unbalanced failure rates.

The system now comprises three types of subsystems $(1, 2, 3)$, each containing 4 components that may still fail independently. The system's evolution begins with no failures and with various probabilistic rates the components fail and are repaired. Each subsystem type is originally modelled by two guarded commands: one for failure and one for repair. The failure rates are $(\epsilon^2, \epsilon, \epsilon)$, $\epsilon = 0.1$, and the repair rates are all equal to 1.0. Repairs are prioritised according to $1 > 2 > 3$ (type 1 has highest priority, etc). For types 1 and 2 group repair starts after two of that type have failed: all failed components of that type are repaired simultaneously. Type 3 components are repaired one by one as soon as one has failed. Finally, the system breaks down as soon as all components of all types have failed. The probability of satisfying the property is 1.179×10^{-7}.

We break guards into several disjoint guards in order to study the effect on the simulations. The algorithm is applied with three different sets of parameters and 50000 samples in the final importance sampling step. In the first case, there are only 6 commands, one for each repair type and one for each failure type. In the second case, we isolate in one command all the transitions violating ϕ. Then, we isolate group repair of two type-1 components, group repair of three type-1 components, group repair of four type-1 components, group repair of two type-2 components, group repair of three type-2 components, group repair of four type-2 components in separate commands. The model is thus equivalent but described with 11 commands. Finally, a perfect description based on individual non-zero transitions contains at

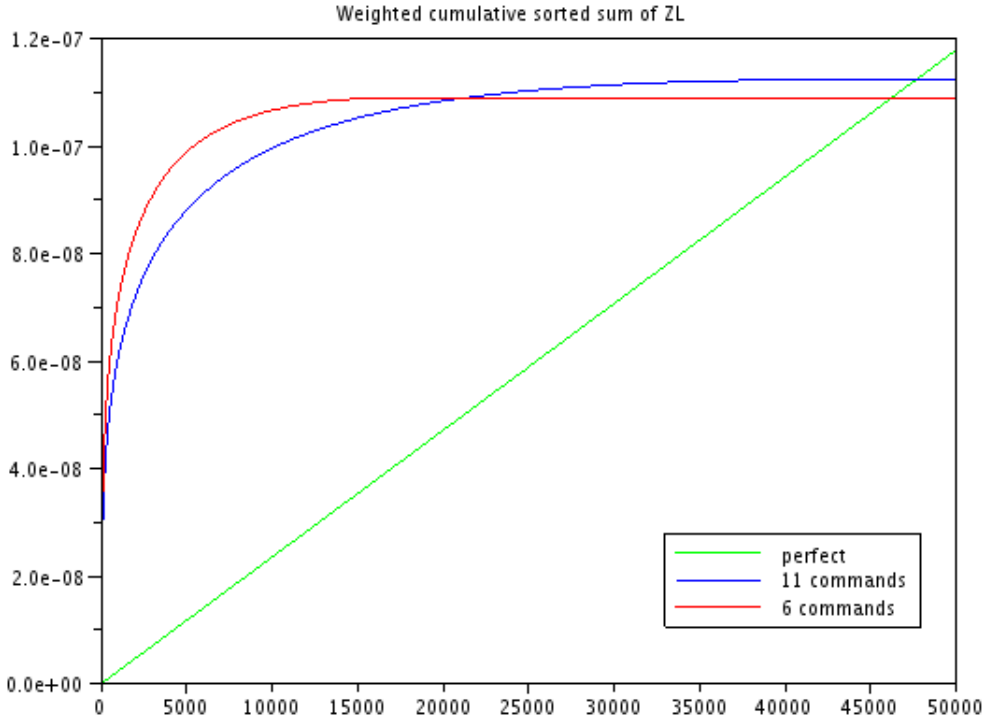


Figure 4.13: Sorted Cumulative normalised Sum of ZL

most 421 commands. As the system is small, we perform Ridder's algorithm in the perfect case and our algorithm with 6 and 11 commands system. We perform simulations and compute for each trace ω_i the product $z(\omega_i)$ with its likelihood ratio $L(\omega_i)$ and plot the sorted ZL normalised by the number of traces.

In Figure 4.13, the three lines represent the cumulative sum of sorted and normalised ZL function, each of them corresponding to a different parametrisation, and tend to their respective estimation. The green line indicates that Ridder's algorithm gives a very accurate result (1.176×10^{-7})and the likelihood ratio is almost constant. With 6 commands (red line), we see that the final estimation is already pretty good but in general slightly underestimates the real value (1.090×10^{-7} in average). Moreover, the likelihood ratio distribution is asymmetric with a large queue of paths with a very low value. The blue line shows that cleverly increasing the number of commands alleviates this effect. The curve is less concave and the final estimate (1.134×10^{-7}) is closer than Ridder's performance.

4.6 Existence of Distributions

Let $(s_i)_{i\geq 0}$ be a discrete-time Markov chain with state space S. In general, properties do not admit a zero-variance estimator at the level of Markov chains with same state space S. In this section, we show however that in a DTMC context, unbounded reachability properties, properties dually equivalent to a unbounded reachability property admit a zero-variance estimator.

In what follows, the chain evolves until a stopping condition SC (reached terminal states or time out) which characterizes the violation or the satisfaction of a temporal property ϕ. Assume that the probability of satisfying a stopping condition in a finite time is equal to 1.

We denote by T the set of transitions between states and by P the transition probability matrix. We write $P(s_{i-1}, s_i)$ to denote the transition probability from state s_{i-1} to s_i. A transition is denoted $\rightarrow$ and a transition between state s_i and s_j is denoted $s_i \rightarrow s_j$.

A trajectory (or a path) ω of the Markov chain is a sequence of transitions and $\tau = |\omega|$ its length (the number of transitions). By definition, $0 < \tau < \infty$. Set of paths is denoted Ω.

Let $\gamma(\omega_i) = Pr(\omega \models \phi)$ the probability that an arbitrary (finite) path satisfies temporal property ϕ, given an initial state $s_i \not\models SC$. Whenever this initial state is the initial state of the system, ω_0, we denote this probability γ.

Let z the function of Ω such that $\forall \omega$, $z(\omega) = \mathbb{1}(\omega \models \phi)$. As traces are stochastic realisations of the system, the behaviour of function z is modelled as a Bernoulli random variable Z. By definition, $\gamma = \mathbb{E}_P[Z]$.

Let Q be a transition probability matrix absolutely continuous with $z(.)P(.)$. The likelihood ratio of path ω is equal to

$$L(\omega) = \prod_{i=1}^{\tau} \frac{P(s_{i-1}, s_i)}{Q(s_{i-1}, s_i)}$$

Under the measure Q,

$$\gamma = \mathbb{E}_Q[ZL]$$

4.6.1 Theorems

Theorem 5 *Given a temporal property ϕ, if there exists a function $c : T \rightarrow \{0, 1\}$ such that $z(\omega) = c(s_{\tau-1} \rightarrow s_\tau)$, there exists an importance sampling estimator of γ with zero-variance in this setting.*

Proof *Let $c : T \rightarrow \{0, 1\}$ such that $c(s_i \rightarrow s_j) = 1$ if $(s_i \rightarrow s_j) \models \phi$ and 0 otherwise.*

As trace ω is checked at each transition, it means that if $z(\omega) = c(s_{\tau-1} \rightarrow s_\tau)$, the property is violated or satisfied in the last transition. The c-value of this last

transition is necessarily independent with respect to time, otherwise function c would not be defined from T to $\{0,1\}$.

So, by construction,

$$Z = \sum_{i=1}^{\tau} c(s_{i-1} \rightarrow s_i) = c(s_{\tau-1} \rightarrow s_\tau) \tag{4.32}$$

Note that Z only depends on the value of the last transition $c(s_{\tau-1} \rightarrow s_\tau)$ as the simulations stop as soon as a stopping criteria is satisfied.

Furthermore, c is equal to 0 all along the path and is equal to 0 or 1 at the last step.

We rewrite random variable ZL as follows:

$$ZL = c(s_{\tau-1} \rightarrow s_\tau) \prod_{i=1}^{\tau} \frac{P(s_{i-1}, s_i)}{Q(s_{i-1}, s_i)} \tag{4.33}$$

Consider $Q(s_{i-1}, s_i)$ proportional to $P(s_{i-1}, s_i)(c(s_{i-1} \rightarrow s_i) + \gamma(s_i))$

In this case,

$$\begin{aligned} Q(s_{i-1}, s_i) &= \frac{P(s_{i-1}, s_i)(c(s_{i-1} \rightarrow s_i) + \gamma(s_i))}{\sum_{s' \in S} P(s_{i-1}, s')(c(s_{i-1} \rightarrow s') + \gamma(s'))} && (4.34) \\ &= \frac{P(s_{i-1}, s_i)(c(s_{i-1} \rightarrow s_i) + \gamma(s_i))}{\gamma(s_{i-1})} && (4.35) \end{aligned}$$

Then,

$$\begin{aligned} ZL &= c(s_{\tau-1} \rightarrow s_\tau) \prod_{i=1}^{\tau} \frac{P(s_{i-1}, s_i)}{Q(s_{i-1}, s_i)} && (4.36) \\ &= c(s_{\tau-1} \rightarrow s_\tau) \prod_{i=1}^{\tau} \frac{P(s_{i-1}, s_i)\gamma(s_{i-1})}{P(s_{i-1}, s_i)(c(s_{i-1} \rightarrow s_i) + \gamma(s_i))} && (4.37) \\ &= c(s_{\tau-1} \rightarrow s_\tau) \frac{\gamma(s_0)}{c(s_{\tau-1} \rightarrow s_\tau) + \gamma(s_\tau)} && (4.38) \\ &= \gamma && (4.39) \end{aligned}$$

The last equality comes from the fact that $s_\tau \models SC$ and so $\gamma(s_\tau) = 0$. It follows that ZL is a constant random variable and so has zero variance.

Consequently, some common and recurrent problems have a zero variance importance sampling estimator. We list a few of them next.

Theorem 6 *Consider the following stopping criteria "hit Δ" a set of states strictly included in S such that that the probability of hitting Δ in a finite time is 1.*

Let $A \subset \Delta$, an initial state $s_0 \notin \Delta$ and $\phi = \Diamond (s \in A)$.

There exists an importance sampling estimator of γ with zero variance in this setting.

Proof *Let $c : T \to \{0, 1\}$ such that $c(s_i \to s_j) = 1$ if $s_j \in A$ and 0 otherwise.*

Trace ω is checked at each transition. So, by construction,

$$z(\omega) = \sum_{i=1}^{\tau} c(s_{i-1} \to s_i) = c(s_{\tau-1} \to s_\tau) \tag{4.40}$$

Then, the theorem is a consequence of theorem 5.

Theorem 7 *Consider the following stopping criteria "hit Δ" a set of states strictly included in S such that that the probability of hitting Δ in a finite time is 1.*

Let $A \subset \Delta$, an initial state $s_0 \notin \Delta$ and $\phi = \Box\,(s \in S \setminus A)$.

There exists an importance sampling estimator of γ with zero-variance in this setting.

Proof *Let $d : T \to \{0, 1\}$ such that $d(s_i \to s_j) = 1$ if $s_j \in A$ and 0 otherwise.*

By construction,

$$z(\omega) = 1 - \sum_{i=1}^{\tau} d(s_{i-1} \to s_i) = 1 - d(s_{\tau-1} \to s_\tau) \tag{4.41}$$

Then, apply theorem 5 with the functional equality $c = 1 - d$.

Theorem 8 *Consider the following stopping criteria "hit A" a set of states strictly included in S such that that the probability of hitting A in a finite time is 1.*

Let B a non empty set such that $A \cap B = \emptyset$, an initial state $s_0 \notin A \cup B$ and $\phi = \neg(s \in B)\, U\ s \in A$.

There exists an importance sampling estimator of γ with zero-variance in this setting.

Proof *This theorem is a corollary of theorem 6. Indeed, the property is not really a reachability in the sense that it is not enough to reach A, B must be avoided. However, as any trace reaching B before A is unsuccessful, the problem is similar to theorem 6 by defining $\Delta = A \sqcup B$.*

Whenever the property is time-bounded, the theorems does not hold in general. Indeed, the same transition could provoke with probability 1 a violation in one case and a satisfaction in another case.

For example, consider the simple transition system depicted in Fig. 4.14, with initial state s_0, together with the property $\bigcirc(\Box^4 \neg(s = s_0 \vee s = s_5))$. No paths containing transitions b, f or g satisfy the property, while paths containing transition e always satisfy the property and transitions a, c, and d exist in paths that both satisfy and do not satisfy the property. If we change the time bound of $\Box$ to 3, the nature of transitions a, b, c, d, e and g is unchanged. However, the nature of transition f depends on the time at which transition is taken. For example, path $acdf$ satisfies the property but afg violates it.

Nevertheless, if the transitions non-takable whenever the property is unbounded are still non-takable whenever the property is bounded, theorems 6, 7 and 8 on φ property stay valid.

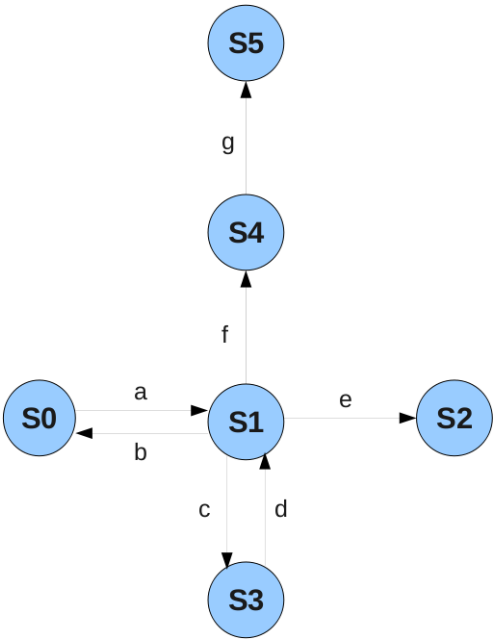


Figure 4.14: A time-bounded counter-example

4.7 Confidence

Bounding the error of estimates produced by importance sampling in the context of statistical model checking remains an open challenge. This is because the importance sampling estimator is based on the likelihood ratio, whose distribution is usually specified implicitly by the transition kernel and property, and has unknown variance.

When using the Monte Carlo estimator (4.2) it is common to bound the error of statistical model checking estimates with the standard confidence interval [116, Section 1.1] or a Hoeffding bound [63]. The confidence interval relies on the fact that the distribution of estimates of a Bernoulli random variable converges rapidly to a normal. With the assumption that the number of samples will always be sufficient for convergence, it is possible to estimate the probability that an N-sample estimate $\hat{\gamma}$ is within ϵ of the true value γ:

$$\mathrm{P}(\parallel \gamma - \hat{\gamma} \parallel \leq \epsilon) \approx 2\Phi(\epsilon\sqrt{(N-1)/\hat{\gamma}(1-\hat{\gamma})}) - 1. \tag{4.42}$$

Function Φ is the cumulative density of a standard normal. Note, in particular, that the accuracy of the approximation relies on $N\hat{\gamma}(1-\hat{\gamma})/(N-1)$ being a good estimate of the true variance of the estimator.

The Hoeffding bound does rely on convergence to normality, requiring only the minimum and maximum possible values of the estimator (denoted a, b, respectively) to relate the number of samples to the probability that the estimate will lie within ϵ of the true value:

$$\mathrm{P}(\parallel \gamma - \hat{\gamma} \parallel \geq \epsilon) \leq 2e^{-2N\epsilon^2/(b-a)^2}. \tag{4.43}$$

In the case of (4.2), $a = 0$ and $b = 1$ and (4.43) reduces to the standard Okamoto bound [101].

With standard Monte Carlo, too few simulations in a particular experiment will result in no traces that satisfy the property (an underestimate), but if enough experiments are performed, a satisfying trace will eventually be observed and the average over many experiments will converge to the true value. Since the distribution is known (Bernoulli), the number of samples needed for convergence can be correctly predicted. In the case of importance sampling with a sub-optimal distribution, too few samples will also result in an underestimate, however the rate of convergence is unknown. Pathological distributions, such as illustrated in Fig. 4.3a, may give an entirely false sense of confidence.

With finite traces the range of values that the likelihood ratio distribution may take is finitely bounded, implying that its variance is also finitely bounded and that the distribution of estimates will converge to normality in the limit of samples (by the central limit theorem). Some authors have thus inferred that a confidence interval may be applied, but there are fundamental problems. With only the guarantee of finite variance, it cannot be assumed that the number of samples will always be sufficient for convergence to normality. Moreover, as demonstrated by the first point in Fig. 4.9, a poor importance sampling distribution may underestimate by tens of orders of magnitude, leading to a poor approximation of true variance and grossly overestimated confidence.

The Hoeffding bound may be correctly applied to importance sampling estimates using the minimum and maximum possible values of the likelihood ratio, i.e., not just the range of values observed during simulation. In practice, however, these values are not known and must be conservatively estimated to ensure correctness (e.g., by assuming worst case likelihood ratio on every transition). In all but exceptional cases, such estimates do not provide bounds that require significantly fewer simulations than standard Monte Carlo. The problems outlined above assume no a priori knowledge about the importance sampling distribution. In the case of distributions produced by the techniques presented in Section (4.4), it is possible to use the assumption that the distributions have minimum cross-entropy with respect to the optimal distribution (f^*) to infer an informal level of confidence. However, for an arbitrary parametrisation, the minimum cross-entropy and the minimum variance distributions are not necessarily the same [65]. They coincide in the case that the optimal distribution (f^*) is a member of the family of distributions.

4.8 Conclusions

Statistical model checking addresses the state space explosion associated with numerical model checking by estimating the parameters of an empirical distribution of executions of a system. By constructing an executable model, rather than an explicit representation of the state space, statistical model checking is able to quantify and

verify the performance of systems that are intractable to an exhaustive approach. Statistical model checking trades certainty for tractability and is often the only feasible means to certify real-world systems. Rare properties pose a particular problem to Monte Carlo simulation methods because the properties are difficult to observe and the error in their estimated probabilities is difficult to bound. Importance sampling is a well-established means to reduce the variance of rare events but requires the construction of a suitable importance sampling distribution without resorting to the exploration of the entire state space.

We have devised a natural parametrisation for importance sampling and have provided a simple algorithm, based on cross-entropy minimisation, to optimise the parameters for use in statistical model checking. We have shown that our parametrisation leads to a unique optimum and have demonstrated that with very few parameters our algorithm can make significant improvements in the efficiency of statistical model checking. We have shown that our approach is applicable to standard reliability models and to the kind of huge state space models found in systems biology. We therefore anticipate that our methodology has the potential to be applied to many complex natural and man-made systems.

An ongoing challenge is to find ways to accurately bound the error of results obtained by importance sampling. Specifically, the sample variance of the results may be a very poor indicator of the true variance (i.e. with respect to the unknown true probability). Recent work has addressed this problem using Markov chain coupling applied to a restricted class of models and logic [5], but a simple universal solution remains elusive. A related challenge is to find precise means to judge the quality of the importance sampling distributions we create. Our algorithm finds an optimum based on an automatic parametrisation of a model described in terms of guarded commands. Linking the importance sampling parametrisation to the description of the model in this way gives our approach an advantage when the rare property is related to semantic features expressed in the syntax. Potentially confounding this advantage is the fact that the syntactic description is likely optimised for compactness or convenience, rather than consideration of importance sampling. As a result, there may be alternative ways of describing the same model that produce better importance sampling distributions. Applying existing work on the robustness of estimators, we hope to adapt our algorithm to provide hints about improved parametrisation.

Chapter 5

Importance splitting for rare properties

5.1 Motivation

It remains an open problem with importance sampling to quantify the performance of apparently 'good' distributions. A further challenge arises from properties and systems that require long simulations. In general, as the length of a path increases, its probability diminishes exponentially, leading to very subtle differences between the original measure f and the alternative measure f' and consequent problems of numerical precision.

A different approach for dealing with rare properties in statistical model checking is to reason on the property of interest instead of reasoning on the model of the system. Importance splitting achieves this by estimating a sequence of conditional probabilities, whose product is the required result. To apply this idea to statistical model checking it is necessary to define a *score function* based on logical properties, and a set of *levels* that delimit the conditional probabilities.

In this chapter, we motivate the use of importance splitting for statistical model checking and describe the necessary and desirable properties of score functions and levels. To our knowledge, this is the first attempt to use this technique in the context of SMC.

Importance splitting procedure The earliest application of importance splitting is perhaps that of [76], where it was used to calculate the probability that neutrons would pass through certain shielding materials. This physical example provides a convenient analogy for the more general case. The system comprises a source of neutrons aimed at one side of a shield of thickness T. It is assumed that neutrons are absorbed by random interactions with the atoms of the shield, but with some small probability γ it is possible for a neutron to pass through the shield. The distance travelled in the shield can then be used to define a set of increasing levels $l_0 = 0 < l_1 < l_2 < \cdots < l_n = T$ that may be reached by the paths of neutrons, with

the property that reaching a given level implies having reached all the lower levels. Though the overall probability of passing through the shield is small, the probability of passing from one level to another can be made arbitrarily close to 1 by reducing the distance between the levels.

These concepts can be generalised to simulation models of arbitrary systems, where a path is a simulation trace. By denoting the abstract level of a path as l, the probability of reaching level l_i can be expressed as $Pr(l \geq l_i) = Pr(l \geq l_i \mid l \geq l_{i-1})Pr(l \geq l_{i-1})$. Defining $\gamma = Pr(l \geq l_n)$ and observing $Pr(l \geq l_0) = 1$, it is possible to write

$$\gamma = \prod_{i=1}^{n} Pr(l \geq l_i \mid l \geq l_{i-1}) \tag{5.1}$$

Each term of the product (5.1) is necessarily greater than or equal to γ. The technique of importance splitting thus uses (5.1) to decompose the simulation of a rare event into a series of simulations of conditional events that are less rare. There have been many different implementations of this idea, but a generalised procedure is as follows.

Assuming a set of increasing levels is defined as above, a number of simulations are generated, starting from a distribution of initial states that correspond to reaching the current level. The procedure starts by estimating $Pr(l \geq l_1 \mid l \geq l_0)$, where the distribution of initial states for l_0 is usually given (often a single state). Simulations are stopped as soon as they reach the next level; the final states becoming the empirical distribution of initial states for the next level. Simulations that do not reach the next level (or reach some other stopping criterion) are discarded. In general, $Pr(l \geq l_i \mid l \geq l_{i-1})$ is estimated by the number of simulation traces that reach l_i, divided by the total number of traces started from l_{i-1}. Simulations that reached the next level are continued from where they stopped. To avoid a progressive reduction of the number of simulations, the generated distribution of initial states is sampled to provide additional initial states for new simulations, thus replacing those that were discarded.

Importance splitting or variants of importance splitting have been applied in many domains since [76], notably in physical systems [38], in telecommunications [128, 129], in watermaking [28], etc. In physical and chemical systems, distances and quantities may provide a natural notion of level that can be finely divided. In the context of model-checking arbitrary systems, variables may be Boolean and temporal properties may not contain an obvious notion of level. To apply importance splitting to statistical model checking it is necessary to define a set of levels based on a sequence of temporal properties, ϕ_i, that have the logical characteristic

$$\phi = \phi_n \Rightarrow \phi_{n-1} \Rightarrow \cdots \Rightarrow \phi_0$$

Each ϕ_i is a strict restriction of the property ϕ_{i-1}, formed by the conjunction of ϕ_i with property ψ_i, such that $\phi_i = \phi_{i-1} \wedge \psi_i$, with $\phi_0 \equiv \top$. Hence, ϕ_i can be written $\phi_i = \bigwedge_{j=1}^{i} \psi_j$. This induces a strictly nested sequence of sets of paths $\Omega_i \subseteq \Omega$:

$$\Omega_n \subset \Omega_{n-1} \subset \cdots \subset \Omega_0$$

where $\Omega_i = \{\omega \in \Omega : \omega \models \phi_i\}$, $\Omega_0 \equiv \Omega$ and $\forall \omega \in \Omega, \omega \models \phi_0$. Thus, for arbitrary $\omega \in \Omega$,

$$\gamma = \prod_{i=1}^{n} Pr(\omega \models \phi_i \mid \omega \models \phi_{i-1}),$$

that is analogous to (5.1).

A statistical model checker works by constructing an automaton to accept traces that satisfy the specified property. In the context of statistical model checking, importance splitting requires that the state of this automaton be included in the final state of a trace that reaches a given level. In practice, this means storing the values of the counters of the loops that implement the time bounded temporal operators.

Structure of the chapter The choice of levels is crucial to the effectiveness of importance splitting. For the purposes of statistical model checking, it is necessary to link levels to temporal logic. Sections 5.2 describes various ways a logical formula may be decomposed into subformulae that may be used to form a nested sequence of levels. To minimise the relative variance of the final estimate it is desirable to choose levels that make $Pr(\omega \models \phi_i \mid \omega \models \phi_{i-1})$ the same for all i (see, e.g., [38]). A simple decomposition of a property may give levels with widely divergent conditional probabilities, hence Section 5.3 introduces the concept of a *score function* and techniques that may be used to increase the possible resolution of levels. We illustrate how a score function may be derived from a property. Given sufficient resolution, a further challenge is to define the levels. In practice, these are often guessed or found by trial and error but we give two importance splitting algorithms in Section 5.4: one that uses fixed levels and one that discovers optimal levels adaptively for a given fixed conditional probability. As score functions can not be grained enough, we present in Section 5.5 an heuristic and an optimal adaptive algorithm. Illustrative examples of fixed and adaptive algorithms are provided in Section 5.6. We discuss the improvement on the optimised algorithm performance compared with previous algorithms in Section 5.7. Finally, we present in Section 5.8 an experimental application of importance splitting combined with state estimation in a hidden Markov model.

5.2 Decomposition of a temporal logic formula

Many existing uses of importance splitting employ a natural notion of levels inherent in a specific problem. Systems that do not have an inherent notion of level may be given quasi-natural levels by 'lumping' states of the model into necessarily consecu-

tive states of an abstracted model. This technique is used in the dining philosophers example in Section 5.6.

For the purposes of statistical model checking, it is necessary to link levels to temporal logic. The following subsections describe various ways a logical formula may be decomposed into subformulae that may be used to form a level-based score function. The techniques may be used independently or combined with each other to give the score function greater resolution. Hence, the term 'property' used below refers both to the overall formula and its subformulae.

Since importance splitting depends on successively reaching levels, the initial estimation problem tends to become one of reachability (as in the case of numerical model checking algorithms). We observe from the following subsections that this does not necessarily limit the range of properties that may be considered.

5.2.1 Simple and natural decomposition

Simple decomposition When a property ϕ is given as an explicit conjunction of n sub-formulae, i.e.:

$$\phi = \bigwedge_{j=1}^{n} \psi_j \tag{5.2}$$

a simple decomposition into nested properties is obtained by:

$$\forall i \in \{1, \ldots, n\}, \quad \phi_i = \bigwedge_{j=1}^{i} \psi_j \tag{5.3}$$

with $\phi_0 \equiv \top$. The associativity and commutativity of conjunction make it possible to choose an arbitrary order of sub-formulae, with the possibility to choose an order that creates levels with equal conditional probabilities. Properties that are not given as conjunctions may be re-written using DeMorgan's laws in the usual way.

Natural decomposition Many rare events are defined with a natural notion of level, i.e., when some quantity in the system reaches a particular value. In physical systems such a quantity might be a distance, a temperature or a number of molecules. In computational systems, the quantity might refer to a loop counter, a number of software objects, or the number of available servers, etc.

Natural levels are thus defined by nested atomic properties of the form:

$$\forall i \in \{0, \ldots, n\}, \quad \phi_i = (l \geq l_i) \tag{5.4}$$

where l is a state variable, $l_0 = 0 < l_1 < \cdots < l_n$ and $\omega \models \phi_n \iff (l \geq l_n)$.

When rarity increases with decreasing natural level, the nested properties have the form:

$$\forall i \in \{0, \ldots, n\}, \quad \phi_i = (l \leq l_i) \tag{5.5}$$

with $l_0 = \max(l) \geq l_1 \geq \cdots \geq l_n$, such that $\omega \models \phi_n \iff l \leq l_n$.

Time may be considered as a natural level if it also happens to be described by a state variable, however in the following subsection it is considered in terms of the bound of a temporal operator.

5.2.2 Decomposition of temporal operators

Since we are using temporal properties, it is necessary to ensure that decompositions are well-defined over properties containing a temporal operator. The following Propositions hold:

1. $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\Diamond^{\leq t}\phi_n \Rightarrow \Diamond^{\leq t}\phi_{n-1})$
2. $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\Box^{\leq t}\phi_n \Rightarrow \Box^{\leq t}\phi_{n-1})$
3. $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\bigcirc\phi_n \Rightarrow \bigcirc\phi_{n-1})$
4. $(\phi_n \Rightarrow \phi_{n-1} \wedge \psi_m \Rightarrow \psi_{m-1}) \Longrightarrow (\phi_n \, U \, \psi_m \Rightarrow \phi_{n-1} \, U \, \psi_{m-1})$
5. $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t}\Box^{\leq s}\phi_{n-1})$

Proof *We denote by $\omega(t)$ the state of execution ω at time t.*

1. $\forall\omega \models \Diamond^{\leq t}\phi_n : \exists t' \leq t \mid \omega(t') \models \phi_n$ *(by definition)*
   $\omega(t') \models \phi_{n-1}$ *(by hypothesis)*
   $\omega \models \Diamond^{t'}\phi_{n-1}$ *hence* $\omega \models \Diamond^{\leq t}\phi_{n-1}$ *and* $\Diamond^{\leq t}\phi_n \Rightarrow \Diamond^{\leq t}\phi_{n-1}$
2. $\forall\omega \models \Box^{\leq t}\phi_n : \forall t' \leq t \mid \omega(t') \models \phi_n$ *(by definition)*
   $\forall t' \leq t : \omega(t') \models \phi_{n-1}$ *(by hypothesis)*
   hence $\omega \models \Box^{\leq t}\phi_{n-1}$ *and* $\Box^{\leq t}\phi_n \Rightarrow \Box^{\leq t}\phi_{n-1}$
3. $\forall\omega(t) \models \bigcirc\phi_n : \omega(t+1) \models \phi_n$ *(by definition)*
   $\omega(t+1) \models \phi_{n-1}$ *(by hypothesis)*
   $\omega(t) \models \bigcirc\phi_{n-1}$ *hence* $\bigcirc\phi_n \Rightarrow \bigcirc\phi_{n-1}$
4. $\forall\omega \models \phi_n \, U \, \psi_m : \exists t, \forall t' \leq t \mid \omega(t') \models \phi_n \wedge \omega(t) \models \psi_m$ *(by definition)*
   $\forall t' \leq t \mid \omega(t') \models \phi_{n-1} \wedge \omega(t) \models \psi_{m-1}$ *(by hypothesis)*
   $\exists t, \forall t' \leq t \mid \omega(t') \models \phi_{n-1} \wedge \omega(t) \models \psi_{m-1}$ *hence* $\phi_n \, U \, \psi_m \Rightarrow \phi_{n-1} \, U \, \psi_{m-1}$
5. $\Box^{\leq s}\phi_n \Rightarrow \Box^{\leq s}\phi_{n-1}$ *(by applying proposition 2 with hypothesis* $\phi_n \Rightarrow \phi_{n-1}$*)*
   $\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t}\Box^{\leq s}\phi_{n-1})$ *(by applying proposition 1 with* $\Box^{\leq s}\phi_n \Rightarrow \Box^{\leq s}\phi_{n-1}$*)*
   hence $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t}\Box^{\leq s}\phi_{n-1})$

Example 5 *In the repair model example (addressed in Section 5.6), given* ***init*** *an initial state and* ***failure*** *an error state characterizing the failure of all components of the system, we consider the property:*

$$\phi = \boldsymbol{init} \wedge \bigcirc(\neg \boldsymbol{init} \, U^{\leq t} \, \boldsymbol{failure}) \tag{5.6}$$

with t infinite.

*The property $\phi = \textbf{\textit{init}} \wedge \bigcirc(\neg \textbf{\textit{init}}\, U^{\leq t}\, \textbf{\textit{failure}})$ has the form of a conjunction, but a simple decomposition is trivial. Using Proposition 3 we can decompose $\bigcirc$ and using Proposition 4 we can decompose U. **failure** can be decomposed in terms of natural levels of failed components. We combine these and consider nested properties based on the total number of failed components **totalfail** (at maximum n).*

We thus define levels $\tau_0 = 0$, $\tau_1 = 2, \ldots$, $\tau_i = i + 1, \ldots$, $\tau_n = n$ and construct nested properties of the form:

$$\phi_i = \textbf{\textit{init}} \wedge \bigcirc(\neg \textbf{\textit{init}}\, U^{\leq t}\, \textbf{\textit{totalfail}} \geq \tau_i). \tag{5.7}$$

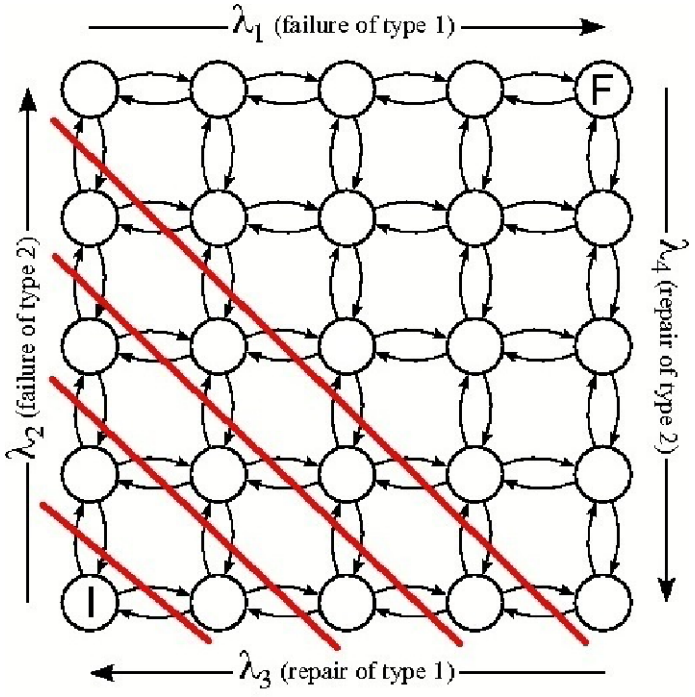


Figure 5.1: A typical repair model parametrised by failure and repair rates.

*In figure 5.1, **I** and **F** are respectively the initial state (all the components are operational) and the failure state (every component is failed). Each red line corresponds to a threshold **totalfail** $= k$.*

5.2.3 Time decomposition

The following propositions hold and can be used for decomposing the time bound of temporal operators:

a) $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\forall \omega \models \Box^{\leq t} \phi_n : \exists t' \geq t \mid \omega \models \Box^{\leq t'} \phi_{n-1})$

b) $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\forall \omega \models \Diamond^{\leq t} \phi_n : \exists t' \leq t \mid \omega \models \Diamond^{\leq t'} \phi_{n-1})$

c) $(\phi_n \Rightarrow \phi_{n-1}) \Longrightarrow (\forall \omega \models \Diamond^{\leq t}\Box^{\leq s}\phi_n : \exists t' \leq t \wedge s' \geq s \mid \omega \models \Diamond^{\leq t'}\Box^{\leq s'}\phi_{n-1})$

d) $(t' \geq t) \Longrightarrow (\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t'}\Box^{\leq s}\phi_n)$

e) $(s' \leq s) \Longrightarrow (\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t}\Box^{\leq s'}\phi_n)$

f) $(t' \geq t \wedge s' \leq s) \Longrightarrow (\Diamond^{\leq t}\Box^{\leq s}\phi_n \Rightarrow \Diamond^{\leq t'}\Box^{\leq s'}\phi_n)$

Proof *a)* $\forall \omega \models \Box^{\leq t}\phi_n, \omega \models \Box^{\leq t}\phi_{n-1}$ *(by proposition 2)*
So $\#\{t' \geq t \mid \omega \models \Box^{t'}\phi_{n-1}\} \geq 1$

b) $\forall \omega \models \Diamond^{\leq t}\phi_n, \omega \models \Diamond^{\leq t}\phi_{n-1}$ *(by proposition 1)*
So $\#\{t' \leq t \mid \omega \models \Diamond^{t'}\phi_{n-1}\} \geq 1$

c) Let $\omega \models \Diamond^{\leq t}\Box^{\leq s}\phi_n$. *Then,* $\exists t' \leq t : \omega(t') \models \Box^{\leq s}\phi_n$ *(at least* $t' = t$*)*
$\exists s' \geq s \mid \omega(t') \models \Box^{\leq s'}\phi_{n-1})$ *(by proposition a)*
Thus, $\omega \models \Diamond^{\leq t'}\Box^{\leq s'}\phi_{n-1}$

d) results from the definition of $\Diamond$

e) results from the definition of $\Box$

f) results from propositions d) and e)

From Proposition a), properties having the form $\phi = \Box^{\leq t}\psi$ may be decomposed in terms of t. For an arbitrary suffix $\omega_{\geq k} = s_k \overset{t_k}{\rightarrow} s_{k+1} \overset{t_{k+1}}{\rightarrow} s_{k+2} \overset{t_{k+2}}{\rightarrow} \cdots$, we have $(\omega_{\geq k} \models \Box^{\leq t}\psi) \leftrightarrow (\omega_{\geq k} \models \psi) \wedge (\omega_{\geq k+1} \models \psi) \wedge \cdots \wedge (\omega_{k+m} \models \psi)$, for some m such that $\sum_{j=k}^{m+k} t_j \leq t \wedge \sum_{j=k}^{m+k+1} t_j \geq t$. This has the form required for a simple decomposition, giving nested properties of the form $\phi_i = \Box^{\leq l_i}\psi, \forall i \in \{1, \ldots, n\}$, where $l_1 = 0 < l_2 < \cdots < l_n = t$, with $\phi_0 \equiv \top$.

Properties having the form $\phi = \Diamond^{\leq t}\psi$ evaluate to disjunctions in terms of time. From Proposition b), it is plausible to construct nested properties of the form $\phi_i = \Diamond^{t+l_i}\psi, \forall i \in \{1, \ldots, n\}$, with $l_1 \geq l_2 \geq \cdots \geq l_n = 0$ and $\phi_0 \equiv \top$. Some caution is required if t is the value given in the overall property.

Indeed, let $t' \geq t$ and $t'' < t'$. Given a set of traces that satisfy $\Diamond^{\leq t'}\phi$, there is no clear way to generate traces that also satisfy $\Diamond^{\leq t}\phi$ (with $t' \geq t$). An idea could be to rebranch traces that do not satisfy $\Diamond^{\leq t'}\phi$ over $\omega_{\leq t''}$, a prefix of a successful trace. Then, the end of the prefix is generated during time $t' - t''$ and we end up with a new trace of length t'. But there is no guarantee that φ is at least satisfied at time $s < t$ and worse, that φ is at least satisfied at time t'. Such situation would not bring any improvement.

5.3 Score functions

Score functions generalise the concept of levels described in Section 5.1. The goal of a score function S is to discriminate good paths from bad with respect to the property of interest. This is often expressed as a function from paths to $\mathbb{R}$, assigning higher values to paths which more nearly satisfy the overall property. Standard statistical model checking can be seen as a degenerate case of splitting, in the sense that computing $P(\omega \models \phi)$ is equivalent to computing $P(S(\omega) \geq 1)$ with the functional equality $S = z$, where z is the Bernoulli distributed model checking function.

Various ways to decompose a temporal logic property are proposed in the previous section. Given a sequence of nested properties $\phi_0 \Leftarrow \phi_1 \Leftarrow \cdots \Leftarrow \phi_n = \phi$, one may design a function which directly correlates logic to score.

Definition 23 *Let $J_0 \supset J_1 \supset \ldots \supset J_n$ be a set of nested intervals of $\mathbb{R}$ and let $\phi_0 \Leftarrow \phi_1 \Leftarrow \cdots \Leftarrow \phi_n = \phi$ be a set of nested properties. The mapping $S : \Omega \to \mathbb{R}$ is a* level-based *score function of property ϕ if and only if $\forall k : \omega \models \phi_k \iff S(\omega) \in J_k$ and $\forall i, j \in \{0, \ldots, |\omega|\} : i < j \Rightarrow S(\omega_{\leq i}) \leq S(\omega_{\leq j})$*

For example, a simple score function may be defined as follows:

$$S(\omega) = \sum_{k=1}^{n} \mathbb{1}(\omega \models \phi_k)$$

$\mathbb{1}(\cdot)$ is an indicator function taking the value 1 when its argument is true and 0 otherwise. Paths that have a higher score are clearly better because they satisfy more of the overall property.

Note that even a simple score function with few levels (e.g., $n = 2$) could provide an unbiased estimate with a likely smaller number of traces than a standard Monte Carlo estimation.

However in many applications the property of interest may not have a suitable notion of levels to exploit; the logical levels may be too coarse or may distribute the probability unevenly. For example, given the dining philosophers problem presented in section 5.6, we know that from a thinking state, a philosopher must pick one fork and then a second one before eating, but there is no obvious way of creating a finer score function from these logical subproperties and actually, the probability of satisfying a subproperty from a state such that the previous subproperty is satisfied is too low (about 0.06, see the simple score function column of Table 5.4). For these cases it is necessary to design a more general score function which maps a larger sequence of nested set of paths to a set of nested intervals of $\mathbb{R}$.

Definition 24 *Let $J_0 \supset J_1 \supset \ldots \supset J_n$ a set of nested intervals of $\mathbb{R}$ and $\Omega = \Omega_0 \supset \Omega_1 \supset \cdots \supset \Omega_n$ a set of nested subsets of Ω. The mapping $S : \Omega \to \mathbb{R}$ is a* general *score function of property ϕ if and only if $\forall k : \omega \in \Omega_k \iff S(\omega) \in J_k$ and $\omega \models \phi \iff \omega \in \Omega_n$ and $\forall i, j \in \{0, \ldots, |\omega|\} : i < j \Rightarrow S(\omega_{\leq i}) \leq S(\omega_{\leq j})$*

$$\begin{array}{ccccc} \phi_n & \Longrightarrow & \phi_{n-1} & \Longrightarrow & \phi_{n-2} \\ \updownarrow & & \updownarrow & & \updownarrow \\ \Omega_n & \subset \cdots \subset & \Omega_{n-1} & \subset \cdots \subset & \Omega_{n-2} \\ \updownarrow & & \updownarrow & & \updownarrow \\ S(\omega_{\phi_n}) \geq \tau_n & & S(\omega_{\phi_{n-1}}) \geq \tau_{n-1} & & S(\omega_{\phi_{n-2}}) \geq \tau_{n-2} \end{array}$$

Table 5.1: Analogy between property, set of paths and thresholds

Informally, Definition 24 states that a general score function requires that the highest scores be assigned to paths that satisfy the overall property and that the score of a path's prefix is non-decreasing with increasing prefix length.

Denoting an arbitrary path by ω and two path prefixes by ω' and ω'', an ideal score function S satisfies the following property:

$$S(\omega') \geq S(\omega'') \Longleftrightarrow Pr(\omega \models \phi \mid \omega') \geq Pr(\omega \models \phi \mid \omega'') \tag{5.8}$$

Intuitively, (5.8) states that prefix ω' has greater score than prefix ω'' if and only if the probability of satisfying ϕ with paths having prefix ω' is greater than the probability of satisfying ϕ with paths having prefix ω''.

Designing a score function which satisfies (5.8) is generally infeasible because it requires a huge analytical work based on a detailed knowledge of the system and, in particular, of the probability of interest. However, the minimum requirement of a score function is much less demanding. Given a set of nested properties $\phi_1, \ldots \phi_i, \ldots, \phi_n$ satisfying (5.1), the requirement of a score function is that $\omega \models \phi_i \iff S(\omega) \geq \tau_i$, with $\tau_i \geq \tau_{i-1}$ a monotonically increasing set of numbers called thresholds.

Denoting ω_{ϕ_k} paths satisfying ϕ_k, table 5.1 gives an intuition of the difference between a designed level-based score function and a designed general score function. In a level-based score function, a set of thresholds $(\tau_k)_{0 \leq k \leq n}$ can easily be derived from the logical levels. In the general case, we try to characterise intermediate sets of paths and underlying thresholds between sets of paths defined by the logical levels.

When no formal levels are available, an effective score function may still be defined using heuristics that only loosely correlate increasing score with increasing probability of satisfying the property. In particular, a score function based on coarse logical levels may be given increased granularity by using heuristics between the levels. We give an example of heuristic that can be used to decompose more finely time bounded properties in Section 5.5.

5.4 Importance splitting algorithms

We give two importance splitting pseudo-algorithms; one with fixed levels defined a priori and one that finds optimal levels adaptively. N denotes the number of

simulations performed at each level. Levels, denoted τ, are defined as values of score function $S(\omega)$, where ω is a path. τ_k is the k^{th} level and ω_i^k is the i^{th} simulation on level k. $\tilde{\gamma}_k$ is the estimate of γ_k, the k^{th} conditional probability $Pr(S(\omega) \geq \tau_k \mid S(\omega) \geq \tau_{k-1})$.

5.4.1 Fixed level algorithm

The fixed level algorithm follows from the general description given in Section 5.1. Its advantages are that it is simple, it has low computational overhead and the resulting estimate is unbiased. Its disadvantage is that the levels must often be guessed by trial and error – adding to the overall computational cost.

In Algorithm 3, $\tilde{\gamma}$ is an unbiased estimate (see, e.g., [38]). Furthermore, Proposition 3 in [27] states:

Proposition 2 *Let $\tilde{\gamma}$ denote the estimate given by the fixed-levels algorithm, then*

$$\sqrt{N}\frac{\tilde{\gamma}-\gamma}{\gamma} \xrightarrow{\mathcal{L}} \mathcal{N}(0,\sigma^2) \quad \textit{with } \sigma^2 \geq \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k} \tag{5.9}$$

The inequality for relative error σ arises because the independence of initial states diminishes with increasing levels: unsuccessful traces are discarded and new initial states are drawn from successful traces. Several possibilities have been provided to minimise this dependence effect in [27]. In the following, for sake of simplicity, we assume that this goal is achieved.

We then deduce the following $(1-\alpha)$ confidence interval:

$$CI = \left[\tilde{\gamma}\left(\frac{1}{1+\frac{z_\alpha \sigma}{\sqrt{N}}}\right), \tilde{\gamma}\left(\frac{1}{1-\frac{z_\alpha \sigma}{\sqrt{N}}}\right)\right] \qquad \text{with} \qquad \sigma^2 \geq \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k}, \tag{5.10}$$

where z_α is the $1-\frac{\alpha}{2}$ quantile of the standard normal distribution. Hence, with confidence $100(1-\alpha)\%$, $\gamma \in CI$. Note that, even if it is not possible to define γ_k arbitrarily, the confidence interval may nevertheless be reduced by increasing N.

In the confidence interval, σ is estimated by the square root of $\sum_{k=1}^{M} \frac{1-\tilde{\gamma}_k}{\tilde{\gamma}_k}$.
To summarise, at each iteration k,

- At line 5 of algorithm 3, traces are generated and a score is assigned to each of them.
- At line 6, traces are sorted in two sets with respect to an iterated threshold τ_k: the set of discarded traces and the set of successful traces.
- At line 7, we estimate γ_k.
- At line 8-9, the discarded traces are replaced by a copy of a successful traces. All traces have now a score greater than τ_k. The traces can be then resampled from their prefix (line 5).

Algorithm 3: Fixed levels

1 Let $(\tau_k)_{1\leq k\leq M}$ be the sequence of thresholds
2 Let *stop* be a termination condition
3 $\forall j \in \{1,\ldots,N\}$, set $\tilde{\omega}_j^1 = \emptyset$
4 **for** $1 \leq k \leq M$ **do**
5 $\quad \forall j \in \{1,\ldots,N\}$, using prefix $\tilde{\omega}_j^k$, generate path ω_j^k until $(S(\omega_j^k) \geq \tau_k) \vee stop$
6 $\quad I_k = \{\forall j \in \{1,\ldots,N\} : S(\omega_j^k) \geq \tau_k\}$
7 $\quad \tilde{\gamma}_k = \frac{|I_k|}{N}$
8 $\quad \forall j \in I_k$, $\tilde{\omega}_j^{k+1} = \omega_j^k$
9 $\quad \forall j \notin I_k$, let $\tilde{\omega}_j^{k+1}$ be a copy of ω_i^k with $i \in I_k$ chosen uniformly randomly
10 $\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

The same process is repeated M times. If at iteration k, all traces have a score lower than τ_k, the algorithm terminates as the conditional probability and so, γ are estimated by a zero value.

5.4.2 Adaptive level algorithm

Given fixed probability γ and fixed number of levels M, the following constrained optimisation problem:

$$\underset{\gamma_1,\cdots,\gamma_M}{\operatorname{argmin}} \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k} \quad \prod_{k=1}^{M} \gamma_k = \gamma \tag{5.11}$$

is solved for all γ_k equal.

Thus given a fixed number of levels, relative error σ is reduced by finding thresholds all equal, that motivates fine grained score functions.

The cost of finding good levels must be included in the overall computational cost of importance splitting. An alternative to trial and error is to use an adaptive level algorithm that discovers its own optimal levels.

Algorithm 4 is an adaptive level importance splitting algorithm based on [29]. It works by pre-defining a fixed number N_k of simulation traces to retain at each level. With the exception of the last level, the conditional probability of each level is then nominally N_k/N. To summarise,

- At line 4 of algorithm 4, traces are generated until a termination condition is satisfied.
- At line 6-7, scores are computed and sorted. We compute the minimal threshold τ_k such that the probability of having a score greater than this threshold is more than N_k/N.

Algorithm 4: Adaptive levels

Let $\tau_\phi = \min\{S(\omega) \mid \omega \models \phi\}$ be the minimum score of paths that satisfy ϕ
Let N_k be the pre-defined number of paths to keep per iteration
$k = 1$
$\forall j \in \{1, \ldots, N\}$, generate path ω_j^k
repeat
    Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$
    Find minimum $\tau_k \in T$ such that $|\{\tau \in T : \tau \geq \tau_k\}| \geq N_k$
    $\tau_k = \min(\tau_k, \tau_\phi)$
    $I_k = \{j \in \{1, \ldots, N\} : S(\omega_j^k) \geq \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
    for $j \notin I_k$ **do**
        choose uniformly randomly $l \in I_k$
        $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in \mathit{pref}(\omega_l^k) : S(\omega) < \tau_k\right\}$
        generate path ω_j^{k+1} with prefix $\tilde{\omega}_j^{k+1}$
    $M = k$
    $k = k + 1$
until $\tau_k \geq \tau_\phi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

- At line 9, traces are sorted in two sets with respect to τ_k: the set of discarded traces and the set of successful traces.
- At line 10, the conditional probability is ideally equal to N_k/N. In practise, for granularity reasons, $\tilde{\gamma}_k \approx N_k/N$.
- At line 12-14, we replace the discarded traces by the smallest successful-trace prefix having a score greater than the scores of discarded traces. All traces have now a score greater than τ_k.
- At line 15, we resample the $N - |I_k|$ prefix until a termination condition is satisfied and their new score can be computed (line 6).

The same process is repeated until τ_φ is exceeded. If between iteration k and k+1, the threshold is not increased, the algorithm terminates as the conditional probability and so, γ are estimated by a zero value.

Use of the adaptive algorithm may lead to gains in efficiency (no trial and error, reduced overall variance), however the final estimate may have a bias of order $\frac{1}{N}$, i.e., $\mathrm{E}(\tilde{\gamma}) = \gamma(1 + \mathcal{O}(N^{-1}))$. The overestimation (potentially not a problem when estimating rare critical failures) is negligible with respect to σ, such that the confidence interval remains that of the fixed level algorithm. Furthermore, under some regularity conditions, the bias can be asymptotically corrected. The estimate of γ has the form $r_0{\gamma_0}^{n_0}$, with γ_0 a constant value of conditional probabilities, $n_0 = M - 1$, $r_0 = \gamma{\gamma_0}^{-n_0}$ a value in $]\gamma_0, 1]$ and $\frac{E[\tilde{\gamma}]-\gamma}{\gamma} \sim \frac{n_0}{N}\frac{1-\gamma_0}{\gamma_0}$ when N goes to infinity. Using the expansion

$$\tilde{\gamma} = \gamma\left(1 + \frac{1}{\sqrt{N}}\sqrt{n_0\frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0}}Z + \frac{1}{N}n_0\frac{1-\gamma_0}{\gamma_0} + o\left(\frac{1}{N}\right)\right), \tag{5.12}$$

with Z a standard normal variable, $\tilde{\gamma}$ is corrected by dividing it by $1 + \frac{n_0(1-\gamma_0)}{N\gamma_0}$. See [27] for more details.

5.5 Efficient heuristic for an optimised algorithm

We then present a fine grained heuristic score function and an optimal adaptive importance splitting algorithm that improve on the performance of previous algorithms. We perform a set of experiments to illustrate both advantages and drawbacks of the technique.

When no formal levels are available, an effective score function may still be defined using heuristics that only loosely correlate increasing score with increasing probability of satisfying the property. In particular, a score function based on coarse logical levels may be given increased granularity by using heuristics between the levels. For example, a time bounded property, not explicitly correlated to time, may become increasingly less likely to be satisfied as time runs out (i.e., with increasing

path length). A plausible heuristic in this case is to assign higher scores to shorter paths. A similar heuristic has been used for importance sampling, under the assumption that the mass of probability in the optimal change of measure is centred on short, direct paths [109]. In the context of importance splitting, the assumption is that shorter paths that satisfy the sub-property at one level are more likely to satisfy the sub-property at the next level because they have more time to do so. We make use of this heuristic in Section 5.7.

5.5.1 Optimised adaptive level algorithm

Algorithm 5 defines an optimised adaptive level importance splitting algorithm. We already know that for an arbitrary number of levels, choosing all the conditional probabilities equal to some value γ_0 reduces the relative variance of the estimate. But a question remains, how should we choose γ_0? Using the expansion 5.12, the variance of the estimate is:

$$Var(\tilde{\gamma}) = \mathrm{E}\left[(\tilde{\gamma} - \gamma)^2\right] = \gamma^2 \left(a^2 Z^2 + b^2 + c^2 + 2abZ + 2acZ + 2bc\right) \tag{5.13}$$

with $a = \frac{1}{\sqrt{N}} \sqrt{n_0 \frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0}}$, $b = \frac{1}{N} n_0 \frac{1-\gamma_0}{\gamma_0}$ and $c = o\left(\frac{1}{N}\right)$. Then, by linearity of the mean, and since $\mathrm{E}[Z] = 0$ and $\mathrm{E}[Z^2] = 1$, we rewrite the variance of the estimate $\tilde{\gamma}$:

$$Var(\tilde{\gamma}) = \frac{\gamma^2}{N} \left(n_0 \frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0} + o(N^{-1}) \right)$$

The variance is minimal when the expression $\left(n_0 \frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0}\right)$ tends to zero. By construction, n_0 is the integer such that $\gamma = r_0 \gamma_0^{n_0}$ with $r_0 \in]\gamma_0, 1]$ and can be defined as equal to $\left\lfloor \frac{\log \gamma}{\log \gamma_0} \right\rfloor$ with $\lfloor \cdot \rfloor$ denoting the floor case of a real number. Substituting $\left\lfloor \frac{\log \gamma}{\log \gamma_0} \right\rfloor$ for n_0, the function

$$f : \gamma_0 \longmapsto \left\lfloor \frac{\log \gamma}{\log \gamma_0} \right\rfloor \frac{1-\gamma_0}{\gamma_0} \tag{5.14}$$

is decreasing on $]0, 1[$. Moreover, $\frac{1-r_0}{r_0}$ is also decreasing to zero when γ_0 tends to 1.

Increasing γ_0 therefore decreases the variance. Ideally, this value is $\gamma_0 = 1 - \frac{1}{N}$ but it is more realistic to fix this value for each iteration k at

$$\gamma_0 = 1 - \frac{N_k}{N} \tag{5.15}$$

with N_k the number of paths achieving the minimal score.

Another advantage of this optimised version is that, although the number of steps before the algorithm terminates is more important, we only rebranch a few discarded traces (ideally only 1) per iteration.

To summarise, the different parameters in each algorithm are:

Algorithm 5: optimised adaptive levels

Let $\tau_\phi = \min\{S(\omega) \mid \omega \models \phi\}$ be the minimum score of paths that satisfy ϕ
$k = 1$
$\forall j \in \{1, \ldots, N\}$, generate path ω_j^k
repeat
  Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$
  $\tau_k = \min T$
  $\tau_k = \min(\tau_k, \tau_\phi)$
  $I_k = \{j \in \{1, \ldots, N\} : S(\omega_j^k) \geq \tau_k\}$
  $\tilde{\gamma}_k = \frac{|I_k|}{N}$
  $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
  for $j \notin I_k$ **do**
    choose uniformly randomly $l \in I_k$
    $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in pref(\omega_l^k) : S(\omega) < \tau_k\right\}$
    generate path ω_j^{k+1} with prefix $\tilde{\omega}_j^{k+1}$
  $M = k$
  $k = k + 1$
until $\tau_k \geq \tau_\phi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

- In the fixed-levels algorithm,
  (a) The number of paths N to estimate each conditional probability,
  (b) The sequence of thresholds $(\tau_k)_{k \in \mathbb{N}}$ (in general induced by a constant step between each threshold)
- In the adaptive-levels algorithm,
  (a) The number of paths N used to determine the next threshold,
  (b) The conditional probability between each threshold γ_0 or equivalently the number of successful paths N_0 per iteration.
- In the optimised algorithm,
  (a) The number of paths N used to determine the next threshold.

5.5.2 Rebranching optimisation

At the end of iteration k, we end up with an estimate of γ_k and an approximation $\tilde{l}_k$ of the first entrance state distribution into level k. The discarded traces must be rebranched over a successful prefix with respect to distribution $\tilde{l}_k$. In practise, to decrease the variance, we do not pick uniformly an index of a successful path but a

cycle of indexes of successful paths. In doing so we avoid the unlikely but possible rebranching of all the discarded traces from the same state.

Let I_k and J_k be respectively the sets of indexes of successful and discarded prefixes. We denote by respectively $I_k(j)$ and $J_k(j)$ the j-th index of I_k and J_k. Let $\mathfrak{S}_{|I_k|}$ be the set of permutations of $\{1, \cdots, |I_k|\}$ and ι an element of $\mathfrak{S}_{|I_k|}$.

We then build randomly a $|J_k|$-length vector $\tilde{J}_k$ with elements of I_k. We choose uniformly cycle ι of $\mathfrak{S}_{|I_k|}$ and repeat the chosen cycle if $N - |I_k| \geq |I_k|$. The first $|J_k|$ elements are the respective elements of $\tilde{J}_k$. Finally, we assign to discarded prefix $\omega_{J_k(j)}$ the successful prefix $\omega_{\tilde{J}_k(j)} = \omega_{I_k((\iota(j)-1 \text{ modulo } |I_k|)+1)}$.

This circular sampling has the advantage to resample perfectly with respect to distribution $\tilde{l}_k$.

5.5.3 Complexity and efficiency

According to [27], the expected complexity C_{ISp} of algorithm 5 is approximately:

$$O(N_{ISp} \log(N_{ISp}) \log(\gamma^{-1})). \tag{5.16}$$

The complexity C_{MC} of a crude Monte Carlo is $O(N_{MC})$ with N_{MC} the number of paths.

So, the importance splitting estimator has smaller variance but greater computational complexity. To take into account both computational complexity and variance, Hammersley and Handscomb have proposed to define the efficiency of a Monte Carlo process as "inversely proportional to the product of the sampling variance and the amount of labour expended in obtaining this estimate" [54]

Hence, the importance splitting estimator $\tilde{\gamma}_{ISp}$ is computationally more efficient than the crude Monte Carlo estimator $\tilde{\gamma}_{MC}$ if:

$$V(\tilde{\gamma}_{MC}) \times C_{MC} \geq V(\tilde{\gamma}_{ISp}) \times C_{ISp} \tag{5.17}$$

Assuming that there exists a constant k such that one importance splitting path is roughly k times more complex that a Monte Carlo path, we approximate the previous expression by the following inequality:

$$\frac{\gamma(1-\gamma)}{N} N \geq -\frac{\gamma^2 \log(\gamma)}{N}(-kN \log(N) \log(\gamma)) \tag{5.18}$$

If we fix for example $k = 10$ and $N = 1000$, the inequality is approximately satisfied for smaller values than 2×10^{-4}. So, if the probability is not so rare, the crude Monte Carlo should be applied.

5.6 Illustrative examples of fixed and adaptive algorithms

The goal of this section is only to illustrate in a simple way the use of importance splitting with statistical model checking and leaves a deeper analysis for the next section. For this purpose, we re-use and re-explain the examples that have been presented in the previous chapter for reading convenience. However, although a comparison between Importance Sampling and Importance Splitting algorithms is tempting, this is left to future investigation as a lot of parametrisations and optimisations are possible and a superficial analysis would not provide relevant results. Moreover, all simulations were performed using a prototype extension of our statistical model checking platform PLASMA [70], in which the previous algorithms have only been partially implemented. A fair PLASMA comparison is thus not yet possible at the moment. All the PLASMA or PRISM models are available online[1][2].

5.6.1 Biochemical network

The network of chemical reactions given below is typical of biochemical systems and demonstrates the potential of SMC to handle the enormous state spaces of biological models.

We consider a well stirred chemically reacting system comprising five reactants (molecules of type A, B, C, D, E), a dimerisation reaction (5.19) and two decay reactions (5.20,5.21).

$$A + B \xrightarrow{1} C \tag{5.19}$$
$$C \xrightarrow{1} D \tag{5.20}$$
$$D \xrightarrow{1} E \tag{5.21}$$

The semantics of (5.19) is that if a molecule of type A encounters a molecule of type B they will combine to form a molecule of type C after a delay drawn from an exponential distribution with mean 1. The decay reactions have the semantics that a molecule of type C (D) spontaneously decays to a molecule of type D (E) after a delay drawn from an exponential distribution with mean 1. A typical simulation run is illustrated in Figure 5.2. A and B combine rapidly to form C, that peaks before decaying slowly to D. The production of D also peaks, while E rises monotonically.

With an initial vector of molecules $(1000, 1000, 0, 0, 0)$, corresponding to types (A, B, C, D, E), the total number of states is less than 10^9, but beyond the current practical capability of exhaustive probabilistic model checking. It is possible for the number of molecules of D to reach 1000, however $D \geq 400$ is unusual. We thus define a suitably rare property to be $\phi = \Diamond^{\leq t} D \geq 460$, with t initially 3000 steps, chosen to be adequately long. To apply Algorithm 3, we set $N = 1000$ and define a nested sequence of properties $\phi_0 = \top$, $\phi_i = \Diamond^{\leq t} D \geq \tau_i$, with $\tau_1 = 390$, $\tau_2 = 400$, $\tau_3 = 410$, $\tau_4 = 420$, $\tau_5 = 430$, $\tau_6 = 440$, $\tau_7 = 450$ and $\tau_8 = 460$. The score function is

[1] http://people.irisa.fr/Cyrille.Jegourel/models.html
[2] https://project.inria.fr/plasma-lab/documentation/examples/

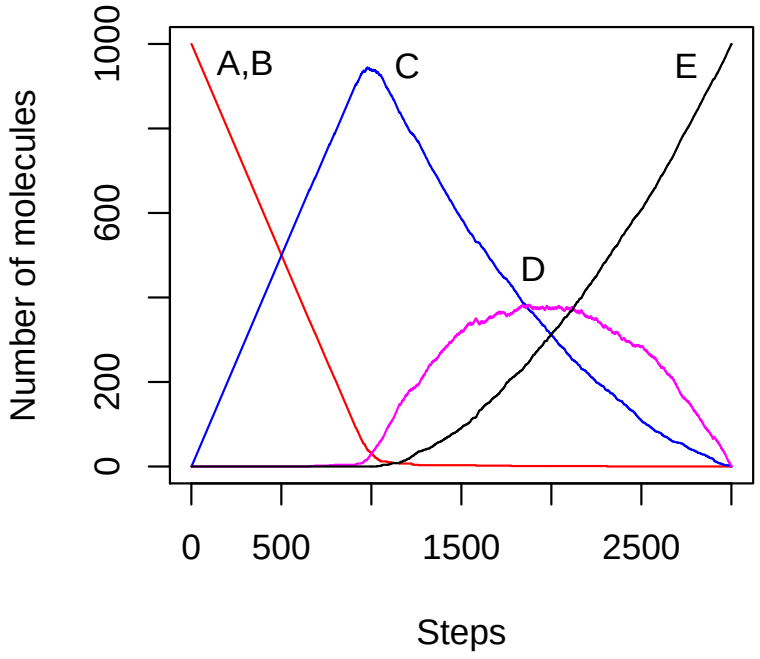


Figure 5.2: A typical stochastic simulation trace of reactions (5.19-5.21).

thus a mapping from paths to τ. τ_1 was found by trial and error, chosen to produce sufficient occurrences of the property on the first level. The other values are equally spaced.

We executed the algorithm 1000 times using the parameters given above. The results are given in Table 5.2. The standard deviation of the estimator, $\sigma_{estimator}$, is estimated in each case using the sample mean. An individual estimate is achieved with 8000 simulation runs; approx. 1.5×10^4 times fewer than the expected number to see a single instance of the rare property.

Algorithm 3 estimates $Pr(D \geq 460) \approx 8.1 \times 10^{-9}$ with 8 levels, implying an optimal (to minimise variance) per-level conditional probability of approx. 0.097. Based on 100 executions, with $N = 1000$ and N_k thus set to 97, Algorithm 4 chose average levels $\hat{\tau}_1 = 396.0$, $\hat{\tau}_2 = 414.5$, $\hat{\tau}_3 = 426.3$, $\hat{\tau}_4 = 434.6$, $\hat{\tau}_5 = 441.8$, $\hat{\tau}_6 = 448.3$, $\hat{\tau}_7 = 454.1$ and $\hat{\tau}_8 = 459.0$. There is apparently some scope with this score function to increase the number of levels and thus increase the confidence of the estimate according to (5.10). This is left to a future investigation.

To compare the estimates of Algorithm 4 and Algorithm 3, we set $N = 1000$ and $N_k = 100$, giving a nominal conditional probability of 0.1 per level. The average levels chosen by Algorithm 4 under these circumstances were $\hat{\tau}_1 = 395.8$, $\hat{\tau}_2 = 414.0$, $\hat{\tau}_3 = 425.4$, $\hat{\tau}_4 = 433.7$, $\hat{\tau}_5 = 440.8$, $\hat{\tau}_6 = 447.3$, $\hat{\tau}_7 = 453.1$ and $\hat{\tau}_8 = 458.2$. These levels have fractionally closer spacing than those with $N_k = 97$, reflecting the marginally increased nominal per-level probability. With 1000 executions, Algorithm 4 estimates $Pr(D \geq 460) \approx 1.4 \times 10^{-8}$, compared to the estimate

Probability	Estimate	$\sigma_{estimator}$
$Pr(D \geq 390)$	0.182	0.012
$Pr(D \geq 400 \mid D \geq 390)$	0.299	0.021
$Pr(D \geq 410 \mid D \geq 400)$	0.201	0.019
$Pr(D \geq 420 \mid D \geq 410)$	0.134	0.017
$Pr(D \geq 430 \mid D \geq 420)$	0.088	0.016
$Pr(D \geq 440 \mid D \geq 430)$	0.057	0.015
$Pr(D \geq 450 \mid D \geq 440)$	0.035	0.012
$Pr(D \geq 460 \mid D \geq 450)$	0.021	0.009
$Pr(D \geq 460)$	8.1×10^{-9}	1.29×10^{-8}

Table 5.2: Chemical network conditional probability estimates based on 1000 runs of Algorithm 3 using $N = 1000$. $\sigma_{estimator}$ is estimated using the sample means.

of 8.1×10^{-9} with Algorithm 3. Given the estimated standard deviation of the fixed level estimator, this empirical difference is ascribed to statistical variance rather than the overestimate predicted by theory.

5.6.2 Repair model

We consider a repair model from the rare event literature (Ex. 1 in [112]), which represents a class of systems that is known to be challenging for parametrised importance sampling; the use of 'group repair' causes them to be 'unbalanced' [112] and renders simple biasing schemes unable to bound the relative error [121].

The model comprises three types of components, with n components per type, that may fail and be repaired at certain probabilistic rates. Each type of component has a different rate of failing and components fail independently. The initial state has no failed components. Repairs are prioritised: components of type 1 are repaired before those of type 2 and type 2 are repaired before type 3. There is a common repair rate, but types 1 and 2 are repaired in groups (all failed components are repaired in one event) while type 3 are repaired singly.

We consider the *total failure entrance probability* (the probability that all components fail, without the system returning to the initial state) expressed as $\gamma = Pr(\omega \models \mathit{init} \wedge \bigcirc(\neg \mathit{init}\, U^{\leq t}\, \mathit{failure}))$, with t infinite. Let fail_1, fail_2 and fail_3 denote the instantaneous number of failed components of types 1, 2 and 3, respectively, then *init* is defined as $\mathit{fail}_1 = 0 \wedge \mathit{fail}_2 = 0 \wedge \mathit{fail}_3 = 0$ and *failure* is defined as $\mathit{fail}_1 = n \wedge \mathit{fail}_2 = n \wedge \mathit{fail}_3 = n$. We set $n = 4$ to create a model with a rare event that is nevertheless tractable to numerical analysis. We thus find that $\gamma = 1.177 \times 10^{-7}$ to four significant figures.

The property $\phi = \mathit{init} \wedge \bigcirc(\neg \mathit{init}\, U^{\leq t}\, \mathit{failure})$ has the form of a conjunction, but a simple decomposition is trivial. Using Proposition 3 we can decompose $\bigcirc$

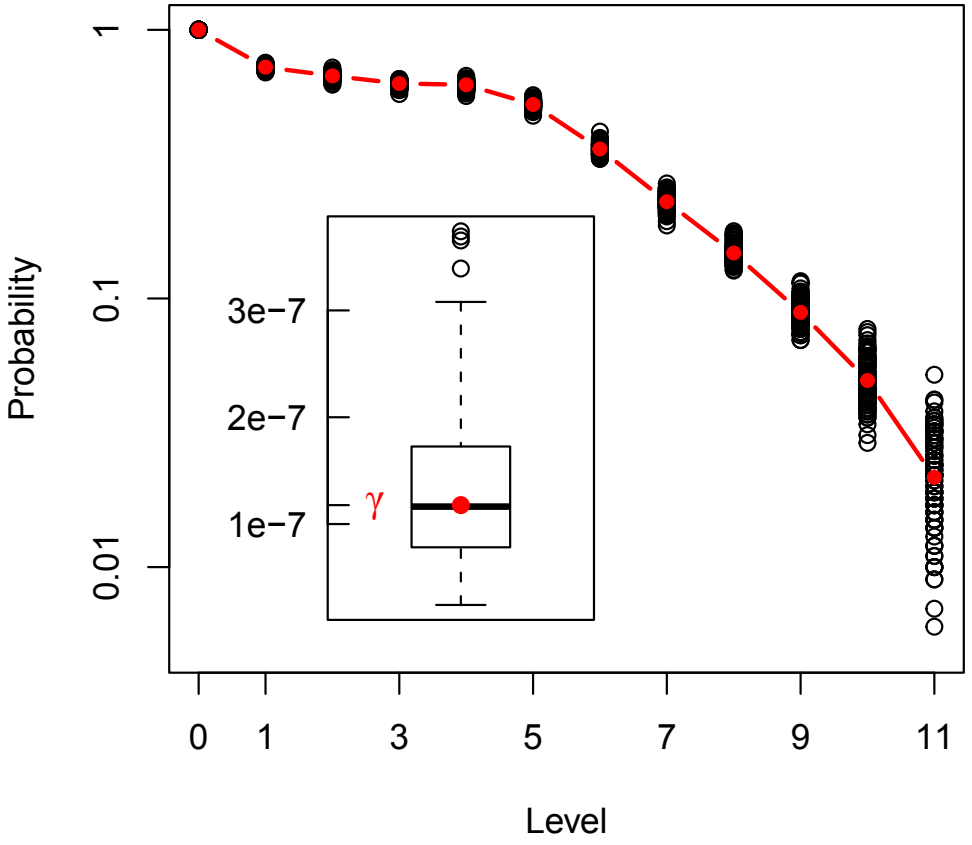


Figure 5.3: Estimated (black) and true (red) conditional probabilities for repair model (line only to guide the eye). Inset, overall estimate (black line) and true value (red dot).

and using Proposition 4 we can decompose U. *init* is a conjunction, but is used negated so can not be usefully decomposed. *failure* can be decomposed as a simple conjunction or in terms of natural levels of failed components. We combine these and consider nested properties based on the total number of failed components $totalfail = fail_1 + fail_2 + fail_3$. The score function is then just a mapping from paths to *totalfail*.

We thus define levels $\tau_0 = 0$, $\tau_1 = 2, \ldots, \tau_i = i + 1, \ldots, \tau_{11} = 12$ and construct nested properties of the form $\phi_i = init \wedge \bigcirc(\neg init\ U^{\leq t}\ totalfail \geq \tau_i)$. We applied Algorithm 3 100 times and achieved the results shown in Table 5.3. Using the numerical model checker PRISM [87] to calculate the true probabilities, we calculate the standard deviations of our estimators ($\sigma_{estimator}$). We conclude that we are able to accurately estimate γ with approx. 800 fewer simulations than would be expected to produce a single example of the rare property.

The results are illustrated in Fig. 5.3, where the inset *box and whisker* plot shows the overall performance of the importance splitting estimator with respect to the true value of γ. The use of a logarithmic scale serves to demonstrate how the

Probability	Estimate	$\sigma_{estimator}$
$\mathrm{P}(\phi_1 \mid \phi_0)$	0.725	0.015
$\mathrm{P}(\phi_2 \mid \phi_1)$	0.673	0.016
$\mathrm{P}(\phi_3 \mid \phi_2)$	0.628	0.015
$\mathrm{P}(\phi_4 \mid \phi_3)$	0.622	0.019
$\mathrm{P}(\phi_5 \mid \phi_4)$	0.529	0.015
$\mathrm{P}(\phi_6 \mid \phi_5)$	0.360	0.017
$\mathrm{P}(\phi_7 \mid \phi_6)$	0.231	0.015
$\mathrm{P}(\phi_8 \mid \phi_7)$	0.149	0.011
$\mathrm{P}(\phi_9 \mid \phi_8)$	0.091	0.010
$\mathrm{P}(\phi_{10} \mid \phi_9)$	0.050	0.010
$\mathrm{P}(\phi_{11} \mid \phi_{10})$	0.023	0.009
$\mathrm{P}(\omega \models \phi_{11})$	1.34×10^{-7}	8.12×10^{-8}

Table 5.3: Estimated conditional and overall probabilities for repair model, based on 100 runs of Algorithm 3 with $N = 1000$. $\sigma_{estimator}$ is calculated w.r.t. the true values.

relative error increases with decreasing estimated probability, motivating the need to find optimal levels. Given the infinite time horizon of the property in this example, we hypothesise that it might be possible to use temporal decomposition to increase the granularity of the score function and thus balance the conditional probabilities of the levels.

5.7 Case study: dining philosophers protocol

We have adapted a case study from the literature to illustrate the use of heuristic-based score functions and of the optimised adaptive splitting algorithm with statistical model checking [72]. We have defined a rare event in the well known probabilistic solution [94] of Dijkstra's dining philosophers problem . In this example, there are no natural counters to exploit, so levels must be constructed by considering 'lumped' states.

A number of philosophers sit at a circular table with an equal number of chopsticks; a chopstick being placed within reach of two adjacent philosophers. Philosophers think and occasionally wish to eat from a communal bowl. To eat, a philosopher must independently pick up two chopsticks: one from the left and one from the right. Having eaten, the philosopher replaces the chopsticks and returns to thinking. A problem of concurrency arises because a philosopher's neighbour(s) may have already taken the chopstick(s). Lehmann and Rabin's solution [94] is to allow the philosophers to make probabilistic choices.

We consider a model of 150 'free' philosophers [94]. The number of states in the

model is more than 10^{177}; 10^{97} times more than the estimated number of protons in the universe. The possible states of an individual philosopher can be abstracted to those shown in Fig. 5.4.

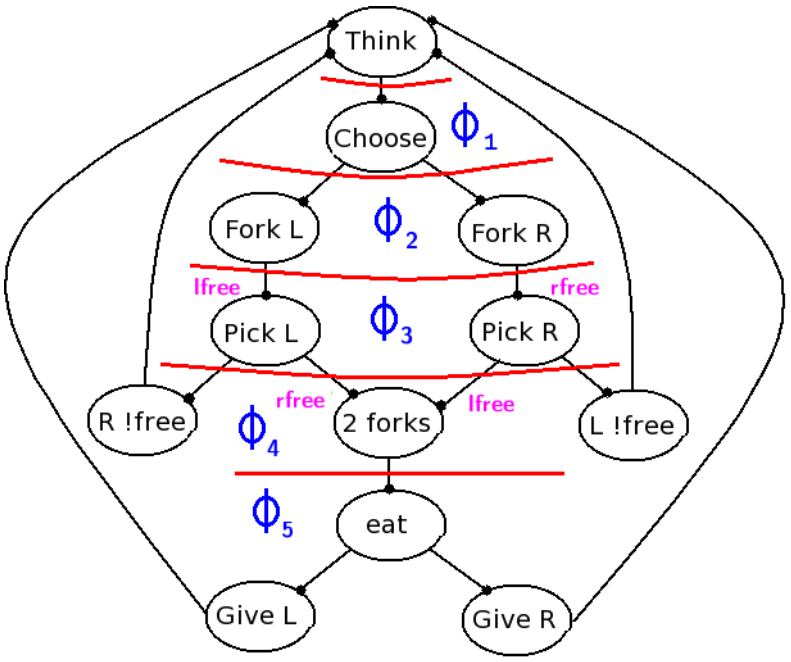


Figure 5.4: Abstract dining philosopher.

Think is the initial state of all philosophers. In state Choose, the philosopher makes a choice of fork he will try to get first. The transitions labelled by lfree or rfree in Fig. 5.4 are dependent on the availability of respectively left or right chopsticks. All transitions are controlled by stochastic rates and made in competition with the transitions of other philosophers. With increasing numbers of philosophers, it is increasingly unlikely that a specific philosopher will be satisfied (i.e., that the philosopher will reach the state eat) within a given number of steps from the initial state. We thus define a rare property $\phi = \Diamond^{\leq t}\mathsf{eat}$, with t initially 30, denoting the property that a given philosopher will reach state eat within 30 steps. Thus, using the states of the abstract model, we decompose ϕ into nested properties $\phi_0 = \Diamond^{\leq t}\mathsf{Think} = \top$, $\phi_1 = \Diamond^{\leq t}\mathsf{Choose}$, $\phi_2 = \Diamond^{\leq t}\mathsf{Try}$, $\phi_3 = \Diamond^{\leq t}1^{\mathsf{st}}\mathsf{stick}$, $\phi_4 = \Diamond^{\leq t}2^{\mathsf{nd}}\mathsf{stick}$, $\phi_5 = \Diamond^{\leq t}\mathsf{eat}$. The red lines crossing the transitions indicate these formal levels on the graph.

Monte Carlo simulations with PLASMA statistical model checker With such a large state space it is not possible to obtain a reference result with numerical model checking. We therefore performed extensive Monte Carlo simulations using the parallel computing capability of the PLASMA statistical model checker [70, 25]. The experiment generated 300 million samples using 255 cores and took about 50

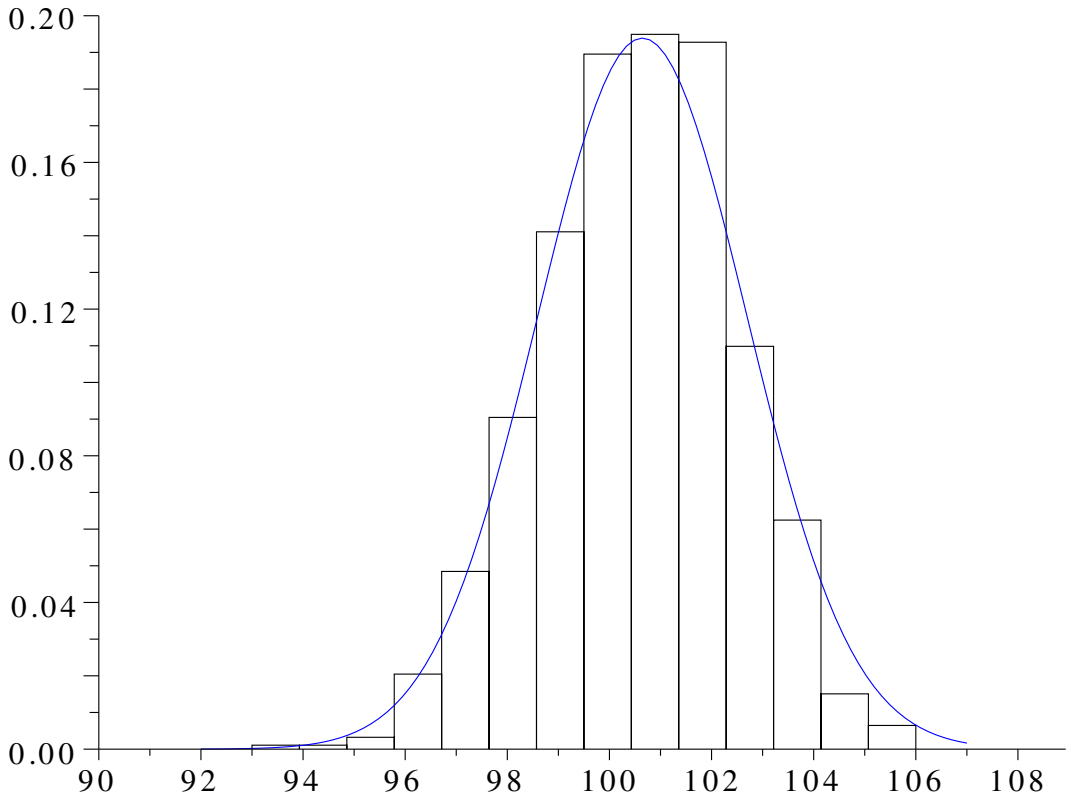


Figure 5.5: Empirical number of levels.

minutes. Our reference probability is thus approximately equal to 1.59×10^{-6} with 95%-confidence interval $[1.44 \times 10^{-6}; 1.72 \times 10^{-6}]$.

At the exception of this previous Monte Carlo experiment, all simulations were performed using SCILAB [3].

5.7.1 Experiment protocol

Four types of importance splitting experiments are driven. The first one uses the simple score function and the fixed algorithm, the second uses the heuristic score function and the fixed-level algorithm (with different step values). The third algorithm uses the adaptive-level algorithm with different γ_0 parameters and finally the fourth set of experiments uses the optimised version of the adaptive algorithm.

For each set of experiments and chosen parameters, experiments are repeated 100 times in order to check the reliability of our results. In what follows, we remind which statistical notions are exploited and why:

- Number of experiments: used to estimate the variance of the estimator.
- Number of path per iteration: it is a parameter of the algorithm, equal to the number of paths used to estimate a conditional probability.

[3] http://www.scilab.org/fr

- Number of levels: known in the fixed algorithm, variable in the adaptive algorithms. In the adaptive case, an average is provided.
- Time in seconds: the average of the 100 experiments is provided.
- The mean estimate is the estimator $\tilde{\gamma}$ of the probability of interest. The average of the 100 estimators is provided.
- The relative standard deviation of $\tilde{\gamma}$ is estimated with the 100 final estimators γ. A reliable estimator must have a low relative standard deviation (roughly ≤ 0.3).
- The mean value of γ_k is the average of the mean values of the conditional probabilities in an experiment. It is variable in the fixed algorithm and supposed to be a constant γ_0 in the adaptive algorithms. Because of the discreteness of the score function, the value is only almost constant and slightly lower than γ_0.
- The relative standard deviation of γ_k is the average of the relative standard deviations of the conditional probabilities in an experiment. By construction, the value in the adaptive algorithms must be low.

5.7.2 Comparison between logical and heuristic score function

Let ω be a path of length $t = 30$. For each prefix $\omega_{\leq j}$ of length j, we define the following function:

$$\Psi(\omega_{\leq j}) = \sum_{k=0}^{n} \mathbb{1}(\omega_{\leq j} \models \phi_k) - \frac{\{\sum_{k=1}^{n} \mathbb{1}(\omega_{\leq j} \models \phi_k)\} - j}{\sum_{k=1}^{n} \mathbb{1}(\omega_{\leq j} \models \phi_k) - (t+1)} \tag{5.22}$$

We define score of ω as follows:

$$S(\omega) = \max_{1 \leq j \leq K} \Psi(\omega_{\leq j}) \tag{5.23}$$

In the following experiment this score function is defined for any path of length $t+1$, starting in the initial state 'all philosophers think'. The second term of Ψ is a number between 0 and 1, linear in j such that the function gives a greater score to paths which satisfy a greater number of sub-properties ϕ_k and discriminates between two paths satisfying the same number of sub-properties by giving a greater score to the shortest path. A score in $]i-1, i]$ implies that a prefix of the path satisfied at most ϕ_i. We then compare results with the simple score function $S(\omega) = \sum_{k=1}^{n} \mathbb{1}(\omega \models \phi_k)$.

The experiments are repeated 100 times in order to demonstrate and improve the reliability of the results. Each conditional probability γ_k is estimated with a sample of 1000 paths.

Statistics	Simple score function	Heuristic score function		
number of experiments	100	100	100	100
number of path per iteration	1000	1000	1000	1000
number of levels	5	20	40	80
Time in seconds (average)	6.95	13.42	16.64	21.56
mean estimate $\times 10^6$ (average)	0.01	0.59	1	1.37
relative standard deviation of $\tilde{\gamma}$	0.77	0.31	0.23	0.19
mean value of $\tilde{\gamma_k}$	0.06	0.53	0.73	0.86
relative standard deviation of $\tilde{\gamma_k}$	1.04	0.36	0.22	0.15

Table 5.4: Comparison between fixed-level algorithms.

For simplicity we consider a linear growing of score thresholds when we use the fixed-level algorithm. The simple score function thresholds increase by 1 between each level. When using the heuristic score function, we performed three sets of experiments involving an increase of 0.2, 0.1 and 0.05 of the thresholds. These partitions imply respectively 5, 20, 40 and 80 levels.

Table 5.4 shows that the simple score function likely gives a strong underestimation. It is due to the huge decrease of value of conditional probabilities between the logical levels. All the estimated conditional probabilities are small and imply a large theoretical relative variance ($V(\tilde{\gamma})/E[\tilde{\gamma}]$). The final levels are difficult to cross and have probabilities close to 0. A sample size of 1000 paths is obviously not enough for the last step. In average $\tilde{\gamma}_5 = 0.003$ and in one case the last step is not satisfied by any trace, such that the estimate is equal to zero.

If a threshold is not often exceeded, it implies that traces will be rebranched from a very small set of first entrance states at the next level. This leads to significant relative variance between experiments. A further problem is that the conditional estimate is less efficient if γ_k is small. Increasing the number of evenly spaced levels decrease *a priori* more smoothly the conditional probabilities and reinforce the reliability of the results as soon as the relative standard deviation of conditional probabilities decreases enough. In the experiments, as expected, the mean value of conditional probabilities is positively correlated to the number of levels (respectively 0.06, 0.53, 0.73 and 0.86) and negatively correlated to the relative standard deviation of conditional probabilities. The results with 40 and 80 levels give results that are apparently close to the reference estimate, but are nevertheless consistently underestimates. This suggests that the number of simulations per level is too low.

Two questions arise: how to detect that the simulation is not efficient or robust and how to improve the results. In answer to the first, there are no general criteria for judging the quality of an importance splitting estimator. However, assuming that experiments are repeated a few times, a large relative error of the estimators (roughly ≥ 0.5), a very low value of conditional probability estimates, or a large relative error

γ_0	0.6	0.75	0.9
number of experiments	100	100	100
number of path per iteration	1000	1000	1000
number of levels (average)	22	34	65
Time in seconds (average)	14.53	16.78	20.05
mean estimate $\times 10^6$ (average)	0.78	1.14	1.58
relative standard deviation of $\tilde{\gamma}$	0.26	0.25	0.23
mean value of $\tilde{\gamma_k}$	0.55	0.68	0.83
relative standard deviation of $\tilde{\gamma_k}$	0.2	0.16	0.12

Table 5.5: Comparison between adaptive algorithms.

of conditional probability estimates (roughly ≥ 0.2) are good warnings. As for the second question, a way to improve results with the fixed level algorithm is simply to increase the number of paths per level or to increase the number of levels, for the reasons given above.

5.7.3 Comparison between fixed and adaptive algorithm

The following section illustrates that adaptive algorithms give significantly more reliable results for slightly increased time. In the following set of experiments we use the adaptive algorithm with three predefined γ_0: 0.6, 0.75 and 0.9. Because of the granularity of the score function, conditional probabilities are not equal at each iteration, but their values are kept under control because their relative standard deviation does not vanish (≤ 0.2). We use 1000 sample paths per level and repeat the experiments 100 times.

As we increased the desired γ_0, the number of levels and time increase. However, the final estimate with $\gamma_0 = 0.9$ matches the Monte Carlo estimator and the relative standard deviation is minimised. In this experiment the number of levels found adaptively is on average 65. Even with mean value of conditional probabilities smaller than in the 80-fixed-level experiment, the results show better convergence, a slightly better speed and lower standard deviation.

5.7.4 Comparison with the optimised adaptive algorithm

This section illustrates a set of experiments using the optimised adaptive algorithm. As previously, we repeated experiments 100 times to check reliability of our results. For each experiment we use a different number of initial paths: 100, 200, 500 and 1000. In order to give an idea of the gain of time, we also executed a Monte Carlo experiment using 10^7 paths. The 95%-confidence intervals are given by (5.10) for the importance splitting experiments and by the standard confidence

Statistics	Importance splitting				MC
nb of experiments	100	100	100	100	1
nb of path per iter.	100	200	500	1000	10 million
Time in sec.	1.73	4.08	11.64	23.77	≥ 5 hours
mean $\times 10^6$ (average)	1.52	1.59	1.58	1.65	1.5
st.dev. $\times 10^6$	1.02	0.87	0.5	0.38	0.39
95%-CI $\times 10^6$	[1.34; 1.74]	[1.48; 1.72]	[1.54; 1.63]	[1.63; 1.67]	[0.74; 2.26]

Table 5.6: Comparison between optimised adaptive algorithms.

interval $\left[\tilde{\gamma} \pm 1.96 \times \sqrt{\frac{\tilde{\gamma}(1-\tilde{\gamma})}{N}}\right]$ for Monte Carlo experiment. As the experiments are repeated several times, we approximate the relative standard deviation σ by the standard deviation of the estimates divided by the average of the estimates, instead of assuming full independence between levels and so taking $\sigma^2 \approx \sum_{k=1}^{m} \frac{1-\gamma_k}{\gamma_k}$. Our approach is more pessimistic and in practise requires the experiment to be repeated a few times. However, even doing so, the results are much more accurate than the Monte Carlo approach. For example, 100 initial paths are used in the first experiment. Roughly speaking, the paths cross on average 100 other levels and only 11% are rebranched each time. So, only 1200 paths are generated and provide in less than 2 seconds an estimate and a confidence interval strictly included in the Monte Carlo confidence interval. This represents a gain greater than 10^4 with respect to the Monte Carlo experiment.

Figure 5.5 illustrates empirically the convergence of the number of levels to a Gaussian with low variance (4.23) with respect to the mean of levels (100.65). Although this fact is only empirical, knowing that the variance is low has some importance whenever the time budget is critical for more extensive experiments.

5.8 An application for systems using state estimation

We present in this section an evaluation methodology combining state estimation and importance splitting for a cyber-physical system (CPS).

5.8.1 Motivation

Applying importance splitting to cyber-physical systems is challenging. In general, the control program of the CPS model is only partially available. Consequently, a finite-model abstraction through static analysis is infeasible. Moreover, the CPS state is generally not known, as the output can only represent a small fraction of the set of state variables. The CPS steering policy towards the rare event is generally not

known, especially if the system model is not available in advance. Last but not least, performing a runtime verification over such program has drawbacks: it alters the timing-related behaviour of the program to check. An overhead is a measure of this alteration. If the original program executes in time T and the monitored program executes in time $T + \eta$, we say that the overhead is η/T. Several techniques have been proposed to reduce the runtime overhead. In [66], the authors introduce an overhead-control technique that selectively turns monitoring on and off in order to guarantee that an overhead budget is never exceeded. However, gaps in monitoring bring uncertainty in the results as it is not possible to assert certainly whether the execution satisfies temporal property ϕ or not [78].

In what follows, we attack these four challenges on a multi-threaded program and a property directly involving one particular thread. We assume that the number of threads is a parameter. Our approach is divided in three steps:

- Learn a hidden Markov Model (HMM) of the CPS to be analyzed by using a set of observation sequences and standard machine-learning techniques [106]. We assume that we can observe the CPS outputs, which are either measurements of the physical part or values output by the cyber part.

- Estimate the CPS state. Having access to the current observation sequence and the learned HMM, we employ statistical inference techniques to determine the hidden state [78]. In order to deal with gaps during the observations, we use the particle filtering algorithm described in [78].

- Drive the simulations until they satisfy the property. We assume that we can start the CPS from a given state, and run it for a given amount of time. In order to steer the system towards the rare event, we use Importance Splitting. However, this requires a property decomposition in a set of levels, such that, the probability of going from one level to the next is essentially equal, and the product of the inter-level probabilities equals the rare event probability. We assume that an heuristic score function has been deduced from observations made on the program containing only the thread of interest. Given the learnt HMM and the score function, we automatically derive an optimal rare event decomposition into levels.

5.8.2 Model description and problem statement

In [124], the authors developed a framework in which an Hidden Markov Model is used to model the program. The internal transition system is theoretically known but detailed internal states are unobservable during simulations. They use standard learning algorithms to learn the HMM from traces that only contain observable actions, relevant with respect to temporal property of interest. They use recursive algorithm for computing the probability that, given an observation sequence $o_1 \cdots o_t$, the HMM is in a particular state. Finally, they compute the uncertainty due to

observation gaps with a state estimation by using a forward algorithm [124] or, alternatively, by using a particle filtering procedure [78].

Hidden Markov model In the 1960's, Leonard Baum introduced Hidden Markov models (HMMs) in a series of articles [13, 12, 15, 14, 11]. It is worth mentioning that hidden Markov models are closely related to an earlier work by Ruslan Stratonovich [125]. HMMs have been proven to be very important for many applications, especially speech recognition [106], character recognition, biological sequence analysis [22], and protein classification problems. Lately, HMMs receive increased attention in the context of communication channel modelling and of QoS properties in wireless networks. In model checking, logic PCTL* has been extended to deal with properties specified on HMMs [140]. An HMM is a statistical Markov model in which the system being modelled is assumed to be a Markov process with hidden states. The occupied hidden state can only be observed through another set of stochastic processes that produce a sequence of observations. Given the sequence of observations, we do not exactly know the occupied state, but we do know the probability distribution over the set of states. This information is captured by a so-called belief state. Formally,

Definition 25 ((Labeled Discrete-Time) Hidden Markov Model) *A Labeled Discrete-Time Hidden Markov Model H is a tuple $(S; P; L; \Theta; \mu; \alpha)$ where:*

- *$(S; P; L)$ is a labeled DTMC,*
- *Θ is a finite set of observations,*
- *$\mu : S \times \Theta \to [0, 1]$ is an observation function satisfying for every $s \in S$,*

$$\sum_{o \in \Theta} \mu(s, o) = 1 \tag{5.24}$$

- *and α is an initial distribution on s such that $\sum_{s \in S} \alpha(s) = 1$.*

The usual learning task in HMMs is to find, given an output sequence of observations, the best set of state transition and output probabilities. No tractable algorithm is known for solving this problem exactly, but a local maximum likelihood can be derived efficiently using forward and backward recursive computations.

Deterministic Finite Automaton The property φ is represented by a deterministic finite automaton:

$$\mathcal{A} = (S_{\mathcal{A}}, s_0, \Sigma, \delta, F) \tag{5.25}$$

where:

- $S_{\mathcal{A}}$ is a set of states,

$$\begin{array}{ccccc} s_{t-1} & \longrightarrow & s_t & \longrightarrow & s_{t+1} \\ \downarrow & & \downarrow & & \downarrow \\ o_{t-1} & & o_t & & o_{t+1} \\ \downarrow & & \downarrow & & \downarrow \\ y_{t-1} & \longrightarrow & y_t & \longrightarrow & y_{t+1} \end{array}$$

Table 5.7: A network composed of an HMM with state (s_t, o_t) and a DFA with states y_t

$$\begin{array}{ccccc} s_{t-1} & \longrightarrow & s_t & \longrightarrow & s_{t+1} \\ \downarrow & & \downarrow & & \downarrow \\ o_{t-1} & & ? & & o_{t+1} \\ \downarrow & & \downarrow & & \downarrow \\ y_{t-1} & \longrightarrow & \tilde{y}_t & \longrightarrow & y_{t+1} \end{array}$$

Table 5.8: A network composed of an HMM with gap $(s_t, ?)$ and a DFA with estimated states $\tilde{y}_t$

- s_0 is an initial state,
- Σ is an alphabet,
- $\delta : S_{\mathcal{A}} \times \Sigma \to S_{\mathcal{A}}$ is a transition function such that $\delta(s, a) = s'$ if (s, a, s') is a transition,
- and F is a set of accepting states.

Table 5.7 illustrates the system evolution. When a transition is taken from s_{t-1} to s_t, a new observation o_t is done and s_t can be estimated by use of $Pr(S_t \mid (S_{t-1}, O_t))$. This distribution is computed by the given distributions $Pr(S_t \mid S_{t-1})$, $Pr(O_t \mid S_t)$ and $Pr(S_0)$. Given the state of the HMM, the state of the DFA can be evaluated.

Table 5.8 illustrates the problem with gaps. As the observations are not all viewable, we estimate the probability distribution in the HMM in order to evaluate the state distribution $\tilde{y}_t$ in the DFA.

Problem statement

The state estimation problem for HMMs adressed in [124, 78] is to compute the sequence of states that have been crossed given an incomplete sequence of observations.

In our problem, given a system model H and $\Diamond^{\leq T}\phi$ a safety property within time T, we aim to estimate probability $Pr(\phi \mid H)$. The satisfaction of ϕ relies on rare sequence of observations $O = (o_1 \cdots o_t)$ within time T. Moreover, the sequence of

observations are incomplete. So, a natural question remains: as importance splitting algorithms are based on branching/resampling algorithms steps, how to use them in simulations with unobservable states to solve the estimation problem?

5.8.3 Running example

In order to illustrate our approach, we use as running example a Dining Philosophers program. This example was chosen because its model is very well known, its complexity nicely scales up, and its rare events are very intuitive. Moreover, the multi-threaded program we use to implement Dining Philosophers illustrates the difficulties encountered when trying to model check real programs, such as their interaction with the operating system and their large state vector. In classic model checking, the former would require checking the associated operating-system functions, and the latter would require some cone-of-influence program slicing. Both are hard to achieve in practice. For monitoring purposes however, all that one needs to do is to instrument the entities of interest (variables, assignments, procedure calls, etc.) and to run the program. Extending monitoring to SSC requires however an HMM, a way of estimating the hidden states, and a way to control the program. To minimize the interference of instrumentation with the program execution, we instrument only one thread. To account for the unknown and possibly distinct executions of the uninstrumented part of the program, we add loops *"do some work"* whose execution time is distributed, for simplicity, according to a uniform probability distribution. We refer to [79] for more details and the full code is available online[4]. Given 150 philosophers, the property of interest is the property that philosopher k succeeds to eat within a given interval of time.

In what follows, we assume that the initial state is 'philosopher k is thinking'. Philosopher k emits symbol $\{T, F, D, @\}$. When the monitor is turned on, symbol @ is thrown with probability 1 whenever she eats. When the monitor is turned off and philosopher k is involved, a gap symbol '?' is however emitted.

Learning the HMM In an HMM, S_{t+1} and O_t only depend on S_t. Learning the HMM implies concretely to run the system (if possible during a long time for more accuracy) and, from the resulting observation sequence $o_1 \cdots o_t$, to use Baum-Welch algorithm to compute the conditional next-state probability distribution $Pr(S_{t+1} \mid S_t)$ and $Pr(O_t \mid S_t)$. The algorithm requires the user to specify the number of states in the HMM and allows the user to provide information about the structure of the system. Note that in our case $Pr(O_t \mid S_t)$ is equal to 0 or 1 in theory but in practise, the learning part can be very approximate. We refer to [124] for more details. The probabilities $Pr(S_{t+1} \mid S_t)$ are stored in a matrix A of size $|S| \times |S|$ and the probabilities $Pr(O_t \mid S_t)$ in a matrix C of size $|S| \times |\Theta|$.

[4] Code repository. https://ti.tuwien.ac.at/tacas2015/

Construction of a score function From the previous (supposedly long enough) trace, we noticed that symbol @ was only emitted after the following sequence of successive observations $(T, F, F, @)$. Thus, we build an heuristic score function as in the dining philosopher case study 5.23 assigning a value in $]0;1]$ whenever a trace emits symbol T for the first time, in $]1;2]$ whenever a trace emits symbol F for the first time, in $]2;3]$ when a trace emits symbol F just after having emitted symbol F and in $]3;4]$ when a trace emits symbol @ for the first time.

State estimation We assign to each trace M particles distributed in a vector π of length $|S|$ with respect to the initial distribution. As a thread-k transition is taken from its current state $S_t = i$, we choose randomly with probability a_{ij} a next state $S_{t+1} = j$. A weight, depending on j and o_{t+1}, is assigned to each particle. This weight is then used in a resampling phase which discards particles that poorly predicted o_{t+1}. The M particles are so redistributed in vector π among the most promising states. When a gap occurs at time $t+1$, we use both distributions $Pr(S_{t+1} \mid S_t)$ and $Pr(O_{t+1} \mid S_t)$ to determine the most likely observation. Note that, given an HMM, $Pr(O_{t+1} \mid S_t)$ may be precomputed. We refer to [78] for more details.

Importance splitting Our trace now complete, we can assign a score and apply the standard importance splitting algorithm whenever the N traces have been generated. The accuracy of this step is of course highly dependent of the reliability of the state estimation. However, the requirement for the state estimation is less demanding. Indeed, we do not need to know with high precision the probability distribution in states but to roughly know which states have been most likely crossed.

The previous steps are repeated for all traces. After the score sorting, we keep the $N - k$ supposedly best traces. The k worst traces are discarded and k instances of successful traces are cloned. The runs indexed by a discarding score are replaced by a clone of a successful trace. The full process is repeated until the score exceeds the final threshold.

5.8.4 Results and discussion

Figure 5.6 illustrates that increasing the probability of gaps impacts the reliability of the estimate and may lead to a significant underestimation. This is mainly due to the fact that a high-scored trace may be likely assigned to a bad trace. These errors add at each iteration. At some point, discarded traces have a high chance to be rebranched on a misleading good trace and the probability to cross a new level severely decreased. In a future work, several strategies could be considered to improve the results:

1. improve the state estimation by increasing the number of particles in the filtering,

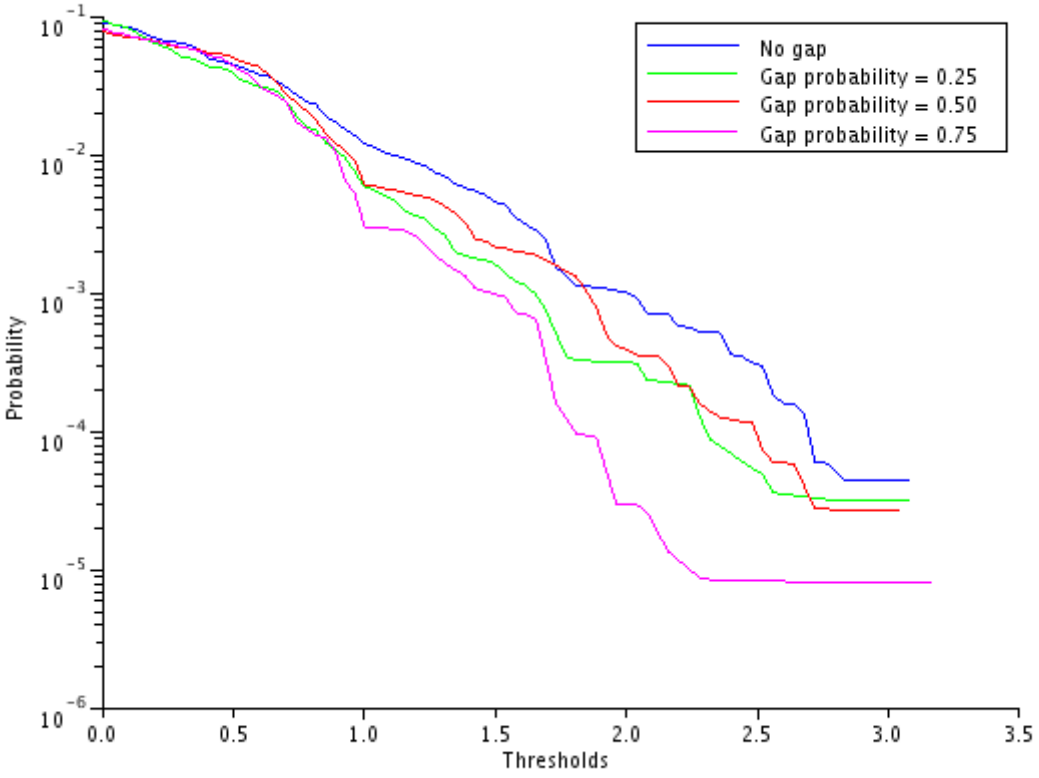


Figure 5.6: Evolution of the estimate after Importance Splitting iterations

2. improve the state estimation by building a 'better' score function more adapted to incomplete traces,

3. improve the important splitting procedure by increasing the number of paths.

If reliability of γ estimate is crucial, for lack of accuracy, it is worth mentioning that ISp-SE gives anyway a lower bound of γ and the system likely satisfies $P_{\geq \tilde{\gamma}_n}\phi$ qualitative property.

5.9 Conclusion

We have introduced the notion of using importance splitting with statistical model checking to verify rare properties. We have described how such properties must be decomposed to facilitate importance splitting and have demonstrated the procedures on several examples. We have described two importance splitting algorithms that may be constrained to give results within confidence bounds.

We have presented an effective heuristic to improve the granularity of score functions for importance splitting. The logical properties used in statistical model checking can thus be decomposed into a greater number of levels, allowing the use of high performance adaptive algorithms. We have presented an optimised adaptive algorithm and shown how, in combination with our heuristic score function, it significantly improves on the performance of the alternatives. Overall, we have shown that the application of importance splitting to statistical model checking has great potential. As future work, we would like to develop a Chernoff bound and sequential hypothesis test to complement the confidence interval presented here.

Conclusion

In this thesis we have defined and presented new results for heterogeneous systems and statistical model checking of rare properties. We summarise the contribution of each chapter and discuss perspectives.

6.1 Contributions

In chapter 3, we presented the syntax and the semantics of SBIP, a stochastic extension of BIP formalism for heterogeneous systems. Statistical model checker SBIP allows the recourse to the stochastic abstraction of components and eliminate the non-determinism. We illustrated the SBIP engine on two experiments out of scope of probabilistic model checking and that can not have been analysed with a pure formal approach. We shall continue the development by implementing new heuristics to speed up simulation and to reduce their number as well as techniques to support unbounded properties. We shall also implement an extension of the stochastic abstraction principle from [6] that allows to compute automatically a small stochastic abstraction from a huge concrete system.

In chapter 4, we presented a simple algorithm that uses cross-entropy minimisation to find an optimal importance sampling distribution. In contrast to previous work, our algorithm is adapted to command-based systems and exploits a naturally defined low dimensional vector of parameters to specify this distribution and thus avoids the intractable explicit representation of a transition matrix. We showed that our parametrisation leads to a unique optimum and can produce many orders of magnitude improvement in simulation efficiency. We demonstrated the efficacy of our methodology by applying it to models from reliability engineering and biochemistry.

Given a model, to improve the quality of our estimator, it is necessary to break the command guards into disjoint guards. We believe that it could be automatised. Of course, at maximum, there would be as commands as individual probability transitions. However, a question still remains: given a maximal intermediate number of commands, how to break guards in a clever way? It is worth mentioning that the general cross-entropy method has received increasing interest in simulation-based verification, notably in testing tools and in SAT solvers [32]. The cross-entropy method was shown to be very efficient in searching for solutions for hard optimization

problems and in locating rare bugs and patterns in large programs [31, 30]. Using a similar approach would help to identify different combinations of parameter values that could be used to define a 'good' break of commands. For example, assume that a rare property is satisfied with same probability by paths dependent on two identical but antagonist parameters (governing 'left' and 'right' movements in a map for example). In this case, the cross-entropy algorithm wouldn't favour one or the other and the optimal parameter wouldn't change. Breaking the commands depending on the position into the graph of transition would be a nice extension of our work.

The reliability of the associated confidence interval is a key challenge for importance sampling as the distribution of likelihood ratio is unknown. For want of exact confidence interval, it may be interesting to propose alternative confidence intervals, more optimistic and more pessimistic than the standard Gaussian or Student confidence interval and to develop more warning signals to indicate that the analysis is suspicious.

In chapter 5, we applied the importance splitting idea to model checking with the help of a score function based on logical properties, and a set of levels that delimit the conditional probabilities. We illustrated how a score function may be derived from a property and gave three importance splitting algorithms: one that uses fixed levels, one that discovers the best levels adaptively and one that optimises the previous algorithms using a heuristic score function. We gave experimental results that demonstrate a significant improvement in performance over alternative approaches and we showed that importance splitting combined with state estimation could be used in hidden Markov models for the verification of rare properties.

We believe that importance splitting has great potential in statistical verification of large systems, as it does not require any change of measure of the system, contrarily to importance sampling. We would like to adapt and extend the applications of the method, especially in cyber-physical systems in which many variables are continuous. A score function depending on these variables would have more likely finer granularity and could give great results.

As for importance sampling, there is no Chernoff bound for our importance splitting algorithms. However, we believe that more research could be done in this direction. Interesting results are potentially exploitable in Guyader, Cérou and Del Moral's work [27][39] and, under some mathematical conditions, a Chernoff-like bound has been proposed by Agnès Lagnoux in [88] for important splitting estimators in the context of Galton-Watson process.

6.2 Future work

We have divided the contents of the future work into three parts. The first concerns future work in statistical model checker PLASMA. The second concerns future work

related to rare event simulation. The third concerns other challenges related to statistical model checking.

Rare event algorithms for Plasma

PLASMA-lab [25] is an efficient SMC library written in Java developed to enable formal analysis of multiple modelling semantics on a single platform and to allow others to integrate our model checking technology into their own software. The software accepts properties described in a form of bounded linear temporal logic (BLTL) extended with custom temporal operators based on concepts such as minimum, maximum and mean of a variable over time. PLASMA-lab has a customisable simulator class that allows rapid prototyping of formal verification solutions using, e.g., Scilab7 or MATLAB8. Model checking modes PLASMA-lab offers three basic modes of model checking: simple Monte Carlo, Monte Carlo using a Chernoff confidence bound and sequential hypothesis testing. There is also a simulation mode for debugging. PLASMA-lab may be instantiated from the command line or from within other software. A graphical user interface (GUI) that exposes the functionality of PLASMA-lab has been constructed and facilitates its use as a standalone application with multiple 'drop-in' modelling languages. To overcome the administrative time needed to distribute SMC on parallel computing architectures, the PLASMA-lab GUI implements a simple and robust client-server architecture, based on Java Remote Method Invocation (RMI) using IPv4/6 protocols. The algorithm will work on dedicated clusters and grids, but can also take advantage of ad hoc networks of heterogeneous computers. PLASMA-lab implements the SMC distribution algorithm of [134], which avoids the statistical bias that might otherwise occur from load balancing.

For now, rare event model checking modes, such as importance sampling and importance splitting, can be implemented as part of the simulator class when the modelling semantics support them. All the algorithms introduced in this thesis have been implemented in Scilab or in the original prototype of PLASMA [70] but, in a future work, we intend to implement, in a more friendly way, the algorithms in the next extension of PLASMA and to test them soon on real case studies.

Rare event simulation for other classes of systems

One of the main difficulty with rare event simulation, also recently mentioned in [108], is to extend the techniques to a larger class of models in an automated way. Indeed, there is no free lunch and these methods exploit knowledge or special features about the system. Approaches that work well in one setting may fail with another setting. A parametrisation for importance sampling may give poor results and a model could require human reasoning to propose an alternative parametrisation. A 'good' score function for importance splitting with respect to a model and a property may be inefficient with another model. It would be interesting to develop automatic

reasoning on the topology of the system in order to propose adequately a simulation technique.

We also believe that importance sampling and splitting could be used in other probabilistic systems, not necessarily described as Markov chains. For example, a timed automaton is a finite automaton extended with a finite set of real-valued clocks. The clock values form guards that may enable or disable transitions and so constrain the possible behaviours of the automaton. A rare property may occur only if several events are triggered in very narrow time intervals. Given large windows of time in which a rare event could occur, we would like to use rare event simulations to identify and 'force' the system to trigger events in narrower and so, more favourable windows. However, the problem of defining a 'good' change of measure or score function still remains and finding a parametrisation in which a cross-entropy method would have a closed-form solution is a non-trivial problem.

Other challenges

Apart from the rare event problem, another limitation of Statistical Model Checking are long traces. Hence, unbounded properties are generally difficult for Statistical Model Checking but not necessarily for Numerical Model Checking (NMC) that takes advantage of the finite model property and finds the static distribution of occupancy for all states. Unbounded properties are therefore no great problem for NMC. For SMC, calculating the steady state distribution is expensive. Unbounded properties must generally be inferred by other means (by some knowledge about the property of or the system), not by actually simulating to infinity. We believe that state lumping techniques could be efficiently used to transform some unbounded until properties into bounded until properties, more accessible to SMC.

Nested probabilistic operators are also challenging for SMC. At each step of a trace, a sub-formula containing a probabilistic operator must be evaluated by its own set of simulations. Nesting further causes an exponential blow-up of simulations. Some work regarding the statistical confidence of nested probabilistic operators of CSL is contained in [134], however the algorithmic complexity remains an open problem.

Basic SMC relies on strong law of large numbers convergence and requires an executable, probabilistic model. Non-deterministic models (e.g. MDPs) characterise the unknown interactions of concurrent systems and deliberately have no single executable semantics. Hence, they are not immediately accessible to SMC. Until recently, SMC has been limited to checking properties of non-deterministic systems using the uniform probabilistic scheduler, to resolving non-determinism in arbitrary ways (e.g., priorities) and to finding optimal schedulers by memory-intensive learning techniques. In [93], however, the authors have developed the basis of the first lightweight sampling-based approach for finding optimal schedulers for MDPs. The technique demonstrates promising results using only $O(1)$ memory, however the

problem of finding rare optimal schedulers remains challenging.

Bibliography

[1] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990.

[2] Rajeev Alur and David L. Dill. The theory of timed automata. In *REX Workshop*, pages 45–73, 1991.

[3] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, J. Sanchez-Gasca, R. Schulz, A. Stankovic, C. Taylor, and V. Vittal. Causes of the 2003 major grid blackouts in North America and Europe, and recommended means to improve system dynamic performance. Technical report, IEEE Power Engineering Society, 2004.

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

[5] Benoît Barbot, Serge Haddad, and Claudine Picaronny. Coupling and Importance Sampling for Statistical Model Checking. In Cormac Flanagan and Barbara König, editors, *TACAS'12*, LNCS, Tallinn, Estonia, March 2012. Springer. To appear.

[6] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *FORTE*, volume 6117 of *LNCS*, pages 32–46. Springer, 2010.

[7] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay, and Emmanuel Sifakis. Verification of an AFDX infrastructure using simulations and probabilities. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 330–344. Springer-Verlag, 2010.

[8] Ananda Basu, Saddek Bensalem, Matthieu Gallien, Felix Ingrand, Charles Lesire, Thanh-Hung Nguyen, and Joseph Sifakis. Incremental component-based construction and verification of a robotic system. In *ECAI*, 2008.

[9] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Systems in BIP. In *SEFM06*, pages 3–12, September 2006.

[10] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *FSTTCS*, pages 260–272, 2006.

[11] Leonard E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[12] Leonard E. Baum and J.A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360, 1967.

[13] Leonard E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[14] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[15] Leonard E. Baum and George R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.

[16] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, pages 147–161, 2001.

[17] Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jegourel, Axel Legay, and journal = Software Tools for Technology Transfer year = 2014 Ayoub Nouri, title = Statistical Model Checking of QoS Properties of Systems in SBIP.

[18] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-Finder: A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] Saddek Bensalem, Lavindra de Silva, Andreas Griesmayer, Félix Ingrand, Axel Legay, and Rongjie Yan. A formal approach for incremental construction with an application to autonomous robotic systems. In *SC'11*, LNCS. Springer, 2011.

[20] Saddek Bensalem, Benoît Delahaye, and Axel Legay. Statistical model checking: Present and future. In *RV*, volume 6418 of *LNCS*. Springer, 2010.

[21] Saddek Bensalem, Axel Legay, Ayoub Nouri, and Doron Peled. Synthesizing distributed scheduling implementation for probabilistic component-based systems. In *MEMOCODE*, pages 87–96, 2013.

[22] M.J. Bishop and Elizabeth A. Thompson. Maximum likelihood alignment of dna sequences. *Journal of Molecular Biology*, 1986.

[23] Simon Bliudze and Joseph Sifakis. The algebra of connectors-structuring interaction in BIP. *IEEE Trans. Comput.*, 57(10):1315–1330, October 2008.

[24] Jonathan Bogdoll, Luis-Maria F. Fiorti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *FORTE*, LNCS. Springer, 2011.

[25] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. PLASMA-lab: A flexible, distributable statistical model checking library. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and PedroR. D'Argenio, editors, *Quantitative Evaluation of Systems*, volume 8054 of *Lecture Notes in Computer Science*, pages 160–164. Springer Berlin Heidelberg, 2013.

[26] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors. *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *Lecture Notes in Computer Science*. Springer, 2005.

[27] Frédéric Cérou, Pierre Del Moral, Teddy Furon, and Arnaud Guyader. Sequential Monte Carlo for rare event estimation. *Statistics and Computing*, 22:795–808, 2012.

[28] Frédéric Cérou, Teddy Furon, Arnaud Guyader, and Cyrille Jegourel. Estimating the probability of false alarm for a zero-bit watermarking technique. In *DSP*. IEEE, 2009.

[29] Frédéric Cérou and Arnaud Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25:417–443, 2007.

[30] Hana Chockler, Karine Even, and Eran Yahav. Finding rare numerical stability errors in concurrent computations. In *ISSTA*, pages 12–22, 2013.

[31] Hana Chockler, Eitan Farchi, Benny Godlin, and Sergey Novikov. Cross-entropy based testing. In *FMCAD*, pages 101–108, 2007.

[32] Hana Chockler, Alexander Ivrii, Arie Matsliah, Simone Fulvio Rollini, and Natasha Sharygina. Using cross-entropy for satisfiability. In *SAC*, pages 1196–1203, 2013.

[33] Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. *Studies in Logic and the Foundations of Mathematics*, 26:118–161, 1963.

[34] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981.

[35] Edmund M. Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In Tevfik Bultan and Pao-Ann Hsiung, editors, *ATVA*, volume 6996 of *LNCS*, pages 1–12. Springer, 2011.

[36] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[37] Pieter-Tjerk De Boer, Victor F. Nicola, and Reuven Y. Rubinstein. Adaptive importance sampling simulation of queueing networks. In *Winter Simulation Conference*, volume 1, pages 646–655, 2000.

[38] Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer, 2004.

[39] Pierre Del Moral and Emmanuel Rio. Concentration inequalities for mean field particle models. *The Annals of Applied Probability*, 21:1017–1052, 2011.

[40] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, August 1975.

[41] Joseph L. Doob. What is a stochastic process? *American Mathematical Monthly*, 49:648–653, 1942.

[42] Joseph L. Doob. *Stochastic process*. Wiley, 1953.

[43] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time. In *POPL*, pages 127–140, 1983.

[44] Yliès Falcone, Mohamad Jaber, Thanh-Hung Nguyen, Marius Bozga, and Saddek Bensalem. Runtime verification of component-based systems. In *SEFM*, pages 204–220, 2011.

[45] Yliès Falcone, Mohamad Jaber, Thanh-Hung Nguyen, Marius Bozga, and Saddek Bensalem. Runtime verification of component-based systems in the BIP framework with formally-proved sound and complete instrumentation. *SOSYM*, pages 1–27, 2013.

[46] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Form. Methods Syst. Des.*, 24(2):101–127, March 2004.

[47] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.

[48] David Gelperin and Bill Hetzel. The growth of software testing. *Commun. ACM*, 31(6):687–695, 1988.

[49] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 412–, Washington, DC, USA, 2001. IEEE Computer Society.

[50] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.

[51] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[52] Patrice Godefroid. Model checking for programming languages using Verisoft. In *POPL*, pages 174–186, 1997.

[53] Radu Grosu and Scott A. Smolka. Monte Carlo model checking. In *TACAS*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.

[54] John M. Hammersley and David C. Handscomb. *Monte Carlo methods*. 1964.

[55] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983.

[56] Klaus Havelund and Thomas Pressburger. Model checking JAVA programs using JAVA PathFinder. *STTT*, 2(4):366–381, 2000.

[57] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *TACAS*, LNCS, pages 342–356. Springer, 2002.

[58] Philip Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Trans. Model. Comput. Simul.*, 5:43–85, January 1995.

[59] Thomas A. Henzinger. The theory of hybrid automata. In *LICS*.

[60] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with BLAST. In *SPIN*, pages 235–239, 2003.

[61] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, pages 73–84, January 2004.

[62] Wassily Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, 58:13–30, 1963.

[63] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[64] Gerard J. Holzmann, Elie Najm, and Ahmed Serhrouchni. SPIN model checking: An introduction. *STTT*, 2(4):321–327, 2000.

[65] Tito Homem-de Mello. A study on the cross-entropy method for rare-event probability estimation. *INFORMS Journal on Computing*, 19(3):381–394, 2007.

[66] Xiaowan Huang, Justin Seyster, Sean Callanan, Ketan Dixit, Radu Grosu, Scott A. Smolka, Scott D. Stoller, and Erez Zadok. Software monitoring with controllable overhead. *STTT*, 14(3):327–347, 2012.

[67] Research Triangle Institute. The economic impacts of inadequate infrastructure for software testing. Technical report, NIST, 2002.

[68] David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Marielle Stoelinga, and Ivan S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *HVC*, volume 4899 of *LNCS*. Springer, 2007.

[69] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross entropy optimisation of importance sampling parameters for statistical model checking. In *CAV*, 2012.

[70] Cyrille Jegourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking - PLASMA. In *TACAS*, LNCS, pages 498–503. Springer, 2012.

[71] Cyrille Jégourel, Axel Legay, and Sean Sedwards. Importance splitting for statistical model checking rare properties. In *CAV*, pages 576–591, 2013.

[72] Cyrille Jegourel, Axel Legay, and Sean Sedwards. An effective heuristic for adaptive importance splitting in statistical model checking. In *ISoLA*, 2014.

[73] Arne Jensen. Markoff chains as an aid in the study of Markoff processes. *Scandinavian Actuarial Journal*, pages 87–91, 1953.

[74] Sandeep Juneja and Perwez Shahabuddin. Fast simulation of Markov chains with small transition probabilities. *Management Science*, 47(4):547–562, 2001.

[75] Herman Kahn. Stochastic (Monte Carlo) Attenuation Analysis. Technical Report P-88, Rand Corporation, July 1949.

[76] Herman Kahn and Theodore E. Harris. Estimation of Particle Transmission by Random Sampling. In *Applied Mathematics*, volume 5 of *series 12*. National Bureau of Standards, 1951.

[77] Herman Kahn and Andy W. Marshall. Methods of Reducing Sample Size in Monte Carlo Computations. *Journal of the Operations Research Society of America*, 1(5), 1953.

[78] Kenan Kalajdzic, Ezio Bartocci, Scott A. Smolka, Scott D. Stoller, and Radu Grosu. Runtime verification with particle filtering. In *RV*, pages 149–166, 2013.

[79] Kenan Kalajdzic, Cyrille Jegourel, Ezio Bartocci, Axel Legay, Scott Smolka, and Radu Grosu. Model checking as control: Feedback control for statistical model checking of cyber-physical systems. submitted to TACAS, 2015.

[80] Joost-Pieter Katoen and Ivan S. Zapreev. Simulation-based CTMC model checking: An empirical evaluation. In *QEST*, pages 31–40. IEEE Computer Society, 2009.

[81] Joost-Pieter Katoen, Ivan S. Zapreev, E. Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. In *QEST*, pages 167–176. IEEE Computer Society, 2009.

[82] Andrey N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Julius Springer, 1933.

[83] Marwan Krunz, Ron Sass, and Herman D. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *INFOCOM*, pages 455–462, April 1995.

[84] Marwan Krunz and Satish K. Tripathi. On the characterization of VBR MPEG streams. In *SIGMETRICS*, pages 192–202, June 1997.

[85] Solomon Kullback. *Information Theory and Statistics*. Dover, 1968.

[86] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.

[87] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 2.0: A tool for probabilistic model checking. In *QEST*, pages 322–323. IEEE, 2004.

[88] Agnès Lagnoux. Rare event simulation. *Probability in the Engineering and Informational Sciences*, 20:45–66, 2006.

[89] Sophie Laplante, Richard Lassaigne, Frédéric Magniez, Sylvain Peyronnet, and Michel de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM TCS*, 8(4), 2007.

[90] Kim G. Larsen. Verification of timed and hybrid systems. In *ICATPN*, pages 39–42, 2000.

[91] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

[92] Axel Legay and Benoît Delahaye. Statistical model checking : An overview. *CoRR*, abs/1005.1327, 2010.

[93] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Scalable verification of Markov decision processes. In *4th Workshop on Formal Methods in the Development of Software (FMDS 2014)*, LNCS. Springer, 2014.

[94] Daniel Lehmann and Michael O. Rabin. On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem (Extended Abstract). In *Proc. 8th Ann. Symposium on Principles of Programming Languages*, pages 133–138, 1981.

[95] Nancy G. Leveson and Clark Savage Turner. Investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[96] Jacques-Louis Lions. Ariane 501 inquiry board report. Technical report, 1996.

[97] Kenneth L. McMillan. *Symbolic model checking*. Kluwer, 1993.

[98] Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, September 1949.

[99] Otto M. Nikodym. Sur une généralisation des intégrales de m.j. radon. *Fundamenta Mathematicae*, 15:131–179, 1930.

[100] Ayoub Nouri, Axel Legay, Saddek Bensalem, and Marius Bozga. SBIP: A statistical model checking extension for the BIP framework. In *First Workshop on Statistical Model Checking*, 2013.

[101] Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10:29–35, 1959.

[102] Emanuel Parzen. *Stochastic Processes*. Holden Day, 1962.

[103] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[104] Arthur N. Prior. *Time and Modality*. The Clarendon Press, 1957.

[105] Diana El Rabih and Nihal Pekergin. Statistical model checking using perfect simulation. In *ATVA*, volume 5799 of *LNCS*, pages 120–134. Springer, 2009.

[106] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[107] Balaji Raman, Ayoub Nouri, Deepak Gangadharan, Marius Bozga, Ananda Basu, Mayur Maheshwari, Axel Legay, Saddek Bensalem, and Samarjit Chakraborty. Stochastic modeling and performance analysis of multimedia socs. In *ICSAMOS*, pages 145–154, 2013.

[108] Daniël Reijsbergen. *Efficient Simulation Techniques for Stochastic Model Checking*. PhD thesis, University of Twente, 2013.

[109] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner Scheinhardt, and Boudewijn Haverkort. Rare event simulation for highly dependable systems with fast repairs. *Performance Evaluation*, 69(7–8):336 – 355, 2012.

[110] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner R. W. Scheinhardt, and Boudewijn R. Haverkort. Rare event simulation for highly dependable systems with fast repairs. *Perform. Eval.*, 69(7-8):336–355, 2012.

[111] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic*. Springer, 1971.

[112] Ad Ridder. Importance sampling simulations of Markovian reliability systems using cross-entropy. *Annals of Operations Research*, 134:119–136, 2005.

[113] Ad Ridder. Asymptotic optimality of the cross-entropy method for Markov chain problems. *Procedia Computer Science*, 1(1):1571 – 1578, 2010.

[114] Christian P. Robert and George Casella. *Monte Carlo statistical methods*. Springer, 2nd edition, 2004.

[115] Grigore Rosu and Saddek Bensalem. Allen linear (interval) temporal logic - translation to LTL and monitor synthesis. In *CAV*, volume 4144 of *LNCS*, pages 263–277. Springer, 2006.

[116] Gerardo Rubino and Bruno Tuffin, editors. *Rare Event Simulation using Monte Carlo Methods*. Wiley, 2009.

[117] Reuven Y. Rubinstein. The Cross-Entropy Method for Combinatorial and Continuous Optimization. 1:127–190, 1999.

[118] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004.

[119] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.

[120] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, pages 251–252. IEEE, September 2005.

[121] Perwez Shahabuddin. Importance Sampling for the Simulation of Highly Reliable Markovian Systems. *Management Science*, 40(3):333–352, 1994.

[122] John E. Shore and Rodney W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26(1):26–37, January 1980.

[123] Joseph Sifakis. A unified approach for studying the properties of transition systems. *Theor. Comput. Sci.*, 18:227–258, 1982.

[124] Scott D. Stoller, Ezio Bartocci, Justin Seyster, Radu Grosu, Klaus Havelund, Scott A. Smolka, and Erez Zadok. Runtime verification with state estimation. In *RV*, pages 193–207, 2011.

[125] Ruslan L. Stratonovich. Conditional Markov processes. *Theory of Probability and its Applications*, 5(2):156–178, 1960.

[126] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338, 1985.

[127] Moshe Y. Vardi. Alternating automata and program verification. In *In Computer Science Today. LNCS 1000*, pages 471–485. Springer-Verlag, 1995.

[128] Manuel Villén-Altamirano and José Villén-Altamirano. RESTART: A Method for Accelerating Rare Event Simulations. In J. W. Cohen and C. D. Pack, editors, *Queueing, Performance and Control in ATM*, pages 71–76. Elsevier, 1991.

[129] Manuel Villén-Altamirano and José Villén-Altamirano. Restart: a straightforward method for fast simulation of rare events. In *Proc. of the 26th Winter Simulation Conference*, pages 282–289. Society for Computer Simulation International, 1994.

[130] Abraham Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[131] Duminda Wijesekera and Jaideep Srivastava. Quality of Service (QoS) Metrics for Continuous Media. *Multimedia Tools and Applications*, 3(2):127–166, July 1996.

[132] Pierre Wolper. Lectures on formal methods and performance analysis. chapter Constructing automata from temporal logic formulas: a tutorial, pages 261–277. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[133] Håkan L. S. Younes. Probabilistic verification for "black-box" systems. In *CAV*, pages 253–265, 2005.

[134] Håkan L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, 2005.

[135] Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF*, pages 144–160, 2010.

[136] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.

[137] Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.

[138] Håkan L. S. Younes. YMER: A Statistical Model Checker. In Kousha Etessami and Sriram Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 171–179. Springer, 2005.

[139] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404, pages 23–39. Springer, 2002.

[140] Lijun Zhang, Holger Hermanns, and David N. Jansen. Logic and model checking for hidden Markov models. In *FORTE*, pages 98–112, 2005.

[141] Paolo Zuliani, Christel Baier, and Edmund M. Clarke. Rare-event verification for stochastic hybrid systems. In *HSCC*, pages 217–226. ACM, 2012.

[142] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to Stateflow/Simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.

Résumé. – Dans cette thèse, nous considérons deux problèmes auxquels le *model checking* statistique doit faire face. Le premier concerne les systèmes hétérogènes qui introduisent complexité et non-déterminisme dans l'analyse. Le second problème est celui des propriétés rares, difficiles à observer et donc à quantifier.

Pour le premier point, nous présentons des contributions originales pour le formalisme des systèmes composites dans le langage BIP. Nous en proposons une extension stochastique, SBIP, qui permet le recours à l'abstraction stochastique de composants et d'éliminer le non-déterminisme. Ce double effet a pour avantage de réduire la taille du système initial en le remplaçant par un système dont la sémantique est purement stochastique sur lequel les algorithmes de *model checking* statistique sont définis.

La deuxième partie de cette thèse est consacrée à la vérification de propriétés rares. Nous avons proposé le recours à un algorithme original d'échantillonnage préférentiel pour les modèles dont le comportement est décrit à travers un ensemble de commandes. Nous avons également introduit les méthodes multi-niveaux pour la vérification de propriétés rares et nous avons justifié et mis en place l'utilisation d'un algorithme multi-niveau optimal. Ces deux méthodes poursuivent le même objectif de réduire la variance de l'estimateur et le nombre de simulations. Néanmoins, elles sont fondamentalement différentes, la première attaquant le problème au travers du modèle et la seconde au travers des propriétés.

Abstract. – In this thesis, we consider two problems that statistical model checking must cope. The first problem concerns heterogeneous systems, that naturally introduce complexity and non-determinism into the analysis. The second problem concerns rare properties, difficult to observe, and so to quantify.

About the first point, we present original contributions for the formalism of composite systems in BIP language. We propose SBIP, a stochastic extension and define its semantics. SBIP allows the recourse to the stochastic abstraction of components and eliminate the non-determinism. This double effect has the advantage of reducing the size of the initial system by replacing it by a system whose semantics is purely stochastic, a necessary requirement for standard statistical model checking algorithms to be applicable.

The second part of this thesis is devoted to the verification of rare properties in statistical model checking. We present a state-of-the-art *importance sampling* algorithm for models described by a set of guarded commands. Lastly, we motivate the use of *importance splitting* for statistical model checking and set up an optimal splitting algorithm. Both methods pursue a common goal to reduce the variance of the estimator and the number of simulations. Nevertheless, they are fundamentally different, the first tackling the problem through the model and the second through the properties.