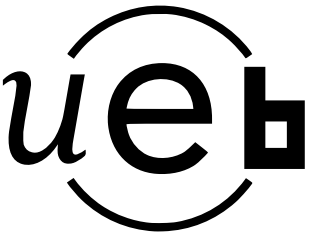
ANNÉE 2015



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

École doctorale Matisse

présentée par

Paulin FOURNIER

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Systèmes Aléatoires ISTIC

Parameterized verification of networks of many identical processes

Soutenance prévue à Rennes
le 17 décembre 2015

devant le jury composé de :

PAROSH ABDULLA
Professeur Université d'Uppsala / *Rapporteur*

JÉROME LEROUX
Directeur de recherche CNRS LaBRI / *Rapporteur*

SADDEK BENSALEM
Professeur Université Joseph Fourier Grenoble / *Examinateur*

JOOST-PIETER KATOEN
Professeur RWTH Université de Aachen / *Examinateur*

SOPHIE PINCHINAT
Professeur Université Rennes 1 / *Examinatrice*

THIERRY JÉRON
Directeur de recherche INRIA Rennes / *Directeur de thèse*

NATHALIE BERTRAND
Chargée de recherche INRIA Rennes / *Encadrante*

ARNAUD SANGNIER
Maître de conférence Université Paris Diderot / *Encadrant*

Contents

Vérification paramétrée de réseaux composés d'une multitude de processus identiques

Durant ces dernières années, on a vu une croissance importante des systèmes informatisés dans notre vie de tous les jours, depuis les ordinateurs et les téléphones intelligents jusqu'aux voitures et avions. En raison de leur utilisation critique, par example dans les centrales nucléaires ou le contrôle du trafic aérien, il est d'une importance primordiale de garantir l'absence de défauts dans ces installations. On constate donc un besoin d'outils qui permettent de vérifier automatiquement la sécurité et la correction de ces systèmes. Une des approches pour prouver leur correction est la vérification formelle. En particulier, la vérification de modèle, ou *model checking*, est une technique de vérification formelle entièrement automatisée. Le model checking consiste à vérifier que le système répond à ses spécifications en vérifiant que le modèle du système satisfait une formule représentant la spécification, voir par example [BK08]. Pour cela, le model checking effectue une exploration exhaustive de tous les comportements possibles du modèle et vérifie si ces comportements satisfont tous la spécification donnée. Si ça n'est pas le cas, un contre-exemple (*i.e.* un comportement qui viole la spécification) est produit à des fins de débogage. Le prix Turing ACM attribué en 2007 à Clarke, Emerson et Sifakis pour leurs travaux sur le model checking témoigne de l'importance de ce sujet. De plus des *model checkers* comme UPPAAL [LPY97] ou Prism [KNP11] sont largement utilisés dans le milieu académique comme dans l'industrie.

Initialement, les techniques de vérification de modèle étaient applicables à des systèmes comportant seulement un nombre restreint d'états. Cependant, de plus en plus de systèmes sont composés de plusieurs sous-systèmes, ce qui entraine une explosion combinatoire du nombre d'états, rendant la vérification de modèle inefficace. Ce problème est connu comme le problème de l'explosion de l'espace d'état. Pour résoudre ce problème, de nombreuses techniques ont été développées comme la représentation symbolique de données, la réduction par ordre partiel ou les abstractions. Un autre axe de développement pour la vérification de modèles est d'envisager des systèmes paramétrés pour lesquels une valeur n'est pas fixée et est considérée comme un paramètre. L'analyse de systèmes paramétrés équivaut à l'étude de toute une famille de systèmes ou chacun d'eux est obtenu par instanciation de la valeur du paramètre. Dans cette thèse, nous

étudions la vérification paramétrée de réseaux aillant pour paramètre leur taille comme paramètre. Notre objectif est de développer des techniques pour vérifier les réseaux indépendamment du nombre de composants.

Systèmes paramétrés Grâce aux progrès matériels et logiciels, les systèmes distribués sont dorénavant parmi les principaux paradigmes lors de l'élaboration de grands systèmes. Fournir des méthodes pour analyser et vérifier les systèmes distribués est une tâche complexe et ce pour plusieurs raisons. Premièrement, il existe différentes familles de systèmes distribués en fonction des moyens de communication (mémoire partagée ou passage de messages), de la puissance de calcul des entités concernées, de la connaissance du système attribuée aux entités (pleine connaissance, ou connaissance locale de leur voisins, ou aucune connaissance) et du type de topologie de communication (anneau, arbre, graphe quelconque, etc). Deuxièmement, la plupart des protocoles développés pour les systèmes distribués sont censés fonctionner pour un nombre illimité de participants ; par conséquent, afin de vérifier que le système se comporte correctement, il faut développer des méthodes qui permettent de faire face à un tel paramètre. En effet, la certification d'un composant indépendamment de l'ensemble du réseau, ou même d'un nombre fini de participants dans un réseau n'est pas suffisante, d'où l'analyse des systèmes paramétrés.

Dans leur article fondateur sur les modèles distribués avec de nombreuses entités identiques [GS92], German et Sistla représentent le comportement d'un réseau de machines à états finis qui interagissent par des rendez-vous. Des variantes ont ensuite été proposées pour gérer les différents moyens de communication, comme la diffusion de messages [EFM99], le passage de jetons [CTTV04, AJKR14] ou la mémoire partagée [EGM13]. Dans son article de synthèse sur les modèles paramétrés [Esp14], Esparza montre que des modifications mineures, telles que la présence ou l'absence d'un contrôleur dans le système, peuvent modifier radicalement la complexité des problèmes de vérification.

Parmi les différents modèles paramétriques de réseaux, les protocoles de diffusion (*broadcast protocols*), étudiés à l'origine par Esparza *et al.* [EFM99], ont plus tard été analysés sous un nouveau point de vue, conduisant à de nouvelles perspectives sur les problèmes de vérification. Plus précisément, un modèle de bas niveau pour représenter les principales caractéristiques des réseaux ad-hoc a été proposé [DSZ10]. Il caractérise les aspects suivants de ces systèmes : le réseau est équipé d'une topologie de communication et les nœuds dans le réseau peuvent communiquer avec leurs voisins à l'aide de messages diffusés. En outre, le nombre de participants est non borné. Dans ce modèle, chaque entité se comporte de manière similaire, en suivant un protocole qui est représenté par un automate fini comprenant trois types d'actions: (1) diffusion d'un message, (2) réception d'un message et (3) action interne. De plus, la topologie de communication ne change pas et aucune entité n'est supprimée ou ajoutée au cours de l'exécution.

Le *problème de l'accessibilité paramétrée* consiste à déterminer s'il existe un nombre initial d'entités et une topologie de communication de telle sorte qu'il est possible d'atteindre une configuration où au moins un processus est dans un état spécifique.

Le problème de l'accessibilité permet ainsi de détecter une erreur dans la conception du protocole en exhibant la possibilité d'atteindre un état d'erreur. La principale difficulté pour résoudre un tel problème réside dans le fait que le nombre de processus ainsi que la topologie de communication initiale sont des paramètres. Il faut donc prouver que quelle que soit la valeur de ces paramètres, l'état d'erreur n'est pas accessible. Dans [DSZ10], il est prouvé que le problème de l'accessibilité paramétrée est indécidable pour les protocoles de diffusion. Cela vaut également pour le *problème de synchronisation paramétrée*, qui demande si tous les processus peuvent converger vers un ensemble d'états spécifiques. Pour les deux problèmes, la décidabilité peut cependant être regagnée, en tenant compte de topologies de communication qui peuvent changer à tout moment de manière non-déterministe [DSTZ12]. Une autre option pour regagner la décidabilité du problème de l'accessibilité est de restreindre les topologies de communication à des graphes complets (appelés cliques) [DSZ11a], ou à des graphes orientés de profondeur bornée [AAR13] .

Cette thèse s'inscrit dans ce cadre. Nous étudions des problèmes de vérification paramétrée afin de vérifier un réseau indépendamment du nombre de composants, sans énumérer toutes les tailles possibles du réseau.

Aspects quantitatifs En plus de la concurrence, l'analyse quantitative des systèmes est vraiment importante. En effet, il n'est pas suffisant de dire qu'une propriété est satisfaite ou non, mais dans quelle mesure. Dans ce but, des modèles probabilistes ont été étudiés. Ils permettent de modéliser des comportements inconnus tels que l'interaction de l'environnement avec le système qui est modélisé par des actions stochastiques. Avec des modèles probabilistes, il est possible d'exprimer à quel point une propriété est satisfaite. Les modèles probabilistes sont également pertinents pour les algorithmes distribués. En effet, les algorithmes randomisés sont largement utilisés car ils permettent de rompre la symétrie des composants. Par exemple, Lehmann et Rabin ont proposé une solution randomisée au problème bien connu du dîner des philosophes [RL94]. Un autre exemple est celui des algorithmes de CSMA, utilisés pour la résolution de la contention dans les réseaux informatiques (voir par exemple la norme ZigBee [Spe]).

Pour les systèmes à états finis avec du non-déterminisme et des probabilités (comme les processus de décision markoviens à états finis), la plupart des problèmes de vérification sont décidables [BK08]. Cependant, quand le nombre d'états est infini, ces problèmes sont beaucoup plus difficiles à aborder. Un cadre général pour les systèmes de transitions infinis purement probabilistes a été proposé dans [AHM07]. Cependant, il semble difficile d'adapter un tel cadre dans le cas combinant des choix probabilistes et non déterministes. En effet, l'introduction des probabilités peut même conduire à l'indécidabilité de problèmes qui sont décidables dans le cas non probabiliste. Par exemple, pour les extensions de systèmes à pile avec des choix non-déterministes et probabilistes, la vérification est indécidable [EY05, BKS05]. D'autre part, l'introduction de transitions probabilistes dans les systèmes infinis ne conduit pas toujours à l'indécidabilité, mais des méthodes de vérification dédiées doivent être développées, comme c'est le cas par exemple pour les systèmes à canaux avec pertes

probabilistes [BBS07].

Un aspect supplémentaire à prendre en compte est le comportement de systèmes temporisés. Les systèmes informatisés sont utilisés pour des applications critiques qui nécessitent des contraintes de temps. Une réponse est non seulement exigée, mais celle-ci doit en plus être donnée dans un certain délai. Les automates temporisés, introduits dans [AD90, AD94], sont des automates à états finis équipés avec des horloges. Ces automates temporisés ont été largement étudiés et la plupart des problèmes de vérification est connue pour être décidable. Toutefois, la vérification devient plus difficile pour les systèmes avec un nombre infini d'états. En effet, la vérification paramétrée de réseaux composés d'un nombre non borné de processus équipés avec des horloges a été étudiée [AJ03, ADM04, ADR$^+$11] et les problèmes de vérification paramétrée sont indécidables lorsque les processus ont plus d'une horloge.

Jusqu'à présent, à notre connaissance, la vérification automatique des protocoles probabilistes temporisés n'a été effectuée que pour un nombre de nœuds fixé et restreint, voir par exemple [Fru06]. En effet, l'approche classique est de construire le réseau en effectuant le produit des protocoles participants, puis d'exécuter un vérificateur de modèle sur ce réseau. Cependant, cette méthode se heurte au problème de l'explosion de l'espace d'état en raison du produit de plusieurs composants. En comparaison, dans cette thèse, nous développons de nouvelles techniques qui permettent de vérifier les réseaux probabilistes indépendamment de leur taille.

Détermination de la frontière entre décidabilité et indécidabilité En plus de l'utilisation pratique de ce travail qui permet de modéliser les systèmes distribués combinant temps et probabilités ainsi que le nombre non borné de participants, ce travail est vraiment intéressant d'un point de vue théorique. En effet, les systèmes paramétrés sont puissants et cela conduit à l'indécidabilité de problèmes de vérification. De nombreuses restrictions ont été étudiées pour être en mesure d'effectuer la vérification de modèle sur des modèles paramétrés. Cette thèse étudie donc la frontière entre décidabilité et indécidabilité des problèmes de vérification pour les problèmes paramétrés probabilistes. En outre, même lorsque les problèmes sont décidables, ils sont souvent d'une grande complexité et ne passent donc pas à l'échelle. Nous étudions donc les restrictions sur la topologie de communication afin d'obtenir des solutions passant à l'échelle.

Organisation et contributions de la thèse

Chapitre I Le premier chapitre est consacré à la mise en place des modèles de base utilisés dans cette thèse. En particulier, nous introduisons les automates temporisés, les chaînes de Markov et les processus de décision markoviens ainsi que certains systèmes de transition à nombre d'états infini. Pour chacun de ces modèles, nous présentons certains résultats ainsi que des techniques intéressantes pour cette thèse.

Chapitre II Ce chapitre est consacré aux travaux existants sur la vérification paramétrée. Cette vue d'ensemble de l'état de l'art présente des modèles existants utilisés dans la littérature pour la modélisation de systèmes construits sur un grand nombre de composants. En particulier, l'attention est portée sur les réseaux de protocoles de diffusion qui sont la base sur laquelle ce travail est construit. En effet, les nouveaux modèles que nous introduisons dans cette thèse sont tous des variantes de protocoles de diffusion.

Nous arrivons ensuite aux chapitres dédiés aux contributions techniques de cette thèse.

Chapitre III La première contribution présentée est l'introduction d'un nouveau modèle, à savoir les réseaux de protocoles temporisés probabilistes, qui combinent trois aspects : le nombre inconnu de participants, les probabilités, et le temps. Dans ce modèle, un nombre non borné de processus est arrangé dans un réseau. Chacun de ses composants exécutent le même protocole décrit par un automate probabiliste fini qui combine actions non-déterministes, actions probabilistes ainsi que des contraintes temporelles. De plus, les composants sont autorisés à communiquer les uns avec les autres par l'intermédiaire de diffusion de messages. Bien que ce modèle soit très intéressant du fait qu'il soit infini (nombre non borné de composants) et qu'il combine probabilités, non-déterminisme et contraintes temporelles, il apparaît que les problèmes basiques de vérification tels que le problème d'accessibilité paramétrée sont indécidables.

Seuls quelques modèles infinis combinent probabilités et non-déterminisme, dont les systèmes de canaux (*channel systems*). Dans ce modèle, une machine à états finis peut écrire et lire les messages dans un canal FIFO non borné. Il a été prouvé que les problèmes de vérification sont indécidables pour les systèmes de canaux [BZ83]. Cependant, en autorisant la présence de défauts dans le système, via la perte de messages, les problèmes de vérification deviennent décidables [AJ93]. En particulier, les systèmes de canaux à pertes probabilistes, où à chaque étape chaque message a indépendamment une probabilité de se perdre, les problèmes de vérification sont décidables. Inspirés par ces travaux, afin de représenter la notion de mobilité dans le réseau, nous introduisons les réseaux dynamiques de protocoles probabilistes temporisés, où les processus peuvent disparaître et être créés selon des lois de probabilité fixes.

Étant donné un réseau (statique ou dynamique) de protocoles probabilistes temporisés, nous considérons les *problèmes de vérification paramétrés* tels que les suivants. Pour un état du modèle des processus, existe-t-il une taille de réseaux pour laquelle, presque sûrement, une configuration contenant un processus dans cet état particulier peut être atteinte, quelle que soit la politique d'ordonnancement ? De manière équivalente, le problème est de savoir si, indépendamment du nombre de processus, la probabilité minimum pour atteindre une configuration cible est 1. Une autre question est de savoir si l'on peut atteindre presque surement l'ensemble de configurations où tous les processus sont réunis dans un certain ensemble d'états? Au-delà de ces problèmes particuliers, nous considérons toutes les variantes où les probabilités minimum ou maximum sont comparées à des seuils entre 0 et 1.

Même si le problème de l'accessibilité paramétrée est connu pour être décidable pour

les réseaux statiques non-probabilistes [DSZ11a], dans les réseaux statiques probabilistes la plupart des problèmes de vérification paramétrés sont indécidables. Ceci est démontré par des réductions du problème de l'arrêt pour les machines à deux compteurs [Min67]. Certains cas restants sont montrés décidables, soit par réduction au cas non-probabiliste, soit grâce à un résultat de monotonie, permettant de considérer uniquement des réseaux composés d'un seul processus.

Pour les réseaux dynamiques de protocoles probabilistes temporisés, nous montrons que l'on peut abstraire la valeur des horloges en régions, ce qui est une adaptation de l'abstraction des régions connue pour les automates temporisés, mais avec un nombre non borné d'horloges. En outre, nous montrons l'existence d'un ordre partiel bien fondé sur ces régions. Nous montrons ensuite que le réseau abstrait grâce aux régions a un attracteur fini. Ainsi, nous pouvons adapter la technique développée pour les systèmes de canaux avec pertes probabilistes afin de résoudre les problèmes d'accessibilité qualitatifs. Ces résultats ont été publiés dans [BF13].

Dans cette thèse, nous étudions également des problèmes quantitatifs paramétrés. Au lieu de comparer la valeur de la probabilité uniquement à des seuils qualitatifs (0 ou 1), nous comparons à toutes les valeurs comprises entre 0 et 1. Bien que les problèmes quantitatifs paramétrés soient encore ouverts, nous obtenons un résultat significatif montrant la décidabilité de l'approximation de la probabilité minimale d'atteindre un état, pour une taille initiale fixée du réseau. On remarquera que, même si la taille initiale est fixée, il peut y avoir un nombre non borné de participants dans le réseau, en raison des créations probabilistes de processus. Ce résultat est intéressant car il est inspiré par le schéma d'approximation dans les systèmes de canaux avec pertes entièrement probabilistes, cependant une telle approximation manquait pour les systèmes infinis combinant non-déterminisme et probabilités.

Chapitre IV Dans ce chapitre, nous présentons un nouveau modèle nommé *réseaux à diffusion sélective de protocoles probabilistes*. Ce modèle étend le modèle des réseaux de diffusion reconfigurables étudiés dans [DSTZ12] en autorisant des actions internes probabilistes, qui permettent à un processus de changer son état interne selon une distribution probabiliste. La principale différence avec les réseaux en clique étudiés dans le chapitre III est que les messages ne parviennent plus à tous les processus, mais seulement à un sous-ensemble choisi de manière non-déterministe.

Dans ce chapitre, nous étudions les versions probabilistes du *problème de l'accessibilité d'un état* qui demande s'il existe une stratégie de résolution du non-déterminisme qui permette d'atteindre une configuration dans laquelle au moins un processus est dans un état spécifique. Nous nous concentrons sur les variantes qualitatives de ce problème en comparant les probabilités seulement à 0 et 1. Ce problème est pertinent de par le fait qu'il permet de vérifier si le protocole est bien conçu, par exemple en vérifiant si un état d'erreur peut être atteint avec une probabilité positive. En plus de l'existence d'une stratégie, nous étudions aussi si toutes les stratégies atteignent le seuil de probabilité. Cela permet de vérifier, par exemple, que quels que soient les choix non déterministes, le protocole évite presque sûrement un état d'erreur.

Dans le chapitre précédent (chapitre III), nous avons vu que les réseaux en clique de

protocoles probabilistes temporisés sont un modèle très puissant conduisant à l'indécidabilité des problèmes qualitatifs d'accessibilité paramétrée. Toutefois, en prenant inspiration des modèles d'états infinis combinant non-déterminisme et probabilités, nous regagnons la décidabilité en considérant des disparitions et créations probabilistes de processus. Cependant, la décidabilité se fait au prix d'une grande complexité. Afin d'obtenir de meilleures bornes de complexité, ce chapitre est consacré à l'étude des réseaux à diffusion sélective. En effet, la complexité du problème de l'accessibilité est beaucoup plus faible (PTIME) dans les réseaux reconfigurables à diffusion. En outre, au lieu d'appliquer un ordre partiel bien fondé qui a une grande complexité, dans ce chapitre, nous adaptons une réduction connue (voir par exemple [CdAFL09]) pour la vérification des modèles probabilistes finis, qui consiste à traduire un processus de décision markovien (MDP) dans un jeu à 2 joueurs, pour lequel l'existence d'une stratégie gagnante est équivalente aux questions d'accessibilité dans le MDP. Cependant, en raison du nombre inconnu de participants, nous avons un modèle infini, donc nous ne pouvons pas appliquer directement cette solution.

Afin de résoudre ce problème, nous introduisons des *jeux de parité distribués*, qui sont des réseaux composés d'un nombre inconnu de processus qui suivent tous le même protocole fini de parité. La nouveauté est qu'il y a maintenant deux joueurs et un objectif de parité. Puisque le deuxième joueur est introduit afin de simuler des choix probabilistes, il est moins puissant que le joueur 1. En effet, dans les jeux distribués le premier joueur est celui qui décide quel processus joue, l'ensemble de processus qui reçoivent les messages et l'action à jouer lorsque le processus sélectionné est dans un état appartenant au joueur 1. Le deuxième joueur, lui, ne peut choisir l'action à jouer que lorsque le premier joueur a choisi un processus dans un état appartenant au joueur 2. Dans ce cadre, nous montrons que l'on peut décider, dans co-NP si le joueur 1 a une stratégie qui répond à un objectif de parité pour tous les choix effectués par le joueur 2. En outre, nous fournissons une traduction, au niveau du protocole, des problèmes d'accessibilité qualitatifs paramétrés dans les réseaux à diffusion sélective de protocoles probabilistes vers le problème de jeu paramétré dans les jeux de parité distribués.

En plus de leur utilité pour résoudre les problèmes probabilistes qualitatifs, nous pensons que les jeux de parité distribués sont un outil intéressant en eux même. En effet, ce modèle de jeu distribué, ainsi que les techniques utilisées pour résoudre le jeu, sont vraiment intéressants et pourraient être utiles dans des contextes différents pour la vérification de réseaux composés d'une multitude de processus identiques. Les résultats présentés dans ce chapitre ont été publiés dans [BFS14].

Chapitre V Le dernier chapitre de cette thèse est basé sur la remarque suivante : les solutions proposées reposent toutes sur des stratégies centralisées qui choisissent les actions des processus avec une pleine connaissance du réseau. En raison du non-déterminisme dans la description du protocole, il peut arriver que deux processus se comportent différemment, même s'ils ont la même information sur ce qui est arrivé jusqu'à présent. Afin d'interdire de tels comportements non réellement distribués, dans le chapitre V, nous forçons les processus à prendre les mêmes décisions dans le cas où ils ont tiré la même séquence de transitions jusqu'ici.

Nous définissons le passé d'un processus comme la séquence de transitions qu'il a prise jusqu'ici. Les *stratégies locales* permettent d'assurer que les processus ayant le même passé prennent la même décision. Par exemple, à partir de la configuration initiale, donc avec un passé vide, si un processus choisit d'effectuer une action interne, par la suite tous les processus avec un passé vide doivent effectuer la même action interne, à moins qu'ils obtiennent des informations supplémentaires en recevant un message.

Nous étudions les problèmes de l'accessibilité et de la synchronisation paramétrées dans les réseaux limités aux stratégies locales. Nous considérons deux types de réseaux différents : d'abord, les réseaux reconfigurables de diffusion pour lesquels les messages atteignent seulement un sous-ensemble des processus choisis de façon non-déterministe. Dans ces réseaux, sans l'hypothèse de stratégies locales, le problème d'accessibilité paramétré est connu pour être PTIME [DSTZ12], de même pour le problème de synchronisation. Dans la deuxième classe de réseaux, la topologie considérée est une clique, *i.e.* les messages atteignent toujours tous les processus.

Nous montrons que les problèmes de l'accessibilité et de la synchronisation avec des stratégies locales dans les réseaux reconfigurables sont NP-complets. Pour obtenir la borne supérieure, nous prouvons que les stratégies locales peuvent être succinctement représentées par des arbres finis que nous appelons les patrons de stratégie. Intuitivement, les patrons de stratégie sont des dépliages finis du protocole avec l'hypothèse supplémentaire que, à partir de chaque nœud, une seule action active est présente. Ils représentent donc les choix de la stratégie locale. En outre, pour répondre aux problèmes d'accessibilité et de synchronisation, nous montrons que nous pouvons équiper les patrons de stratégie avec un ordre représentant l'ordre dans lequel une exécution peut visiter les nœuds du modèle. Enfin, nous obtenons un algorithme NP en montrant que nous pouvons nous concentrer sur les patrons de stratégie de tailles polynomiales afin de résoudre les problèmes paramétrés.

Pour les réseaux en clique, sans surprise, les problèmes sont plus difficiles car le problème d'accessibilité paramétré a déjà été démontré non primitif récursif, sans l'hypothèse de la localité. Nous montrons que le problème est en fait indécidable avec la restriction à des stratégies locales. La preuve est basée sur la simulation d'une machine à deux compteurs pour laquelle le problème de l'arrêt est indécidable [Min67]. Néanmoins, avec l'hypothèse supplémentaire que chaque message peut être reçu dans chaque état, donc que toutes les communications apportent des informations à tous les processus dans le réseau, nous pouvons montrer que le problème de l'accessibilité est décidable. La preuve de décidabilité est basée sur une abstraction que nous prouvons être un système de transition bien structuré, ce qui implique que le problème de l'accessibilité est décidable [ACJT96, AJ01, FS01]. Ces résultats ont été publiés dans [BFS15].

Il est à noter que le problème notablement difficile de synthèse de contrôleurs distribués [PR90] est relativement proche du problème de l'existence d'une stratégie locale. En effet, une stratégie locale correspond à un contrôleur local pour les processus dont le rôle est de résoudre les choix non déterministes. De plus les stratégies locales sont d'un grand intérêt pour mettre en œuvre des algorithmes distribués. En effet, le non-déterminisme est difficile à mettre en œuvre en pratique, on peut donc voir les

stratégies locales comme une version déterministe implémentable d'une spécification non-déterministe pour un algorithme distribué.

La suite du manuscrit détaille ces contributions. La langue employée est l'anglais pour pouvoir être lu par un jury de thèse international.

Introduction

In recent years, computerized systems have been more and more present in our every day life from computers and smart-phones to cars and planes. Due to their critical use, in power plants or air traffic control for example, it is of paramount importance to guarantee the absence of flaws in the implementations. There is thus a need in developing tools that allow to automatically check the safety and correctness of these systems. One of the approaches to prove their correctness is formal verification. In particular, model checking is a fully automated formal verification technique. Model checking consists in verifying that the system meets its specification by verifying that a model of the system satisfies a formula representing the specification, see *e.g.* [BK08]. It performs an exhaustive exploration of all possible behaviors of the model and checks if those behaviors all satisfy the given specification. In the negative case, a counter-example (*i.e.* some behavior that violates the specification) is output for debugging purposes. As a witness of the importance of model checking, Clarke, Emerson, and Sifakis were awarded in 2007 the ACM Turing Award for their work on the subject. Model checkers such as UPPAAL [LPY97] or Prism [KNP11] are widely used.

At the beginning, model checking techniques were applicable to systems with few states. However, considering systems composed of many sub-systems, the number of states grows rapidly making the model checking inefficient. This issue is known as the state space explosion problem. To tackle this problem, many techniques were developed such as symbolic data representation, partial order reduction or abstractions. An other development for model checking was to consider parameterized systems for which some value is unfixed and considered as a parameter. Analyzing parameterized systems is equivalent to studying a whole family of systems where each of them is obtained by instantiating the value of the parameter. In this thesis, we investigate parameterized verification of networks and we consider the size of the networks as a parameter. Our objective is to develop techniques to verify networks independently of the number of components.

Parameterized systems

Thanks to the advances in hardware and software architectures, distribution and concurrency have become one of the main paradigms when developing large systems. Providing methods to analyze and verify distributed systems is a complex task and this for several reasons. First, there are different families of distributed systems depending

on the communication means (shared memory or message passing), on the computing power of the involved entities, on the knowledge of the system provided to the entities (full knowledge, or local knowledge of their neighbors, or no knowledge at all) and on the type of communication topology (ring, tree, arbitrary graph, etc). Second, most of the protocols developed for distributed systems are supposed to work for an unbounded number of participants; hence in order to verify that a system behaves correctly, one needs to develop methods which allow to deal with such a parameter. Indeed, one can no longer be satisfied by a certification of one component independently of the whole network, or even of a finite number of participants in a network. This motivates the framework of parameterized systems.

In their seminal paper on distributed models with many identical entities [GS92], German and Sistla represent the behavior of a network by finite state machines interacting via 'rendezvous' communications. Variants have then been proposed to handle different communication means, like broadcast communication [EFM99], token-passing [CTTV04, AJKR14], message passing [BGS14] or shared memory [EGM13]. In his survey on parameterized models [Esp14], Esparza shows that minor changes, such as the presence or absence of a controller in the system, can drastically modify the complexity of the verification problems.

Broadcast protocols. Among the various parametric models of networks, broadcast protocols, originally studied by Esparza *et al.* [EFM99], have later been analyzed under a new viewpoint, leading to new insights on the verification problems. Specifically, a low level model to represent the main characteristics of ad-hoc networks has been proposed [DSZ10]. It characterizes the following aspects of such systems: the network is equipped with a communication topology and the nodes in the network can only communicate with their neighbors using broadcast communication. Moreover, the number of participants is unbounded. In this model, each entity behaves similarly, following a protocol which is represented by a finite state machine performing three kinds of actions: (1) broadcast of a message, (2) reception of a message and (3) internal action. Furthermore, the communication topology does not change and no entity is deleted or added during an execution.

The *reachability problem* consists in determining whether there exist an initial number of entities and a communication topology such that it is possible to reach a configuration where at least one process is in a specific control state. The reachability problem thus allows to detect an error in the conception of the protocol by detecting the possibility to reach an error state for some process. The main difficulty in solving such a problem lies in the fact that both the number of processes and the initial communication topology are parameters for which one wishes to prove that the state is not reachable for all instantiations. In [DSZ10], it is proven that the parameterized reachability problem is undecidable in broadcast protocols. The same holds for the *synchronization problem*, which asks whether all processes can converge to a set of target states. For both the reachability and the synchronization problems, decidability can however be regained, by considering communication topologies that can change non-deterministically at any moment [DSTZ12]. Another option to recover decidability of the reachability problem is to restrict the topologies to complete graphs (aka cliques),

or bounded depth graphs [DSZ11a], or acyclic directed graphs [AAR13].

This thesis falls within this framework. We study parameterized verification problems in order to verify a network independently of the number of components, without enumerating all possible sizes of the network.

Quantitative aspects

In addition to concurrency, quantitative analysis of systems is really important. Indeed, it may not be enough to say that a property is satisfied or not, but to what extent. To this aim probabilistic models have been studied. They allow to model unknown behaviors, typically the environment interaction with the system is modeled by stochastic actions. With probabilistic models it is possible to express how likely a property is satisfied. Probabilistic models are also of interest for distributed algorithms. Indeed, randomized algorithms are widely used since they allow to break the symmetry of components for example in Lehmann-Rabin's randomized solution to the well-known problem of the dining philosophers [RL94] or in the CSMA algorithms used for contention resolution in computer networks (see *e.g.* the standard ZigBee [Spe]).

For finite state systems with non-determinism and probabilities (like finite state Markov Decision Processes), most verification problems are decidable [BK08], but when the number of states is infinite, they are much harder to tackle. A general framework for purely probabilistic infinite state transition systems has been proposed in [AHM07]. However, it seems hard to adapt such a framework to the case with probabilistic and non-deterministic choices. Indeed, the introduction of probabilities might even lead to the undecidability of problems that are decidable in the non-probabilistic case. For instance for extensions of pushdown systems with non-deterministic and probabilistic choices, the model checking problems for linear time or branching time logics are undecidable [EY05, BKS05]. On the other hand, it is not always the case that the introduction of probabilistic transitions leads to undecidability, but then dedicated verification methods have to be developed, as it is the case for instance for non-deterministic probabilistic lossy channel systems [BBS07].

An additional aspect of nowadays systems is timed behaviors. Computerized systems are used for critical applications that require timing constraints. One not only requires that the system answers but that the answer is given in a certain delay. Timed automata, introduced in [AD90, AD94], are finite state automata equipped with dense time or discrete time clock variables. These automata have been widely investigated and most verification problems are known to be decidable. However, when turning towards infinite state systems with dense time clock variables, the problems become more difficult. Indeed, parameterized verification of networks composed of an unbounded number of processes equipped with clocks has been studied [AJ03, ADM04, ADR$^+$11] and the parameterized verification problems are undecidable when the processes have more than one clock.

So far, up to our knowledge, the automated verification of timed probabilistic protocols has only been performed for a fixed, and rather small, number of nodes, see *e.g.* [Fru06]. Indeed, the classical approach is to build the network as the product of the participating protocols, and then to run a model checker on this product. However, this method faces the state space explosion problem due to the product of several components. In comparison, in this thesis we develop new techniques that allow to verify probabilistic networks, independently of their size.

Determining the frontier between decidability and undecidability In addition to the practical use of this work allowing to model distributed systems combining time and probabilities as well as unbounded number of participants, this work is really interesting on a theoretical point of view. Indeed, parameterized systems are really powerful and this leads to undecidability of verification problems. Many restrictions have been studied to be able to perform model checking on parameterized models. This thesis thus investigates the frontier of decidability and undecidability of the verification problems for probabilistic parameterized problems. Moreover, even when the problems are decidable it often turns out that they are non-tractable because of high complexity. We thus investigate restrictions on the communication topology in order to obtain tractable solutions.

Organization and contributions of the thesis

This thesis is organized as follows.

Chapter I The first chapter is dedicated to the introduction of the basic models used in the thesis. In particular, we introduce timed automaton, Markov chains and Markov decision processes as well as some infinite state transition systems. For each of these models, some key results or techniques are presented.

Chapter II This chapter is dedicated to existing work on parameterized verification. This overview of the state of the art shows the existing models used in the literature for modeling systems built on a large number of components. In particular, the attention is focused on broadcast protocol networks which are the basis on which this work is built. The new models we introduce in this thesis are all variations of broadcast protocols.

We then come to chapters dedicated to the technical contributions of this thesis.

Chapter III The first contribution presented in this thesis is the introduction of a new model, namely networks of probabilistic timed protocol, that combines three aspects: unknown number of components, probabilities and time. In this model, an unbounded number of components are set in a network, each of these components runs the same protocol described by a finite state machine that combines probabilistic actions as well as timing constraints. Moreover, the components are allowed to communicate with each other via broadcasts of a finite number of messages. Although this model

is really interesting by the fact that it is an infinite system combining probabilities, non-determinism and time, it turns out that the basic verification problems, such as the parameterized reachability problem, are undecidable.

Only few infinite models combine probabilities and non-determinism, one of them are channel systems. In this model, a finite state machine can write and read messages in an unbounded fifo channel. It was shown that the verification problems are undecidable for channel systems [BZ83]. However, allowing faults in the system via loss of messages leads to decidability [AJ93]. In particular, considering probabilistic lossy channel system, where at each step each message has an independent probability to get lost, the verification problems are decidable. Inspired by these works, in order to represent the notion of mobility in the network, we further introduce *dynamic* probabilistic timed networks, where processes can disappear and be created, according to fixed probability laws.

Given a (static or dynamic) probabilistic timed network, we consider relevant *parameterized verification problems*, such as the following. For a distinguished state of the process model, can a configuration with some process in this particular state be reached almost surely, for any initial number of processes and for every scheduling policy? Equivalently, the problem is whether, independently of the number of processes, the minimum probability to reach a target configuration is 1. An other question is whether, for a set of target states, the configurations with all processes gathered in this target set can be reached almost surely? Beyond these particular problems, we consider all variants where the minimum or maximum probabilities are compared to the thresholds between 0 or 1.

Even though the parameterized reachability problem has been shown to be decidable for clique networks [DSZ11a], in static probabilistic timed networks most of the parameterized verification problems are undecidable. This is shown via reduction of the halting problem for two-counter machines, which is known undecidable [Min67]. The remaining cases are decidable, either by reduction to the non-probabilistic case or thanks to a monotonicity result, stating that it is enough to consider networks composed of a unique process in order to answer the problem.

For dynamic probabilistic timed networks, we show that one can abstract the value of the clock by considering regions, which are an adaption of the region abstraction in timed automata but with an unbounded number of clocks. Moreover, we can equip these regions with a well quasi order, and show that the abstracted network enjoys the finite attractor property. Thus we can adapt the technique developed for non-deterministic lossy channel systems in order to solve the qualitative problems. These results were published in [BF13].

New in this thesis, we also investigate quantitative parameterized problems. Instead of comparing the value of probability only to qualitative thresholds (0 or 1), we allow to compare to any bound between 0 and 1. Although the quantitative parameterized problems are still open, we obtain a nice result showing the decidability of the approximation of the minimal probability to reach a state, for a given fixed initial size of network. Notice that, even if the initial size is fixed, there can be an unbounded number of participants in the network, due to probabilistic creation of processes. This result is

interesting since it is inspired by the approximation scheme in fully probabilistic lossy channel systems, however such an approximation was missing for non-deterministic and probabilistic infinite state systems.

Chapter IV In this chapter, we present a new model named *selective broadcast networks of probabilistic protocols*. This model extends the model of reconfigurable broadcast networks studied in [DSTZ12] by allowing probabilistic internal actions, that is, a process can change its internal state according to a probabilistic distribution. The main difference with the clique networks studied in Chapter III is that the broadcast no longer reaches all processes, but a subset of processes chosen non-deterministically. Whereas the semantics of reconfigurable broadcast networks was given in terms of an infinite state system with non-determinism, due to the different possibilities of sending messages from different nodes and also to the non-determinism of the protocol itself, we obtain here an infinite state system with probabilistic and non-deterministic choices.

In selective broadcast networks, a broadcast reaches a subset of processes. This allows to model peer to peer applications in networks equipped with a switch that chooses to whom the message is intended. This is in opposition to clique networks that are closer to networks equipped with a hub that redistributes all the messages to every one. Selective broadcasts can also be useful to model wireless systems with high mobility. For example, in a flock of birds all equipped with a wireless sensor, the reception of a message broadcast by a sensor is limited to the sensors on the birds near the broadcaster. Moreover, due to the movement of the birds, the set of birds within reach between two communications can be totally different, from the whole flock if all the birds are close together, to no one if the bird sending the message took its distance with the flock.

In this chapter, we study the probabilistic versions of the *state reachability problem* that asks whether there exists a strategy resolving non-determinism that allows to reach a configuration in which at least one process is in a specific state. We focus on the qualitative variants of this problem by comparing probabilities only with 0 and 1. This problem is relevant since it allows to check whether the protocol is well designed, for example by checking whether a bad state can be reached with positive probability. In addition to the existence of a strategy, we also study the problem asking whether all the strategies meet the probability threshold. This allows to check, for example, that whatever the non-deterministic choices, the protocol avoids a bad state almost surely.

In the previous chapter (Chapter III), we have seen that clique networks of timed probabilistic protocols are a powerful model leading to undecidability of the qualitative parameterized reachability problems. However, taking inspiration from a well known infinite state model combining non-determinism and probabilities, namely lossy channel systems, we regain decidability when considering networks with probabilistic disappearance and creation of processes *i.e.* networks for which the size evolves randomly over time. However, the decidability comes at the price of high complexity. In order to achieve better complexity bounds, this chapter is devoted to the study of selective broadcast networks. Indeed, the complexity for the reachability problem is much lower (PTIME) in reconfigurable broadcast networks. Moreover, instead of applying well

quasi order which have a high complexity, in this chapter we adapt a known reduction (see for instance [CdAFL09]) for verification of finite probabilistic models, which consists in translating a Markov decision process into a 2-player game, for which the existence of a winning strategy is equivalent to the reachability questions in the MDP. However, due to the unknown number of participants in the parameterized verification problem, we have an infinite model, hence we cannot apply directly this solution.

In order to solve this problem, we introduce *selective broadcast of parity protocols*, which are networks composed of an unknown number of processes that all run the same finite state parity protocol. The novelty is that there are now two players and a parity objective. Since the second player is introduced in order to simulate probabilistic choices, it is less powerful than player 1. Indeed, in selective broadcast of parity protocols, the first player is the one deciding which process plays, which processes receive the messages and the action to play when the selected process is in a state belonging to player 1. However, the second player can only choose the action to play when the first player has chosen a process in a state belonging to player 2. In this setting, we show that one can decide in co-NP whether player 1 has a strategy that fulfills a parity objective for any choices made by player 2. Moreover, we provide a translation, at the protocol level, from the parameterized qualitative reachability problems in selective broadcast networks of probabilistic protocols to the parameterized game problem in selective broadcast networks of parity protocols.

In addition to their usefulness to solve the qualitative probabilistic problems, we believe that selective broadcast of parity protocols are an interesting tool on their own. Indeed, this model of a distributed game, as well as the techniques used to solve the game, are really interesting and could be of use in different settings of the many identical processes field. The results presented in this chapter were published in [BFS14].

Chapter V The last chapter presenting contributions of this thesis is based on the following remark: the solutions given for parameterized broadcast protocols all rely on centralized strategies to choose the actions of the processes with a full knowledge of the network. Due to the non-determinism in the description of the protocol, it may happen that two processes behave differently, even if they have the same information on what has happened so far in an execution. To forbid such non-truly distributed behaviors, in Chapter V, we constrain processes to take the same decisions in case they fired the same sequence of transitions so far.

Here, we consider networks composed of an arbitrary number of components, or processes, all running the same protocol. These protocols are finite state machines with three kinds of transitions: internal actions that affect only the process performing it, broadcast that send a message to the other processes, and receptions that allow processes to receive the messages sent by broadcast. In this setting, the past of a process is the sequence of transitions it has taken so far. The *local strategies* ensure that processes with the same past take the same decision. As an example, from the initial configuration, hence with an empty past, if a process chooses to perform an internal action, then all the processes with an empty past should perform the same internal action, unless they get additional information by receiving a message.

We study the parameterized reachability and synchronization problems in broadcast protocol networks restricted to *local strategies*. The first problem asks whether there exist a network size and a local strategy such that a configuration with at least one process in a given state is reached. The latter asks whether a configuration with all processes gathered in a given set of states is reached. We consider two different settings for the networks: first, reconfigurable broadcast networks for which the messages reach only a subset of the processes chosen non-deterministically. In these networks, without the assumption of local strategies, the parameterized reachability problem is known to be PTIME [DSTZ12], as well as the synchronization problem. The second class networks are clique broadcast networks, in which the messages always reach all processes.

We show that the reachability and synchronization problems under local strategies in reconfigurable broadcast networks are NP-complete. To obtain the upper bound, we prove that local strategies can be succinctly represented by finite trees that we call strategy patterns. Intuitively, strategy patterns are finite unfoldings of the protocol with the additional assumption that, from each node, only one active action is present, thus they represent the choices of the local strategy. Moreover, to answer the reachability and synchronization problems, we show that we can equip the strategy patterns with an order representing the order in which an execution can visit the nodes of the pattern. Finally, we obtain an NP algorithm by showing that we can focus on polynomial size strategy patterns in order to solve the parameterized problems.

For clique networks, with no surprise, the problems are harder since the parameterized reachability problem was already shown to be non-primitive recursive, without the locality assumption. We show that the problem is in fact undecidable when restricting to local strategies. The proof is based on the simulation of two-counter machines for which the termination is known to be undecidable [Min67]. The proof relies on the fact that some processes can wait in some states without any receptions, hence without getting any new information. With these processes, and thanks to the locality assumption, we can run exactly the same simulation of the two-counter machine a second time to check that it was correct. This property is a key for undecidability. Indeed, with the additional assumption that every message can be received in every state, hence that all communications bring information to all processes in the network, we can show that the reachability problem is decidable. The decidability proof is based on a counting abstraction on which we can show that it is a well structured transition system and thus that the reachability problem is decidable [ACJT96, AJ01, FS01]. These results were published in [BFS15].

Interestingly, the notably difficult distributed controller synthesis problem [PR90] is relatively close to the problem of existence of a local strategy. Indeed, a local strategy corresponds to a local controller for the processes executing the protocol, and whose role is to resolve the non-deterministic choices. Local strategies are of interest to implement distributed algorithms. Indeed, non-determinism is hard to implement in real life, thus one can see the local strategy as an implementable deterministic version of a non-deterministic specification for a distributed algorithm.

We conclude this thesis by listing some possible extensions of our work.

Personal publications

[BF13] Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *FSTTCS*, volume 13, pages 501–513, 2013.

[BFS14] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *Foundations of Software Science and Computation Structures*, pages 134–148. Springer, 2014.

[BFS15] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Distributed local strategies in broadcast networks. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4*, 2015, pages 44–57, 2015.

Chapter I

Preliminaries

Several models have been proposed in the literature in order to represent complex systems. Typically, one can be interested in modeling time, randomness, and unbounded data structures. We review here models and results for some quantitative or infinite systems that we will use in the rest of the document.

After introducing some preliminary notations, we present timed automata that are finite state machines equipped with clock variables that range over positive reals. Timed automata allow to model systems with timing constraints. The dense time clock variables give rise to infinite state systems, however abstraction techniques have been proposed in order to verify some properties on these systems.

Then we present the quantitative model of Markov chains that are state machines for which the transition relation is given by probabilistic distributions. Markov chains allow to model randomness of events, typically the actions of the environment. Moreover, in Markov chains it is possible to quantify over the set of executions and thus to express, *e.g.* that a property is almost surely satisfied. In addition to probabilities, it may be useful to have some non-determinism in the model. Indeed, the non-determinism allows to model controllable events. For example to model a faulty communicating machine, whether message a or message b is sent is done thanks to non-determinism, but the loss of messages is uncontrollable and modeled by probabilistic distributions. We thus introduce Markov decision processes that combine both probabilities and non-determinism. The probabilities in Markov decision processes can be seen as a random adversary to the scheduler that resolves the non-determinism. We then introduce 2-player games, where the adversary is given full power and not only fixed probabilistic distributions.

More general classes of infinite transition systems have been proposed for which some verification questions are decidable, such as well-structured transition systems for which the monotonicity of the transition relation with respect to a well-quasi order allows to run a backward analysis to decide coverability. Finally, we present results for two particular well structured transition systems: lossy channel systems and vector addition systems with states. We also review the undecidability results for two-counter machines.

1 Basic definitions

We use $\mathbb{R}^+$ to denote the set of non-negative real numbers. For two reals $a, b \in \mathbb{R}^+$, we denote by $[a, b] = \{x \mid a \leq x \leq b\}$ the set of all reals between a and b.

A *probability distribution* over a countable set E is a function $d : E \to [0, 1]$, such that $\sum_{x \in E} d(x) = 1$. We denote by $Dist\,(E)$ the set of all probability distributions over E.

For an arbitrary set E, we write $\mathbb{M}(E)$ for the set of finite basis *multisets* over E, *i.e.* the set of multiplicity functions $M : E \to \mathbb{N}$ such that the set $\{x \in E \mid M(x) > 0\}$ is finite. We also use an alternative notation for multisets, for example $M = \langle x, x, x, y\rangle$ denotes the multiset defined by $M(x) = 3$, $M(y) = 1$ and $\forall z \in E \setminus \{x, y\}, M(z) = 0$. We now introduce simple operations on multisets. For $M \in \mathbb{M}(E)$ and $x \in E$ we write $M + x$ for the multiset M' defined by $M'(x) = M(x) + 1$ and $M'(y) = M(y)$ for $y \neq x$. Symmetrically, and assuming $M(x) > 0$, $M - x$ is a notation for M' such that $M' + x = M$. Given an integer $n \in \mathbb{N}$, we denote by $\mathbb{M}^n(E)$ the set of multisets with n elements *i.e* the set of multisets $M \in \mathbb{M}(E)$ such that $\sum_{x \in E} M(x) = n$. Notice that $\mathbb{M}(E) = \bigcup_{n \in \mathbb{N}} \mathbb{M}^n(E)$.

We now introduce transition systems which are used to describe the potential behaviors of discrete systems.

Definition 1.1 (Transition systems) *A* transition system *is a tuple* $\mathcal{TS} = (\mathrm{S}, \mathsf{s}_0, \to)$ *where:*

- S *is a set of states,*
- $\mathsf{s}_0 \in \mathrm{S}$ *is the initial state, and*
- $\to \subseteq \mathrm{S} \times \mathrm{S}$ *is the transition relation.*

The transition system $\mathcal{TS}$ is said to be finite if S is finite, and infinite otherwise.

2 Timed automata

We now introduce timed automata that are infinite state systems due to clock variables that range over the non-negative reals. During an execution, clock values all increase at the same speed. Timed automata are useful to model timing constraints since the values of clocks can be compared to integers to enable or disable transitions. Timed automata can be used to model and analyze the timing behavior of computer systems, *e.g.*, real-time systems or networks.

Given a finite set of real-valued clocks C, we denote by $\mathcal{G}(\mathrm{C})$ the set of guards composed of conjunctions of constraints of the form $\mathbf{c} \sim n$ where $\mathbf{c} \in \mathrm{C}$, $n \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$. Moreover, $Up(\mathrm{C})$ denotes the set of updates of the clocks. An update $up : \mathrm{C} \to \{\emptyset, \mathbf{c} := 0\}$ assigns to each clock an update action that either leaves the clock unchanged ($up(\mathbf{c}) = \emptyset$) or resets it to zero ($up(\mathbf{c}) = \mathbf{c} := 0$).

We now give the definition of timed automata introduced in [AD90, AD94].

Definition 2.1 (Timed automata) *A* timed automaton *is a tuple* $\mathcal{A} = (\mathrm{L}, \ell_0, \mathrm{C}, \Sigma, \Delta)$ *where:*

- L *is a finite set of locations;*
- $\ell_0 \in \mathrm{L}$ *is the initial location;*
- C *is a finite set of real valued clock variables;*
- Σ *is a finite alphabet, and*
- $\Delta \subseteq \mathrm{L} \times \mathcal{G}(\mathrm{C}) \times \Sigma \times Up(\mathrm{C}) \times \mathrm{L}$ *is a transition relation.*

Semantics In order to define the semantics of timed automata, we introduce clock valuations which are assignments of the clocks to real values. Precisely, a valuation is a mapping $v : \mathrm{C} \to \mathbb{R}^+$ representing that the clock $\mathbf{c} \in \mathrm{C}$ has the real value $v(\mathbf{c})$. Given a real $d \in \mathbb{R}^+$, we denote by $v + d$ the valuation v' in which all the clock values are increased by d *i.e.* for every clock $\mathbf{c} \in \mathrm{C}$, $v'(\mathbf{c}) = v(\mathbf{c}) + d$. Given a guard $g \in \mathcal{G}(\mathrm{C})$, we say that a valuation v satisfies the guard if and only if, for every constraint $\mathbf{c} \sim n$ of the guard g, we have $v(\mathbf{c}) \sim n$. Given an update up and a valuation v, we define the effect of the update on the valuation as the valuation v' in which all the clocks reset by the update are set to zero. Formally, $up(v) = v'$ where for every clock $\mathbf{c} \in \mathrm{C}$ if $up(\mathbf{c}) = \mathbf{c} := 0$ then $v'(\mathbf{c}) = 0$ and otherwise $v'(\mathbf{c}) = v(\mathbf{c})$.

The semantics of a timed automaton is given as a transition system $\mathcal{T} = (\mathrm{S}, (\ell_0, \mathbf{0}), \to)$ where the states are pairs $\mathsf{s} = (\ell, v)$ from $\mathrm{S} = \mathrm{L} \times \mathbb{R}^{+\mathrm{C}}$ and such that:

- $(\ell, v) \xrightarrow{d} (\ell, v + d)$ for every $d \in \mathbb{R}^+$, and
- $(\ell, v) \xrightarrow{a} (\ell', v')$ for every $a \in \Sigma$ such that there exists $(\ell, g, a, up, \ell') \in \Delta$ with v satisfying the guard g, and $v' = up(v)$.

Notice that the transition system $\mathcal{T}$ has infinitely many states due to the valuations taking values in $\mathbb{R}^+$. However one can abstract valuations into equivalence classes called regions in order to obtain a finite abstraction.

Region abstraction The region abstraction, introduced in the seminal paper [AD94], forms a finite partition of the valuations over C. This abstraction is based on the observation that, for reachability problems, the relevant information in clock valuations consists of the integer part of each clock (up to the maximal constant appearing in guards) and the ordering of their fractional parts.

Given $x \in \mathbb{R}^+$ a non-negative real, we denote by $\lfloor x \rfloor$ its integer part and $\{x\}$ its fractional part. Note that $x = \lfloor x \rfloor + \{x\}$.

Definition 2.2 (Region equivalence) *Let* $b \in \mathbb{N}$ *and let* $\mathsf{s}_1 = (\ell_1, v_1)$ *and* $\mathsf{s}_2 = (\ell_2, v_2)$ *be two states.*

The states s_1 *and* s_2 *are* b-region equivalent*, denoted* $\mathsf{s}_1 \approx_b \mathsf{s}_2$ *if:*

(i) $\ell_1 = \ell_2$*: the locations match,*

(ii) *for every clock* $\mathbf{c} \in \mathrm{C}$, $(\lfloor v_1(\mathbf{c}) \rfloor \leq b) \vee (\lfloor v_2(\mathbf{c}) \rfloor \leq b) \Rightarrow \lfloor v_1(\mathbf{c}) \rfloor = \lfloor v_2(\mathbf{c}) \rfloor$*: integer parts of clocks agree up to* b*,*

(iii) *for every clock* $\mathbf{c} \in \mathrm{C}$, $(\{v_1(\mathbf{c})\} = 0) \Leftrightarrow (\{v_2(\mathbf{c})\} = 0)$*: the set of clocks with integer values are equal,*

(iv) *for* $\sim \in \{<, =, >\}$*, and for every pair of clocks* $\mathbf{c}, \mathbf{c}'$, $(\{v_1(\mathbf{c})\} \sim \{v_1(\mathbf{c}')\}) \Leftrightarrow (\{v_2(\mathbf{c})\} \sim \{v_2(\mathbf{c}')\})$*: the orderings of fractional parts in* v_1 *and* v_2 *coincide.*

Notice that the region equivalence is indexed by a bound b. For a given timed automaton, this bound is set to the maximal constant appearing in guards, we will thus omit this bound in the sequel. Given a state $\mathsf{s} \in \mathrm{S}$, $[\mathsf{s}]$ denotes its equivalence class for $\approx$, and is called a *region*. It is shown in [AD94] that one can quotient $\mathcal{T}$ by the relation $\approx$ to obtain a finite automaton called region automaton and denoted $[\mathcal{T}]$. Moreover, for any execution in a timed automaton there exists an execution in the region automaton that consists in abstracting the value of the clocks and the delays. Reciprocally, for any execution in the region automaton, one can concretize the delays in order to obtain an execution in the timed automaton. This shows, in particular, that to decide the location reachability in the infinite transition system $\mathcal{T}$, one can check for reachability of a region containing the goal location in the region abstraction $[\mathcal{T}]$.

3 Markov chains, Markov decision processes and games

Model checking techniques focus on certifying correctness of the system for all possible behaviors. However, systems are subject to random phenomena such as message losses or failures of components, thus the non-deterministic approach is too pessimistic and does not take into account how likely these events happen. In these settings, probabilistic models have been studied in order, not to absolutely guarantee that a property is satisfied, but to what extent the property holds. With these models it is possible to express how likely a property is satisfied. In this section, we present Markov chains which are a fully probabilistic model. The successor of a state is given by a fixed probability distribution. In addition to the probabilistic behaviors it may be useful to have non-determinism in order to represent non-random events *e.g.* whether a controller switches on or off a component is model by non-determinism and whether a component fails by a probabilistic distribution. We thus present Markov decision processes that combine non-determinism and probabilities. An other way to model the environment is to see it as purely random, see it as a non-deterministic adversary. This is the case for 2-player games, that give the environment full power and not just probabilistic distributions. We introduce this model at the end of this section.

3.1 Markov chains

We consider homogeneous discrete time Markov chains with a countable state space. These are Markov chains in which the probability to go from one state to another,

representing a step, is given by a fixed probabilistic distribution.

Definition 3.1 (Markov chains) *A* Markov chain *is a tuple* $\mathcal{M} = (\mathrm{S}, \mathsf{s}_0, Prob)$ *where:*

- S *is a countable set of states,*
- $\mathsf{s}_0 \in \mathrm{S}$ *is the initial state, and*
- $Prob : \mathrm{S} \to Dist\,(\mathrm{S})$ *is the transition probability function that assigns to each state* $\mathsf{s} \in \mathrm{S}$ *a probability distribution over* S*.*

A path in a Markov chain $\mathcal{M} = (\mathrm{S}, \mathsf{s}_0, Prob)$ is a finite or infinite sequence $\mathsf{s}_0\mathsf{s}_1 \ldots$ of states in S starting with the initial state s_0. The set of all finite paths is denoted $Paths_*$ and the set of all infinite paths is denoted $Paths_\omega$. Markov chains naturally induce a probability measure on their set of infinite paths. We use the classical probability space (see [KSK66]) $\{\Omega, \Delta, \mathbb{P}\}$ on infinite paths, which is defined as follows:

- $\Omega = \{\mathsf{s}_0\mathsf{s}_1 \cdots \mid \mathsf{s}_i \in \mathrm{S}\} = Paths_\omega$ is the set of infinite paths,
- Δ is the smallest σ-algebra that contains all cylinder sets $Cyl(\mathsf{s}_0\mathsf{s}_1 \ldots \mathsf{s}_n) = \{\mathsf{s}_0\mathsf{s}_1 \ldots \mathsf{s}_n \ldots\}$
- $\mathbb{P}$ is the probability measure defined by $\mathbb{P}(Cyl(\mathsf{s}_0\mathsf{s}_1 \ldots \mathsf{s}_n)) = \prod_{i=1}^{n} Prob(\mathsf{s}_{i-1}, \mathsf{s}_i)$, and which is extended to Δ by Carathéodory's extension theorem.

It was shown in [Var85] that the set of paths eventually reaching a state s, denoted $\Diamond\mathsf{s}$, is measurable. Given a Markov chain $\mathcal{M}$ and a state s we thus denote $\mathbb{P}(\mathcal{M} \models \Diamond\mathsf{s})$ the measure of the set of infinite paths reaching s.

3.2 Markov decision processes

A natural extension of Markov chains is Markov decision processes. In addition to the transition probability function, Markov decision processes allow for some non-determinism. The set of states is thus partitioned into probabilistic states, from which the successor is given as in Markov chains by the transition probability function, and non-deterministic states, from which the successor is defined by the non-deterministic transition relation.

Definition 3.2 (Markov decision processes) *A* Markov decision process *is a tuple* $\mathcal{M} = (\mathrm{S}, \mathrm{S}^{(p)}, \mathrm{S}^{(n)}, \mathrm{S}_0, T, Prob)$ *where*

- S *is a countable set of states,*
- $\mathrm{S}^{(p)}$ *and* $\mathrm{S}^{(n)}$ *partition* S *in respectively,* probabilistic states *and* non-deterministic states*,*
- $\mathrm{S}_0 \subseteq \mathrm{S}$ *is the set of initial states,*

- $T \subseteq \mathrm{S}^{(n)} \times \mathrm{S}$ *is the* non-deterministic transition relation*, and*
- $\mathit{Prob} : \mathrm{S}^{(p)} \to \mathit{Dist}\,(\mathrm{S})$ *is the* transition probability function *that assigns to each state* $\mathsf{s} \in \mathrm{S}^{(p)}$ *a probability distribution over* S*.*

One cannot directly define a probability space on a Markov decision process due to the non-determinism. However, once the non-deterministic choices are fixed we obtain a Markov chain for which we can define a probability space. We thus define the schedulers which are in charge of resolving the non-determinism, they associate a transition with every finite path ending in a non-deterministic state.

Definition 3.3 (Schedulers) *A* scheduler *is a function* $\sigma : \mathit{Paths}_* \to T$ *such that for every finite path* $\mathsf{s}_0\mathsf{s}_1 \dots \mathsf{s}_n$ *with* $\mathsf{s}_n \in \mathrm{S}^{(n)}$*,* $\sigma(\mathsf{s}_0\mathsf{s}_1 \dots \mathsf{s}_n) \in (\{\mathsf{s}_n\} \times \mathrm{S}) \cap T$*.*

A Markov decision process $\mathcal{M}$ together with a scheduler σ give rise to a Markov chain $\mathcal{M}_\sigma$. Indeed, the non-determinism being resolved by the scheduler, one can consider the Markov chain $\mathcal{M}_\sigma$, that has for state space the set of finite paths in $\mathcal{M}$, the transition probability function being naturally lifted from $\mathrm{S}^{(p)}$ for paths ending in probabilistic states and reflecting the choices of the scheduler for paths ending in non-deterministic states.

An interesting subclass of schedulers are *memoryless schedulers* which are scheduler for whose the decisions do not depend on the whole execution but only on the current state.

Definition 3.4 (Memoryless schedulers) *A scheduler* σ *is* memoryless *if for every pair of finite paths* $\rho = \mathsf{s}_0 \dots \mathsf{s}_n$ *and* $\rho' = \mathsf{s}'_0 \dots \mathsf{s}'_m$*, if* $\mathsf{s}_n = \mathsf{s}'_m$ *then* $\sigma(\rho) = \sigma(\rho')$*.*

Notice that the state space of the Markov chain induced by memoryless schedulers can be restricted to S. Indeed, the decision of the scheduler only depends on the state and not on the finite path.

Given a scheduler σ and a state s, the probability to reach the state s is defined as the probability to reach a path ending in s in $\mathcal{M}_\sigma$ and is denoted $\mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$. The reachability problems for Markov decision processes amount to establishing the best lower or upper probability bounds for reaching a state, when ranging over all schedulers. Formally, it amounts to computing $\sup_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$ and $\inf_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$.

For finite state Markov decision processes, it is well known that all quantitative reachability problems are decidable in PTIME [KNPS08, BK08]. Moreover, an interesting result is that the extremal probabilities are attained by memoryless schedulers. One can thus *e.g.* compute $\max_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$ together with an optimal memoryless scheduler. This is done by linear programming.

Theorem 3.1 *Given a finite-state Markov decision process* $\mathcal{M}$ *and a state* s

- *computing* $\inf_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$ *and* $\sup_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$ *can be done in polynomial time.*
- *there exists a memoryless scheduler* σ *such that* $\mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s}) = \inf_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$*, and similarly for* $\sup_\sigma \mathbb{P}_\sigma(\mathcal{M} \models \Diamond\mathsf{s})$*.*

One can see a Markov decision process as a $1-\frac{1}{2}$ player game. Indeed the first player is in charge of resolving the non-determinism and thus plays in the non-deterministic states, whereas the second player is purely random and chooses the outcome of probabilistic choices. Notice that in these games the role of the second player is really limited since its decision can only be taken according to fixed probability distributions.

3.3 2-player games

We introduce now 2-player games in which, contrary to Markov decision processes, both players have the same decision power.

Definition 3.5 (2-player games) *A* 2-player game *is a tuple* $\mathcal{G} = (\mathrm{S}, \mathrm{S}^{(1)}, \mathrm{S}^{(2)}, \mathrm{T}, col)$ *where:*

- S *is a countable set of states, partitioned into* $\mathrm{S}^{(1)}$ *the states of player* 1*, and* $\mathrm{S}^{(2)}$ *the states of player* 2*;*
- $\mathsf{s}_0 \in \mathrm{S}$ *is the initial state;*
- $\mathrm{T} \subseteq \mathrm{S} \times \mathrm{S}$ *is the transition relation;*
- $col : \mathrm{T} \to \mathbb{N}$ *is the parity function.*

A play ρ in a 2-player game is an infinite sequence of states $\mathsf{s}_0\mathsf{s}_1 \dots$ such that for every $i \in \mathbb{N}$, $(\mathsf{s}_i, \mathsf{s}_{i+1}) \in \mathrm{T}$. The parity associated with a play is the maximal parity seen infinitely often. Formally, the parity of play $\rho = \mathsf{s}_0, \mathsf{s}_1, \dots$ is defined as $col(\rho) = \max\{c \in \mathbb{N} \mid \forall i, \exists j \geq i, col(\mathsf{s}_j, \mathsf{s}_{j+1}) = c\}$. A play is said to be winning for player 1 if its associated parity is even. Otherwise it is winning for player 2.

Similarly to schedulers for Markov decision processes, we define the notion of strategies that, given a finite prefix of play, called finite run, dictate the choice of the players.

Definition 3.6 (Strategies) *A* strategy α *for player* 1 *is a function that associates with every finite run* $\rho = \mathsf{s}_0, \mathsf{s}_1, \dots, \mathsf{s}_n$*, with* $\mathsf{s}_n \in \mathrm{S}^{(1)}$ *a transition* $(\mathsf{s}_n, \mathsf{s}_{n+1}) \in \mathrm{T}$*.*

Symmetrically, a strategy β *for player* 2 *is a function that associates with every finite run* $\rho = \mathsf{s}_0, \mathsf{s}_1, \dots, \mathsf{s}_n$*, with* $\mathsf{s}_n \in \mathrm{S}^{(2)}$ *a transition* $(\mathsf{s}_n, \mathsf{s}_{n+1}) \in \mathrm{T}$*.*

Given a 2-player game $\mathcal{G}$, a pair of strategies (α, β) defines a single play $\rho(\mathcal{G}, \alpha, \beta) = \mathsf{s}_0, \mathsf{s}_1, \dots$ such that for every $n \in \mathbb{N}$ with $\mathsf{s}_n \in \mathrm{S}^{(1)}$ we have $\alpha(\mathsf{s}_0, \mathsf{s}_1, \dots \mathsf{s}_n) = (\mathsf{s}_n, \mathsf{s}_{n+1})$, and otherwise, when $\mathsf{s}_n \in \mathrm{S}^{(2)}$, $\beta(\mathsf{s}_0, \mathsf{s}_1, \dots \mathsf{s}_n) = (\mathsf{s}_n, \mathsf{s}_{n+1})$. Strategy α for player 1 is said to be winning against strategy β for player 2, if the run $\rho(\mathcal{G}, \alpha, \beta)$ is winning for player 1, that is, if $\mathrm{col}(\rho(\mathcal{G}, \alpha, \beta))$ is even. A strategy for player 1 is winning if it is winning against all strategies of player 2, and symmetrically for player 2.

As for Markov decision processes, an interesting result in finite games is that: first the game is determined, in other words, either there is a winning strategy for player 1 or there is one for player 2. Moreover, winning strategies are memoryless. The problem of deciding whether player 1 or player 2 is winning is in NP∩co-NP. Whether it is in PTIME is a long-standing open problem.

4 Infinite transition systems

4.1 Well-structured transition systems

An interesting subclass of infinite transition systems is the one of well-structured transition systems, for which several verification problems can be decided algorithmically. In particular the coverability of a state is decidable. Well-structured transition systems include a large number of well-known infinite-state systems such as Petri nets, lossy channel systems, ...

The intuition behind such systems is that some monotonicity of the system, allows us to consider finite symbolic representations of infinite sets of states. More precisely, a well-quasi ordering on the states allows to represent finitely upward closed sets of states by their minimal elements. The transition relation is monotonic with respect to this order, thus one can simulate the effect of one step on an infinite upward closed set by only computing the effect on the finite set of minimal elements. To decide coverability, one can thus apply a backward analysis, computing iteratively one step predecessors of the upward closure of the goal state. Finally, this symbolic computation is guaranteed to terminate since, at a certain point, newly computed upward sets will be subsumed by older ones.

In order to define such systems we first introduce well-quasi orders. Note that there are several equivalent definitions for such orderings but we introduce here only one.

Definition 4.1 (Well quasi orders) *A* well quasi order $\sqsubseteq$ *over a set* E *is a partial order such that for every infinite sequence* $x_1, x_2, \cdots \in E^\omega$ *there exist two indices* i *and* j *such that* $i < j$ *and* $x_i \sqsubseteq x_j$.

Definition 4.2 (Well structured transition systems)*[ACJT96, AJ01, FS01] A* well structured transition system *is a transition system,* $\mathcal{TS} = (\mathrm{S}, \mathsf{s}_0, \rightarrow)$ *equipped with a well quasi order* $\sqsubseteq$ *such that* $\rightarrow$ *is compatible with* $\sqsubseteq$ i.e. *if* $\mathsf{s}_1 \rightarrow \mathsf{s}_2$ *and* $\mathsf{s}_1 \sqsubseteq \mathsf{s}_3$ *then there exists* s_4 *such that* $\mathsf{s}_2 \sqsubseteq \mathsf{s}_4$ *and* $\mathsf{s}_3 \rightarrow^* \mathsf{s}_4$.

In this definition, the monotonicity of the transition system is defined by $\mathsf{s}_3 \rightarrow^* \mathsf{s}_4$ *i.e.* a step of lower configuration can be simulated from higher configurations in an arbitrary number of steps. There are several variants for this condition, depending on the desired property, such as for example the exact simulation that requires that the step can be simulated in one step, *i.e* that $\mathsf{s}_3 \rightarrow \mathsf{s}_4$. However this assumption does not affect the general algorithm for deciding the coverability problem, which asks, given a configuration s, whether there exists $\mathsf{s} \sqsubseteq \mathsf{s}'$ such that $\mathsf{s}_0 \rightarrow^* \mathsf{s}'$. Under the assumption that the predecessor of an upward-closed set is effectively computable, the algorithm consisting in iteratively computing the set of predecessors of the upward closure of a configuration is ensured to terminate and allows to decide the coverability problem.

Theorem 4.1 ([ACJT96, AJ01, FS01]) *The coverability problem is decidable for well structured transition systems.*

4.2 Lossy channel systems

Lossy channel systems form a class of well structured transition systems. They are finite state machines equipped with an unbounded FIFO channel on which we can write and read messages from a finite alphabet. Moreover, faults in the communication are modeled by the loss of messages in the channel.

Definition 4.3 (Lossy channel systems) *A* lossy channel system *is a tuple* $\mathcal{L} = (\mathrm{Q}, \mathsf{q}_0, \mathrm{M}, \delta)$ *composed of a finite set of states* Q*, an initial state* $\mathsf{q}_0 \in \mathrm{Q}$*, a finite set of messages* M *and a transition relation* $\delta \subseteq \mathrm{Q} \times \{?, !\} \times \mathrm{M} \times \mathrm{Q}$*.*

A configuration is a pair $(\mathsf{q}, w) \in \mathrm{Q} \times \mathrm{M}^*$ composed of the current state q and the current contents of the channel w. The initial configuration is the configuration (q_0, ϵ) composed of the initial state and the empty word. Given two words over M representing channel contents, $w = m_1 \dots m_n$ and $w' = m'_1 \dots m'_m$, w is smaller than w' with respect to the sub-word order, denoted $w \leq_l w'$, if $n \leq m$ and there exists an increasing function f such that for all $i \in [1 \dots n]$, $m_i = m'_{f(i)}$. Transitions between configurations of $\mathcal{L}$ derive from the reading and writing rules in δ. There is a transition from (q, mw) to (q, w') as soon as $w' \leq_l w$ and $(\mathsf{q}, ?, m, \mathsf{q}') \in \delta$. Also there is a transition from (q, w) to (q, w') as soon as $w' \leq_l wm$ and $(\mathsf{q}, !, m, \mathsf{q}') \in \delta$.

Theorem 4.2 ([AJ93, CS08]) *Given a lossy channel system* $\mathcal{L} = (\mathrm{Q}, \mathsf{q}_0, \mathrm{M}, \delta)$ *and a state* $\mathsf{q} \in \mathrm{Q}$*, the problem of whether* q *is reachable from the initial configuration is decidable and* F_{ω^ω}*-complete.*

Probabilistic lossy channel systems A probabilistic variant of lossy channel systems was studied in [ABRS05] where losses are no longer non-deterministic but at each step each message in the channel has an independent probability to be lost. Moreover, a weight function on transitions allows to derive their probability to be fired.

Definition 4.4 (Probabilistic lossy channel systems)*[ABRS05] A* probabilistic lossy channel system *is a tuple* $\mathcal{L} = (\mathrm{Q}, \mathsf{q}_0, \mathrm{M}, \delta, \lambda_-, \mathrm{w})$ *composed of a lossy channel system* $(\mathrm{Q}, \mathsf{q}_0, \mathrm{M}, \delta)$*, a loss rate* λ_-*, and a weight function* $\mathrm{w} : \delta \to \mathbb{N}^+$*.*

Here again, a configuration is a pair $(\mathsf{q}, w) \in \mathrm{Q} \times \mathrm{M}^*$ composed of the current state q and the current contents of the channel w, and the initial configuration is the pair (q_0, ϵ). From a given configuration, the probability to pick an enabled transition is defined as the weight of this transition divided by the sum of the weights of all the possible transitions. The probabilistic distribution ruling the evolution of the channel contents is derived from the loss rate λ_-; each message has an independent probability of λ_- to be lost. The probabilistic transition between configurations is thus defined by the probability to pick a given transition multiplied by the probability to get a given channel contents.

An important property to notice is that, since message losses are independent events, the more messages in the channel, the higher the probability to lose some.

Definition 4.5 (Finite attractor) *A set $E \subseteq \mathrm{S}$ is a finite attractor for a Markov chain $\mathcal{M} = (\mathrm{S}, \mathsf{s}_0, Prob)$ if E is finite and from every state $\mathsf{s} \in \mathrm{S}$, E is almost surely reachable from s,* i.e. $\mathbb{P}((\mathrm{S}, \mathsf{s}, Prob) \models \Diamond E) = 1$.

The finite attractor property states that Markov chains induced by lossy channel systems admit a finite recurrent set [ABRS05]. Indeed, from any configuration, a configuration with no message in the channel will be almost surely reached. Roughly said, some results for finite Markov chains extend to infinite Markov chains with a finite attractor. In particular, qualitative state reachability problems are decidable for probabilistic channel systems.

An other interesting result on probabilistic lossy channel systems is that one can compute, with an arbitrary precision, the probability to reach a state (see [IN97, Rab03, AHM07]). The idea of the proof of this result is to consider the set of executions of length n, and to partition it into three sets: $Reach_n$ the executions that already reached the goal q_f, $Escape_n$ the executions that have not and cannot be extended to reach the goal, and $Undecided_n$ for the remaining executions. It was shown that if the Markov chain has a finite attractor then the probability to be undecided (denoted $p_n^?$) decreases towards 0 as n increases. Moreover, writing $p_n^+ = \mathbb{P}(Reach_n)$ for the probability to have reached the target before n steps, p_n^+ is an under approximation and $p_n^+ + p_n^?$ an over-approximation of $\mathbb{P}(\Diamond \mathsf{q}_f)$. Hence, this gives an algorithm to approximate the probability to reach the goal up to any desired precision.

A more complex model for lossy channel systems is the one of non-deterministic probabilistic lossy channel systems [BS03, BBS07, BS13] which combines non-determinism and probabilities in an infinite state system. The notion of attractor can be extended to Markov decision processes, asking that the finite set is an attractor for any scheduler. One can still achieve decidability of the qualitative control state reachability in non-deterministic probabilistic lossy channel systems. However, for the quantitative problems to our knowledge no approximation scheme is known for this variant.

4.3 Vector addition systems with states

An other example of well structured transition system is the model of vector addition system with states. Vector addition systems with states are finite state machines manipulating unbounded counters that can be incremented and decremented while staying non-negative.

Definition 4.6 (Vector addition system with states (VASS)) *A VASS of dimension n is defined as a tuple $\mathcal{V} = (\mathrm{S}, \mathrm{E})$ where:*

- S *is a finite set of control states;*
- $\mathrm{E} \subseteq \mathrm{S} \times \mathbb{N}^n \times \mathrm{S}$ *is the transition relation.*

A configuration of a VASS, is a pair $(\mathsf{s}, v) \in \mathrm{S} \times \mathbb{N}^n$ composed of a state and an n-tuple of integers. With the VASS $\mathcal{V}$ is associated the transition system $T_{\mathcal{V}} = (\mathrm{S} \times \mathbb{N}^n, \hookrightarrow)$ where

$\hookrightarrow \subseteq (\mathrm{S} \times \mathbb{N}^n) \times (\mathrm{S} \times \mathbb{N}^n)$ is the transition relation, defined as follows: $(\mathsf{s}, v) \hookrightarrow (\mathsf{s}', v')$ if there exists a transition $(\mathsf{s}, \mathbf{v}, \mathsf{s}') \in \mathrm{E}$ such that $v' = v + \mathbf{v}$, where $+$ is the addition component by component. Note that the vectors in configurations of $T_{\mathcal{V}}$ are tuples of naturals, so that a transition can only be fired if the resulting configuration ensures non-negativity of all components. An *infinite run* of $\mathcal{V}$ starting from a configuration (s_0, v_0) is an infinite sequence of the form $(\mathsf{s}_0, v_0) \hookrightarrow (\mathsf{s}_1, v_1) \hookrightarrow (\mathsf{s}_2, v_2) \ldots$. A cycle $(\mathsf{s}, v) \hookrightarrow (\mathsf{s}_1, v_1) \ldots \hookrightarrow (\mathsf{s}, v')$ starting and ending in the same control state is said to have positive effect on each component if $v \leq v'$, *i.e.* for all $i \in [1 \ldots n]$ we have $v[i] \leq v'[i]$.

An interesting result on VASS is recalled in the following theorem. It states that the existence of a cycle with positive effect on each component, is decidable in polynomial time.

Theorem 4.3 ([KS88]) *Given a VASS $\mathcal{V} = (\mathrm{S}, \mathrm{E})$ of dimension n, deciding whether there exist $\mathsf{s} \in \mathrm{S}$ and a vector $v \in \mathbb{N}^n$ such that there exists a cycle $(\mathsf{s}, v) \hookrightarrow^+ (\mathsf{s}, v')$ with $v' \geq v$, is decidable in PTIME.*

The detection of cycles with positive effect is done thanks to linear programming. Since VASS are well structured transition systems, when considering the natural order on n-tuples, positive effect cycles are interesting since they can be iterated. Indeed since the last configuration of the cycle is bigger than the first configuration, one can repeat the sequence of transitions of the cycle, and this as many time as desired.

4.4 two-counter machines

two-counter machines are also finite states machines manipulating counters. However, contrary to VASS, in addition to increments and decrements, one can also test to zero the value of a counter.

Definition 4.7 (Minsky machines [Min67]) *A* Minsky machine *is a tuple $\mathcal{M} = (\mathrm{K}, \mathsf{L}_0, \mathsf{L}_{acc}, c_1, c_2)$ manipulating two integer variables c_1 and c_2 called counters and is composed of a finite set of instructions* K. *Each instruction $\mathsf{L} \in \mathrm{K}$ is either of the form:*

Increment $\mathsf{L} : c_i := c_i + 1;\ \texttt{goto}\ \mathsf{L}'$, *or*

Decrement $\mathsf{L} : \texttt{if}\ c_i = 0\ \texttt{then goto}\ \mathsf{L}'\ \texttt{else}\ c_i := c_i - 1;\ \texttt{goto}\ \mathsf{L}''$

where $i \in \{1, 2\}$ and $\mathsf{L}, \mathsf{L}', \mathsf{L}''$ are labels preceding each instruction. L_0 is the initial label and there is furthermore a special label L_{acc} from which nothing can be done.

In this thesis we will consider *deterministic Minsky machines i.e.* Minsky machines for which the label is unique for each instruction. We will hence use the labels to refer to the instructions.

A configuration is a tuple of $\mathrm{K} \times \mathbb{N} \times \mathbb{N}$and the configuration $\pi = (\mathsf{L}, n_1, n_2)$ means that the machine is at instruction L with counter c_1 with value n_1 and counter c_2 with

value n_2. For a configuration $\pi = (\mathsf{L}, n_1, n_2)$ we write $\mathrm{K}(\pi) = \mathsf{L}$ for the instruction, and $c_i(\pi) = n_i$ for $i \in \{1, 2\}$ the value of counter i in π.

A deterministic Minsky machine $\mathcal{M} = (\mathrm{K}, \mathsf{L}_0, \mathsf{L}_{acc}, c_1, c_2)$ gives rise to a unique run, finite or infinite, which is a sequence of configurations $\pi_0\pi_1\ldots$ such that $\mathrm{K}(\pi_0) = \mathsf{L}_0$, and $c_i(\pi_0) = 0$ for $i \in \{1, 2\}$, and for all j, if $\mathrm{K}(\pi_j) = \mathsf{L}$ is an increment of counter c_1, *i.e.* of the form $\mathsf{L} : c_1 := c_1 + 1;\ \texttt{goto}\ \mathsf{L}'$, then $c_1(\pi_{j+1}) = c_1(\pi_j) + 1$, $c_2(\pi_{j+1}) = c_2(\pi_j)$ and $\mathrm{K}(\pi_{j+1}) = \mathsf{L}'$ (similarly for increment of counter c_2); if $\mathrm{K}(\pi_j) = \mathsf{L}$ is a decrement of counter c_1, *i.e.* of the form $\mathsf{L} : \texttt{if}\ c_1 = 0\ \texttt{then goto}\ \mathsf{L}'\ \texttt{else}\ c_1 := c_1 - 1;\ \texttt{goto}\ \mathsf{L}''$ then if $c_1(\pi_j) = 0$ then $c_i(\pi_{j+1}) = c_i(\pi_j)$ for $i \in \{1, 2\}$ and $\mathrm{K}(\pi_{j+1}) = \mathsf{L}'$, otherwise $c_1(\pi_{j+1}) = c_1(\pi_j) - 1$, $c_2(\pi_{j+1}) = c_2(\pi_j)$ and $\mathrm{K}(\pi_{j+1}) = \mathsf{L}''$ (similarly for decrement of counter c_2);

The halting problem then asks whether or not the unique run of $\mathcal{M}$ reaches L_{acc}; we assume here, without loss of generality, that $\mathcal{M}$ stops in L_{acc} or does not terminate.

A Minsky machine is said to be bounded if there exists a bound $B \in \mathbb{N}$ such that the value of the counters is always smaller than B, *i.e.* $\forall j \in \mathbb{N}, c_1(\pi_j) + c_2(\pi_j) \leq B$. Notice that if the machine halts it is bounded. The boundedness problem asks whether a Minsky machine is bounded.

Theorem 4.4 ([Min67]) *Both the boundedness problem and the halting problem are undecidable for (deterministic) Minsky machines.*

Contrary to VASS which are monotonic, the tests to zero of counters in Minsky machines breaks the monotonicity. Indeed there are transitions that are enabled for configurations where one of the counter has value zero but not for bigger configurations where the counter is not zero. Thus two-counter machines are not well structured transition systems. This difference is crucial since it even leads to undecidability of the reachability problem which was decidable for VASS.

Chapter II

Parameterized verification

Generally speaking, parameterized systems are systems in which some value is unfixed and considered as a parameter. Analyzing parameterized systems is equivalent to studying a whole family of systems where each of them is obtained by instantiating the value of the parameter. In the recent years, an increasing interest has been shown in the verification of such systems for a variety of parameters.

As an example, the parameter can be the value present in guards of timed automata, as in parameterized timed automata [AHV93]. This allows to model number of systems with unknown timing constraints. It also brings some robustness in the verification of the systems by proving correctness for different values of the parameter. Most of the interesting problems are undecidable for parameterized timed automata. However, several restrictions, such as L/U-automata [HRSV02] where the parametric constraints are either used as upper or lower bound but not both, have been proposed for which the parameterized verification is decidable.

An other example is a parameter for the probability of an event. In [Daw05] the authors introduce parametric Markov chains for which the probability of an action is parametric. Numerous works have been published in this field and some tools for parameter synthesis have been developed, such as PROPhESY [DJJ+15].

In general, considering parameterized programs allows to ensure robustness of the verification. Indeed, one can use the parameter as an unknown variable but with known bounds for its possible values. The parameterized verification thus ensures the correctness of the whole family of systems defined by the different instances of the parameter. One can also consider the parameter synthesis problem, that consists in computing the set of parameter values for which the property is satisfied.

In this thesis, we focus on parameterized systems modeling networks, where the parameter is the number of components of the network. We thus present in the next sections the state of the art in parameterized verification for models with many identical processes.

1 Many identical processes

Already in the 80's, Apt and Kozen showed the undecidability of very general parameterized model-checking problems [AK86]. In their seminal paper on distributed models with many identical entities [GS92], German and Sistla represent the behavior of a network by finite state machines interacting via 'rendezvous' communications. Variants have then been proposed to handle different communication means, like broadcast communication [EFM99], token-passing [CTTV04, AJKR14], message passing [BGS14] or shared memory [EGM13]. In his survey on parameterized models [Esp14], Esparza shows that minor changes, such as the presence or absence of a controller in the system, can drastically modify the complexity of the parameterized verification problems. In addition to finite state protocols, some works have been done to investigate infinite-state machines such as [Hag11] where pushdown automata communicate via finite shared memory. Surprisingly, some undecidable problems for non-parameterized systems become decidable in the parameterized setting.

Another model for parameterized systems, combining dense time clock variables and a parametric number of processes has been proposed in [AJ03]. Such networks, called networks of timed processes, are composed of a finite state controller and an arbitrary number of processes equipped each with one real-valued clock. The evolution of the network is described by a finite set of rules that allow the controller and arbitrary set of processes to move simultaneously. The rules may also be conditioned by the values of the clock and may manipulate them by resetting the clocks to zero. It was shown that one can abstract the configurations of such a network into constraints in order to represent finitely the real valued clock variable. Moreover, these constraints can be equipped with a well-quasi order and it was shown that the abstract network is a well structured transition system, hence that the reachability problem is decidable. In continuation with this work, networks where the processes have multiple real-valued clocks have been studied [ADM04]. The reachability problem is undecidable for such networks. Indeed, with two clock variables per process, one can use the clocks to link the processes into a tape. Intuitively, two processes are linked if the first clock of a process shares the value as the second clock of the second process. This construction together with the parametric number of components and the dense valued clocks allow to link arbitrarily many processes in a tape and thus to simulate Turing machines.

More recently, in [AdFE15], the authors consider parameterized networks, called population protocols, under a new point of view. They consider only fair executions, *i.e.* executions for which all the processes execute an action infinitely often. The processes also synchronize via a finite set of rules, however the studied questions differ since the authors do not look for reachability but ask for a consensus of all the processes, *i.e.* they ask whether all the processes can gather in the same state for all fair executions. They say that the protocol computes a predicate if for all initial configurations a consensus is reached. It is shown that the problems of deciding whether a protocol ensures a consensus for all initial configurations and deciding whether it computes a given predicate are both decidable.

Among the various parametric models of networks, broadcast protocols, originally

studied by Esparza *et al.* [EFM99], have later been analyzed under a new viewpoint, leading to new insights on the verification problems. Specifically, a low level model to represent the main characteristics of ad-hoc networks has been proposed [DSZ10]: the network is equipped with a communication topology and processes communicate via broadcast to their neighbors.

2 Ad Hoc networks

Ad Hoc networks introduced in [DSZ10] are networks composed of an arbitrary number of communicating processes, all running the same finite state protocol, distributed over an undirected graph representing the communication topology of the network. Every process behaves similarly following a protocol which is represented by a finite state machine, performing three kinds of actions: (1) broadcast of a message from a finite alphabet, (2) reception of a message and (3) internal action. Furthermore, the communication topology does not change during an execution and no entity is deleted or added during an execution. When a process broadcasts a message, all its neighbors with an enabled reception for this message must take a reception transition.

The *control state reachability problem* consists then in determining whether there exist an initial number of entities and an initial communication topology, such that it is possible to reach a configuration where at least one process is in a specific control state (considered for instance as an error state). The main difficulty in solving such a problem lies in the fact that both the number of processes and the initial communication topology are parameters, for which one wishes to find an instantiation. In [DSZ10], it is proven that this problem is undecidable. Indeed, it is possible to create a gadget that extract a chain of processes in the topology graph. Note that it may not be possible for some topology graphs, in such case the gadget just ends the computation. Once such a chain is extracted, simulation of a 2-counter machine follows easily thanks to the unbounded number of participants.

The shape of the communication topology is crucial here for the undecidability proof. Restricting the set of possible communication topologies to complete graphs (aka cliques) or bounded depth graphs, the reachability problem is decidable [DSZ10, DSZ11a]. Indeed in such graphs, it is no longer possible to design a protocol that will extract an unbounded simple path from the topology, in the first case, because all the messages reach all the components and in the second case, because there are no such path in bounded depth graphs. Moreover, it is possible to equip the ad hoc networks restricted to these topologies with a well-quasi order and to show that they are well-structured transition systems. The class of topologies for which the reachability problem is decidable is even wider since it was shown in [DSZ11a] that decidability extends to bounded path maximal clique graphs, *i.e.* to graphs composed of cliques arranged in such a way that the underlying clique graph is a bounded path graph. An other way to regain decidability is to consider acyclic directed graphs of bounded depth [AAR13].

A timed version of parameterized broadcast networks was studied in [ADR$^+$11]. In this work, the authors augment the protocols with clock variables, representing the

elapse of time, and all growing at the same rate. Interestingly, the topology of the network also plays a role in the decidability status. Indeed, even for bounded path graphs the reachability problem is undecidable with dense time clocks. However, in the case of clique topologies, the problem becomes decidable assuming only one clock per process as for timed networks [AJ03, ADM04].

An other constraint on the communication topology leading to decidability of the parameterized reachability is to consider non-deterministic *reconfiguration* of the communication topology. Reconfigurable networks, are networks in which the nodes can, at any moment, move and change their neighborhood. In [DSTZ12], the parameterized reachability problem is shown to be PTIME-complete. Moreover, the synchronization problem, which asks whether all processes can gather at the same time in a given set of states is shown to be in NP. The key idea is that in parameterized reconfigurable networks, one can use the parametric number of components to double the number of processes and use the reconfiguration of the topology to partition the network in two sub networks. Thus, from an execution reaching a set of states, one can build an execution with twice as much processes in each of these states. This remark allows to design a PTIME algorithm for the reachability problem by building the set of reachable states: one start with the initial state, and adds all the reachable states with internal action, then the reachable states with a broadcast, and finally the reachable states thanks to a reception of a message that can be sent from an accessible state. Repeating these three steps, one obtain a saturation algorithm computing in polynomial time the set of all reachable states.

In fact reconfigurable broadcast networks enjoy the following properties.

Lemma 2.1 ([DSTZ12]) *Let $\mathcal{P}$ be a broadcast protocol and $Occur(\mathcal{P})$ denote the set of reachable states.*

1. *The set $Occur(\mathcal{P})$ is effectively computable in PTIME.*
2. *For every $k \in \mathbb{N}$, there exist a network size and a finite execution such that for all $\mathsf{q} \in Occur(\mathcal{P})$, there are at least k processes in state q at the end of the execution.*
3. *There exists $K \in \mathbb{N}$ such that for all $k \geq K$, there exist a network size and a finite execution such that for all $\mathsf{q} \in Occur(\mathcal{P})$, there are exactly k processes in state q at the end of the execution.*

A logic, called cardinality constraint, was introduced in [DSTZ12] in order to express the presence or absence of processes in given states in a configuration. Qualitative cardinality constraints are expressed by the following grammar:

$$\phi := \#\mathsf{q} = 0 \mid \#\mathsf{q} \geq 1 \mid \phi \wedge \phi \mid \phi \vee \phi \ .$$

The semantics of cardinality constraints is rather natural, *e.g.* a configuration satisfies the formula $\phi = (\#\mathsf{q}_1 \geq 1 \wedge \#\mathsf{q}_2 = 0) \vee (\#\mathsf{q}_3 = 0)$ if either some process is in state q_1 and none are in state q_2, or no process is in state q_3. The authors studied the problem of

satisfiability of cardinality constraint formulas, meaning the existence of a network size for which there exists an execution reaching a configuration satisfying the formula. This problem subsumes the reachability and the synchronization problems since they can be encoded into such formulas. Indeed, the reachability problem of a state q corresponds to the cardinality constraint formula $\#\mathsf{q} \geq 1$, and the synchronization problem for a set of states S to the formula $\bigwedge_{\mathsf{q} \notin S} \#\mathsf{q} = 0$. It was shown in [DSTZ12] that the satisfiability of cardinality constraint is NP-complete. However, we sketch here a proof stating that the complexity comes in fact from the complexity of the formula. Indeed, considering disjunctive normal form formulas, the problem is PTIME-complete in the size of the protocol and the formula. Disjunctive normal form formulas can be seen as disjunctions of synchronization problems, the algorithm thus consists in checking each of these synchronization problems until one is satisfied or that the formula is proven false.

To check the synchronization problem in PTIME, we use the algorithm of [DSTZ12], computing the set of reachable states. The idea is to remove from the protocol the non-reachable states, after what we transform the protocol by reverting all its transitions, and considering the set of target states as the initial states. One can compute the set of reachable states in this reverted protocol called co-reachable states. Intuitively, co-reachable states correspond to states for which, starting from an initial configuration with 'enough' processes in each state, one can find an execution reaching a configuration satisfying the synchronization problem. Reciprocally, a state which is not co-reachable cannot be visited by an execution satisfying the synchronization problem since there is no way to 'bring back' the process to a state of the target set. Thus, we can remove the non-co-reachable states from the protocol. Iterating this procedure, we end with a protocol containing only the states that are both reachable and co-reachable. Assuming that the obtained protocol is not empty, using point 3. of Lemma 2.1 on both the protocol an the reverted protocol, one can exhibit an execution satisfying the synchronization property. We thus obtain a PTIME procedure to decide the synchronization problem, and more generally, thus show cardinality constraint problems to be PTIME for disjunctive normal form formulas.

In comparison with these works, in this thesis, we extend the model of reconfigurable networks with probabilities by allowing probabilistic transitions in the description of the protocol. We also consider a new restriction for the reconfigurable networks by considering only local strategies, *i.e.* enforcing the processes to behave in a truly distributed way. Local strategies assume that the processes choose their action only according to their knowledge of the current execution, *i.e.* the transitions they have taken so far, and not according to the whole network history. But first, we investigate a probabilistic and timed extension of clique protocols, where the number of processes is parametric and all the messages reach every component.

Chapter III

Clique networks of probabilistic timed protocols

1 Introduction

In this chapter, we introduce the model of clique networks of probabilistic timed protocols and study the parameterized verification of this model. A clique network of probabilistic timed protocols is formed of many identical probabilistic timed automata(see *e.g.* [KNPS08]), with a single clock. Interaction between processes is modeled by broadcasts of messages to all the processes in the network. This model is a probabilistic and timed extension of the models studied in [DSZ11b, DSZ11a, DSZ12]. Notice that this formalism combines infinite-state space, due to an unknown number of processes in the network, as well as data structures with infinite domains (dense clocks), and probabilistic behaviors. Moreover, in order to represent the notion of mobility in the network, we further introduce *dynamic* probabilistic timed networks, where processes can disappear and be created, according to fixed probability laws.

This model of probabilistic timed networks is quite natural since a potential application domain for parameterized verification is the one of wireless sensor networks (WSN). WSN consist of a large number of nodes measuring and transmitting data. The number of nodes is a significant parameter while setting up the network, since it affects in particular the performances by influencing the risk of collision in communications. Most protocols for WSN include probabilistic choices and timing constraints. Indeed, the probabilities are used to break the symmetry between the nodes and the timing constraint to ensure reactivity of the system. Also, in many cases the number of nodes in the network evolves over time due to nodes breaking down, or nodes refilling their battery using *e.g.* solar energy. Moreover, in some applications, the placement of sensors and their exact number are unknown. We therefore advocate that (dynamic) probabilistic timed networks with a parametric number of processes make a quite suitable model for WSN protocols.

So far, up to our knowledge, the automated verification of such protocols, combining probabilities and time, has only been studied for a fixed, and rather small, number of

nodes, as in [Fru06] where Prism [KNPS08] is used to verify the contention resolution protocol in IEEE standard 802.15.4. In comparison, the parameterized verification of probabilistic timed networks that we investigate in this chapter would provide answers for an arbitrary number of processes.

Given a probabilistic timed network, we consider two kinds of relevant parameterized verification problems: parameterized reachability problems and parameterized synchronization problems. An example of parameterized reachability problem is the following. Given a distinguished state of the protocol model, can a configuration with some process in this particular state be reached almost surely, for any initial number of processes and for every scheduling policy? Equivalently, the problem is whether, independently of the number of processes, the minimum probability to reach a target configuration is 1. An other interesting problem is the synchronization problem: for a set of target states, can the configurations with all processes gathered in this target set be reached almost surely? Note that our model, clique networks of probabilistic timed protocols, combines both probabilities and non-determinism, we thus need to quantify over the possible resolution of the non-determinism. Hence, beyond these two particular problems, we consider all the existential and universal variants for quantifying over strategies and all the thresholds between 0 or 1.

The rest of this chapter is organized as follows. We formally define the model of probabilistic timed protocols in Section 2 and we detail the semantics we attach to clique networks of such protocols: in a first part, a static semantics is considered, where the number of processes is unknown but fixed; and in a second part, a dynamic semantics is detailed where the number of processes evolves in a probabilistic way.

Section 3 is devoted to parameterized verification for the static semantics. We show that most qualitative parameterized verification problems are undecidable when the topology is static, that is, if the number of processes is constant (but unknown). These undecidability results already hold in the untimed case, *i.e.* when the individual processes are Markov decision processes (MDP) rather than probabilistic timed automata. To establish the undecidability results, we explain how a clique network of probabilistic protocols can simulate a 2-counter machine [Min67], using the processes to encode the counter values.

Section 4 is devoted to the dynamic case, where the initial number of processes is unknown and in which the processes are created and disappear according to fixed probability distributions. In contrast to the static case, parameterized verification becomes decidable in this setting, and we provide a decision procedure for all the qualitative parameterized verification problems of interest. The algorithm consists in computing, by fix point iteration, the set of configurations for which the property holds. In each case, the termination of the algorithm is ensured by a dedicated well-quasi-order on configurations of the network, and the correctness of the algorithm relies on a finite attractor property [ABRS05] in our model. We also establish a complexity lower-bound by reducing the reachability problem in lossy channel systems [AJ93, CS08]: the qualitative parameterized verification problems in the dynamic case are thus non-primitive recursive.

Afterwards, we move to the more complex problem of quantitative parameterized verification. Unfortunately, we left the quantitative parameterized verification problems open. However we provide a partial answer: for a fixed initial configuration, we give an approximation procedure of the minimal probability to reach a target state. This result is in a non-parameterized setting, yet it is still of interest since it is, to the best of our knowledge, the first result on the probability approximation on infinite state models combining probabilities and non-determinism. Moreover, this result can be generalized to other infinite state models combining probabilities and non-determinism such as lossy channel systems with insertion.

2 Modeling probabilistic networks

This section is devoted to the introduction of probabilistic timed protocols and to the semantics we attach to clique networks of such protocols. We define both the static semantics where the number of processes is unknown but fixed, and the dynamic semantics where the number of processes evolves in a probabilistic way.

2.1 Probabilistic timed protocols

In order to model the behaviors of processes evolving in a network, we introduce probabilistic timed protocols. This model is an extension of the broadcast protocol model studied in [DSZ11b, DSZ11a, DSZ12] but with probabilities and one clock variable. For each process, its behavior may now be guarded by some constraints on the value of its own clock. We restrict the number of clocks to one by process because of the undecidability result of the reachability problem with multiple clocks for a similar non-probabilistic model [ADM04]. Moreover, the probabilistic extension allows us to model randomized algorithms as well as the uncertainty of the result of an action.

We recall that (see Section I.2) given a clock variable $\mathbf{x}$, $\mathcal{G}(\mathbf{x})$ denotes the set of guards on the clock composed of conjunctions of constraints of the form $\mathbf{x} \sim n$ where $n \in \mathbb{N}$ is an integer and $\sim \in \{\leq, <, =, >, \geq\}$ is a comparison operator. Moreover, $Up(\mathbf{x}) \in \{\emptyset, \mathbf{x} := 0\}$ denotes the set of updates on the clock variable that either (for $\emptyset$) leaves the clock value unchanged or (for $\mathbf{x} := 0$) resets it to zero.

Definition 2.1 (Probabilistic timed protocols) *A* probabilistic timed protocol *is a tuple* $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \mathbf{x}, \Sigma, \Delta)$ *where:*

- Q *is a finite set of control states, partitioned into probabilistic states* $\mathrm{Q}^{(p)}$ *and non-deterministic states* $\mathrm{Q}^{(n)}$*,*
- $\mathsf{q}_0 \in \mathrm{Q}^{(n)}$ *is the initial state,*
- $\mathbf{x}$ *is a clock variable,*
- Σ *is a finite message alphabet,*
- Δ *is the edge relation, partitioned into*

- *internal actions:* $\Delta_i \subseteq \mathrm{Q}^{(n)} \times \mathcal{G}(\mathbf{x}) \times \{\varepsilon\} \times Up(\mathbf{x}) \times \mathrm{Q}^{(p)}$,
- *broadcasts:* $\Delta_b \subseteq \mathrm{Q}^{(n)} \times \mathcal{G}(\mathbf{x}) \times \{!!m \mid m \in \Sigma\} \times Up(\mathbf{x}) \times \mathrm{Q}^{(n)}$,
- *receptions:* $\Delta_r \subseteq \mathrm{Q}^{(n)} \times \mathcal{G}(\mathbf{x}) \times \{??m \mid m \in \Sigma\} \times Up(\mathbf{x}) \times \mathrm{Q}^{(n)}$,
- *probabilistic transitions:* $\Delta_p : \mathrm{Q}^{(p)} \to Dist\left(\mathrm{Q}^{(n)}\right)$ *that, for each probabilistic state* $\mathsf{q} \in \mathrm{Q}^{(p)}$*, associates a unique distribution.*

In our model, a control state can be the source of several internal actions, each giving rise to a probability distribution for the successor state, whereas broadcasts and receptions are not probabilistic. This is not a real restriction:

Remark 2.1 (Probabilistic receptions and broadcasts) *Notice that models with probabilistic and non-deterministic choices for broadcasts and receptions can be encoded in our model by introducing intermediary states and additional internal actions.*

Notice that the model of probabilistic timed protocol is equivalent to the model of probabilistic timed automata [KNPS08] with one clock. However, we choose to redefine it here to emphasize two points: first the particularities of our model such as the restriction to one clock and broadcast communication, and second the link with broadcast protocol models [DSZ11b, DSZ11a, DSZ12] for which probabilistic timed protocols form a timed probabilistic extension.

Example 2.1 *A simple example of probabilistic timed protocol modeling mutual exclusion over two resources is given in Figure 2.1. For simplicity sake, in the example, non specified guards are the trivial guards, and only reset updates are specified.*

The initial state of this protocol is the state idle*. From here, there is an internal transition* $(\mathsf{idle}, true, \varepsilon, \mathbf{x} := 0, \mathsf{p}) \in \Delta_i$ *going to a probabilistic state* $\mathsf{p} \in \mathrm{Q}^{(p)}$*. When taking this transition, the clock is reset to zero. The probabilistic transition* d *associated with* p*,* $(\mathsf{p}, d) \in \Delta_p$*, is such that* $d(\mathsf{req}_1) = p$ *and* $d(\mathsf{req}_2) = 1 - p$*.*

The non-deterministic states $\mathsf{req}_1 \in \mathrm{Q}^{(n)}$ *and* $\mathsf{req}_2 \in \mathrm{Q}^{(n)}$ *represent the requests of (respectively) resource one and two. Once the clock has a value greater than 1, the transition* $(\mathsf{req}_1, \mathbf{x} > 1, !!1, \mathbf{x} := 0, \mathsf{CS}_1) \in \Delta_b$ *can be taken. This transition broadcasts the message 1 and allows to move to* CS_1*, resetting the clock to zero.*

In state req_1*, one can also move back to the state* idle *by receiving the message 1 thanks to the transition* $(\mathsf{req}_1, true, ??1, \emptyset, \mathsf{idle}) \in \Delta_r$*.*

The semantics of this example protocol will be explained in Example 2.2.

In the semantics (formally defined in the next section), we consider non-blocking broadcasts, meaning that the absence of reception of a message does not forbid the broadcast of the message. Hence a self-loop on any state where a reception is not defined would give the same semantics. Moreover, in the case where several reception transitions are defined for the same state, we can 'split' this state thanks to internal actions in order to consider only one possible reception at a time in every state. These two points allow to assume that protocols have complete deterministic edge relation for receptions as stated in the following remark.

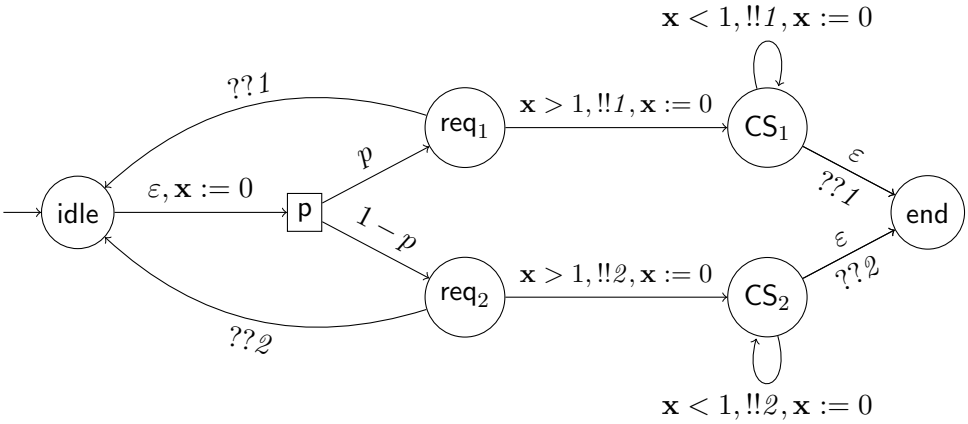


Figure 2.1: A probabilistic timed protocol modeling mutual exclusion over two resources.

Remark 2.2 (Simplification: complete deterministic receptions) *The edge relation is assumed, for simplicity, complete and deterministic for receptions: from every state* q*, for every value* x *of clock variable* $\mathbf{x}$ *and for every message* $m \in \Sigma$*, there exists a single edge* $(\mathsf{q}, g, ??m, Up(\mathbf{x}), \mathsf{q}') \in \Delta_r$ *with* $x \models g$*.*

2.2 Static semantics for clique networks of probabilistic timed protocols

In this section, we now introduce networks composed of probabilistic timed protocols. We consider that the components are arranged in a clique, *i.e.* that a broadcast reaches all the processes and that they all execute the same probabilistic timed protocol $\mathcal{P}$.

Definition 2.2 (Clique network of probabilistic timed protocol) *A* clique network $C(\mathcal{P}^N)$ *is composed of* $N \in \mathbb{N}$ *copies, called* processes*, of a probabilistic timed protocol* $\mathcal{P}$*.*

Note that in a clique network $C(\mathcal{P}^N)$ the number of processes is fixed to N along all computations; hence we will, sometimes, refer to this network as *static* networks in opposition to dynamic networks presented in the next section.

Example 2.2 *The intuitive semantics of a clique network composed of processes running the protocol given in Figure 2.1 is the following:*

Each process starts in state idle *with clock value* 0*. At any time, any process in state* idle *can request access to a resource. The choice of the allocated resource is probabilistic. The requesting process must then stay in the corresponding request state (*req_i*) at least one time unit before moving to the critical section state (*CS_i *representing usage of resource* i*). If a requesting process receives a message indicating that the resource is already used, it moves back to the initial state* idle*. While using a resource, a process informs the others by broadcast, at least once every time unit.*

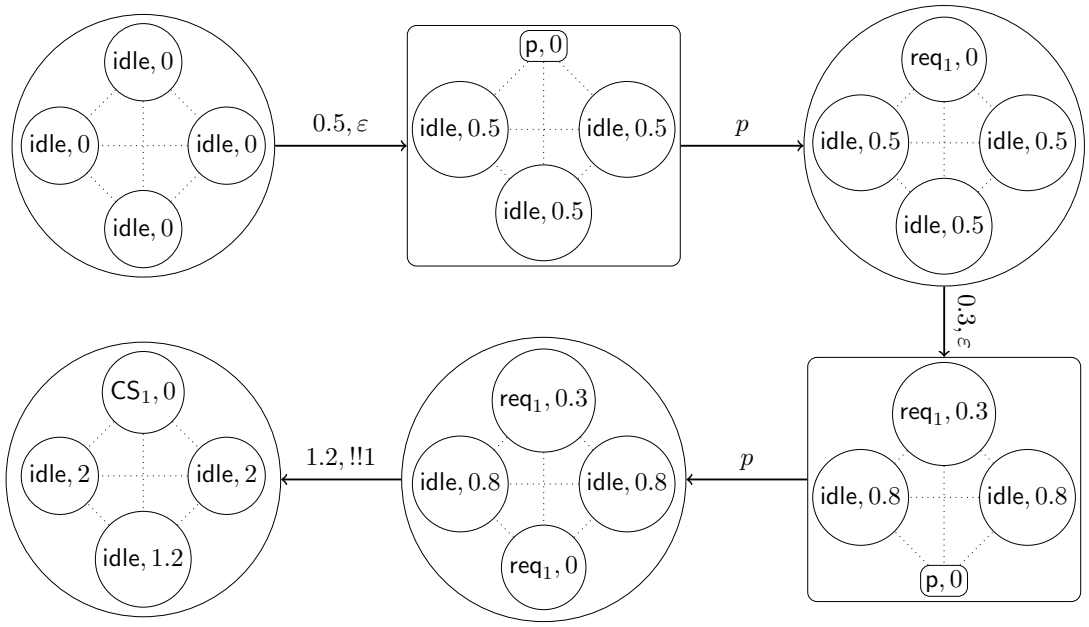


Figure 2.2: An execution of a clique network of 4 processes running the protocol represented in Figure 2.1.

*An example of an execution is given in Figure 2.2. In this figure, the pair corresponding to the current state and clock value of a process is represented inside a circle if the state is non-deterministic (*e.g $(\mathsf{idle}, 0)$ *in the first configuration) and in a square if it is probabilistic (*e.g. $(\mathsf{p}, 0)$ *in the second configuration). Symmetrically, the probabilistic configurations (here the second and fourth configurations) are boxed whereas the non-deterministic configurations are circled.*

In this execution, first, after 0.5 *time unit, a process requests access to a resource, with probability p it moves to the request state* req_1*. After* 0.3 *time unit, an other process also requests access to a resource and ends in state* req_1*. Finally, after* 1.2 *time units (since its clock is now greater than* 1*), the first process accesses the resource by broadcasting the message 1 , the other process thus moves back to the state* idle*.*

We now formalize this semantics. The semantics of a clique network $C(\mathcal{P}^N)$ of N processes running the probabilistic timed protocol $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \mathbf{x}, \Sigma, \Delta)$ is given as a Markov decision process, denoted $\mathscr{C}(\mathcal{P}^N)$, which is detailed below.

Recall that processes do not carry identifiers, so that one cannot distinguish between two processes with same control state and same clock value. As a consequence, we define a *configuration* as a finite multiset of size N, $\gamma \in \mathbb{M}^N(\mathrm{Q} \times \mathbb{R}^+)$ over the set of pairs composed of a control state and a real value for the clock. Intuitively, if configuration γ contains exactly two occurrences of $(\mathsf{q}, 0.5)$, written $\gamma(\mathsf{q}, 0.5) = 2$, then two processes have current state q and current clock value $x = 0.5$. Abusing notations, we denote by $\gamma(\mathsf{q}) = \sum_{x \in \mathbb{R}^+} \gamma(\mathsf{q}, x)$ the number of processes in state q. Moreover, since N processes

are involved in the network, $\sum_{\mathsf{q}\in \mathrm{Q}} \gamma(\mathsf{q}) = N$.

The semantics of a clique network $C(\mathcal{P}^N)$ is the Markov decision process $\mathscr{C}(\mathcal{P}^N) = (\Gamma, \Gamma^{(p)}, \Gamma^{(n)}, \{\gamma_0\}, T^{(n)}, T^{(p)})$, where:

- Γ is the set of configurations partitioned into probabilistic configurations $\Gamma^{(p)}$ and non-deterministic configurations $\Gamma^{(n)}$;
  - $\Gamma^{(p)} = \{\gamma \in \mathbb{M}^N(\mathrm{Q} \times \mathbb{R}^+) \mid \sum_{\mathsf{q}\in \mathrm{Q}^{(p)}} \gamma(\mathsf{q}) = 1\}$ is the set of probabilistic configurations, *i.e.* configurations in which exactly one process (the process performing the probabilistic action) is in a probabilistic state;
  - $\Gamma^{(n)} = \mathbb{M}^N(\mathrm{Q}^{(n)} \times \mathbb{R}^+)$ is the set of non-deterministic configurations;
- $\gamma_0 \in \Gamma^{(n)}$ is the initial configuration, defined by $\gamma_0(\mathsf{q}_0, 0) = N$ and $\gamma_0(\mathsf{q}, x) = 0$ otherwise;
- $T^{(p)}$ is the transition probability function that is naturally lifted from Δ_p, *i.e.* if $(\mathsf{q}, d) \in \Delta_p$ and there exists $x \in \mathbb{R}^+$ such that $\gamma(\mathsf{q}, x) = 1$ then $(\gamma, d') \in T^{(p)}$ with d' such that for all $\gamma' \in \Gamma$, if there exists $\mathsf{q}' \in \mathrm{Q}$ such that $\gamma' = \gamma - (\mathsf{q}, x) + (\mathsf{q}', x)$ then $d'(\gamma') = d(\mathsf{q}')$ and $d'(\gamma') = 0$ otherwise;
- $T^{(n)} \subseteq \Gamma^{(n)} \times \mathbb{R}^+ \times (\mathrm{Q} \times \mathbb{R}^+) \times \Delta \times \Gamma$ is the non-deterministic transition relation defined as follow: $(\gamma, y, (\mathsf{q}, x), \delta, \gamma') \in T^{(n)}$ if δ is a broadcast or an internal action and denoting $\delta = (\mathsf{q}', g, a, up, \mathsf{q}_\delta)$, we have $\mathsf{q} = \mathsf{q}'$ and there exists $\gamma_1 \in \mathbb{M}^N(\mathrm{Q} \times \mathbb{R}^+)$ and $\gamma_2, \gamma_3 \in \mathbb{M}^{N-1}(\mathrm{Q} \times \mathbb{R}^+)$ such that:

  Time elapsing: for all $\bar{\mathsf{q}} \in \mathrm{Q}$ and all $z \in \mathbb{R}^+$, $\gamma_1(\bar{\mathsf{q}}, z + y) = \gamma(\bar{\mathsf{q}}, z)$, *i.e.* all the clock values are incremented by y.

  Non-deterministic choice of a process: there exists a process in state q with clock value x, *i.e.* $\gamma_1(\mathsf{q}, x) > 0$, the transition δ is enabled for this process, *i.e.* $x \vDash g$, and the process is treated separately $\gamma_2 = \gamma_1 - (\mathsf{q}, x)$.

  Execution of the action: if δ is an internal action, *i.e.* if $a = \varepsilon$, then the other processes are unaffected hence $\gamma_2 = \gamma_3$. Otherwise, $a = !!m$ and all the processes have to receive the message m by taking a reception transition, hence we update their states and their clock value according to the upgrade specified on the reception. Formally, for all states $\mathsf{q}_3 \in \mathrm{Q}$ and all clock values $z \in \mathbb{R}^+$, $\gamma_3(\mathsf{q}_3, z) = \sum\{\gamma_2(\mathsf{q}'_3, z') \mid (\mathsf{q}'_3, g', ??m, up', \mathsf{q}_3) \in \Delta_r \texttt{ and } z' \vDash g' \texttt{ and } z = up(z')\}$[1].
  Finally, the state of the process performing the action and its clock value are updated according to δ, *i.e.* $\gamma' = \gamma_3 + (\mathsf{q}_\delta, up(x))$.

  Moreover, in order to forbid finite runs, we use a special action **unlock** that is enabled only when no other active action (either internal action or broadcast) is enabled, even after some time elapsing, *i.e.* if $(\{\gamma\} \times \mathbb{R}^+ \times (\mathrm{Q} \times \mathbb{R}^+) \times (\Delta_i \cup \Delta_b) \times \Gamma) \cap T^{(n)} = \emptyset$ we add a special transition $(\gamma, \textbf{unlock}, \gamma)$ in $T^{(n)}$ that loops on the configuration γ.

[1] Recall that we consider complete deterministic edge relation for reception (Remark 2.2).

Informally, the non-deterministic transition relation is split in three parts; in the first part, we let time elapse (possibly 0 unit of time) and all the clocks in the networks are incremented by the same value, this leads to the configuration γ_1; in the second part, a process is chosen non-deterministically to perform an action, we thus divide the multiset into a multiset and a pair (q, x) representing the selected process; in the last part, this process performs an enabled action (*i.e.* an action such that its clock value satisfies the guard), it can either be an internal action that affects only itself and leads to a probabilistic configuration (for which the probability distribution is given by $T^{(p)}$), or a broadcast, that reaches all other processes in the network, in which case they all update their clocks and state according to the reception transition they use. These steps are graphically represented in Figure 2.3. However, since we do not distinguish

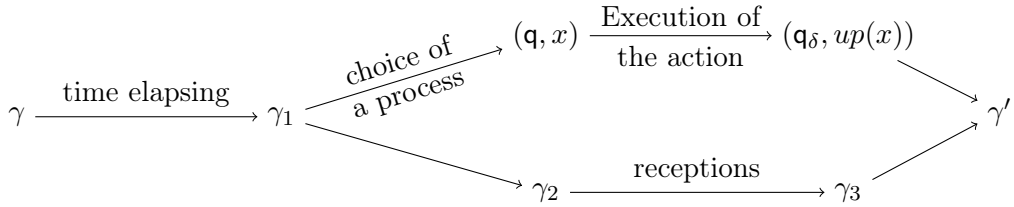


Figure 2.3: Graphical representation of a non-deterministic transition.

between processes with same control state and same clock value, the intuitive semantics of non-deterministic steps is not to select a process, but rather to select a pair (q, x) with $\gamma(\mathsf{q}, x) > 0$ and then to fire for any process in state (q, x) an enabled action.

In order to simplify notations in the case where $(\gamma, y, (\mathsf{q}, x), \delta, \gamma') \in T^{(n)}$ (resp. $(\gamma, \mathbf{unlock}, \gamma') \in T^{(n)}$), we will write $\gamma \xrightarrow{y,(\mathsf{q},x),\delta} \gamma'$ (resp. $\gamma \xrightarrow{\mathbf{unlock}} \gamma'$) or even simply $\gamma \to \gamma'$.

An *execution* in $\mathscr{C}(\mathcal{P}^N)$ is a finite, or infinite, sequence of configurations $\rho = \gamma_0\gamma_1\gamma_2\ldots$ starting in the initial configuration γ_0 and such that for all i, if $\gamma_i \in \Gamma^{(n)}$ then $\gamma_i \to \gamma_{i+1}$ and if $\gamma_i \in \Gamma^{(p)}$, then denoting d the probability distribution such that $(\gamma_i, d) \in T^{(p)}$ then $d(\gamma_{i+1}) > 0$. The set of all executions is denoted $\mathcal{E}$.

2.3 Dynamic semantics for clique networks of probabilistic timed protocols

Clique networks of probabilistic timed protocols from Definition 2.2 are networks where the number of processes is fixed along any execution. We now discuss a reasonable way to introduce some mobility in the network. In the application to wireless sensor networks, the number of nodes may change during the execution of the system, since nodes may break down or run out of battery, but also, may be newly inserted or have their battery refilled. We therefore propose a model of probabilistic timed networks, in which the number of processes evolves over time, and in which, disappearances and creations of processes are independent random events following fixed probability distributions. Abstracting mobility by random events seems to be a good trade-off between simplicity

and realism of the model.

Moreover, in such systems with dynamism there is often a base station that is fixed all along the execution. For example, in the case of wireless sensor networks, there can be a fixed antenna connected to the network in order to collect data. We therefore consider a model with a fixed base in addition to a parametric number of mobile processes.

Definition 2.3 (Dynamic clique networks of probabilistic timed protocols) *A* dynamic clique network $DC(B, \mathcal{P}_\Lambda^N)$ *is composed of one probabilistic timed protocol* B *called the base and several (initially* $N \in \mathbb{N}$*) copies, called* processes*, of a probabilistic timed protocol* $\mathcal{P}$*. The number of processes evolves according to probability distributions given by the pair* $\Lambda = (\lambda_+, \lambda_-)$*, consisting of a creation rate* $\lambda_+ \in]0,1[$ *and a deletion rate* $\lambda_- \in (0,1)$*.*

The semantics of a **D**ynamic **C**lique network, composed of a base running the protocol B and processes running the protocol $\mathcal{P}$ (with initially N processes) with pair of mobility rates Λ, is given as a Markov decision process $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$ which is detailed below.

The rates λ_+ and λ_- represent respectively dynamic creation and disappearance of processes according to fixed probabilistic laws: after each discrete action, each process disappears with probability λ_-, followed by the creation of k processes (in control state q_0 with clock value 0) with probability $\lambda_+^k(1-\lambda_+)$, for every integer $k \in \mathbb{N}$.

Notice that the case $\lambda_+ = \lambda_- = 0$ is a particular case ruled out here in order to avoid the static case. Indeed, with $\lambda_+ = \lambda_- = 0$, the number of processes would be constant and if the base also runs the protocol $\mathcal{P}$, we recover the model of (non-dynamic) clique network of probabilistic timed protocols from Definition 2.2, *i.e* $\mathscr{C}(\mathcal{P}^N) = \mathscr{DC}(\mathcal{P}, \mathcal{P}_{(0,0)}^{N-1})$.

Example 2.3 *An execution of* $\mathscr{DC}(\mathcal{P}, \mathcal{P}_{(0.1,0.2)}^3)$ *is represented in Figure 2.4. For simplicity, both the base and the processes run the protocol* $\mathcal{P}$*. Notice that each step is similar to the execution in the static network except for the configuration indexed by* (d) *from which the dynamism takes place. In this example, the base is represented by the top state. The base first performs an internal action that leads with probability* p *to the state* req_1*. After that, the dynamism takes place, one process disappears and none are created, thus the probability is* $\binom{3}{1}0.1^1(1-0.1)^2(0.2)^0(1-0.2) = 0.1944$*. Next, after* 1.1 *time units the base sends the message 1 that all processes receive. Then, one process disappears and four are created, the probability is thus* $\binom{2}{1}0.1^1(1-0.1)^1(0.2)^4(1-0.2) = 0.0002304$*.*

We now detail the semantics $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$ of a dynamic clique network $DC(B, \mathcal{P}_\Lambda^N)$, with $B = (\mathrm{Q}_B, \mathrm{Q}_B^{(p)}, \mathrm{Q}_B^{(n)}, \mathsf{b}_0, \mathbf{x}_B, \Sigma, \Delta_B)$ and $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \mathbf{x}, \Sigma, \Delta)$. The semantics of non-deterministic actions and probabilistic transitions is really close to the semantics for static networks, the only difference is that we have to take into account the base here; however there is a major difference after each non-deterministic action and probabilistic transition since there are random creations and disappearances of processes.

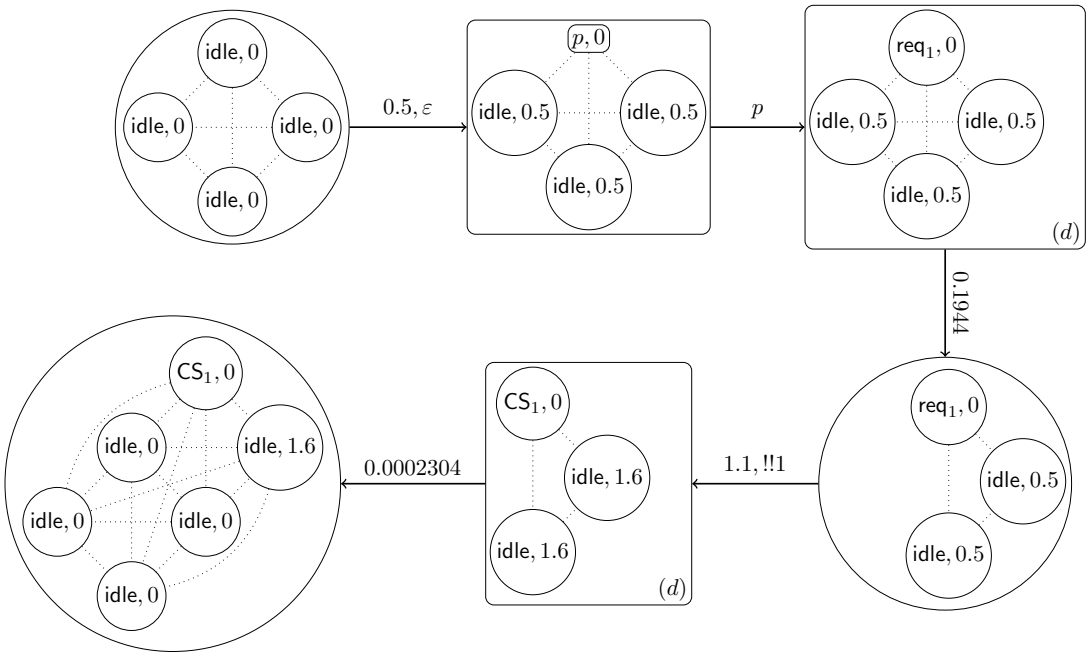


Figure 2.4: An execution of a dynamic clique network of (initially) three processes and a base all running the protocol represented in Figure 2.1.

A *configuration* is now a pair $\boldsymbol{\gamma} = ((\mathsf{b}, x_B), \gamma)$ composed of the state of the base and its clock value, together with a finite multiset $\gamma \in \mathbb{M}(\mathrm{Q} \times \mathbb{R}^+)$ over the set of pairs composed of a control state and a real value for the clock. Notice that now, we consider multisets that can be of any size. Since, thanks to the dynamism, even if we start with N processes, there can later be any number of processes in the network.
In order to simplify definitions, we sometimes abuse notation and see the pair $\boldsymbol{\gamma} = ((\mathsf{b}, x_B), \gamma)$ as the multiset $\gamma + \langle(\mathsf{b}, x_B)\rangle$.

The semantics of a dynamic network $DC(B, \mathcal{P}_\Lambda^N)$ is given in terms of a Markov decision process $\mathscr{DC}(B, \mathcal{P}_\Lambda^N) = (\Gamma, \Gamma^{(p)}, \Gamma^{(n)}, \{((\mathsf{b}_0, 0), \gamma_0)\}, T^{(n)}, T^{(p)})$, where:

- $\Gamma^{(n)} = (\mathrm{Q}_B^{(n)} \times \mathbb{R}^+) \times \mathbb{M}(\mathrm{Q}^{(n)} \times \mathbb{R}^+)$ is the set of non-deterministic configurations;
- $\Gamma^{(p)}$ is the set of probabilistic configurations that is partitioned in two sets: first, $\{\boldsymbol{\gamma} \in (\mathrm{Q}_B \times \mathbb{R}^+) \times \mathbb{M}(\mathrm{Q} \times \mathbb{R}^+) \mid \sum_{\mathsf{q} \in \mathrm{Q}^{(p)} \cup \mathrm{Q}_B^{(p)}} \boldsymbol{\gamma}(\mathsf{q}) = 1\}$ configurations in which exactly one process is in a probabilistic state; and second $\{\boldsymbol{\gamma}^{(d)} \mid \boldsymbol{\gamma} \in \Gamma^{(n)}\}$ a copy of the non-deterministic configurations (the dynamism will take place from these configurations as explained in the following).
- $((\mathsf{b}_0, 0), \gamma_0) \in \Gamma^{(n)}$ is the initial configuration, defined by $\gamma_0(\mathsf{q}_0, 0) = N$ and $\gamma_0(\mathsf{q}, x) = 0$ otherwise;

- The non-deterministic transition relation is defined exactly as for the static case but now leads to a probabilistic configuration from which the dynamism will take place: $T^{(n)} \subseteq \Gamma^{(n)} \times \mathbb{R}^+ \times ((\mathrm{Q} \cup \mathrm{Q}_B) \times \mathbb{R}^+) \times (\Delta \cup \Delta_B) \times \Gamma^{(p)}$ is such that $(\boldsymbol{\gamma}, y, (\mathsf{q}, x), \delta, \boldsymbol{\gamma}'') \in T^{(n)}$ if and only if, either δ is an internal action denoted $\delta = (\mathsf{q}', g, \varepsilon, up, \mathsf{q}_\delta)$ then $\boldsymbol{\gamma}'' = \boldsymbol{\gamma}' \in \Gamma^{(p)}$, or δ is a broadcast action denoted $\delta = (\mathsf{q}', g, !!m, up, \mathsf{q}_\delta)$ then $\boldsymbol{\gamma}'' = \boldsymbol{\gamma}'^{(d)}$ with $\boldsymbol{\gamma}' \in \Gamma^{(n)}$ and we have, $\mathsf{q} = \mathsf{q}'$ and there exists $\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \boldsymbol{\gamma}_3 \in \mathbb{M}(\mathrm{Q} \times \mathbb{R}^+ \cup \mathrm{Q}_B \times \mathbb{R}^+)$ such that:

  Time elapsing: for all $\mathsf{q}_1 \in \mathrm{Q} \cup \mathrm{Q}_B$ and all $z \in \mathbb{R}^+$, $\boldsymbol{\gamma}_1(\mathsf{q}_1, z+y) = \boldsymbol{\gamma}(\mathsf{q}_1, z)$, *i.e.* all the clock values are incremented by y.

  Non-deterministic choice of a process: there exists a process (or the base) in state q with clock value x, *i.e.* $\boldsymbol{\gamma}_1(\mathsf{q}, x) > 0$, the transition δ is enabled for this process, *i.e.* $x \vDash g$ and the process is treated separately $\boldsymbol{\gamma}_2 = \boldsymbol{\gamma}_1 - (\mathsf{q}, x)$.

  Execution of the action: if δ is an internal action, *i.e.* if $a = \varepsilon$ then the other processes are unaffected, hence $\boldsymbol{\gamma}_2 = \boldsymbol{\gamma}_3$. Otherwise, $a = !!m$ and all the processes have to receive the message m by taking a reception transition, hence we update their states and their clock values according to the reception. Formally, for all states $\mathsf{q}_3 \in \mathrm{Q} \cup \mathrm{Q}_B$ and all clock values $z \in \mathbb{R}^+$, $\boldsymbol{\gamma}_3(\mathsf{q}_3, z) = \sum\{\boldsymbol{\gamma}_2(\mathsf{q}_3', z') \mid (\mathsf{q}_3', g', ??m, up', \mathsf{q}_3) \in \Delta \cup \Delta_B \texttt{ and } z' \vDash g' \texttt{ and } z = up'(z')\}$. Finally, the state of the process performing the action and its clock value are updated according to δ, *i.e.* $\boldsymbol{\gamma}' = \boldsymbol{\gamma}_3 + (\mathsf{q}_\delta, up(x))$.

  Moreover, as in the static case, in order to forbid finite runs, we use a special action **unlock** that is enabled only when no other active action (either internal action or broadcast) is enabled, even after some time elapsing.

- The probabilistic transition function $T^{(p)}$ is defined differently depending on the type of the probabilistic configuration. First, for a configuration $\boldsymbol{\gamma}$ such that $\sum_{\mathsf{q} \in \mathrm{Q}^{(p)} \cup \mathrm{Q}_B^{(p)}} \boldsymbol{\gamma}(\mathsf{q}) = 1$ the probabilistic transition function is, as before, naturally lifted from Δ_p, *i.e.* if $(\mathsf{q}, d) \in \Delta_p$ and there exists $x \in \mathbb{R}^+$ such that $\gamma(\mathsf{q}, x) = 1$ then $(\boldsymbol{\gamma}, d') \in T^{(p)}$ with d' such that for all $\mathsf{q}' \in \mathrm{Q}$, $d'(((\mathsf{b}, \mathbf{x}_B), \gamma - (\mathsf{q}, x) + (\mathsf{q}', x))^{(d)}) = d(\mathsf{q}')$ and $d'(\boldsymbol{\gamma}') = 0$ otherwise. Similarly, if $\mathsf{b} \in \mathrm{Q}_B^{(p)}$ and $(\mathsf{b}, d) \in \Delta_B$ then $(\boldsymbol{\gamma}, d') \in T^{(p)}$ with d' such that for all $\mathsf{b}' \in \mathrm{Q}_B$, $d'(((\mathsf{b}', \mathbf{x}_B), \gamma)^{(d)}) = d(\mathsf{b}')$ and $d'(\boldsymbol{\gamma}') = 0$ otherwise.

Dynamism We now explain how the dynamism in the network is modeled. With each configuration $\boldsymbol{\gamma} = ((\mathsf{b}, x_B), \gamma)^{(d)} \in \Gamma^{(p)}$, we associate the probability distribution d, *i.e.* $(\boldsymbol{\gamma}, d) \in T^{(p)}$ where d is such that for all configurations $\boldsymbol{\gamma}' = ((\mathsf{b}', {x_B}'), \gamma') \in \Gamma$.

- $d(\boldsymbol{\gamma}') = 0$ if $\mathsf{b}' \neq \mathsf{b}$ or ${x_B}' \neq x_B$ or $\boldsymbol{\gamma}' \notin \Gamma^{(n)}$;
- otherwise $d(\boldsymbol{\gamma}') = \sum_{\gamma_1} P_-(\gamma, \gamma_1) \cdot P_+(\gamma_1, \gamma')$ where we write $P_-(\gamma_1, \gamma_2)$ for the probability to obtain γ_2 from γ_1 when processes disappear; recall that each

process can disappear with probability λ_-. Hence $P_-(\gamma_1,\gamma_2)=0$ if $\gamma_2 \not\subseteq \gamma_1$ and $P_-(\gamma_1,\gamma_2)=\prod_{(\mathsf{q},x)\in\gamma_1}\binom{\gamma_1(\mathsf{q},x)}{\gamma_2(\mathsf{q},x)}\lambda_-^{\gamma_1(\mathsf{q},x)-\gamma_2(\mathsf{q},x)}(1-\lambda_-)^{\gamma_2(\mathsf{q},x)}$ otherwise. Similarly, $P_+(\gamma_1,\gamma_2)$ is the probability to obtain γ_2 from γ_1 when processes are created; recall that for every integer k, k processes are created (in $(\mathsf{q}_0,0)$) with probability $\lambda_+^k(1-\lambda_+)$.

As before, to simplify notations in the case where $(\boldsymbol{\gamma},y,(\mathsf{q},x),\delta,\boldsymbol{\gamma}')\in T^{(n)}$, we will write $\boldsymbol{\gamma}\xrightarrow{y,(\mathsf{q},x),\delta}\boldsymbol{\gamma}'$ or even simply $\boldsymbol{\gamma}\to\boldsymbol{\gamma}'$.

As in the static case, an *execution* in $\mathscr{DC}(B,\mathcal{P}_\Lambda^N)$ is a finite or infinite sequence of configurations $\rho=\boldsymbol{\gamma}_0\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2\dots$ starting in the initial configuration $\boldsymbol{\gamma}_0$ and such that for all i, if $\boldsymbol{\gamma}_i\in\Gamma^{(n)}$ then $\boldsymbol{\gamma}_i\to\boldsymbol{\gamma}_{i+1}$ and if $\boldsymbol{\gamma}_i\in\Gamma^{(p)}$, denoting d the distribution such that $(\boldsymbol{\gamma}_i,d)\in T^{(p)}$ then $d(\boldsymbol{\gamma}_{i+1})>0$. We denote by $\mathcal{E}$ the set of all executions.

2.4 Parameterized probabilistic verification problems

In this section, we consider a protocol $B=(\mathrm{Q}_B,\mathrm{Q}_B^{(p)},\mathrm{Q}_B^{(n)},\mathsf{b}_0,\mathbf{x}_B,\Sigma,\Delta_B)$ for the base and a protocol $\mathcal{P}=(\mathrm{Q},\mathrm{Q}^{(p)},\mathrm{Q}^{(n)},\mathsf{q}_0,\mathbf{x},\Sigma,\Delta)$ for the processes. The definitions here are given for a dynamic network but still hold for a static network considering $B=\mathcal{P}$ and $\lambda_-=\lambda_+=0$.

Recall that a scheduler resolves the non-determinism in Markov decision processes (see Definition 3.3). In networks, decisions are taken by a strategy.

Definition 2.4 (Strategy) *A* strategy *for a dynamic clique network is a scheduler for the Markov decision process representing its semantics. Formally a strategy is a function* $\sigma:\mathcal{E}\to(\mathbb{R}^+\times((\mathrm{Q}\cup\mathrm{Q}_B)\times\mathbb{R}^+)\times(\Delta\cup\Delta_B))\cup\{\mathbf{unlock}\}$*, specifying for each execution a delay, a process and a discrete action to perform next, and such that for all executions* $\rho=\boldsymbol{\gamma}_0...\boldsymbol{\gamma}_n$ *with* $\boldsymbol{\gamma}_n\in\Gamma^{(n)}$ *there exists a configuration* $\boldsymbol{\gamma}'\in\Gamma$ *and a transition* $\boldsymbol{\gamma}_n\xrightarrow{\sigma(\rho)}\boldsymbol{\gamma}'$.

In words, a strategy resolves the non-determinism by choosing in each configuration a delay τ, a process represented by the pair (q,x) followed by some discrete action $\delta\in(\Delta\cup\Delta_B)$, or if the system is in deadlock the strategy chooses the special action **unlock**. The condition $\boldsymbol{\gamma}_n\xrightarrow{\sigma(\rho)}\boldsymbol{\gamma}'$ ensures that σ only chooses delays and actions consistent with the semantics of the network or the special action **unlock** when the network is in deadlock so that all executions are infinite.

We are now in a position to introduce relevant verification questions for clique networks of probabilistic timed protocols. First, we define quantitative reachability problems.

Let $\mathsf{q}_f\in\mathrm{Q}\cup\mathrm{Q}_B$ be a state, and $\rho=\boldsymbol{\gamma}_0\boldsymbol{\gamma}_1\dots$ be an infinite execution of $\mathscr{DC}(B,\mathcal{P}_\Lambda^N)$. The execution ρ satisfies $\Diamond\mathsf{q}_f$, denoted $\rho\models\Diamond\mathsf{q}_f$, if there exists a configuration $\boldsymbol{\gamma}_i=((\mathsf{b},\mathbf{x}_B),\gamma)$ along ρ with $\boldsymbol{\gamma}_i(\mathsf{q}_f)>0$.

Given a strategy σ for $\mathscr{DC}(B,\mathcal{P}_\Lambda^N)$, we write $\mathbb{P}_\sigma(\mathscr{DC}(B,\mathcal{P}_\Lambda^N)\models\Diamond q_f)$ for the probability, under the scheduler σ, of the set of infinite executions ρ such that $\rho\models\Diamond q_f$.

The aim of this work is to verify a good property in the networks independently of the number of participants. Dually, this amounts to check for the existence of a network size satisfying a bad property. We thus define the following family of decision problems to decide the reachability of a particular control state for an arbitrary number of components. Formally, for $\exists\!\!\!\forall \in \{\exists, \forall\}$ a quantifier, $p \in [0,1]$ a probability threshold and $\sim \in \{<, =, >\}$ a comparison operator:

$REACH^{\exists\!\!\!\forall}_{\sim p}(\mathscr{DC})$
Input: Two probabilistic timed protocols B and $\mathcal{P}$, a rate pair Λ, and a control state $\mathsf{q}_f \in \mathrm{Q} \cup \mathrm{Q}_B$.
Question: Does there exist $N \in \mathbb{N}_{>0}$ such that $\exists\!\!\!\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}^N_\Lambda) \models \Diamond \mathsf{q}_f) \sim p$?

Second, we define the quantitative synchronization problem. Let $T_B \subseteq \mathrm{Q}_B$, $T \subseteq \mathrm{Q}$ be subsets of states, and $\rho = \boldsymbol{\gamma}_0 \boldsymbol{\gamma}_1 \dots$ be an infinite execution of $\mathscr{DC}(B, \mathcal{P}^N_\Lambda)$. The execution ρ satisfies $\Diamond\!\!\downarrow (T_B, T)$, denoted $\rho \models \Diamond\!\!\downarrow (T_B, T)$, if there exists a configuration $\boldsymbol{\gamma}_i$ along ρ such that, whenever $\boldsymbol{\gamma}_i(\mathsf{q}) > 0$ then $\mathsf{q} \in T_B \cup T$.

Given a strategy σ for $\mathscr{DC}(B, \mathcal{P}^N_\Lambda)$, we write $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}^N_\Lambda) \models \Diamond\!\!\downarrow (T_B, T))$ for the probability under σ of the set of infinite executions ρ such that $\rho \models \Diamond\!\!\downarrow (T_B, T)$.

We define the following family of decision problems to answer whether the processes can gather at the same time in some target states for an arbitrary number of components. For $\exists\!\!\!\forall \in \{\exists, \forall\}$ a quantifier, $p \in [0,1]$ a probability threshold and $\sim \in \{<, =, >\}$ a comparison operator:

$SYNC^{\exists\!\!\!\forall}_{\sim p}(\mathscr{DC})$
Input: Two probabilistic timed protocols B and $\mathcal{P}$, a rate pair Λ, and a pair of sets of control states $T_B \subseteq \mathrm{Q}_B$, $T \subseteq \mathrm{Q}$.
Question: Does there exist $N \in \mathbb{N}_{>0}$ such that $\exists\!\!\!\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}^N_\Lambda) \models \Diamond\!\!\downarrow (T_B, T)) \sim p$?

Notice that here the problems are defined as the existence of an initial network size that satisfies the property. But since we consider all the variants for $\exists\!\!\!\forall \in \{\exists, \forall\}$, $p \in [0,1]$ and $\sim \in \{<, =, >\}$, the problems where one asks whether the property is satisfied for any number of processes can be obtained by negation. For example, if the answer is negative for $REACH^{\exists}_{=1}(\mathscr{DC})$ that means that for all $N \in \mathbb{N}_{>0}$ and for all strategies σ, $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}^N_\Lambda) \models \Diamond \mathsf{q}_f) < 1$.

The decidability results obtained are presented in Table III.1 where decidable problems are in green and undecidable ones in red.

3 Parameterized verification of static clique networks of probabilistic timed protocols

In this section, we consider the static case *i.e.*, when $\lambda_- = \lambda_+ = 0$, and $B = \mathcal{P}$. We start by establishing simple decidability results, before showing the undecidability of the other problems.

			$=0$	$=1$	>0	<1	$=p$	$>p$	$<p$
REACH	$\mathscr{C}$	$\exists$	Theorem 3.4	Theorem 3.3	Theorem 3.2	Theorem 3.4	Theorem 3.3	open	Theorem 3.4
		$\forall$	Theorem 3.1	Theorem 3.5	Theorem 3.5	Theorem 3.1	Theorem 3.5	Theorem 3.5	Theorem 3.1
	$\mathscr{DC}$	$\exists$	Theorem 4.2	Theorem 4.4	Theorem 4.1	Theorem 4.3	open	open	open
		$\forall$	Theorem 4.1	Theorem 4.3	Theorem 4.2	Theorem 4.4	open	open	open
SYNC	$\mathscr{C}$	$\exists^\forall$	Theorem 3.6						
	$\mathscr{DC}$	$\exists^\forall$	Theorem 4.6				open		

Table III.1: Summary of the decidability results of this chapter. In green decidable problems and in red undecidable ones. In this table $p \in]0,1[$.

3.1 Some decidability results using monotonicity

First, let us establish a monotonicity result.

Lemma 3.1 *For every strategy with N processes σ_N, there exist a strategy σ_{N+1} for $N+1$ processes such that $\mathbb{P}_{\sigma_N}(\mathscr{C}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) \leq \mathbb{P}_{\sigma_{N+1}}(\mathscr{C}(\mathcal{P}^{N+1}) \models \Diamond \mathsf{q}_f)$.*

Proof Any strategy σ_N in $\mathscr{C}(\mathcal{P}^N)$ can be mimicked by a strategy σ_{N+1} in $\mathscr{C}(\mathcal{P}^{N+1})$ by ignoring a process, until a deadlock is reached in $\mathscr{C}(\mathcal{P}^N)$, after what the choice in σ_{N+1} can be arbitrary.

The probability to reach q_f under σ_{N+1} is thus greater than under σ_N. Indeed, σ_{N+1} captures all executions reaching q_f with the same probability, and all executions reaching a deadlock are extended (thanks to the additional process); moreover, the ignored processes may reach q_f (due to receptions) along executions previously avoiding q_f. □

Using this Lemma, we obtain the following decidability result.

Theorem 3.1 *The problems $REACH^{\forall}_{=0}(\mathscr{C})$, $REACH^{\forall}_{<p}(\mathscr{C})$ for all $p \in [0,1]$ are decidable for static clique networks of probabilistic timed protocols.*

Proof Lemma 3.1 allows us to bring the parameterized problems back to their equivalent problems in finite Markov decision process. Indeed, since the parameterized problems ask for the existence of a network size, one can first check if the property already hold for a network composed of only one process, hence in a finite MDP. Moreover, if the property does not hold, for example, if $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^1) \models \Diamond \mathsf{q}_f) = 0$ does not hold, that means that $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^1) \models \Diamond \mathsf{q}_f) > 0$. Then applying Lemma 3.1 multiple times, we obtain that for all network sizes N, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) > 0$. Thus the answer of the parameterized problem $REACH^{\forall}_{=0}(\mathscr{C})$ is negative. Hence the parameterized problem $REACH^{\forall}_{=0}(\mathscr{C})$ is equivalent to $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^1) \models \Diamond \mathsf{q}_f) = 0$.

With the same arguments one can show that for any threshold $p \in [0,1]$, $\exists N \in \mathbb{N},\ \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) < p$ is equivalent to $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^1) \models \Diamond \mathsf{q}_f) < p$.

The decidability of $REACH^{\forall}_{=0}(\mathscr{C})$, and $REACH^{\forall}_{<p}(\mathscr{C})$ thus derives from the decidability of the following verification questions for finite state timed Markov decision process [KNPS08, BK08]: $\forall \sigma,\ \mathbb{P}_\sigma(M \models \Diamond \mathsf{q}_f) = 0$ (resp. $< p$). □

$REACH^{\exists}_{>0}(\mathscr{C})$ is an other problem that can be easily resolved thanks to preexisting works. Indeed, this problem asks for the existence of a finite path to the target state and is thus equivalent to the reachability problem in the non-probabilistic case, hence this problem is decidable as stated in the following theorem.

Theorem 3.2 *The problem $REACH^{\exists}_{>0}(\mathscr{C})$ is decidable for static clique networks of probabilistic timed protocols.*

Proof Observe that, for a fixed size $N \in \mathbb{N}$, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) > 0$ if and only if there exist an execution ρ in $\mathscr{C}(\mathcal{P}^N)$ with $\rho \models \Diamond \mathsf{q}_f$. The decidability of $REACH^{\exists}_{>0}(\mathscr{C})$ is therefore a consequence of the decidability of the parameterized reachability problem in the non-probabilistic case, established in [ADR+11]. □

3.2 Undecidability results

We now consider the remaining cases, and prove their undecidability, already for the restricted class of untimed probabilistic protocols. Untimed probabilistic protocols can be seen as probabilistic timed protocols for which all the guards are the trivial guard. Since the clock does not play any role in this protocol, in this section we omit the clock (together with the guards and updates) in the description of the protocols.

In order to show the undecidability of the remaining problems, we use a reduction from deterministic Minsky machines (see Section I.4.4) for which the halting problem and the boundedness problem are known to be undecidable (Theorem 4.4 [Min67]). Each of the problems involves a different reduction relying on its specificity but we use the same backbone that is weak simulation of the Minsky machine thanks to clique networks of probabilistic (untimed) protocols. This general idea is presented in the following, and the proofs of undecidability of each case are stated in Theorems 3.3, 3.4, and 3.5.

Simulation of a Minsky machine. Let $\mathcal{M} = (\mathrm{K}, \mathsf{L}_0, \mathsf{L}_{acc}, \mathsf{c}_1, \mathsf{c}_2)$ be a Minsky machine, we sketch here how to build a protocol that weakly simulates $\mathcal{M}$, *i.e.* that either simulates correctly $\mathcal{M}$ if possible, or for which we are able to detect an error in the simulation.

First, notice that in a clique network it is easy to isolate one process *i.e.* to reach a configuration in which there is one and only one process in a particular state. Indeed, by making a process perform a broadcast, since all the other processes receive this broadcast, we can ensure that the process is alone in a particular state. We will use this idea to simulate increment and decrement of the counter values, and to initially isolate one process to keep track of the current instruction.

At the beginning of the simulation, one process is isolated to play the role of the *controller*, that keeps track of the current instruction in $\mathcal{M}$. The other processes will serve to encode the values of the counters, and are grouped in state idle. The increment of counter c_i is represented by moving a process from idle to state c_i where the counter value is encoded. This can be done by two communications: one from the controller to the processes in idle, followed by one from one "counter" process to the controller.

Symmetrically, the decrement of counter c_i is represented by moving a process state c_i to idle.

The test to zero of counter c_i is more tricky since there is no way to test whether there is a process in a given state or not. However, we rely on non-deterministic guesses of the controller to establish whether the counter value is zero or not. The idea is to detect errors when they occur. If the guess is zero while $c_i > 0$, all processes in c_i move to an error state $\mathsf{err_z}$ that will be used to detect that an error occurred in the simulation. Symmetrically, if the guess is not zero while $c_i = 0$, the unfeasibility of the decrement will allow us to detect the error. If the guess is correct, the simulation proceeds, and no process is in an error state.

This general reduction gives us a way to weakly simulate a Minsky machine. In the following, we tune this sketch in order to show undecidability of the different problems. But first, we define more formally this construction to capture the important properties of the reductions.

Formally, from a Minsky machine $\mathcal{M} = (\mathrm{K}, \mathsf{L}_0, \mathsf{L}_{acc}, c_1, c_2)$, we build the non probabilistic protocol $\mathcal{P}_\mathcal{M} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \Sigma, \Delta)$ represented in Figure 3.5, and such that:

- $\mathrm{Q}^{(p)} = \emptyset$,
- $\mathrm{Q}^{(n)} = \mathrm{K} \cup \mathrm{K} \times \mathrm{K} \cup \bigcup_{i=1}^{2} \{\mathsf{c}_i, \mathsf{c}_i{+}{+}, \mathsf{c}_i{-}{-}\} \cup \{\mathsf{idle}, \mathsf{err_z}, \mathsf{q}_0\}$,
- $\Sigma = \bigcup_{i=1}^{2} \{c_i{+}{+}, c_i{-}{-}, z(c_i)\} \cup \{ok, ctrl\}$,
- $\Delta = \Delta_i \cup \Delta_c \cup \Delta_\mathrm{K}$ is partitioned into three parts:

  Initialization: $\Delta_i = \{(\mathsf{q}_0, !!ctrl, \mathsf{L}_0), (\mathsf{q}_0, ??ctrl, \mathsf{idle})\}$.
  The initialization transitions are represented in purple in the figure.

  Counter part: Δ_c is composed of the following transitions; $(\mathsf{c}_i, ??z(c_i), \mathsf{err_z})$ for tests to zero; $(\mathsf{c}_i, ??c_i{-}{-}, \mathsf{c}_i{-}{-})$, $(\mathsf{c}_i{-}{-}, ??ok, \mathsf{c}_i)$, and $(\mathsf{c}_i{-}{-}, !!ok, \mathsf{idle})$ for decrements; and $(\mathsf{idle}, ??c_i{+}{+}, \mathsf{c}_i{+}{+})$, $(\mathsf{c}_i{+}{+}, ??ok, \mathsf{idle})$, and $(\mathsf{c}_i{+}{+}, !!ok, \mathsf{c}_i)$ for increments.
  The increment transitions are represented in red, and the decrement transitions are represented in blue in the figure.

  Controller part: Δ_K is composed of the following transitions:

  - for all increment instructions $\mathsf{L} : c_i := c_i + 1;\ \mathtt{goto}\ \mathsf{L}'$ there are two transitions $(\mathsf{L}, !!c_i{+}{+}, (\mathsf{L}, \mathsf{L}'))$ and $((\mathsf{L}, \mathsf{L}'), ??ok, \mathsf{L}')$ and
  - for all decrement instructions $\mathsf{L} : \mathtt{if}\ c_i = 0\ \mathtt{then\ goto}\ \mathsf{L}'\ \mathtt{else}\ c_i := c_i - 1;\ \mathtt{goto}\ \mathsf{L}''$ there are transitions $(\mathsf{L}, !!z(c_i), (\mathsf{L}, \mathsf{L}'))$, $((\mathsf{L}, \mathsf{L}'), \varepsilon, \mathsf{L}')$, $(\mathsf{L}, !!c_i{-}{-}, (\mathsf{L}, \mathsf{L}''))$, and $((\mathsf{L}, \mathsf{L}''), ??ok, \mathsf{L}'')$.

  The test to zero transitions are represented in green in the figure.

We now give some properties on the executions of $\mathscr{C}(\mathcal{P}_\mathcal{M}^N)$. We say that a configuration γ of the network encodes a configuration $\pi = (\mathsf{L}, v_1, v_2)$ of the Minsky machine $\mathcal{M}$ if $\gamma(\mathsf{L}) = 1$, $\gamma(\mathsf{c}_1) = v_1$, $\gamma(\mathsf{c}_2) = v_2$ and for all $\mathsf{q} \in \mathrm{Q} \setminus \{\mathsf{L}, \mathsf{c}_1, \mathsf{c}_2, \mathsf{idle}\}$ we have $\gamma(\mathsf{q}) = 0$. In

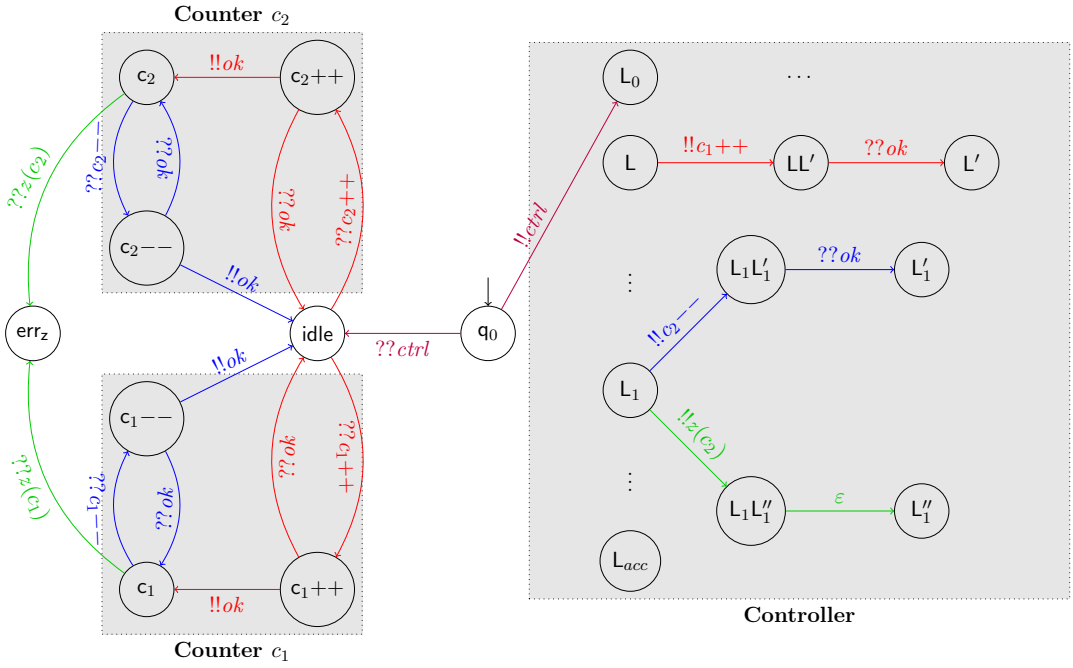


Figure 3.5: General framework of the reductions.

words, if the controller points to the right instruction, the number of processes encoding each counter is correct and all remaining processes are in state idle.

We say that an execution $\rho = \gamma_0\gamma_1\ldots$ of the network simulates a run of the Minsky machine $\pi_0\pi_1\ldots$ if for all finite prefixes $\pi_0\ldots\pi_n$ of the run there exists a sub-sequence $\gamma_{j_0}\ldots\gamma_{j_n}$ of ρ such that for all $k \in [0\ldots n]$, γ_{j_k} encodes π_k.

Let us now detail how the reduction works:

Initialization: From the initial configuration, the only possible transition is for one process to broadcast the message *ctrl* and move to L_0; all the other processes receive this message and move to the state idle. Notice that the resulting configuration encodes the initial configuration π_0 of $\mathcal{M}$.

Increment: In a configuration in which the controller is in a state representing an increment instruction, the only possible transition is, for the controller, to broadcast the message $c_i{+}{+}$ that is received by all processes in state idle. One of these processes will then broadcast the message *ok* making all the processes in state $\mathsf{c}_i{+}{+}$ move back to idle and the controller moves to the next instruction state. In the new configuration, there is now one more process in state c_i (the process that broadcasts *ok*). Notice that in the case where there was no process in state idle, no process can send the message *ok* and the network reaches a deadlock configuration.

Decrement: In a configuration in which the controller is in a state representing a

decrement instruction, a possible transition is, for the controller, to broadcast the message $c_i{-}{-}$ that is received by all the processes in state c_i, one of these processes will then broadcast the message *ok* making all the processes in state $\mathsf{c}_i{-}{-}$ move back to c_i and the controller moves to the next instruction state. In the new configuration, there is now one less process in state c_i (the process that broadcasts *ok*). Notice that in the case where the controller chooses the decrement although there is no process in state c_i, no process can send the message *ok* and the network reaches a deadlock configuration.

Zero test: Lastly, for a configuration in which the controller is in a state representing a decrement instruction, the only other possible transition is for the controller to broadcast the message $z(c_i)$, and then move to the next state representing the next instruction. However, notice that in the case where the controller makes a wrong guess (*i.e.* assumes the counter to be zero while it is not) all the processes in state c_i receive the message $z(c_i)$ and thus move to $\mathsf{err_z}$, and since it is a sink state, they will stay there for the rest of the execution.

Example 3.1 *Consider the deterministic Minsky machine with initial label* L_0*, accepting label* L_{acc} *and composed of the following instructions:*

$$\begin{aligned}
&\mathsf{L}_0 : c_1 := c_1 + 1;\ \texttt{goto}\ \mathsf{L}_1\\
&\mathsf{L}_1 : c_1 := c_1 + 1;\ \texttt{goto}\ \mathsf{L}_2\\
&\mathsf{L}_2 : \texttt{if}\ c_1 = 0\ \texttt{then goto}\ \mathsf{L}_1\ \texttt{else}\ c_1 := c_1 - 1;\ \texttt{goto}\ \mathsf{L}_3\\
&\mathsf{L}_3 : \texttt{if}\ c_2 = 0\ \texttt{then goto}\ \mathsf{L}_{acc}\ \texttt{else}\ c_2 := c_2 - 1;\ \texttt{goto}\ \mathsf{L}_1
\end{aligned}$$

Its associated run is $\Pi = (\mathsf{L}_0, 0, 0)(\mathsf{L}_1, 1, 0)(\mathsf{L}_2, 2, 0)(\mathsf{L}_3, 1, 0)(\mathsf{L}_{acc}, 1, 0)$.

We give examples of possible executions for a clique network composed of processes running the protocol $\mathcal{P}_{\mathcal{M}}$*. First, an execution simulating* Π*, with 4 processes:*

$$\begin{aligned}
\langle \mathsf{q}_0^4 \rangle &\xrightarrow{!!ctrl} \langle \mathsf{L}_0, \mathsf{idle}^3 \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_0\mathsf{L}_1, \mathsf{c}_1{+}{+}^3 \rangle \xrightarrow{!!ok} \langle \mathsf{L}_1, \mathsf{c}_1, \mathsf{idle}^2 \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_1\mathsf{L}_2, \mathsf{c}_1, \mathsf{c}_1{+}{+}^2 \rangle\\
&\xrightarrow{!!ok} \langle \mathsf{L}_1, \mathsf{c}_1^2, \mathsf{idle} \rangle \xrightarrow{!!c_1{-}{-}} \langle \mathsf{L}_2\mathsf{L}_3, \mathsf{c}_1{-}{-}^2, \mathsf{idle} \rangle \xrightarrow{!!ok} \langle \mathsf{L}_3, \mathsf{c}_1, \mathsf{idle}^2 \rangle \xrightarrow{!!z(c_2)} \langle \mathsf{L}_3\mathsf{L}_{acc}, \mathsf{c}_1, \mathsf{idle}^2 \rangle\\
&\xrightarrow{\varepsilon} \langle \mathsf{L}_{acc}, \mathsf{c}_1, \mathsf{idle}^2 \rangle
\end{aligned}$$

Then, an execution with 2 processes simulating Π*, but lacking processes to do an increment, hence reaching a deadlock:*

$$\langle \mathsf{q}_0^2 \rangle \xrightarrow{!!ctrl} \langle \mathsf{L}_0, \mathsf{idle} \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_0\mathsf{L}_1, \mathsf{c}_1{+}{+} \rangle \xrightarrow{!!ok} \langle \mathsf{L}_1, \mathsf{c}_1 \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_1\mathsf{L}_2, \mathsf{c}_1 \rangle$$

Lastly, an execution with 4 processes making a wrong guess for a zero test, hence reaching the error state:

$$\begin{aligned}
\langle \mathsf{q}_0^4 \rangle &\xrightarrow{!!ctrl} \langle \mathsf{L}_0, \mathsf{idle}^3 \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_0\mathsf{L}_1, \mathsf{c}_1{+}{+}^3 \rangle \xrightarrow{!!ok} \langle \mathsf{L}_1, \mathsf{c}_1, \mathsf{idle}^2 \rangle \xrightarrow{!!c_1{+}{+}} \langle \mathsf{L}_1\mathsf{L}_2, \mathsf{c}_1, \mathsf{c}_1{+}{+}^2 \rangle\\
&\xrightarrow{!!ok} \langle \mathsf{L}_1, \mathsf{c}_1^2, \mathsf{idle} \rangle \xrightarrow{!!z(c_1)} \langle \mathsf{L}_2\mathsf{L}_1, \mathsf{err_z}^2, \mathsf{idle} \rangle \xrightarrow{!!\varepsilon} \langle \mathsf{L}_1, \mathsf{err_z}^2, \mathsf{idle} \rangle \dots
\end{aligned}$$

The following proposition summarizes the properties of the reduction. The proofs are straightforward from the definition and are omitted.

Proposition 3.1 *Let $\mathcal{M} = (\mathrm{K}, \mathsf{L}_0, \mathsf{L}_{acc}, c_1, c_2)$ be a Minsky machine and $\pi_0\pi_1\dots$ be its associated run. In $\mathscr{C}(\mathcal{P}_{\mathcal{M}}^N)$, the clique network composed of N processes running protocol $\mathcal{P}_{\mathcal{M}}$, the following properties hold: for an execution $\rho = \gamma_0 \dots \gamma_n$*

1. *There is a unique process playing the role of the controller,* i.e. *if $n \geq 1$ then $\sum_{\mathsf{q}\in \mathrm{K}\cup \mathrm{K}\times\mathrm{K}} \gamma_n(\mathsf{q}) = 1$.*
2. *When the controller is in a state representing an instruction, there are no processes in intermediary states $\mathsf{c}_i{+}{+}$ and $\mathsf{c}_i{-}{-}$ for $i \in \{1,2\}$,* i.e. *if $\sum_{\mathsf{q}\in \mathrm{K}} \gamma_n(\mathsf{q}) = 1$ then $\sum_{\mathsf{q}\in\{\mathsf{c}_i{+}{+},\mathsf{c}_i{-}{-}|i\in\{1,2\}\}} \gamma_n(\mathsf{q}) = 0$.*
3. *For each infinite execution $\rho' = \gamma_0\gamma_1\dots$, either:*
   - *ρ'* **simulates the run** *$\pi_0\pi_1\dots$, or*
   - **there are not enough processes to simulate the run**, i.e. *there is an index k such that γ_k encodes a configuration $\pi = (\mathsf{L}, v_1, v_2)$ and such that L corresponds to an increment and $v_1 + v_2 = N-1$, hence the execution reaches a deadlock in configuration γ_{k+1}, or*
   - **the controller wrongly guessed zero**, i.e *there is an index k such that γ_k encodes a configuration $\pi = (\mathsf{L}, v_1, v_2)$ and such that L corresponds to a test to zero decrement of counter $i \in \{1,2\}$ and $v_i > 0$ and the next action of the controller is a broadcast of $z(c_i)$ then all the processes in state c_i move to err_z and stay there for the rest of the execution, or*
   - **the controller did not guess zero when it was**, i.e *there is an index k such that γ_k encodes a configuration $\pi = (\mathsf{L}, v_1, v_2)$ and such that L corresponds to a test to zero decrement of counter $i \in \{1,2\}$ and $v_i = 0$ and the next action of the controller is a broadcast of $c_i{-}{-}$, then, the execution reaches a deadlock in configuration γ_{k+1}.*
4. *There exists an execution that simulates the run $\pi_0\pi_1\dots$ if and only if $N - 1 \geq \max_k c_1(\pi_k) + c_2(\pi_k)$.*

In the following paragraphs, we explain how to use and adapt, for the particularity of each problem, the above reduction in order to show undecidability of the reachability problems.

Undecidability of $REACH^{\exists}_{=1}(\mathscr{C})$. We first focus on the problem $REACH^{\exists}_{=1}(\mathscr{C})$, *i.e.* whether, for some number of processes, there exists a strategy that allows to reach, almost surely, a goal state. We then prove the undecidability of the problems $REACH^{\exists}_{=p}(\mathscr{C})$ for $p \in]0,1]$.

To establish the undecidability of $REACH^{\exists}_{=1}(\mathscr{C})$, we slightly modify $\mathcal{P}_{\mathcal{M}}$ into $\mathcal{P}' = (\mathrm{Q}', \mathrm{Q}^{(p)\prime}, \mathrm{Q}^{(n)\prime}, \mathsf{q}_0, \Sigma', \Delta')$ such that: $\mathcal{P}'$ contains now a probabilistic state $\mathrm{Q}^{(p)\prime} = \{\mathsf{p}\}$

and two more non-deterministic states $\mathsf{Q}^{(n)'} = \mathsf{Q}^{(n)} \cup \{\mathsf{reset}, \mathsf{w}\}$. The distribution associated with p is d (*i.e* $(\mathsf{p}, d) \in \Delta'$) and is such that $d(\mathsf{w}) = 0.5$ and $d(\mathsf{reset}) = 0.5$. Moreover, $\mathcal{P}'$ contains the additional transitions $(\mathsf{L}_{acc}, \varepsilon, \mathsf{p})$, $(\mathsf{reset}, !!r, \mathsf{L}_0)$ and $(\mathsf{c}_i, ??r, \mathsf{idle})$.

See Figure 3.6 for a representation of $\mathcal{P}'$. In this figure, the new elements in comparison with Figure 3.5 are in brown. The idea is that after reaching L_{acc} with probability 0.5, the controller moves to the state w, and with probability 0.5, it starts again the computation from L_0. However, since err_z is a sink state, the new computation may not use the processes stuck in err_z. Hence, if the execution was not a simulation of the Minsky machine, the new computation will start with strictly less processes.

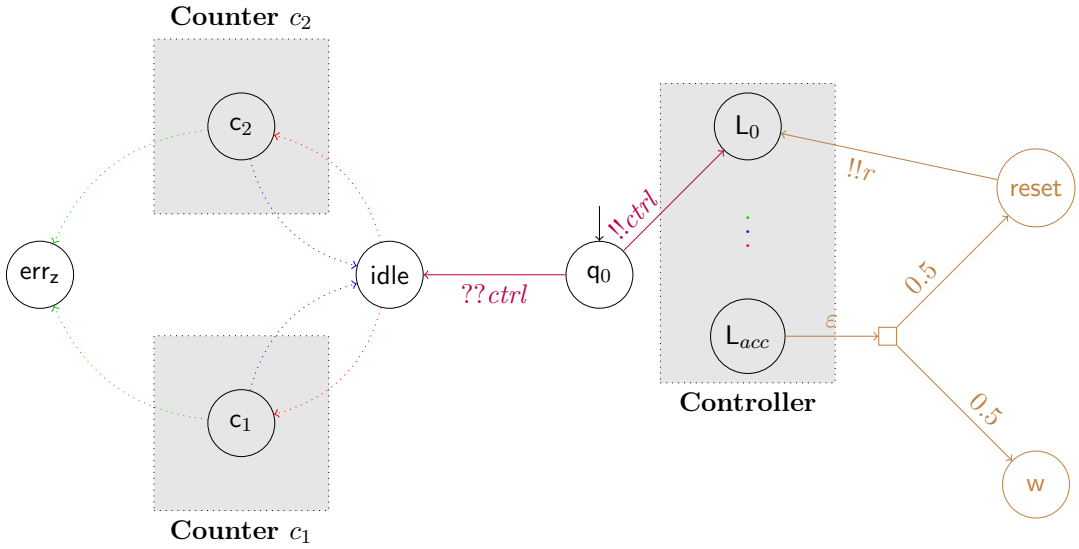


Figure 3.6: Reduction for $REACH^{\exists}_{=1}(\mathscr{C})$.

In order to prove that $REACH^{\exists}_{=1}(\mathscr{C})$ is undecidable, we show the following lemma.

Lemma 3.2

$$\exists N \in \mathbb{N}_{>0},\ \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(N^{\mathcal{P}'}) \models \Diamond \mathsf{w}) = 1 \iff \mathcal{M} \textit{ terminates.}$$

Proof Assume that $\mathcal{M}$ terminates and let $\Pi = \pi_0 \dots \pi_n$ be its run. Let N be an integer greater than the maximal value of the sum of the counters along Π *e.g.* $N = \max_j(c_1(\pi_j) + c_2(\pi_j)) + 1$. From point 4 of Proposition 3.1, we know that there is an execution ρ_Π in $\mathscr{C}(\mathcal{P}^N_\mathcal{M})$ that simulates Π. But since $\mathcal{P}_\mathcal{M}$ is a sub-protocol of $\mathcal{P}'$, we can define a strategy σ such that when the controller is in a state of $\mathsf{Q} \setminus \{\mathsf{L}_{acc}\}$, σ plays as ρ_Π and whenever the controller reaches L_{acc}, the strategy makes the controller go to the probabilistic state p; if the probabilistic transition leads to reset then σ starts over the computation by choosing the broadcast of r. Hence from L_0, under σ, the controller reaches L_{acc} and no process is in the error state err_z. From L_{acc} with probability 0.5, it

moves to w and with probability 0.5, it starts again with the same number of processes. Hence, the probability to eventually reach w is 1.

Assume now that $\mathcal{M}$ does not terminate, and consider an execution ρ of $\mathscr{C}(\mathcal{P}'^N)$ for an arbitrary number of processes $N \in \mathbb{N}$. Assume that ρ ends in L_{acc}. This execution does not faithfully simulate Π and thus, some processes must be in the error state $\mathsf{err_z}$. As a consequence, each time the network tries to reach w with probability 0.5, it needs to retry from L_0 with at least one more process stuck in $\mathsf{err_z}$. Therefore, each execution of $\mathscr{C}(\mathcal{P}'^N)$ cannot visit L_{acc} more than $N-1$ times. We thus obtain $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{w}) \leq 0.5^{N-1}$, which concludes the proof of Lemma 3.2. □

Theorem 3.3 *The problems $REACH^{\exists}_{=p}(\mathscr{C})$ for $p \in]0,1]$ are undecidable for static clique networks of probabilistic (timed) protocols.*

Proof Using a small gadget (represented in Figure 3.7) one can, from a protocol $\mathcal{P}$ and $p \in]0,1[$, design a new protocol $\mathcal{P}_p$ that behaves exactly as $\mathcal{P}$ with probability p and deadlocks with probability $1-p$. This gadget consists in a new initial state from which the only possibility is to broadcast a message s, after that the process has probability p to broadcast again the message s and probability $1-p$ to reach a deadlock state; the other processes in the initial state, at the reception of s move to a state w from which the reception of s leads to the initial state of $\mathcal{P}$; if the second s is not broadcast, the processes are stuck in the state w.

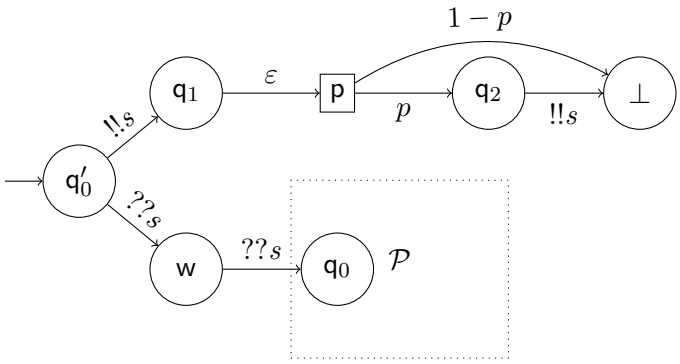


Figure 3.7: Gadget to run $\mathcal{P}$ with probability $p \in]0,1[$.

This gadget, together with Lemma 3.2, gives the undecidability of the problems $REACH^{\exists}_{=p}(\mathscr{C})$ for $p \in]0,1]$, and thus concludes the proof of Theorem 3.3. □

Undecidability of $REACH^{\exists}_{=0}(\mathscr{C})$. We now focus on the problem $REACH^{\exists}_{=0}(\mathscr{C})$, *i.e.* whether, for some number of processes, there exists a strategy that allows to avoid, almost surely, a goal state. We then prove undecidability of the problems $REACH^{\exists}_{=0}(\mathscr{C})$ and $REACH^{\exists}_{<p}(\mathscr{C})$ for $p \in]0,1]$.

To prove undecidability of $REACH^{\exists}_{=0}(\mathscr{C})$, we redefine $\mathcal{P}'$ from $\mathcal{P}_{\mathcal{M}}$ by adding for all $\mathsf{q} \in \mathrm{K} \times \mathrm{K}$ a transition $(\mathsf{q}, \varepsilon, \mathsf{err_z})$. See Figure 3.8 for a representation of $\mathcal{P}'$. In this

figure, the new elements in comparison with Figure 3.5 are in brown. Note that $\mathcal{P}'$ is not probabilistic in this case. The idea is that in this protocol, the only way to avoid the state $\mathsf{err_z}$ is to simulate the run of $\mathcal{M}$. However, we know that this is only possible if the counters are bounded.

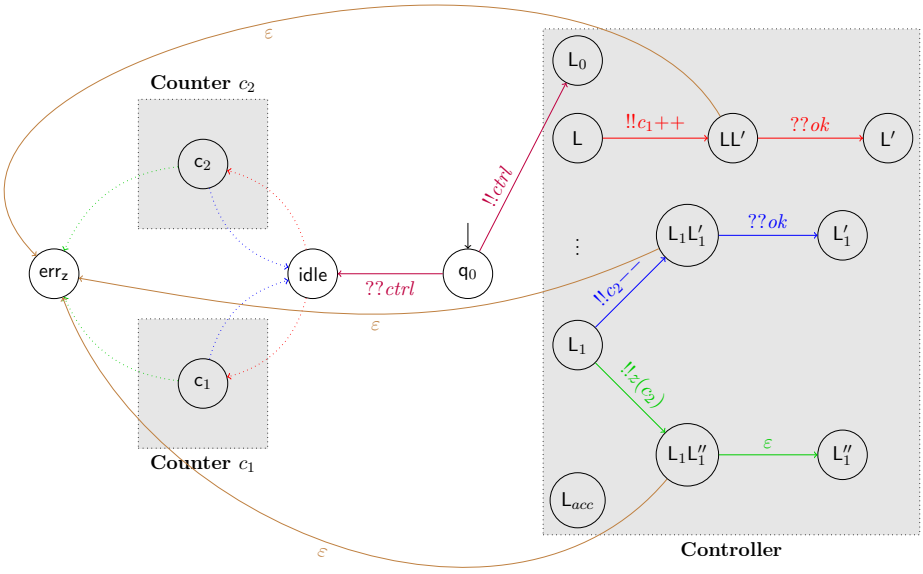


Figure 3.8: Reduction for $REACH^{\exists}_{=0}(\mathscr{C})$.

In order to prove that $REACH^{\exists}_{=0}(\mathscr{C})$ is undecidable, we show the following lemma.

Lemma 3.3

$$\exists N \in \mathbb{N}_{>0},\ \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0 \iff \mathcal{M} \textit{ is bounded.}$$

Proof Assume that $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0$. Since $\mathcal{P}'$ is not probabilistic, it means that there is a maximal execution that avoids $\mathsf{err_z}$. Indeed, in the absence of probabilities, a strategy leads to a unique execution, hence under a strategy the probability to reach $\mathsf{err_z}$ is either 0 or 1. Since, with the transition added in $\mathcal{P}'$, the controller cannot be blocked elsewhere than in L_{acc}. The fact that an execution avoids $\mathsf{err_z}$ means that the execution faithfully simulates Π. Using point 4 of Proposition 3.1, we know that, for such an execution to exist, necessarily $N-1 \geq \max_j(c_1(\pi_j)+c_2(\pi_j))$. Hence, $\mathcal{M}$ is bounded.

Assume that $\mathcal{M}$ is bounded and let $N = \max_j(c_1(\pi_j) + c_2(\pi_j)) + 1$. By point 4 of Proposition 3.1, we know that there exists an execution ρ_Π that faithfully simulates

Π, and hence avoids $\mathsf{err_z}$. Thus, defining σ as the strategy that simulates $\mathcal{M}$, we have $\mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0$, which concludes the proof of Lemma 3.3. □

Theorem 3.4 *The problems $REACH^{\exists}_{=0}(\mathscr{C})$ and $REACH^{\exists}_{<p}(\mathscr{C})$ for $p \in]0,1]$ are undecidable for static clique networks of probabilistic (timed) protocols.*

Proof Remark that the proof of Lemma 3.3 also gives a proof for the undecidability of $REACH^{\exists}_{<p}(\mathscr{C})$ for $p \in]0,1]$. Hence we showed Theorem 3.4. Indeed, we recall that since $\mathcal{P}'$ is not probabilistic, the probability to reach $\mathsf{err_z}$ is either 0 or 1. □

Undecidability of $REACH^{\forall}_{>0}(\mathscr{C})$. We now focus on the problem $REACH^{\forall}_{>0}(\mathscr{C})$, *i.e.* whether, for some number of processes, all the strategies reach a target state with positive probability. We then prove the undecidability of the problems $REACH^{\forall}_{>p}(\mathscr{C})$ for $p \in [0,1[$ and $REACH^{\forall}_{=p}(\mathscr{C})$ for $p \in]0,1]$.

For these cases, we define $\mathcal{P}'$ from $\mathcal{P}_\mathcal{M}$ by adding an error state $\mathsf{err_c}$, for all $\mathsf{q} \in \mathrm{K} \cup \mathrm{K} \times \mathrm{K}$ a transition $(\mathsf{q}, \varepsilon, \mathsf{err_c})$, and the transitions: $(\mathsf{err_c}, !!err, \mathsf{err_c})$, $(\mathsf{idle}, ??err, \mathsf{err_z})$, $(\mathsf{c}_i{+}{+}, ??err, \mathsf{err_z})$ and $(\mathsf{c}_i{-}{-}, ??err, \mathsf{err_z})$ for $i \in \{1,2\}$. See Figure 3.9 for a representation of $\mathcal{P}'$. Note that here again $\mathcal{P}'$ is not probabilistic. The idea is to force all the executions to eventually reach $\mathsf{err_z}$ except for: 1) the executions that simulate the run of $\mathcal{M}$ when it is infinite and 2) the executions that simulate the run but lack processes to perform an increment.

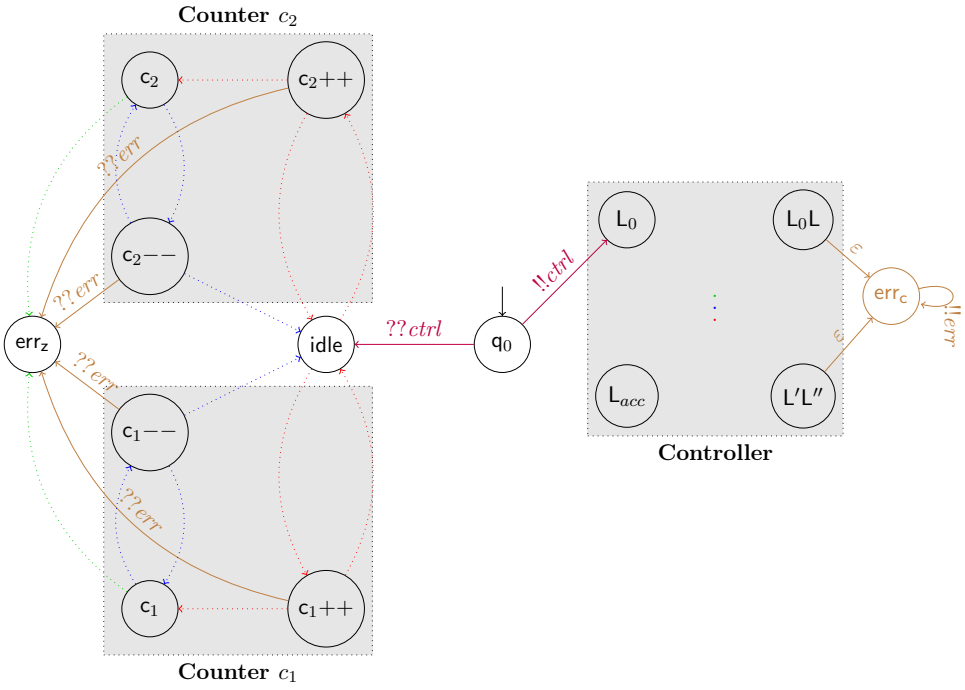


Figure 3.9: Reduction for $REACH^{\forall}_{>0}(\mathscr{C})$.

In order to prove that $REACH^{\inf}_{>0}(\mathscr{C})$ is undecidable, we show the following lemma.

Lemma 3.4

$$\exists N \in \mathbb{N}_{>0},\ \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) > 0 \iff \mathcal{M} \text{ terminates.}$$

Proof Assume that $\mathcal{M}$ terminates, we let $N = \max_j(c_1(\pi_j)+c_2(\pi_j))+2$. We distinguish between executions of $\mathscr{C}(\mathcal{P}'^N)$ that faithfully simulate Π and others. Any execution that faithfully simulates Π reaches L_{acc} with at least one process in state idle. From L_{acc}, the only choice is to take one of the additional transitions and move to $\mathsf{err_c}$ and then to send the message *err*, forcing the processes in idle to move to $\mathsf{err_z}$. Consider now an execution that does not faithfully simulate Π. Then, either one process was sent to $\mathsf{err_z}$, or the controller moved to $\mathsf{err_c}$ with at least one process in idle. Again, from $\mathsf{err_c}$ the only choice is to send an *err* message, forcing the processes in idle to move to $\mathsf{err_z}$. Therefore, all executions reach $\mathsf{err_z}$, hence $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\models^{\mathcal{P}',N})\Diamond \mathsf{err_z}) > 0$, and the probability is even 1 since $\mathcal{P}'$ is not probabilistic.

Assume now that $\mathcal{M}$ does not terminate, and let us show that for all network sizes $N \in \mathbb{N}_{>0}$, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0$. We distinguish between two cases. The first case is when $\mathcal{M}$ is not bounded. For all N, consider the execution ρ_N that first simulates the prefix $\pi_0 \dots \pi_n$ of Π, where π_n is the first configuration such that $c_1(\pi_n) + c_2(\pi_n) = N - 1$ and $c_1(\pi_{n+1}) + c_2(\pi_{n+1}) = N$, and then, the controller moves to $\mathsf{err_c}$ and loops there by broadcasting *err*. This execution ρ_N does not reach $\mathsf{err_z}$ since, it first simulates a prefix of Π, and then, when *err* is sent, there are no process in state idle to receive it, since all the processes are either in state c_1 or c_2. Hence $\forall N, \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0$. Second, if $\mathcal{M}$ is bounded, then for $N < \max_j(c_1(\pi_j) + c_2(\pi_j)) + 1$, we again consider ρ_N and get the same result. And for $N \geq \max_j(c_1(\pi_j) + c_2(\pi_j)) + 1$, using point 4 of Proposition 3.1, there exists an execution ρ_Π that faithfully simulates Π and therefore never reaches $\mathsf{err_z}$. Hence, for all N, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 0$. This concludes the proof of Lemma 3.4. □

Theorem 3.5 *The problems $REACH^{\forall}_{>p}(\mathscr{C})$ for $p \in [0,1[$ and $REACH^{\forall}_{=p}(\mathscr{C})$ for $p \in]0,1]$ are undecidable for static clique networks of probabilistic (timed) protocols.*

Proof Notice that since $\mathcal{P}'$ (defined in the proof of Lemma 3.4) is not probabilistic, for all strategies σ, $\mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) > 0$ if and only if $\mathbb{P}_\sigma(\mathscr{C}(\mathcal{P}'^N) \models \Diamond \mathsf{err_z}) = 1$. We hence deduce the undecidability of the two following problems: $REACH^{\forall}_{=1}(\mathscr{C})$ and $REACH^{\forall}_{>p}(\mathscr{C})$ for $p \in [0,1[$. Moreover, the gadget presented in Figure 3.7 also allows one to obtain the undecidability of $REACH^{\forall}_{=p}(\mathscr{C})$ for $p \in]0,1[$. Which concludes the proof of Theorem 3.5. □

3.3 Undecidability of synchronization

We now consider the synchronization property, *i.e.* whether the processes can gather at the same time in some specific states.

Notice that we can use the target set to check whether the simulation of a Minsky machine presented in Section 3.2 has reached the error state (that is a deadlock state).

Theorem 3.6 *The following problems are undecidable:*

- $SYNC^{\exists}_{=p}(\mathscr{C})$ *and* $SYNC^{\forall}_{=p}(\mathscr{C})$ *for* $p \in [0,1]$*;*
- $SYNC^{\exists}_{>p}(\mathscr{C})$ *and* $SYNC^{\forall}_{>p}(\mathscr{C})$ *for* $p \in [0,1[$*; and*
- $SYNC^{\exists}_{<p}(\mathscr{C})$ *and* $SYNC^{\forall}_{<p}(\mathscr{C})$ *for* $p \in]0,1]$*,*

for static clique networks of probabilistic (timed) protocols.

Proof To prove undecidability of these problems, we use the protocol $\mathcal{P}_{\mathcal{M}}$ defined in Section 3.2, represented in Figure 3.5. Notice that we can use the synchronization property to ensure that the error state is absent from the last configuration, hence that the run of the Minsky machine was simulated without error. Considering the target set $T = \{\mathsf{L}_{acc}, \mathsf{c}_1, \mathsf{c}_2, \mathsf{idle}\}$, we know, thanks to Proposition 3.1, that there exists an execution synchronizing in T if and only if the Minsky machine terminates. Since there is no probability in this protocol, the existence of an execution is equivalent to the existence of a strategy that satisfies synchronization with probability 1. We hence get the undecidability of $SYNC^{\exists}_{>p}(\mathscr{C})$ for $p \in [0,1[$. Moreover, using the gadget from Figure 3.7, we also get the undecidability of $SYNC^{\exists}_{=p}(\mathscr{C})$ for $p \in]0,1]$.

Consider now the protocol represented in Figure 3.8 for which there is an internal transition to the error state from all intermediary states of the controller. For this protocol, the only executions for which the controller does not move to the error state are of two kinds: either (1) executions that simulate the Minsky machine or (2) executions that wrongly guessed zero and then either the controller reaches the final instruction or the controller can simulate infinitely many instructions, and hence avoid the error state. To detect these two false simulations with a target set, we modify the protocol: first, we add a self loop broadcasting a message *err* on the error state and a reception of this message from L_{acc} to $\mathsf{err_z}$. Doing so, if an execution reaches L_{acc} while there are some processes in the error state, the controller is forced to move to the error state. Second, we allow to use the processes modeling the counters only once: this is done by moving them in a new deadlock state idle' at the end of a decrement instead of idle. When reaching idle', a process is said to be consumed. In order to prevent infinite simulations, we add before all original instructions two additional instructions that increment then decrement the same counter. That way, the simulation of each original instruction consumes at least one process, hence there is no infinite simulation. We thus ruled out the two cases and now the only execution where the controller does not reach the error state is the execution that simulates the Minsky machine. For the target set $T = \mathrm{Q} \setminus (\mathrm{K} \cup \mathrm{K} \times \mathrm{K} \cup \{\mathsf{q}_0\})$, there is an execution where all the processes do not gather in T if and only if the Minsky machine terminates. Hence $SYNC^{\exists}_{=0}(\mathscr{C})$ is undecidable, and since there are no probabilities in the protocol, the problems $SYNC^{\exists}_{<p}(\mathscr{C})$ are equivalent to $SYNC^{\exists}_{=0}(\mathscr{C})$ and hence also undecidable.

Now, replace the internal transition from LL' to the error state when L is an increment, by a reception of message *err*, and add an internal transition from idle to $\mathsf{err_z}$. If the number of processes is smaller than the maximum sum of the two counters plus one then, there is an execution in which the controller is stuck while doing an increment. Otherwise, for all the executions, the controller eventually reaches $\mathsf{err_z}$. Hence, for the

target set $T = \mathrm{Q} \setminus (\mathrm{K} \cup \mathrm{K} \times \mathrm{K} \cup \{\mathsf{q}_0\})$, the problem $SYNC^{\forall}_{=1}(\mathscr{C})$ is equivalent to the non-boundedness of the Minsky machine and is thus undecidable. Since there are no probabilities, the problems $SYNC^{\forall}_{>p}(\mathscr{C})$ for $p \in [0,1]$ are equivalent to $SYNC^{\forall}_{=1}(\mathscr{C})$, and hence also undecidable. Moreover, using the small gadget from Figure 3.7 we also obtain the undecidability of $SYNC^{\forall}_{=p}(\mathscr{C})$ for $p \in]0,1]$.

Last, consider the protocol represented in Figure 3.9, remove the internal transition LL' to the error state when L is a decrement and replace the internal transition LL' to the error state when L is an increment by a broadcast of err. Consider the target set $T = \{\mathsf{c}_1, \mathsf{c}_2, \mathsf{idle}, \mathsf{err}_\mathsf{c}\}$. If the number of processes is smaller than the maximum of the two counters sum plus one, then there is an execution that simulates $\mathcal{M}$ until it tries to perform an increment and lack processes, and the controller then moves to err_c. Thus, there exists a strategy that gathers all processes in the target set with probability 1 (since the protocol is not probabilistic). Otherwise, if the number of processes is greater than the maximum of the two counters sum plus one, then the execution that simulates $\mathcal{M}$ avoids err_c, and the other executions are either stuck in a state LL', or some processes are in state err_z. Thus, for all strategies, the probability to gather all the processes in the target set is 0. Hence, the problem $SYNC^{\forall}_{=0}(\mathscr{C})$ is equivalent to the non-boundedness of the Minsky machine, and is thus undecidable. Since there are no probabilities, the problems $SYNC^{\forall}_{<p}(\mathscr{C})$ for $p \in]0,1]$ are equivalent to $SYNC^{\forall}_{=0}(\mathscr{C})$, and hence also undecidable. □

In this section, we considered the parameterized verification of (static) clique networks of probabilistic timed protocols. We first have established the decidability of the problems $REACH^{\forall}_{=0}(\mathscr{C})$, and $REACH^{\forall}_{<p}(\mathscr{C})$ for all $p \in [0,1]$ thanks to a monotonicity property. Then, we have shown the undecidability of all the remaining reachability problems as well as the undecidability of the synchronization problems thanks to encodings of Minsky machines.

A well known infinite state model share this undecidability result: channel systems. However, in channel systems if one consider that message can be lost, the reachability problems become decidable. We will see in the following Section that similar results hold also in our case: if we consider unreliability in the model by introducing loss of process the problems are decidable.

4 Parameterized verification of dynamic networks of probabilistic timed protocols

We now turn to dynamic clique networks of probabilistic timed protocols, and will see that the decidability of the qualitative parameterized verification problems is recovered thanks to probabilistic disappearance and creation of processes. To establish this result, we first abstract the Markov decision process (MDP for short) $\mathscr{DC}(B, \mathcal{P}^N_\Lambda)$ into a discrete Markov decision process, using an *ad-hoc* region abstraction which preserves probabilities. This abstraction allows one not to consider all the possible real values of the clocks, but only a finite number of integer parts as well as an ordering on their fractional parts. Then, similarly to what is done for lossy channel systems [BBS07, BS13],

we prove that the so-called region MDP enjoys the finite attractor property, that it can be equipped with a well-quasi-ordering on its region-configurations, and that we can effectively compute the predecessor operator (thus perform a backward reachability analysis). These two properties entail the decidability of the qualitative verification questions in the region MDP that are equivalent to the parameterized verification problems in the network.

4.1 Region abstraction

Recall that the classical region abstraction for timed automata, presented in the seminal paper [AD94] (see Section 2), is based on the observation that the relevant information in clock valuations consists of the integral part of each clock (up to the maximal constant appearing in guards) and the ordering of their fractional parts. In our context, since the number of processes is unbounded (hence the number of clocks is unbounded), the region abstraction cannot be used directly. Still, based on classical regions, we present an equivalence relation over configurations.

Recall that (see Section 2) for $x \in \mathbb{R}^+$ a non-negative real, we denote by $\lfloor x \rfloor$ its integral part and $\{x\}$ its fractional part. Note that $x = \lfloor x \rfloor + \{x\}$.

Definition 4.1 (Region equivalence) *Let $b \in \mathbb{N}$ and let $\boldsymbol{\gamma}_1 = ((\mathsf{b}_1, x_{B1}), \gamma_1)$ and $\boldsymbol{\gamma}_2 = ((\mathsf{b}_2, x_{B2}), \gamma_2)$ be two configurations, with $\gamma_1 = \langle(\mathsf{q}_1, x_1), \ldots, (\mathsf{q}_n, x_n)\rangle$ and $\gamma_2 = \langle(\mathsf{p}_1, y_1), \ldots, (\mathsf{p}_m, y_m)\rangle$. For convenience, we denote $\mathsf{b}_1 = \mathsf{q}_0$, $x_{B1} = x_0$, $\mathsf{b}_2 = \mathsf{p}_0$, and $x_{B2} = y_0$.*

The configurations $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ are region equivalent*, denoted $\boldsymbol{\gamma}_1 \approx_b \boldsymbol{\gamma}_2$ if $\boldsymbol{\gamma}_1 = \boldsymbol{\gamma}^{(d)}$ if and only if $\boldsymbol{\gamma}_2 = \boldsymbol{\gamma}'^{(d)}$, and there exist a bijection $h : [0 \ldots n] \to [0 \ldots m]$ (hence $m = n$) such that the following conditions hold, $\forall i, j \in [0 \ldots n]$:*

(i) $h(0) = 0$*: the indices for the two base clocks match,*

(ii) $\mathsf{q}_i = \mathsf{p}_{h(i)}$*: states of processes agree,*

(iii) $(\lfloor x_i \rfloor \leq b) \vee (\lfloor y_{h(i)} \rfloor \leq b) \Rightarrow \lfloor y_{h(i)} \rfloor = \lfloor x_i \rfloor$*: integral parts of clocks agree up to b,*

(iv) $(\{x_i\} = 0) \Leftrightarrow (\{y_{h(i)}\} = 0)$*: clocks with integer values agree,*

(v) for $\sim \in \{<, =, >\}$, $(\{x_i\} \sim \{x_j\}) \Leftrightarrow (\{y_{h(i)}\} \sim \{y_{h(j)}\})$*: the orderings of fractional parts coincide.*

In Definition 4.1, the region equivalence is indexed by a bound b. For two given protocols B and $\mathcal{P}$, this bound is set to the maximal constant appearing in guards. For simplicity, we omit it in what follows and simply write $\approx$. Given $\boldsymbol{\gamma} \in \Gamma$ a configuration, $[\boldsymbol{\gamma}]$ denotes its equivalence class for $\approx$, and is called a *region-configuration*. As an example with $b = 1$, $\langle(q_1, 0), (q_2, 2.1)\rangle \approx \langle(q_1, 0), (q_2, 4.5)\rangle \not\approx \langle(q_1, 0.8), (q_2, 4.5)\rangle$.

Similarly to classical regions in timed automata, two region-equivalent configurations exhibit similar future behaviours in $\mathscr{DC}(B, \mathcal{P}^N_\Lambda)$. This is formalized in the following proposition:

Proposition 4.1 *Let $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ be two configurations. If $\boldsymbol{\gamma}_1 \approx \boldsymbol{\gamma}_2$, then $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ are time-abstract bisimilar,* i.e.*:*

- *for all $(y, (\mathsf{q}, x), \delta) \in \mathbb{R}^+ \times ((\mathrm{Q} \cup \mathrm{Q}_B) \times \mathbb{R}^+) \times (\Delta \cup \Delta_B)$ such that $(\boldsymbol{\gamma}_1, y, (\mathsf{q}, x), \delta, \boldsymbol{\gamma}_1') \in T^{(n)}$ there exist $y', x' \in \mathbb{R}^+$ and $\boldsymbol{\gamma}_2' \in [\boldsymbol{\gamma}_1']$ such that $(\boldsymbol{\gamma}_2, y', (\mathsf{q}, x'), \delta, \boldsymbol{\gamma}_2') \in T^{(n)}$,*
- *if $(\boldsymbol{\gamma}_1, \mathbf{unlock}, \boldsymbol{\gamma}_1') \in T^{(n)}$ then there exist $\boldsymbol{\gamma}_2' \in [\boldsymbol{\gamma}_1']$ such that $(\boldsymbol{\gamma}_2, \mathbf{unlock}, \boldsymbol{\gamma}_2') \in T^{(n)}$,*
- *if $(\boldsymbol{\gamma}_1, d_1) \in T^{(p)}$ then $(\boldsymbol{\gamma}_2, d_2) \in T^{(p)}$ and for all $\boldsymbol{\gamma}_1' \in \Gamma$ there exist $\boldsymbol{\gamma}_2' \in [\boldsymbol{\gamma}_1']$ such that $d_1(\boldsymbol{\gamma}_1') = d_2(\boldsymbol{\gamma}_2')$.*

Proof Let $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ be two configurations with $\boldsymbol{\gamma}_1 = \langle(\mathsf{q}_0, x_0), (\mathsf{q}_1, x_1), \ldots, (\mathsf{q}_n, x_n)\rangle$ and $\boldsymbol{\gamma}_2 = \langle(\mathsf{p}_0, y_0), (\mathsf{p}_1, y_1), \ldots, (\mathsf{p}_m, y_m)\rangle$ and such that $\boldsymbol{\gamma}_1 \approx \boldsymbol{\gamma}_2$. Let h be a bijection witnessing the equivalence of $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ *i.e.* satisfying the conditions of Definition 4.1. Let $\boldsymbol{\gamma}_1'$ be a configuration.

If $(\boldsymbol{\gamma}_1, \mathbf{unlock}, \boldsymbol{\gamma}_1') \in T^{(n)}$, by definition we know that $\boldsymbol{\gamma}_1' = \boldsymbol{\gamma}_1^{(d)}$ and that no transition is enabled in $\boldsymbol{\gamma}_1$. Since the bijection h ensures that the processes of $\boldsymbol{\gamma}_2$ are in the same state as the processes of $\boldsymbol{\gamma}_1$ and with clock values satisfying the same guards (recall that we compare clock values to integers only) we know that no transition is enabled in $\boldsymbol{\gamma}_2$ hence $(\boldsymbol{\gamma}_2, \mathbf{unlock}, \boldsymbol{\gamma}_2^{(d)}) \in T^{(n)}$ and the same bijection h satisfies the conditions of Definition 4.1 for $\boldsymbol{\gamma}_1^{(d)}$ and $\boldsymbol{\gamma}_2^{(d)}$ hence $\boldsymbol{\gamma}_1^{(d)} \approx \boldsymbol{\gamma}_2^{(d)}$.

If $(\boldsymbol{\gamma}_1, d_1) \in T^{(p)}$, there are two cases. Either $\boldsymbol{\gamma}_1 = \boldsymbol{\gamma}^{(d)}$ then d_1 corresponds to dynamism and $\boldsymbol{\gamma}_2 = \boldsymbol{\gamma}'^{(d)}$. Note that the probabilities of disappearance and creation of processes will be equal from two equivalent configurations, and will lead again to equivalent configurations. Or, there is a process that is in a probabilistic state, *i.e.* there exist $k \in [0 \ldots n]$ such that $q_k \in \mathrm{Q}^{(p)} \cup \mathrm{Q}_B^{(p)}$, d_1 is thus directly lifted from the distribution d such that $(\mathsf{q}_k, d) \in \Delta_p$. Since $\boldsymbol{\gamma}_1 \approx \boldsymbol{\gamma}_2$ we know that $\mathsf{q}_k = \mathsf{p}_{h(k)}$, hence $(\boldsymbol{\gamma}_2, d_2) \in T^{(p)}$ where d_2 is lifted from d. Since the clock value as well as the state of the other processes do not change, we obtain that the reached configurations are equivalent.

If $(\boldsymbol{\gamma}_1, t, (\mathsf{q}, x), \delta, \boldsymbol{\gamma}_1'') \in T^{(n)}$ for some $(t, (\mathsf{q}, x), \delta) \in \mathbb{R}^+ \times ((\mathrm{Q} \cup \mathrm{Q}_B) \times \mathbb{R}^+) \times (\Delta \cup \Delta_B)$, we first focus on time elapsing. We distinguish between three cases:

- Assume first that there is some index $k \in [1 \ldots n]$ such that $\{x_k + t\} = 0$. We partition the indices in three sets depending on whether the fractional part of their clock is lower, equal or greater to the fractional part of x_k. Formally, for $\sim \in \{<, =, >\}$, we define the set $I_\sim = \{i \in [0 \ldots n] \mid \{x_i\} \sim \{x_k\}\}$.Intuitively, after elapsing the delay t, all clocks in $I_=$ will have null fractional part, all clocks in $I_>$ will increase by $\lfloor t \rfloor + 1$ their integral parts, and have positive fractional parts, and all clocks in $I_<$ will increase by $\lfloor t \rfloor$ their integral parts, and have positive fractional parts. The sets $I_<$, $I_=$, and $I_>$ as well as their evolutions after time elapsing are represented in Figure 4.10. In this figure, $t = n + 0.55$ for some $n \in \mathbb{N}$, and $I_= = \{5, 6\}$ since $\{x_5\} = \{x_6\}$ and $\{x_5 + t\} = 0$, $I_> = \{7\}$, and $I_< = \{1, 2, 3, 4\}$.

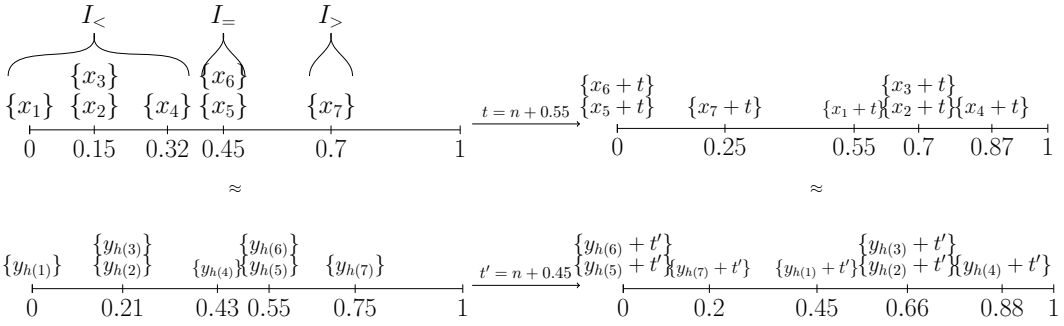


Figure 4.10: Evolution of the fractional parts.

We now define $t' = \lfloor t \rfloor + 1 - \{y_{h(k)}\}$, and prove that $\boldsymbol{\gamma}'_2 = \boldsymbol{\gamma}_2 + t'$ satisfies $\boldsymbol{\gamma}'_2 \approx \boldsymbol{\gamma}'_1$ (using the same bijection h).

First, notice that for i, j in the same set $I_\sim$ the ordering of $\{x_i\}$ and $\{x_j\}$ is the same as the ordering of $\{x_i + t\}$ and $\{x_j + t\}$ and that the ordering of $\{y_{h(i)}\}$ and $\{y_{h(j)}\}$ is the same as the ordering of $\{y_{h(i)} + t'\}$ and $\{y_{h(j)} + t'\}$. Second, notice that $\forall i_< \in I_<,\ \forall i_= \in I_=,\ \forall i_> \in I_>$ we get that:

- $\{x_{i_<}\} < \{x_{i_=}\} < \{x_{i_>}\}$,
- $\{x_{i_=} + t\} < \{x_{i_>} + t\} < \{x_{i_<} + t\}$, and
- $\{y_{h(i_<)}\} < \{y_{h(i_=)}\} < \{y_{h(i_>)}\}$,
- $\{y_{h(i_=)} + t'\} < \{y_{h(i_>)} + t'\} < \{y_{h(i_<)} + t'\}$.

Moreover,

(i) $\forall i \in [0 \dots n],\ \mathsf{q}_i = \mathsf{p}_{h(i)}$, still holds because control states did not change;

(ii) The integral parts still agree since:
   - The integral parts agree for $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$, *i.e* $\forall i \in [0 \dots n],\ (\lfloor x_i \rfloor \leq b) \vee (\lfloor y_{h(i)} \rfloor \leq b) \Rightarrow \lfloor y_{h(i)} \rfloor = \lfloor x_i \rfloor$;
   - By definition of the set $I_<$ and of t', we know that $\forall i \in I_<, \lfloor x_i + t \rfloor = \lfloor x_i \rfloor + \lfloor t \rfloor$ and $\lfloor y_{h(i)} + t' \rfloor = \lfloor y_{h(i)} \rfloor + \lfloor t' \rfloor = \lfloor y_{h(i)} \rfloor + \lfloor t \rfloor$;
   - And finally, by definition of the set $I_>$, $I_=$ and of t', we know that $\forall i \in I_= \cup I_>, \lfloor x_i + t \rfloor = \lfloor x_i \rfloor + \lfloor t \rfloor + 1$ and $\lfloor y_{h(i)} + t' \rfloor = \lfloor y_{h(i)} \rfloor + \lfloor t \rfloor + 1$.

   We can thus conclude that for all $i \in [0 \dots n],\ (\lfloor x_i + t \rfloor \leq b) \vee (\lfloor y_{h(i)} + t' \rfloor \leq b) \Rightarrow \lfloor y_{h(i)} + t' \rfloor = \lfloor x_i + t \rfloor$;

(iii) $\forall i \in [0 \dots n], (\{x_i + t\} = 0) \Leftrightarrow (i \in I_=) \Leftrightarrow (\{y_{h(i)}\} = \{y_{h(k)}\}) \Leftrightarrow (\{y_{h(i)} + t'\} = 0)$,

(iv) $\forall i, j \in [0 \dots n]$ for $\sim \in \{<, =, >\}$, $(\{x_i + t\} \sim \{x_j + t\}) \Leftrightarrow (\{y_{h(i)} + t'\} \sim \{y_{h(j)} + t'\})$, is straightforward from the two remarks above.

- Assume now that there is no index k such that $\{x_k + t\} = 0$. Then let k be an index such that $\forall i, \big(\{x_i\} \geq \{x_k\} \implies \lfloor x_i \rfloor + \lfloor t \rfloor + 1 = \lfloor x_i + t \rfloor\big)$ and $\forall i, \big(\{x_i\} < \{x_k\} \implies \lfloor x_i \rfloor + \lfloor t \rfloor = \lfloor x_i + t \rfloor\big)$. The above proof also works with $I_< = \{i \in [0 \dots n] \mid \{x_i\} < \{x_k\}\}, I_= = \emptyset$ and $I_> = \{i \in [0 \dots n] \mid \{x_i\} \geq \{x_k\}\}$.
- Last, in the remaining cases, defining $t' = \lfloor t \rfloor + (1 - \max_i(\{y_i\}))/2$, it is straightforward that $\boldsymbol{\gamma}'_2 \approx \boldsymbol{\gamma}_2 + t'$.

We now focus on the execution of the discrete action. Since δ is enabled in $\boldsymbol{\gamma}'_1$, there exist an index i such that $(\mathsf{q}_i, x_i) = (\mathsf{q}, x)$ from where δ is performable. Since $\boldsymbol{\gamma}'_1 \approx \boldsymbol{\gamma}'_2$ (with witness bijection h), $y_{h(i)}$ satisfies the same guards as x_i. Moreover, $\mathsf{p}_{h(i)} = \mathsf{q}_i$ and hence transition δ is also enabled in $(\mathsf{p}_{h(i)}, y_{h(i)})$. If δ is a communication, a similar argument implies that the $y_{h(i')}$'s satisfy the same guards as the $x_{i'}$'s, so that all the processes in $\boldsymbol{\gamma}'_2$ will receive the message and take the same transition as their corresponding processes in $\boldsymbol{\gamma}'_1$. In particular, the clock updates are the same.

Hence for all $t \in \mathbb{R}^+$, $(\mathsf{q}, x) \in (\mathrm{Q} \cup \mathrm{Q}_B) \times \mathbb{R}^+$, and $\delta \in (\Delta \cup \Delta_B)$ such that $(\boldsymbol{\gamma}_1, t, (\mathsf{q}, x), \delta, \boldsymbol{\gamma}''_1) \in T^{(n)}$ there exist two reals $t', x' \in \mathbb{R}^+$ and a configuration $\boldsymbol{\gamma}''_2 \in [\boldsymbol{\gamma}''_1]$ such that $(\boldsymbol{\gamma}_2, t', (\mathsf{q}, x'), \delta, \boldsymbol{\gamma}''_2) \in T^{(n)}$.

□

Thanks to Proposition 4.1, the MDP $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$ can be abstracted into its quotient by $\approx$, a countable MDP, formally defined as follows:

Definition 4.2 (Region MDP) *The* region MDP *of a dynamic clique network of probabilistic timed protocols* $\mathscr{DC}(B, \mathcal{P}_\Lambda^N) = (\Gamma, \Gamma^{(p)}, \Gamma^{(n)}, \{((\mathsf{b}_0, 0), \gamma_0)\}, T^{(n)}, T^{(p)})$ *is the MDP* $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) = (\Gamma_r, \Gamma_r^{(p)}, \Gamma_r^{(n)}, \mathbf{r}_0, T_r^{(n)}, T_r^{(p)})$ *defined by:*

- $\Gamma_r = \Gamma_r^{(p)} \cup \Gamma_r^{(n)}$,
- $\Gamma_r^{(p)} = \{[\boldsymbol{\gamma}] \mid \boldsymbol{\gamma} \in \Gamma^{(p)}\}$ *and* $\Gamma_r^{(n)} = \{[\boldsymbol{\gamma}] \mid \boldsymbol{\gamma} \in \Gamma^{(n)}\}$,
- $\mathbf{r}_0 = \{[((\mathsf{b}_0, 0), \gamma_0)]\}$,
- $T_r^{(n)}$ *is such that*
  - *if* $(\boldsymbol{\gamma}_1, y, (\mathsf{q}, x), \delta, \boldsymbol{\gamma}'_1) \in T^{(n)}$ *then* $([\boldsymbol{\gamma}_1], \delta, [\boldsymbol{\gamma}'_1]) \in T_r^{(n)}$
  - *if* $(\boldsymbol{\gamma}_1, \mathbf{unlock}, \boldsymbol{\gamma}'_1) \in T^{(n)}$ *then* $([\boldsymbol{\gamma}_1], \mathbf{unlock}, [\boldsymbol{\gamma}'_1]) \in T_r^{(n)}$
- $T_r^{(p)}$ *is such that if* $(\boldsymbol{\gamma}, d) \in T^{(p)}$ *then* $([\boldsymbol{\gamma}], d') \in T_r^{(p)}$ *with* $d'([\boldsymbol{\gamma}']) = \sum_{\boldsymbol{\gamma}_1 \in [\boldsymbol{\gamma}']} d(\boldsymbol{\gamma}_1)$.

Remark that the region MDP $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ is well-defined, thanks to Proposition 4.1. First, the existence of successors by discrete actions is uniform inside an equivalence class. Moreover, the sum $\sum_{\boldsymbol{\gamma}_1 \in [\boldsymbol{\gamma}']} d(\boldsymbol{\gamma}_1)$ does not depend on $\boldsymbol{\gamma}'$ but only on $[\boldsymbol{\gamma}']$: since there is no clock update in probabilistic transitions and only states change, it implies that for $\boldsymbol{\gamma}_1 \neq \boldsymbol{\gamma}_2$ with $d(\boldsymbol{\gamma}_1) > 0$ and $d(\boldsymbol{\gamma}_2) > 0$ we have $\boldsymbol{\gamma}_1 \notin [\boldsymbol{\gamma}_2]$. Last, the above sum is well-defined since there is at most a unique $\boldsymbol{\gamma}_1 \in [\boldsymbol{\gamma}']$ such that $d(\boldsymbol{\gamma}_1) > 0$.

As before, we refer to schedulers in the region MDP $\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N))$ as *strategies*.

Given $\mathsf{q}_f \in \mathrm{Q} \cup \mathrm{Q}_B$, and $\rho = [\boldsymbol{\gamma}_0][\boldsymbol{\gamma}_1]\ldots$ an infinite execution of $\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N))$, we say that ρ satisfies $\Diamond\mathsf{q}_f$, denoted $\rho \models \Diamond\mathsf{q}_f$, if there exist a region-configuration $[\boldsymbol{\gamma}_i] = [((\mathsf{b},\mathbf{x}_B),\gamma)]$ along ρ with either $\mathsf{b} = \mathsf{q}_f$ or $\gamma(\mathsf{q}_f,\mathbf{x}) > 0$ for some arbitrary clock value $\mathbf{x}$. Given a strategy σ_r for $\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N))$, we write $\mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N)) \models \Diamond\mathsf{q}_f)$ for the probability under σ of the set of infinite executions ρ with $\rho \models \Diamond\mathsf{q}_f$. We extend, in a straightforward way, the region abstraction from configuration to executions. Given an execution $\rho = \boldsymbol{\gamma}_0\boldsymbol{\gamma}_1\ldots$ of $\mathscr{DC}(B,\mathcal{P}_\Lambda^N)$, $[\rho]$ denotes the execution $[\boldsymbol{\gamma}_0][\boldsymbol{\gamma}_1]\ldots$.

As intended, the region MDP $\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N))$ is equivalent to $\mathscr{DC}(B,\mathcal{P}_\Lambda^N)$ in the following sense:

Proposition 4.2 *Let $\sim \in \{<,=,>\}$ be a comparison operator, $p \in [0,1]$ be a probabilistic threshold, and E be a set of executions.*

$$\exists\sigma_r\ \mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N)) \models [E]) \sim p \iff \exists\sigma\ \mathbb{P}_\sigma(\mathscr{DC}(B,\mathcal{P}_\Lambda^N) \models E) \sim p\ .$$

Proof We fix $N \in \mathbb{N}$ an initial number of processes.

The right-to-left implication of the equivalence is the easiest: from a strategy σ_r in the region MDP, one can define a scheduler σ for the original network by mimicking the discrete actions, and turning the abstract delays into concrete ones. This construction ensures the same probability of obtaining an execution of E. Thus

$$\forall\sigma_r,\ \exists\sigma,\ \mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N)) \models [E]) = \mathbb{P}_\sigma(\mathscr{DC}(B,\mathcal{P}_\Lambda^N) \models E).$$

For the other implication, first we show that a strategy in the network cannot lead to two different executions that are region equivalent. Afterwards, we can simply lift the strategy to a strategy for the region MDP and show that their probabilities of reaching an execution of E agree.

We show that for two executions, $\rho = \boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n$ and $\rho' = \boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n$, respecting σ, if $[\rho] = [\rho']$ (*i.e* if for all $i \in [0\ldots n], [\boldsymbol{\gamma}_i] = [\boldsymbol{\gamma}'_i]$) then $\rho = \rho'$, by induction on the length of the executions. First, consider $\rho = \boldsymbol{\gamma}_0$ and $\rho' = \boldsymbol{\gamma}_0$. Since there is an unique initial configuration, we obtain $\rho = \rho'$.

Assume now that, for all pairs ρ and ρ' of executions of length n respecting σ, we have: $[\rho] = [\rho']$ implies $\rho = \rho'$.

Consider the two executions of length $n+1$ respecting σ, $\rho = \boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n\boldsymbol{\gamma}_{n+1}$ and $\rho' = \boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n\boldsymbol{\gamma}'_{n+1}$. Assume that $[\rho] = [\rho']$, by induction hypothesis, we have that $\boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n = \boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n$. We distinguish between two cases on whether $\boldsymbol{\gamma}_n$ is a probabilistic or non-deterministic configuration. First, when $\boldsymbol{\gamma}_n$ is a non-deterministic configuration, $\boldsymbol{\gamma}_{n+1}$ is given by $\sigma(\boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n) = \sigma(\boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n)$ hence $\boldsymbol{\gamma}_{n+1} = \boldsymbol{\gamma}'_{n+1}$. Second, when $\boldsymbol{\gamma}_n$ is a probabilistic configuration, since there is no clock update in probabilistic transitions but only one state change it implies that if $\boldsymbol{\gamma}_{n+1} \neq \boldsymbol{\gamma}'_{n+1}$ then $[\boldsymbol{\gamma}_{n+1}] \neq [\boldsymbol{\gamma}'_{n+1}]$; which contradicts $[\rho] = [\rho']$. Hence in both cases $\rho = \rho'$.

As a consequence, we can naturally lift a strategy σ into a region strategy σ_r by letting $\sigma_r([\boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n])$ as $\sigma(\boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n)$ if $[\boldsymbol{\gamma}'_0\ldots\boldsymbol{\gamma}'_n] = [\boldsymbol{\gamma}_0\ldots\boldsymbol{\gamma}_n]$, and be arbitrary if there is no such execution. By construction we get:

$$\forall\sigma,\ \exists\sigma_r,\ \mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B,\mathcal{P}_\Lambda^N)) \models [E]) = \mathbb{P}_\sigma(\mathscr{DC}(B,\mathcal{P}_\Lambda^N) \models E).$$

This ends the proof of the second implication, and thus the proof of Proposition 4.2. □

A consequence of Proposition 4.2 is that for $\exists\!\!\!\checkmark \in \{\exists, \forall\}$,

$$\exists\!\!\!\checkmark\sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) \sim p \quad \iff \quad \exists\!\!\!\checkmark\sigma_r,\ \mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) \sim p$$

and

$$\exists\!\!\!\checkmark\sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \downarrow (T_1, T_2)) \sim p \iff \exists\!\!\!\checkmark\sigma_r,\ \mathbb{P}_{\sigma_r}(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \downarrow (T_1, T_2)) \sim p\ .$$

Therefore, the parameterized verification problems are equivalent in the dynamic probabilistic timed network and its region MDP.

4.2 Deciding parameterized problems on the region MDP

We have seen, with Proposition 4.2, that the qualitative reachability problems are equivalent in $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$ and in $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$. We now expose how to decide them in the region MDP. The decidability result relies on two key arguments: first, $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ admits a finite attractor, and second, in $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$, the predecessor operator is effective and preserves upward-closure for some well-quasi-ordering. The decidability can then be derived by applying similar techniques as for non-deterministic and probabilistic lossy channel systems [BBS07, BS13].

The finite attractor property was introduced originally for probabilistic lossy channel systems (LCS) and states that Markov chains induced by LCS admit a finite recurrent set [ABRS05, BBS06a] (see Section 4.2). Roughly said, some results for finite Markov chains extend to infinite Markov chains with a finite attractor. A Markov chain is said to have a *finite attractor*, if there exist a finite set of states that is visited infinitely often almost-surely (*i.e.*, with probability 1). Further, a Markov decision process has a finite attractor if there exist a finite set of states which is an attractor under all possible schedulers.

Proposition 4.3 *The set* $\{[((\mathsf{q}, n), \emptyset)] \mid \mathsf{q} \in \mathrm{Q}_B, n \in [0 \dots b]\}$ *is an attractor for* $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$.

Proof This proof is similar to the proof of existence of a finite attractor for LCS with insertion errors [ABRS05].

Let us fix a strategy σ for the region abstraction $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$. The state space of $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ can be partitioned according to the number of processes in the region-configuration. For this partition, the Markov chain defined by $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ under scheduler σ is almost left-oriented: there exist $M_0 \in \mathbb{N}$ and $\eta \in \mathbb{R}^+$ such that from any region-configuration $\boldsymbol{\gamma}$ with $M \geq M_0$ processes, the expected number of processes of the next region-configuration under σ is smaller than or equal to $M - \eta$. To see this, write K for the expected number of processes that are created in one step. Starting from $\boldsymbol{\gamma}$ of size M, the expected size of the next region-configuration is bounded by $M(1-\lambda_-)+K$. Taking M_0 large enough, there exist $\eta > 0$ such that $\forall M \geq M_0,\ M(1-\lambda_-) + K \leq M-\eta$. This shows that the Markov chain is almost left-oriented, and applying [BBS06a],

$\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ thus admits a finite attractor: the set of region-configurations with size smaller than M_0.

From every region-configuration of this attractor there is a positive probability that all processes disappear in the next step, and none are created. Hence the region-configurations with the base only and with no process also form a finite attractor. □

Now that the existence of a finite attractor has been established, we define an appropriate partial order $\trianglelefteq$ on all the region-configurations Γ_r. Intuitively, region-configuration $\mathbf{r}$ is smaller than $\mathbf{r}'$, if we can remove some processes of a configuration of $\boldsymbol{\gamma}' \in \mathbf{r}'$ and obtain a configuration $\boldsymbol{\gamma} \in \mathbf{r}$. We also add a side-condition on the clocks with integer value, which is needed to obtain a good property on the predecessor operator. Formally:

Definition 4.3 (Ordering on region-configurations) *The order* $\trianglelefteq \subseteq \Gamma_r \times \Gamma_r$ *is defined as follows: for* $\mathbf{r}, \mathbf{r}' \in \Gamma_r$, $\mathbf{r} \trianglelefteq \mathbf{r}'$ *if there exist* $\boldsymbol{\gamma} = ((\mathsf{b}, x_B), \gamma) \in \mathbf{r}$ *and* $\boldsymbol{\gamma}' = ((\mathsf{b}', {x_B}'), \gamma') \in \mathbf{r}'$ *such that:*

(i) $\mathsf{b} = \mathsf{b}'$ *and* $x_B = {x_B}'$*; and*

(ii) $\forall \mathsf{q} \in \mathrm{Q},\ \forall x \in \mathbb{R}^+,\ \gamma(\mathsf{q}, x) \leq \gamma'(\mathsf{q}, x)$*; and*

(iii) $\sum_{\mathsf{q} \in \mathrm{Q}} \sum_{n \in \mathbb{N}} \gamma(\mathsf{q}, n) = 0 \implies \sum_{\mathsf{q} \in \mathrm{Q}} \sum_{n \in \mathbb{N}} \gamma'(\mathsf{q}, n) = 0.$

Adapting the proof of Higman's lemma, one obtains:

Proposition 4.4 *The partial order* $\trianglelefteq$ *is a well-quasi-ordering.*

Recall that Higman's lemma states that the sub-word relation on words is a well-quasi-ordering. We adapt here its proof to establish that $\trianglelefteq$ is a well-quasi-ordering. Our proof of Proposition 4.4 is technical, but follows the same lines as the one for Higman's lemma, and is just given here for the sake of completeness.

Proof First, remark that $\trianglelefteq$ is well-founded. Indeed, if $\mathbf{r} \trianglelefteq \mathbf{r}'$ then $\mathbf{r}$ necessarily has less processes than $\mathbf{r}'$, and thus there cannot be any infinite decreasing sequence for $\trianglelefteq$.

Let us now show that there are no infinite sequences $(\mathbf{r}^i)_{i \in \mathbb{N}}$ such that for all $i < j \in \mathbb{N}$, $\neg(\mathbf{r}^i \trianglelefteq \mathbf{r}^j)$. Such a sequence is called a *bad sequence*, and we will prove that bad sequences do not exist. To do so, we first introduce some notations. Let $F : \mathbb{N} \times \Gamma \to [0, 1)$ be the function such that $F(0, \boldsymbol{\gamma}) = 0$, and for $i > 0$, $F(i, \boldsymbol{\gamma})$ is the i-th smallest fractional part greater than 0 appearing in $\boldsymbol{\gamma}$.

We define $\mathsf{rChunk} : \mathbb{N} \times \Gamma \to \langle Q \times [0, .., b+1] \rangle$ as the function such that

$$\begin{array}{lll} \mathsf{rChunk}(i, ((\mathsf{b}, x_B), \gamma)) & \{(\mathsf{q}, n)^m \mid & n \in [0 \ldots b], \mathsf{q} \in \mathrm{Q}, \gamma(\mathsf{q}, n + F(i, \gamma)) = m \\ & & \quad \text{or } n = b+1 \text{ and } \sum_{k > b} \gamma(\mathsf{q}, k + F(i, \gamma)) = m\} \cup \\ & \{(\mathsf{b}, n) \mid & n \in [0 \ldots b], \mathsf{b} \in \mathrm{Q}_B, n + F(i, \gamma) = x_B \\ & & \quad \text{or } n = b+1 \text{ and } \exists k > b, k + F(i, \gamma) = x_B\} \end{array}$$

with the convention that $\mathsf{rChunk}(i, \boldsymbol{\gamma}) = \emptyset$ if there are less than i different fractional parts in $\boldsymbol{\gamma}$.

Intuitively, $\mathsf{rChunk}(i, \boldsymbol{\gamma})$ is the multiset composed of the pairs of states and integral parts of clocks (up to $b+1$) for processes or the base that have the i-th smallest fractional parts.

Notice that $\forall \boldsymbol{\gamma}' \in [\boldsymbol{\gamma}], \forall i \in \mathbb{N}, \mathsf{rChunk}(i, \boldsymbol{\gamma}') = \mathsf{rChunk}(i, \boldsymbol{\gamma})$, and whenever $\boldsymbol{\gamma}' \notin [\boldsymbol{\gamma}]$, there exist an index $i \in \mathbb{N}$ such that $\mathsf{rChunk}(i, \boldsymbol{\gamma}') \neq \mathsf{rChunk}(i, \boldsymbol{\gamma})$. Hence $[\boldsymbol{\gamma}]$ is characterized by the multisets $\mathsf{rChunk}(i, \boldsymbol{\gamma})$. Also observe that $\mathrm{Q} \times [0 \dots b+1] \cup \mathrm{Q}_B \times [0 \dots b+1]$ is finite, hence $\subseteq$, the natural inclusion on multisets is a well-quasi-ordering on $\mathbb{M}(\mathrm{Q} \times [0 \dots b+1] \cup \mathrm{Q}_B \times [0 \dots b+1])$.

Let us now go back to the proof of Proposition 4.4. Assume, towards a contradiction that there are bad sequences for $\trianglelefteq$. We define $\mathbf{r}^0$ as one of the regions with the minimal number of processes that starts such a bad sequence. Then, we let $\mathbf{r}^1$ as one of the regions with minimal number of processes that can follow $\mathbf{r}^0$ in a bad sequence. Inductively, $\mathbf{r}^k$ is defined as one of the regions with the smallest number of processes that can follow $\mathbf{r}^0 \cdots \mathbf{r}^{k-1}$ in a bad sequence. The infinite bad sequence built this way exists by assumption, and is written $\mathcal{S} = \mathbf{r}^0, \mathbf{r}^1, \cdots$.

Since $\subseteq$ is a well-quasi-ordering, we can extract a subsequence $\mathbf{r}^{k_0}, \mathbf{r}^{k_1}, \cdots$ such that $\mathsf{rChunk}(0, \mathbf{r}^{k_0}) \subseteq \mathsf{rChunk}(0, \mathbf{r}^{k_1}) \subseteq \cdots$. We now define $\mathcal{S}' = (u^{k_n})_{n \in \mathbb{N}}$ as a sequence where u^{k_n} is the region such that $\mathsf{rChunk}(0, u^{k_n}) = \mathsf{rChunk}(0, \mathbf{r}^{k_n}) \cap \mathrm{Q}_B \times [0 \dots b+1]$ and $\forall i > 0, \mathsf{rChunk}(i, u^{k_n}) = \mathsf{rChunk}(i, \mathbf{r}^{k_n})$.

To reach a contradiction, we observe that $\mathcal{S}'$ is also a bad sequence and has "less processes" than $\mathcal{S}$, contradicting the minimality of $\mathcal{S}$.

First, $\mathcal{S}'$ is a bad sequence: indeed, assuming $u^{k_i} \trianglelefteq u^{k_j}$ for some $k_i < k_j$, then there exist $\boldsymbol{\gamma}_i \in u^{k_i}$ and $\boldsymbol{\gamma}_j \in u^{k_j}$ that satisfy the three conditions of Definition 4.3. Letting $\boldsymbol{\gamma}'_i = \boldsymbol{\gamma}_i + \mathsf{rChunk}(0, \mathbf{r}^{k_i}) \cap \mathrm{Q} \times [0 \dots b+1]$ and $\boldsymbol{\gamma}'_j = \boldsymbol{\gamma}_j + \mathsf{rChunk}(0, \mathbf{r}^{k_j}) \cap \mathrm{Q} \times [0 \dots b+1]$, a direct consequence is that $\boldsymbol{\gamma}'_i \in \mathbf{r}^{k_i}$, $\boldsymbol{\gamma}'_j \in \mathbf{r}^{k_j}$ and that $\boldsymbol{\gamma}'_i$, $\boldsymbol{\gamma}'_j$ also satisfy the three conditions of Definition 4.3, since $\mathsf{rChunk}(0, \mathbf{r}^{k_i}) \subseteq \mathsf{rChunk}(0, \mathbf{r}^{k_j})$. Hence, $\mathbf{r}^{k_i} \trianglelefteq \mathbf{r}^{k_j}$, although $\mathcal{S}$ is assumed to be a bad sequence.

Last, let us show that $\mathcal{S}'$ contradicts the definition of $\mathcal{S}$ as "minimal" bad sequence. There are two cases:

- If $\mathsf{rChunk}(0, \mathbf{r}^{k_0}) \not\subseteq \mathrm{Q}_B \times [0 \dots b+1]$, the sequence $\mathbf{r}^0, \cdots, \mathbf{r}^{k_0-1}, u^{k_0}, u^{k_1}, \cdots$ is a bad sequence and u^{k_0} has less processes than v^{k_0}. This contradicts the definition of $\mathbf{r}^{k_0}$.
- If $\mathsf{rChunk}(0, \mathbf{r}^{k_0}) \subseteq \mathrm{Q}_B \times [0 \dots b+1]$, we repeat the construction of the subsequence on the second factor in $\mathsf{rChunk}(1, \mathbf{r}^{k_i})$. With the same arguments, we extract a subsequence $\mathcal{S}'' = (u^{k'_n})_{n \in \mathbb{N}}$ with $\mathsf{rChunk}(0, u^{k'_n}) = \mathsf{rChunk}(0, v^{k'_n})$ and $\forall i > 0, \mathsf{rChunk}(i, u^{k'_n}) = \mathsf{rChunk}(i+1, v^{k'_n})$. We obtain a contradiction since $\mathbf{r}^0, \cdots, \mathbf{r}^{k'_0-1}, u^{k'_0}, u^{k'_1}, \cdots$ is a bad sequence and its k'_0-th element has less processes than the one of $\mathcal{S}$.

To sum up, we proved that $\trianglelefteq$ is well-founded and every infinite sequence contains a pair of ordered elements, so that $\trianglelefteq$ is a well-quasi-ordering. □

We now consider the upward-closure operator $\uparrow$ w.r.t. $\trianglelefteq$: given $C \subseteq \Gamma_r$,

$$\uparrow C = \{\mathbf{r}' \in \Gamma_r \mid \exists \mathbf{r} \in C \text{ such that } \mathbf{r} \trianglelefteq \mathbf{r}'\} \ .$$

Notice that the condition $\mathbf{r}' \in \Gamma_r$ ensures that the base is represented only once, and that there is at most one process (or the base) in a probabilistic state. A set $C \subseteq \Gamma_r$ is said to be *upward-closed* whenever $C = \uparrow C$. Since $\trianglelefteq$ is a well-quasi-ordering, any non-decreasing sequence of upward-closed sets eventually stabilizes. This property will be useful in the sequel.

Last, we define the predecessor operator: for $\mathbf{r} \in \Gamma_r^{(n)}$, $Pre(\mathbf{r})$ denotes the set of region-configurations $\mathbf{r}' \in \Gamma_r^{(n)}$ that can reach $\mathbf{r}$ after a non-deterministic action and the probabilistic transition that follows (from mobility and possibly from probabilistic states). Notice that here, the initial number of processes is not specified, since the predecessor operator is independent of this initial number. It happens that for any upward-closed set $C \subseteq \Gamma_r^{(n)}$, the set $Pre(C)$ can be computed, and is upward closed.

Proposition 4.5 *The predecessor operator $Pre(.)$ preserves upward-closure, is effective and satisfies for any set $C \subseteq \Gamma_r^{(n)}$, $Pre(C) = Pre(\uparrow C)$.*

Proof To prove Proposition 4.5, we have to show that given an upward-closed set C, then $Pre(C)$ is upward-closed, and can be computed effectively from C and the protocols B and $\mathcal{P}$. Moreover, we will show that $Pre(C) = Pre(\uparrow C)$ for any set $C \subseteq \Gamma_r^{(n)}$.

The predecessor operator Pre can be split into two operators Pre_d and Pre_s such that $Pre = Pre_s \circ Pre_d$ where Pre_d is the dynamism predecessor operator that applies to the non-deterministic region-configurations $\mathbf{r} \in \Gamma_r^{(n)}$, and Pre_s is the static predecessor operator that applies to probabilistic region-configurations $\mathbf{r} \in \Gamma_r^{(p)}$. The static predecessor operator can itself be split into two operators Pre_t and Pre_a, for predecessors by time elapsing, or by a discrete action, such that $Pre_s = Pre_t^* \circ Pre_a$.

Let C be an upward-closed set of configurations. To prove that $Pre_t^*(C)$ is upward-closed and effective, we explain how it can be computed, starting from the finitely many minimal elements of C. Let $\mathbf{r}$ be a region-configuration.

- If $\mathsf{rChunk}(0, \mathbf{r}) = \emptyset$, meaning that no clock has an integer value in $\mathbf{r}$, then

$$Pre_t(\uparrow \mathbf{r}) = \uparrow \mathbf{r} \cup \uparrow \mathbf{r}' \cup \bigcup_{(\mathsf{q},n) \in \mathrm{Q} \times [0 \ldots b+1]} \uparrow \mathbf{r}_{(\mathsf{q},n)}.$$

  Where $\mathbf{r}'$ is the region defined as $\forall i \in \mathbb{N}, \mathsf{rChunk}(i, \mathbf{r}') = \mathsf{rChunk}(i+1, \mathbf{r})$. And where $\mathbf{r}_{(\mathsf{q},n)}$ stands for the region defined as $\mathsf{rChunk}(0, \mathbf{r}_{(\mathsf{q},n)}) = \langle (\mathsf{q}, n) \rangle$ and $\forall i > 0, \mathsf{rChunk}(i, \mathbf{r}_{(\mathsf{q},n)}) = \mathsf{rChunk}(i, \mathbf{r})$.

- If $\mathsf{rChunk}(0, \mathbf{r}) \cap ((\mathrm{Q} \cup \mathrm{Q}_B) \times \{0\}) \neq \emptyset$, meaning that some clocks have value 0 then

$$Pre_t(\uparrow \mathbf{r}) = \uparrow \mathbf{r}.$$

- If $\mathsf{rChunk}(0, \mathbf{r}) \neq \emptyset$ and $\mathsf{rChunk}(0, \mathbf{r}) \cap ((\mathrm{Q} \cup \mathrm{Q}_B) \times \{0\}) = \emptyset$, meaning that some clocks have non null integer value in $\mathbf{r}$, then

$$Pre_t(\uparrow \mathbf{r}) = \uparrow \mathbf{r} \cup \uparrow \mathbf{r}'$$

where, $\mathbf{r}'$ is defined as: $\mathsf{rChunk}(0,\mathbf{r}') = \emptyset$, for the minimal index i_{max} such that $\forall i > i_{max}, \mathsf{rChunk}(i,\mathbf{r}) = \emptyset$, $\mathsf{rChunk}(i_{max},\mathbf{r}') = \mathsf{rChunk}(0,\mathbf{r}) - 1$ and $\forall i \neq 0, i \neq i_{max}, \mathsf{rChunk}(i,\mathbf{r}') = \mathsf{rChunk}(i,\mathbf{r})$. Assuming $\mathsf{rChunk}(0,\mathbf{r}) = \langle(\mathsf{q}_1,x_1)...(\mathsf{q}_n,x_n)\rangle$, then $\mathsf{rChunk}(0,\mathbf{r}) - 1$ is a notation for $\langle(\mathsf{q}_1,x_1-1)...(\mathsf{q}_n,x_n-1)\rangle$.

This case inspection provides an upward-closed set as result of the computation of the time predecessor of a basic upward-closed set. One concludes that Pre_t preserves upward-closure and is effective.

Now, considering predecessors by a discrete action, the above proof can be adapted to show that $Pre_a(\uparrow C)$ is upward-closed and effective. The case inspection is only technical and omitted here.

Finally, let us consider the dynamism predecessor operator. We show that for any set C, $Pre_d(C) = Pre_d(\uparrow C)$. Let us detail the non-trivial inclusion $Pre_d(\uparrow C) \subseteq Pre_d(C)$. Let $\mathbf{r} \in Pre_d(\uparrow C)$, then there exist $\mathbf{r}_1 \trianglelefteq \mathbf{r}$ and $\mathbf{r}_2$ with $\mathbf{r}_1 \trianglelefteq \mathbf{r}_2$ and $\mathbf{r}_2 \in \uparrow C$, these regions are obtained by the sequence of steps $\mathbf{r} \xrightarrow{loss} \mathbf{r}_1 \xrightarrow{creation} \mathbf{r}_2$. Therefore $\mathbf{r}_2 = \mathbf{r}'_2 + \mathbf{r}''_2$ with $\mathbf{r}'_2 \in C$. We then define $\mathbf{r}'_1$ such that $\mathbf{r}'_1 \trianglelefteq \mathbf{r}'_2$ and $\mathbf{r}'_1 \trianglelefteq \mathbf{r}_1$. The sequence of steps $\mathbf{r} \xrightarrow{loss} \mathbf{r}'_1 \xrightarrow{creation} \mathbf{r}'_2$ then witnesses that $\mathbf{r} \in Pre_d(C)$ as it is shown in the graphical representation in Figure 4.11.

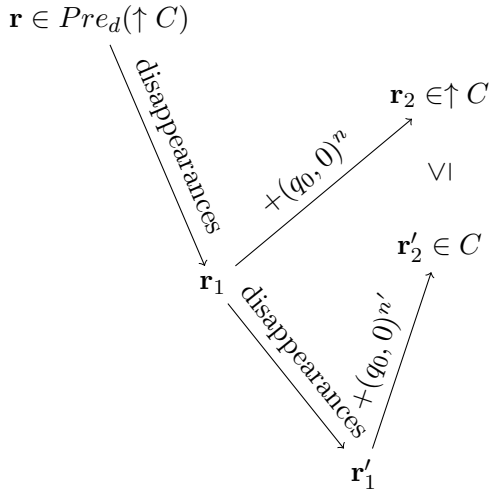


Figure 4.11: Graphical representation to show $Pre_d(\uparrow C) \subseteq Pre_d(C)$.

□

4.2.1 Solving reachabilty

Propositions 4.3, 4.4 and 4.5 are decisive to obtain the decidability of qualitative parameterized reachability problems in the region MDP $\mathcal{R}(\mathscr{DC}(B,\mathcal{P}^N_\Lambda))$. Combined with Proposition 4.2 we obtain the decidability of the qualitative reachability problems for dynamic clique networks as stated in Theorems 4.1, 4.2, 4.3, and 4.4.

We first fix some notations. We denote by $\boldsymbol{\gamma}_0^N = ((\mathsf{b}_0, 0), \langle(\mathsf{q}_0, 0)^N\rangle)$ the initial configuration in $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$, and $[\boldsymbol{\gamma}_0^N]$ its equivalence class for $\approx$. $\boldsymbol{\gamma}_0^{\mathbb{N}} = \bigcup_{N\in\mathbb{N}} \boldsymbol{\gamma}_0^N$ denotes the set of all initial configurations and $[\boldsymbol{\gamma}_0^{\mathbb{N}}]$ the union of their equivalence classes.

Now, for each qualitative problem, we reduce to a decidable reachability question in the region MDP.

Theorem 4.1 *The problems $REACH^{\exists}_{>0}(\mathscr{DC})$ and $REACH^{\forall}_{=0}(\mathscr{DC})$ are decidable for dynamic probabilistic timed networks.*

Proof As stated above, by Proposition 4.2, we can consider the same decision problem in the region MDP. To show decidability of $REACH^{\exists}_{>0}(\mathscr{DC})$ (resp. $REACH^{\forall}_{=0}(\mathscr{DC})$), it thus suffices to show the decidability of whether there exist $N \in \mathbb{N}$ and a scheduler σ such that $\mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) > 0$ (resp. $\mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) = 0$). Given $N \in \mathbb{N}$, we have the series of equivalences:

$$\begin{aligned} \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) > 0 \iff & \quad \exists \sigma,\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) > 0 \\ \iff & \quad [\boldsymbol{\gamma}_0^N] \in Pre^*(\uparrow \mathsf{q}_f). \end{aligned}$$

Recall that $\uparrow \mathsf{q}_f$ represents the set of equivalence classes where at least one process is in state q_f, and is thus upward-closed. Since the Pre operator preserves upward-closure and is effective, and because $\trianglelefteq$ is a well-quasi-ordering, the set $Pre^*(\uparrow \mathsf{q}_f)$ can be computed effectively by successive iterations of Pre.

To answer whether there exist a network size $N \in \mathbb{N}$ and a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) > 0$, it then suffices to test whether one of the region-configurations of $[\boldsymbol{\gamma}_0^{\mathbb{N}}]$ belongs to $Pre^*(\uparrow \mathsf{q}_f)$. Hence, it suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}] \cap Pre^*(\uparrow \mathsf{q}_f) \neq \emptyset$ to decide $REACH^{\exists}_{>0}(\mathscr{DC})$. As a consequence, we obtain the decidability of $REACH^{\exists}_{>0}(\mathscr{DC})$.

In the same way, one can answer whether for all $N \in \mathbb{N}$ there exist a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) > 0$ by testing whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}] \subseteq Pre^*(\uparrow \mathsf{q}_f)$. And since

$$\exists N, \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) = 0 \Leftrightarrow \quad \neg[\forall N, \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) > 0]$$

it suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}] \subseteq Pre^*(\uparrow \mathsf{q}_f)$ to decide $REACH^{\forall}_{=0}(\mathscr{DC})$. As a consequence, we obtain the decidability of $REACH^{\forall}_{=0}(\mathscr{DC})$. □

For the remaining cases, as for qualitative verification of LCS [BBS06b], apart from the predecessor operator Pre, we also introduce a kind of controlled predecessor operator, $SPre$ called the *safe predecessor operator*, and defined as follows:

$$SPre_A(B) = \{\mathbf{r} \in \Gamma_r^{(n)} \mid Post[\alpha](\mathbf{r}) \cap B \neq \emptyset \text{ and } Post[\alpha](\mathbf{r}) \subseteq A\}\ ,$$

where $Post[\alpha](\mathbf{r})$ stands for the set of successors of $\mathbf{r}$ after the non-deterministic action α (and probabilistic transition for internal actions) and after the following probabilistic

transitions (from dynamism). In words, a region-configuration $\mathbf{r}$ belongs to $SPre_A(B)$ if there exist an action which can lead to B (if the processes disappearances and creations are favourable) and necessarily leads to A (whatever the process disappearances and creations).

Also, we define the following notation for the *downward-kernel* of a set of configurations with respect to $\trianglelefteq$. Given $C \subseteq \Gamma_r$,

$$\Downarrow C = \{\mathbf{r} \in C \mid \forall \mathbf{r}' \trianglelefteq \mathbf{r},\ \mathbf{r}' \in C\} \ .$$

The operator $\Downarrow$ is the dual of the upward-closure one $\uparrow$, in the following sense: for every set $C \subseteq \Gamma_r$, $\uparrow (\Gamma_r \setminus C) = \Gamma_r \setminus (\Downarrow C)$.

Using this notation, the safe predecessor operator enjoys the following properties:

Proposition 4.6 *$SPre$ is effective and satisfies $SPre_A(B) = SPre_{\Downarrow A}(\uparrow B)$.*

Proof The effectivity of $SPre$ derives from the one of Pre, since

$$SPre_A(B) = \bigcup_\alpha Pre[\alpha](B) \cap \Gamma_r \setminus Pre[\alpha]\big(\Gamma_r \setminus A\big) \ .$$

Now, we establish that $SPre_A(B) = SPre_{\Downarrow A}(\uparrow B)$.

$$\begin{aligned} SPre_A(B) &= \bigcup_\alpha Pre[\alpha](B) \cap \Gamma_r \setminus Pre[\alpha]\big(\Gamma_r \setminus A\big) \\ &= \bigcup_\alpha Pre[\alpha](\uparrow B) \cap \Gamma_r \setminus Pre[\alpha]\Big(\uparrow \big(\Gamma_r \setminus A\big)\Big) \\ &= \bigcup_\alpha Pre[\alpha](\uparrow B) \cap \Gamma_r \setminus Pre[\alpha]\Big(\Gamma_r \setminus \big(\Downarrow A\big)\Big) \\ &= SPre_{\Downarrow A}(\uparrow B) \ . \end{aligned}$$

□

To prove the decidability of the remaining cases, we will rely on a general convergence result for terms defined in the μ-calculus [BS13]. Roughly, the convergence is guaranteed for any closed term defined by a μ-calculus expression in which (1) the variables for least fix-points are under the scope of an upward-closure operator, and (2) the variables for greatest fix-points are under the scope of a downward-kernel operator.

Theorem 4.2 *The problems $REACH^{\exists}_{=0}(\mathscr{DC})$ and $REACH^{\forall}_{>0}(\mathscr{DC})$ are decidable for dynamic probabilistic timed networks.*

Proof Once again, we use Proposition 4.2 in order to consider the same decision problem in the region MDP. We then use a greatest fix-point and the safe pre operator to

compute the set of region-configurations from which there is a scheduler avoiding almost surely q_f. Given $N \in \mathbb{N}$ we have:

$$\begin{aligned}\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) = 0 \iff & \quad \exists \sigma\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) = 0 \\ \iff & \quad [\boldsymbol{\gamma}_0^N] \in W_1,\end{aligned}$$

where W_1 is defined as the following greatest fix-point: $W_1 = \nu X.(\neg \uparrow \mathsf{q}_f) \cap SPre_X(\Gamma_r)$. In words, the greatest fix-point W_1 computes the set of region-configurations that do not contain q_f and have an action that allows to stay almost surely in this set. The proof for non-deterministic and probabilistic lossy channel systems can be adapted *mutatis mutandis* to show the correctness as well as the effective computability of W_1 in our case. More generally, the correctness and effective computation hold for any MDP with a finite attractor, and such that $Pre(A) = Pre(\uparrow A)$. Thanks to Proposition 4.6, W_1 can be rewritten as

$$W_1 = \nu X.(\neg \uparrow \mathsf{q}_f) \cap SPre_{\Downarrow X}(\Gamma_r)\ ,$$

which is a guarded term: X, a greatest fix-point variable, is downward-guarded.

To decide whether there exist $N \in \mathbb{N}$ and a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) = 0$, it then suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}]$ intersects W_1. This proves the decidability of $REACH^{\exists}_{=0}(\mathscr{DC})$.

Similarly, to decide whether for all $N \in \mathbb{N}$ there exist a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) = 0$, it then suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}]$ is a subset of W_1. And since:

$$\exists N, \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) > 0 \iff \quad \neg[\forall N, \exists \sigma,\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) = 0],$$

it suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}] \subseteq W_1$ to decide $REACH^{\forall}_{>0}(\mathscr{DC})$. As a consequence, we obtain the decidability of $REACH^{\forall}_{>0}(\mathscr{DC})$. □

To obtain the decidability of $REACH^{\exists}_{<1}(\mathscr{DC})$ and $REACH^{\forall}_{=1}(\mathscr{DC})$, we use the combination of a least fix-point and a greatest fix-point in order to obtain the set of region-configurations from which there exist a strategy that avoids q_f with positive probability.

Theorem 4.3 *The problems $REACH^{\exists}_{<1}(\mathscr{DC})$ and $REACH^{\forall}_{=1}(\mathscr{DC})$ are decidable for dynamic probabilistic timed networks.*

Proof Once again, we use Proposition 4.2 in order to consider the same decision problem in the region MDP. Here we reuse the greatest fix-point W_1 and combine it with a least fix-point, in order to obtain that given $N \in \mathbb{N}$:

$$\begin{aligned}\exists \sigma\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) < 1 \iff & \quad \exists \sigma\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) < 1 \\ \iff & \quad [\boldsymbol{\gamma}_0^N] \in W_2,\end{aligned}$$

where W_2 is the least fix-point defined by $W_2 = \mu X.W_1 \cup (Pre(X) \cap \neg(\uparrow \mathsf{q}_f))$. Intuitively, W_2 is the set of region-configurations from which we can reach W_1 before reaching

q_f, hence from which there exist a strategy that reaches W_1 before q_f with positive probability. Moreover, from the previous case we know that from W_1 there is a strategy avoiding q_f almost surely. Equivalently, $W_2 = \mu X.W_1 \cup (Pre(\uparrow X) \cap \neg(\uparrow \mathsf{q}_f))$. Again, following what was proven for non-deterministic and probabilistic lossy channel systems, W_2 can be effectively computed.

To decide whether there exist $N \in \mathbb{N}$ and a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) < 1$, it then suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}]$ intersects W_2. This implies the decidability of $REACH^{\exists}_{<1}(\mathscr{DC})$.

Similarly, to decide whether for all network sizes $N \in \mathbb{N}$ and a scheduler σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) < 1$, it then suffices to test whether $[\boldsymbol{\gamma}_0^{\mathbb{N}}] \subseteq W_2$. And since

$$\exists N,\ \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) = 1 \Leftrightarrow \quad \neg[\forall N, \exists \sigma,\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) < 1],$$

this implies the decidability of $REACH^{\forall}_{=1}(\mathscr{DC})$. □

The decidability of the last two cases is proved with the combination of a least-fix point and a greatest fix-point in order to obtain the set of region configurations from which there exist a strategy that reaches q_f almost surely.

Theorem 4.4 *The problems $REACH^{\exists}_{=1}(\mathscr{DC})$ and $REACH^{\forall}_{<1}(\mathscr{DC})$ are decidable for dynamic probabilistic timed networks.*

Proof Once again, we use Proposition 4.2 in order to consider the same decision problem in the region MDP. Here, we introduce W_3 which combines a greater and a least fix-point, in order to obtain that given $N \in \mathbb{N}$:

$$\begin{aligned} \exists \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}, 0, \Lambda, \models^{\Diamond}_{\mathsf{q}})_f) = 1 \iff & \quad \exists \sigma\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) = 1 \\ \iff & \quad [\boldsymbol{\gamma}_0^N]\ \in W_3 \end{aligned}$$

where W_3 is defined as the following fix-point: $W_3 = \nu X.[\mu Y.(SPre_X(Y) \vee (\uparrow \mathsf{q}_f))]$. Intuitively, the inner least fix-point computes the region-configuration from where there is a strategy reaching q_f, and the outer greater fix-point ensures that the probability to stay in these region-configurations is 1. Hence we cannot reach a region-configuration from where q_f is not accessible. Here again, applying the proof techniques for non-deterministic and probabilistic lossy channel systems, one obtains that W_3 is correct (justifying the last equivalence). Also, W_3 can be rewritten into a guarded term using Proposition 4.6:

$$W_3 = \nu X.[\mu Y.(SPre_{\Downarrow X}(\uparrow Y) \vee (\uparrow \mathsf{q}_f))]\ .$$

Deciding $REACH^{\exists}_{=1}(\mathscr{DC})$ thus boils down to the computation of W_3 followed by the test: $[\boldsymbol{\gamma}_0^{\mathbb{N}}]\ \cap W_3 \neq \emptyset$?

Equivalently, deciding whether for all N there exist σ such that $\mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^0) \models \Diamond \mathsf{q}_f) = 1$ boils down to the computation of W_3 followed by the test: $[\boldsymbol{\gamma}_0^{\mathbb{N}}]\ \subseteq W_3$? And since

$$\exists N,\ \forall \sigma,\ \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f) < 1 \Leftrightarrow \quad \neg[\forall N, \exists \sigma,\ \mathbb{P}_\sigma(\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N)) \models \Diamond \mathsf{q}_f) = 1],$$

this implies the decidability of $REACH^{\forall}_{<1}(\mathscr{DC})$. □

In this section we showed decidability of the qualitative reachability problems. Although the quantitative problems are still open, we are able to compute, given an initial network size, the value of the infimum on the strategies to reach the goal state within arbitrary precision. The next section is dedicated to this result.

4.2.2 Approximation of minimal probability

We now move to quantitative analysis, *i.e.* to problems where the probabilistic threshold is not only 0 or 1 but can be all possible values between 0 and 1. The quantitative reachability problems are still open, however given an initial network size we are able to compute, within arbitrary precision, the minimal probability to reach the target.

This proof is inspired by the approximation scheme for the probability to reach a goal in fully probabilistic LCS [ABRS05]. The idea is to consider the set of executions of length n, and to partition it into three sets: $Reach_n$ the executions that already reached the target, $Escape_n$ the executions that have not and cannot be prolongated to reach the goal, and $Undecided_n$ for the remaining executions. It was shown that if the Markov chain has a finite attractor then the probability to be undecided (denoted $p_n^?$) decreases towards 0 as n increases. Moreover, denoting $p_n^+ = \mathbb{P}(Reach_n)$ the probability to have reach the target before n steps, p_n^+ is an under approximation and $p_n^+ + p_n^?$ an over-approximation of $\mathbb{P}(\Diamond \mathsf{q}_f)$. Hence, this gives an algorithm to approximate the probability to reach the target.

In order to adapt this approximation scheme to our setting, we must overcome two main difficulties. First, we have to lift the approximation to Markov decision processes rather than Markov chains. Second, due to creations, from any region-configuration one can, in one step, reach an unbounded number of region-configurations. In order to tackle this problem, in beyond bounding the length of the executions, we also bound the number of processes along executions.

Let us fix some notations. In the following, we consider a fixed region MDP $\mathcal{R}(\mathscr{DC}(B, \mathcal{P}_\Lambda^N))$ and we omit it in the notations. Notice that we thus consider a fixed initial number of processes, N.

For $n \in \mathbb{N}$, let ${\Gamma_r}^{\leq n}$ be the set of region-configurations with less than n processes. Let W_1 be the set of region-configurations from which there is a scheduler avoiding almost surely q_f (note that this set is effectively computable, see the proof of Theorem 4.2). Clearly, from W_1 the infimum probability is 0.

Let $Reach_n = \{\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_n \cdots \mid \exists k \leq n,\ (q_f \in \boldsymbol{\gamma}_k) \wedge (\forall i \leq k,\ \boldsymbol{\gamma}_i \in {\Gamma_r}^{\leq n})\}$ be the set of infinite executions that reach q_f in less than n steps and with less than n processes.

Let $Escape_n = \{\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_n \cdots \mid \exists k \leq n,\ (\boldsymbol{\gamma}_k \in W_1) \wedge (\forall i \leq k,\ \boldsymbol{\gamma}_i \in {\Gamma_r}^{\leq n})\}$ be the set of executions that reach W_1 in less than n steps and with less than n processes.

Let $Greater_n = \{\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_n \cdots \mid \exists k \leq n,\ \boldsymbol{\gamma}_k \in \Gamma_r \setminus {\Gamma_r}^{\leq n}\}$ be the set of executions that have more than n processes before step n.

Let $Undecided_n = \{\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_n \cdots \mid \forall k \leq n,\ q_f \notin \boldsymbol{\gamma}_k \wedge \boldsymbol{\gamma}_k \notin W_1 \wedge \boldsymbol{\gamma}_k \in {\Gamma_r}^{\leq n}\}$ be the rest of the executions.

To obtain an approximation scheme for $\inf_\sigma \mathbb{P}_\sigma(\Diamond \mathsf{q}_f)$, we first show the following inequalities:

Lemma 4.1

$$\inf_\sigma\ \mathbb{P}_\sigma(Reach_n) \le \inf_\sigma \mathbb{P}_\sigma(\Diamond \mathsf{q}_f) \le \inf_\sigma\ \mathbb{P}_\sigma(Reach_n) + \sup_\sigma\ \mathbb{P}_\sigma(Greater_n) + \sup_\sigma\ \mathbb{P}_\sigma(Undecided_n)\ .$$

Proof To prove the first inequality, notice that for all $n \in \mathbb{N}$, all executions ρ in $Reach_n$ satisfy $\Diamond \mathsf{q}_f$. Hence, for all strategies σ, $\mathbb{P}_\sigma(Reach_n) \le \mathbb{P}_\sigma(\Diamond \mathsf{q}_f)$. We thus deduce that $\inf_\sigma\ \mathbb{P}_\sigma(Reach_n) \le \inf_\sigma \mathbb{P}_\sigma(\Diamond \mathsf{q}_f)$.

To prove the second inequality we define $S_{W_1=0}$ as the family of strategies that once they reach a configuration of W_1 play to avoid q_f almost surely. Notice that, for any strategy σ, there exists a strategy $\sigma' \in S_{W_1=0}$ such that σ' plays as σ until a configuration of W_1 is reached and then plays to avoid q_f almost surely. Formally, for any execution $\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k$, if $\boldsymbol{\gamma}_k \in W_1$, $\sigma'(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k) = \sigma_0(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k)$ where $\mathbb{P}_{\sigma_0}(\boldsymbol{\gamma}_k \models \Diamond \mathsf{q}_f) = 0$ and $\sigma'(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k) = \sigma(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k)$ otherwise. On the one hand, under this definition of σ', we have $\mathbb{P}_{\sigma'}(Reach_n) \le \mathbb{P}_\sigma(Reach_n)$, hence $\inf_{\sigma' \in S_{W_1=0}} \mathbb{P}_{\sigma'}(Reach_n) \le \inf_\sigma \mathbb{P}_\sigma(Reach_n)$. On the other hand, we have $\inf_\sigma \mathbb{P}_\sigma(Reach_n) \le \inf_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(Reach_n)$ since $S_{W_1=0}$ is a subset of strategies. Thus, $\inf_\sigma \mathbb{P}_\sigma(Reach_n) = \inf_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(Reach_n)$.

Now, notice that for all $n \in \mathbb{N}$, $Reach_n, Escape_n, Greater_n$, and $Undecided_n$ form a partition of the set of all executions $\mathcal{E}$. Thus, for every strategy $\sigma \in S_{W_1=0}$, $\mathbb{P}_\sigma(\Diamond \mathsf{q}_f) = \mathbb{P}_\sigma(Reach_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Escape_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Greater_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Undecided_n \cap \Diamond \mathsf{q}_f)$. And by definition of $S_{W_1=0}$, $\mathbb{P}_\sigma(Escape_n \cap \Diamond \mathsf{q}_f) = 0$.

Hence, we obtain the following inequalities:

$$\begin{aligned}
&\inf_\sigma \mathbb{P}_\sigma(\Diamond \mathsf{q}_f) \\
&\le \inf_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(\Diamond \mathsf{q}_f) \\
&= \inf_{\sigma \in S_{W_1=0}} \big(\mathbb{P}_\sigma(Reach_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Greater_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Undecided_n \cap \Diamond \mathsf{q}_f)\big) \\
&= \inf_{\sigma \in S_{W_1=0}} \big(\mathbb{P}_\sigma(Reach_n) + \mathbb{P}_\sigma(Greater_n \cap \Diamond \mathsf{q}_f) + \mathbb{P}_\sigma(Undecided_n \cap \Diamond \mathsf{q}_f)\big) \\
&\le \inf_{\sigma \in S_{W_1=0}} \big(\mathbb{P}_\sigma(Reach_n) + \mathbb{P}_\sigma(Greater_n) + \mathbb{P}_\sigma(Undecided_n)\big) \\
&\le \inf_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(Reach_n) + \sup_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(Greater_n) + \sup_{\sigma \in S_{W_1=0}} \mathbb{P}_\sigma(Undecided_n) \\
&\le \inf_\sigma \mathbb{P}_\sigma(Reach_n) + \sup_\sigma \mathbb{P}_\sigma(Greater_n) + \sup_\sigma \mathbb{P}_\sigma(Undecided_n)
\end{aligned}$$

□

Lemma 4.2 $\inf_\sigma\ \mathbb{P}_\sigma(Reach_n)$, $\sup_\sigma\ \mathbb{P}_\sigma(Undecided_n)$ *and* $\sup_\sigma,\ \mathbb{P}_\sigma(Greater_n)$ *are effectively computable for every* $n \in \mathbb{N}$.

Proof For every $n \in \mathbb{N}$, one can build a finite MDP $M_n = (Q, P)$ with state space $\{\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k \mid k \le n \wedge \forall i \le k,\ \boldsymbol{\gamma}_i \in {\Gamma_r}^{\le n}\} \cup \{\bot\}$ and the probabilistic function P is such that $P(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k, \alpha, \boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k \boldsymbol{\gamma}_{k+1}) = \mathbb{P}(\boldsymbol{\gamma}_k \xrightarrow{\alpha} \boldsymbol{\gamma}_{k+1})$ and $P(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k, \alpha, \bot) = \sum_{\boldsymbol{\gamma} \notin {\Gamma_r}^{\le n}} \mathbb{P}(\boldsymbol{\gamma}_k \xrightarrow{\alpha} \boldsymbol{\gamma})$.

For every strategy σ in the region MDP, there exists a strategy σ' in M_n such that $\mathbb{P}_\sigma(Reach_n) = \mathbb{P}^{M_n}_{\sigma'}(\Diamond Reach_n)$ and reciprocally. Idem for $Undecided_n$. On the other hand for $Greater_n$, we obtain $\mathbb{P}_\sigma(Greater_n) = \mathbb{P}^{M_n}_{\sigma'}(\Diamond \bot)$.

Hence

$$\inf_\sigma \ \mathbb{P}_\sigma(Reach_n) = \inf_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond Reach_n) = \min_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond Reach_n),$$

$$\sup_\sigma \ \mathbb{P}_\sigma(Undecided_n) = \sup_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond Undecided_n) = \max_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond Undecided_n),$$

and

$$\sup_\sigma \ \mathbb{P}_\sigma(Greater_n) = \sup_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond \bot) = \max_\sigma \mathbb{P}^{M_n}_\sigma(\Diamond \bot)$$

are effectively computable. □

Lemma 4.3 $\lim_{n\to\infty} \sup_\sigma \ \mathbb{P}_\sigma(Greater_n) = 0$.

Proof Notice that the probability to have executions with more than n processes before step $\log(n)$ does not depend on the strategy since the strategy has no influence on the probability of creation and deletion. Thus, there exists K such that for all strategies σ, $\mathbb{P}_\sigma(Greater_n) = K$. In the following, we denote by $\mathbb{P}(Greater_n)$ this value.

We denote by $p_{k \xrightarrow{\ell} j}$ (resp. $p_{k \xrightarrow{\ell} >j}$) the probability to start in a configuration with k processes and end in a configuration with j (resp. more than j) processes in at most ℓ steps.

First, remark that $\mathbb{P}(Greater_n) = 1$ for any $n \leq N$, since the first configuration has already more than n processes.

We now prove that for $n > N$, $\mathbb{P}_\sigma(Greater_n) \leq \log(n).p_{n \xrightarrow{1} >n}$. We prove this by showing by induction on ℓ that for any $k < n$, $p_{k \xrightarrow{\ell} >n} \leq \ell.p_{n \xrightarrow{1} >n}$.

First notice that for all $k \leq n$ we have $p_{k \xrightarrow{1} >n} \leq p_{n \xrightarrow{1} >n}$ showing the base case. Intuitively, whatever the rates λ_+ and λ_-, there is a greater probability to obtain more processes starting from a bigger configuration.

Now, assume the induction hypothesis $p_{k \xrightarrow{\ell} >n} \leq \ell.p_{n \xrightarrow{1} >n}$.

$$\begin{aligned}
p_{k \xrightarrow{\ell+1} >n} &= p_{k \xrightarrow{1} >n} + \sum_{k' \leq n} p_{k \xrightarrow{1} k'} \cdot p_{k' \xrightarrow{\ell} >n} \\
&\leq 1 \cdot p_{n \xrightarrow{1} >n} + \sum_{k' \leq n} p_{k \xrightarrow{1} k'} \cdot \ell \cdot p_{n \xrightarrow{1} >n} \\
&\leq p_{n \xrightarrow{1} >n} + (1 - p_{k \xrightarrow{1} >n}) \cdot p_{n \xrightarrow{1} >n} \\
&\leq p_{n \xrightarrow{1} >n} + 1 \cdot p_{n \xrightarrow{l} >n} \\
&\leq (l+1) \cdot p_{n \xrightarrow{1} >n}.
\end{aligned}$$

This proves that for any $\ell \geq 1$ and any $k \leq n$, $p_{k \xrightarrow{\ell} >n} \leq \ell \cdot p_{n \xrightarrow{1} >n}$.

Hence $\mathbb{P}_\sigma(Greater_n) = p_{N \xrightarrow{\log(n)} >n} \leq n \cdot p_{n \xrightarrow{1} >n}$.

We now show that $\lim_{n\to\infty} n \cdot p_{n \xrightarrow{1} >n} = 0$.

$$\begin{aligned}
\lim_{n\to\infty} n.p_{n \xrightarrow{1} >n} &= \lim_{n\to\infty} n.\sum_{k=0}^{n} \binom{n}{k} \lambda_-^k (1-\lambda_-)^{n-k} \sum_{i=k+1}^{\infty} \lambda_+^i (1-\lambda_+) \\
&= \lim_{n\to\infty} n.(1-\lambda_-)^n \lambda_+ (1 + \frac{\lambda_-\lambda_+}{1-\lambda_-})^n \\
&= \lim_{n\to\infty} n.\lambda_+.(1-\lambda_-(1-\lambda_+))^n \\
&= 0
\end{aligned}$$

Which concludes the proof of Lemma 4.3, since $\lim_{n\to\infty} \sup_\sigma \mathbb{P}_\sigma(Greater_n) \leq \lim_{n\to\infty} n \cdot p_{n \xrightarrow{1} >n} = 0$. □

Lemma 4.4 $\lim_{n\to\infty} \sup_\sigma \mathbb{P}_\sigma(Undecided_n) = 0$.

Proof We now show that $\lim_{n\to\infty} \sup_\sigma \mathbb{P}_\sigma(Undecided_n) = 0$.

Let $\sigma_n = argmax_\sigma \mathbb{P}_\sigma^{M_n}(\Diamond Undecided_n)$. Notice that for any $k \geq n$, $Undecided_k \subseteq Undecided_n$, hence $\mathbb{P}_{\sigma_k}(Undecided_n) \geq \mathbb{P}_{\sigma_k}(Undecided_k)$.

Let us define inductively a sequence $(S_n)_{n\in\mathbb{N}}$ of sets of strategies such that $S_{W_1=0} = \{\sigma_n \mid n \in \mathbb{N}\}$, and for all $k \in \mathbb{N}$, S_{k+1} is an infinite subset of S_k such that, for all strategies $\sigma, \sigma' \in S_{k+1}$ and all executions $\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k$ such that for all $i \leq k$, $\boldsymbol{\gamma}_i \in \Gamma_r(\leq k)$ and $\sigma(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k) = \sigma'(\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k)$. This subset exists by the pigeonhole principle, since there is a finite number of executions $\boldsymbol{\gamma}_0 \dots \boldsymbol{\gamma}_k$, a finite number of actions from each region-configuration of $\Gamma_r(\leq k)$ and an infinite number of strategies in S_k.

Let σ_{max} be a strategy such that $\forall n \in \mathbb{N}$, $\sigma_{max} \in S_k$.

Assume now toward a contradiction that there exists $\epsilon > 0$ such that $\forall n \in \mathbb{N}$, $\mathbb{P}_{\sigma_n}(Undecided_n) > \epsilon$. Remark that $\mathbb{P}_{\sigma_{max}}(Undecided_n) = \mathbb{P}_{\sigma_k}(Undecided_n)$ for some $k \geq n$ such that $\sigma_k \in S_n$. Since $\mathbb{P}_{\sigma_k}(Undecided_n) \geq \mathbb{P}_{\sigma_k}(Undecided_k) > \epsilon$, we obtain that for every $n \in \mathbb{N}$, $\mathbb{P}_{\sigma_{max}}(Undecided_n) > \epsilon$ hence $\lim_{n\to\infty} \mathbb{P}_{\sigma_{max}}(Undecided_n) = \mathbb{P}_{\sigma_{max}}(Undecided) > \epsilon$ which is impossible since $\mathscr{DC}(B, \mathcal{P}_\Lambda^N)$ together with the strategy σ_{max} is a Markov chain with finite attractor.

Hence $\sup_\sigma \mathbb{P}_\sigma(Undecided_n) \xrightarrow[n\to\infty]{} 0$, which concludes the proof of Lemma 4.4. □

Putting together the Lemmas 4.1, 4.2, 4.3, and 4.4, we obtain the following theorem.

Theorem 4.5 *Given two protocols $\mathcal{P}$ and B, a pair of rates $\Lambda \in]0,1[^2$, an initial number of processes $N \in \mathbb{N}$, and a bound $\epsilon \in]0,1]$, one can effectively compute a value $p \in [0,1]$ such that $|(\inf_\sigma \mathbb{P}_\sigma(\mathscr{DC}(B, \mathcal{P}_\Lambda^N) \models \Diamond \mathsf{q}_f)) - p| < \epsilon$.*

4.2.3 Synchronization analysis

For the synchronization analysis, we do not just want to reach an upward closed set as we did for the reachability analysis; we also require that all the processes are in the target set. This rather corresponds to the complementary of an upward closed set. In [BBS06b], the authors consider simple symbolic sets, which in words consist of an upward closed set minus a sum of upward closed sets, *i.e.* a symbolic set of the form $\uparrow X - \uparrow Y_1 \cdots - \uparrow Y_n$. They have shown that for LCS these symbolic sets are closed under Boolean operations, upward closure, and predecessor operator and also that all these operations are effective.

One can adapt the proofs given in Section 4.2.1 to show that in our case we can also effectively compute the predecessor of a symbolic set. Moreover, synchronization in sets T_B and T can be expressed as the symbolic set $\uparrow \Gamma_r - \sum_{\mathsf{b} \in \mathrm{Q}_B \setminus T_B} \uparrow \mathsf{b} - \sum_{\mathsf{q} \in \mathrm{Q} \setminus T} \uparrow \mathsf{q}$. Last, it is shown in [BBS06b] that symbolic sets defined by fix-points expressions can be computed effectively (under guardedness conditions for fix-point terms). We can thus reuse the fix-point expressions presented in section 4.2.1 and adapt it to symbolic sets to obtain the decidability of the qualitative synchronization analysis.

Theorem 4.6 *The problems* $SYNC^{\exists\forall}_{\sim p}(\mathscr{D}\mathscr{C})$ *are decidable for* $p \in \{0, 1\}$.

4.2.4 Complexity

In the previous sections, we have shown the reachability and synchronization problems to be decidable. However, the proofs rely on a backward analysis, for which we showed termination thanks to a well-quasi-ordering. This gives an algorithm of high complexity. We show in this section that these problems are in fact F_{ω^ω}-complete using a reduction from reachability in LCS that is known to be F_{ω^ω}-complete (see section 4.2).

Theorem 4.7 *The problems* $REACH^{\exists\forall}_{\sim p}(\mathscr{D}\mathscr{C})$ *are* F_{ω^ω}*-hard.*

Proof The idea to prove F_{ω^ω}-hardness, is to reduce the reachability problem in LCS, which is F_{ω^ω}-complete. The main idea of the reduction is to use the clock of each process to order them: intuitively, the order given by the clock values corresponds to the order in which messages appear in the channel of a LCS configuration. To do so, we define two (non-probabilistic) protocols, and use the base to encode the state of the LCS, and the processes to encode the messages. The dynamism in the network will correspond to loss in the LCS.

Let $\mathcal{L} = (\mathrm{Q}, \mathsf{q}_0, \mathrm{M}, \delta)$ be a lossy channel system. First, we define the base protocol $B_{\mathcal{L}} = (\mathrm{Q}_B, \mathrm{Q}_B^{(p)}, \mathrm{Q}_B^{(n)}, \mathsf{b}_0, \mathbf{x}, \Sigma, \Delta_B)$ as (see Figure 4.12):

- $\mathrm{Q}_B^{(p)} = \emptyset$
- $\mathrm{Q}_B^{(n)} = \mathrm{Q} \cup \delta$
- $\mathsf{b}_0 = \mathsf{q}_0$
- $\Sigma = \{ok, r\} \cup \{m_r, m_w \mid m \in \mathrm{M}\}$

- Δ_B is such that for every transition $t = (\mathsf{q}, ?, m, \mathsf{q}') \in \delta$, there are transitions $(\mathsf{q}, tt, ??m_r, \emptyset, t)$ and $(t, tt, \varepsilon, \emptyset, \mathsf{q}')$ to read a message in the channel and for every transition $t = (\mathsf{q}, !, m, \mathsf{q}') \in \delta$ there are transitions $(\mathsf{q}, \mathbf{x}_B > 0, !!m_w, \mathbf{x}_B := 0, t)$ and $(t, tt, ??ok, \emptyset, \mathsf{q}')$ to write a message in the channel. Finally, from any state $\mathsf{b} \in \mathrm{Q}_B$, there is a transition $(\mathsf{b}, tt, !!r, \mathbf{x}_B := 0, \mathsf{b}_0)$ that allows one to restart the simulation from an initial configuration.

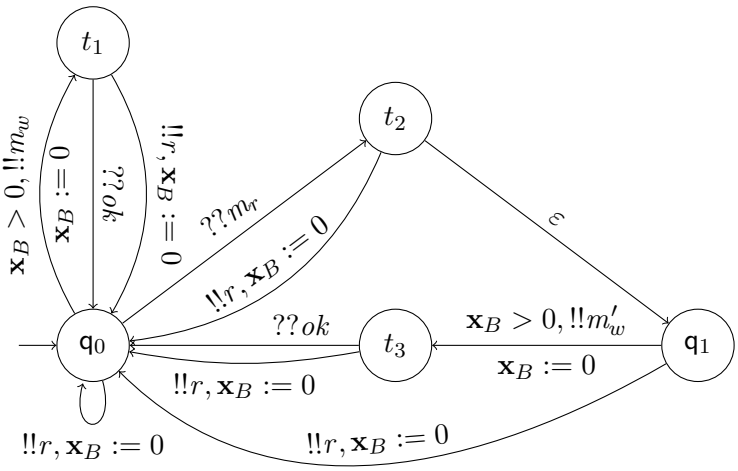


Figure 4.12: Base protocol obtained from a LCS with two states q_0 and q_1 and three transitions $t_1 = (\mathsf{q}_0, !m, \mathsf{q}_0)$, $t_2 = (\mathsf{q}_0, ?m, \mathsf{q}_1)$, and $t_3 = (\mathsf{q}_1, !m', \mathsf{q}_0)$.

Then, the protocol $\mathcal{P}_{\mathcal{L}} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \mathbf{x}, \Sigma, \Delta)$ is defined as (see Figure 4.13):

- $\mathrm{Q}^{(p)} = \emptyset$
- $\mathrm{Q}^{(n)} = \{\mathsf{q}_0\} \cup \{m, \bar{m} \mid m \in \mathrm{M}\}$
- $\Sigma = \{ok\} \cup \{m_r, m_w \mid m \in \mathrm{M}\}$
- Δ is such that for every message $m \in \mathrm{M}$, there are transitions: $(\mathsf{q}_0, tt, ??m_w, \mathbf{x} := 0, \bar{m})$, $(\bar{m}, tt, ??ok, \mathbf{x} := 0, \mathsf{q}_0)$, and $(\bar{m}, \mathbf{x} = 0, !!ok, \mathbf{x} := 0, m)$ that correspond to writing a message to the channel and $(m, \mathbf{x} = 1, !!m_r, \mathbf{x} := 0, \mathsf{q}_0)$ that corresponds to reading a message from the channel. And from any state $\mathsf{q} \in \mathrm{Q}$, there is a transition $(\mathsf{q}, tt, ??r, \mathbf{x} := 0, \mathsf{q}_0)$ that allows one to restart the simulation from an initial configuration.

We now prove that a given state q is reachable in $\mathcal{L}$ if and only if $\mathsf{q} \in \mathrm{Q}_B$ is reachable with probability 1 in the dynamic clique network $\mathscr{DC}(B_{\mathcal{L}}, \mathcal{P}_{\mathcal{L}}^N)$ with arbitrary (positive) rates Λ.

We say that a configuration $\boldsymbol{\gamma} = ((\mathsf{b}, \mathbf{x}_B), \gamma)$ encodes a configuration (q, w) if $\mathsf{b} = \mathsf{q}$ and denoting $w = m_1 \dots m_n$, there exist $x_1, \dots, x_n \in [0, 1]$ such that $x_1 > \dots > x_n$ and

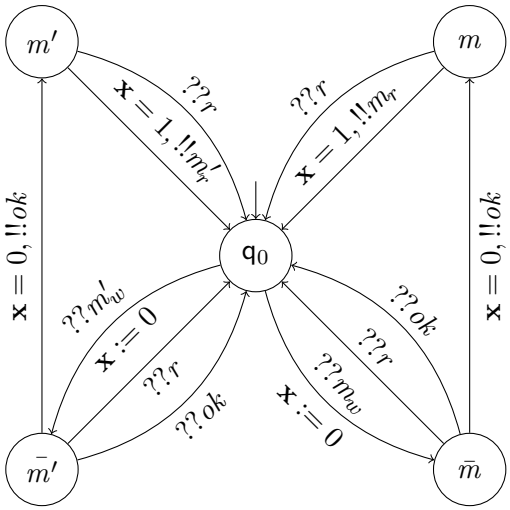


Figure 4.13: Protocol obtained from a LCS with two messages m and m'.

for all $i \in [1 \dots n]$ we have $\gamma(m_i, x_i) = 1$. Moreover, all the other processes are either in the initial state or in a message state (*i.e* a state of M) but with a clock value greater than one (these processes correspond to lost messages, and cannot be used anymore). Formally, if $\gamma(\mathsf{q}, x) > 0$ with $(\mathsf{q}, x) \neq (m_i, x_i)$ then $\mathsf{q} = \mathsf{q}_0$ or $\mathsf{q} = m \in \mathrm{M}$ and $x > 1$.

Remark 4.1 *The construction of $B_{\mathcal{L}}$ and $\mathcal{P}_{\mathcal{L}}$ ensures that if $\boldsymbol{\gamma}$ encodes (q, mw) and if there is a transition $t = (\mathsf{q}, ?, m, \mathsf{q}')$ in the LCS then there exist $\boldsymbol{\gamma}'$ and $\boldsymbol{\gamma}''$ such that $\boldsymbol{\gamma} \xrightarrow{x,(m,1),\delta} \boldsymbol{\gamma}' \xrightarrow{y,(t,\mathbf{x}_B),\delta'} \boldsymbol{\gamma}''$ for some $y \in \mathbb{R}^+$ and where $x = 1 - x_1$, $\delta = (m, x = 1, !!m_r, \mathbf{x} := 0, \mathsf{q}_0)$ and $\delta' = (t, tt, \varepsilon, \emptyset, \mathsf{q}')$. Moreover $\boldsymbol{\gamma}''$ encodes a configuration (q', w') such that $(\mathsf{q}, mw) \xrightarrow{?,m} (\mathsf{q}', w')$.*

Similarly, if $\boldsymbol{\gamma}$ encodes (q, w) and if there are some processes in the initial state and if there is a transition $t = (\mathsf{q}, !, m, \mathsf{q}')$ in the LCS then there exist $\boldsymbol{\gamma}'$ and $\boldsymbol{\gamma}''$ such that $\boldsymbol{\gamma} \xrightarrow{x,(\mathsf{q},x'),\delta} \boldsymbol{\gamma}' \xrightarrow{0,(m',0),\delta'} \boldsymbol{\gamma}''$ where $x' > 0$, $\delta = (\mathsf{q}, \mathbf{x}_B > 0, !!m_w, \mathbf{x} := 0, t)$ and $\delta' = (m', x = 0, !!ok, \emptyset, m)$. Moreover $\boldsymbol{\gamma}''$ encodes a configuration (q', w') such that $(\mathsf{q}, w) \xrightarrow{!,m} (\mathsf{q}', w')$.

Assume that q is reachable in $\mathcal{L}$ and let $(\mathsf{q}_0, \epsilon) \xrightarrow{a_1} (\mathsf{q}_1, w_1) \dots \xrightarrow{a_n} (\mathsf{q}_n, w_n)$ be the execution such that $\mathsf{q}_n = \mathsf{q}$, and where the a_i's are of the form $?m$ or $!m$, thus read or write actions.

We define the strategy σ such that in a configuration $\boldsymbol{\gamma}$ that encodes (q_i, w_i), the strategy simulates a_{i+1} with the two transitions given by Remark 4.1. There are two cases where this strategy can fail: first, if there are no process in the initial state to simulate the writing; second, if the resulting configuration does not encode the next configuration in the LCS. This case happens when the dynamism (in particular disappearances) in the network does not match the non-deterministic losses in the LCS. In these two cases, when the strategy fails, it resets the network and starts again from

the beginning. This strategy has a positive probability to simulate the execution of the LCS and since it resets the network each time it fails to simulate, the probability to eventually simulate the LCS is 1. Therefore, q is reached almost-surely.

Assume now that there is a strategy σ that reaches q with probability 1. Let ρ be a finite execution respecting σ that reaches q, and consider a sequence of transitions between the initial configuration or a reset and a configuration containing q. By construction, this sequence follows the pattern of transitions given in Remark 4.1. Hence we can extract from this sequence an execution in the LCS that reaches q. This proves that $REACH^{\exists}_{=1}(\mathscr{DC})$ is F_{ω^ω}-hard.

In order to prove the hardness of the other problems, one can use this idea to encode LCS and either reduce from the reachability problem or the termination problem (asking whether all runs of the lossy channel system are finite) which is better suited to verification problems with an universal quantifier on strategies. □

5 Conclusion

In this chapter, we investigated parameterized verification of networks of probabilistic timed protocols. Such networks form a probabilistic and timed extension of the broadcast networks studied in [DSZ11b, DSZ11a, DSZ12].

We have shown that most of the qualitative parameterized verification problems are undecidable for a static semantics. The proofs of undecidability are done through simulation of 2-counter machines, using the processes to encode the counter values. Then, taking inspiration from lossy channel systems, which are an other model combining probabilities and non-determinism, we considered dynamic networks, *i.e.* networks in which the number of processes evolves along the computation according to probabilistic distributions. We have shown that the dynamic networks enjoy a finite attractor property similar to the one of lossy channel systems. This is achievable only through a first abstraction, namely the region abstraction, that allows to abstract away the real values of the clocks. This abstraction generalizes the region abstraction for timed automata in a non trivial way in order to cope with the unbounded and dynamic number of clocks. We further provided a well-quasi-order on the extended regions, and together with the effective computation of the predecessor operation and the finite attractor property, obtained the decidability of the parameterized qualitative problems. Notice that the termination of the decision procedure is obtained via the classical backward computation on well-structured transition systems and is thus of high complexity (Ackerman-hard). However, we also provide a reduction from the reachability problem in lossy channel systems to show that these problems are in fact non primitive recursive.

An attempt towards the decision of quantitative properties has been made. Even though the parameterized quantitative problems are still open, we gave a procedure to approximate the minimal probability to reach a control state for a fixed number of processes. This approximation scheme is adapted from one of the probability approximations for fully probabilistic lossy channel systems [ABRS05] or decisive Markov chains [AHM07]. The adaptation is not trivial since we possibly have to deal with an

infinite number of schedulers that resolve non determinism. Our approximation scheme is interesting in itself since it only exploits general properties, for examples the effectiveness of the predecessor operator, the effectiveness of computing the set of states from where we can avoid the target with probability one, and the finite attractor property. Therefore, it can easily be generalized to a larger class than dynamic networks, that would in particular include probabilistic lossy channel systems with insertions.

The decision algorithms we have developed have a high worst-case complexity. In order to establish whether they are applicable in practice, I started to implement them into a prototype tool. In any case, it would be interesting to have an answer for every number of processes, even to a high cost, compared to a cheaper solution but only for a finite bounded number of processes. This tool is developed in C in order to use the Antichain and Pseudo-Antichain Library Aapal [AaP], a C library dedicated to the representation and manipulation of antichains and pseudo-antichains, which are well-suited data structures for partially ordered sets.

The high complexity lower-bounds are not really surprising since parameterized reachability in untimed non-probabilistic clique networks is already Ackermann hard. In order to achieve better complexity bounds, we considered an other model, reconfigurable broadcast networks, where the broadcast does not reach all the components but only a subset, chosen non-deterministically [DSTZ12]. Since parameterized reachability is tractable for this model, there is hope that a probabilistic extension will have manageable complexity decision procedures. This is the subject of the next chapter.

Chapter IV

Selective broadcast networks of probabilistic protocols

1 Introduction

In this chapter, we introduce and study the parameterized verification of selective broadcast networks of probabilistic protocols. This model extends the model of reconfigurable broadcast networks studied in [DSTZ12] by allowing probabilistic internal actions: a process can change its state according to a probabilistic distribution. The main difference with the clique networks presented in the previous chapter (see Chapter III) is that the broadcast no longer reaches all processes but a subset of them, chosen non-deterministically. The semantics of reconfigurable broadcast networks was given in terms of an infinite state transition system. Here, due to the different possibilities of sending messages from different nodes and also to the non-determinism of the protocol itself, we obtain an infinite state system with probabilistic and non-deterministic choices.

In selective broadcast networks, sent messages may reach a subset of processes. This allows to model peer to peer applications in networks equipped with a switch that redirect the messages to a subset of participants. This is in contrast with clique networks that are closer to networks equipped with a hub that redistributes all the messages to everyone. Selective broadcast networks can also be useful to model wireless systems with high mobility. For example, in a flock of birds all equipped with a wireless sensor, the reception of a message broadcast by a sensor is limited to the sensors on the birds near the emitter. Moreover, due to the movement of the birds, the set of birds in reach between two messages can be totally different, from the whole flock if all the birds are close together, to no one if the bird sending the message took its distance with the flock.

We propose, in this chapter, to study the probabilistic version of the parameterized control state reachability by seeking for the existence of a scheduler, resolving non-determinism, which minimizes or maximizes the probability to reach a configuration in which at least one process is in a specific state. We focus on the qualitative aspects of this problem by comparing extremal probabilities only to 0 and 1.

In the previous chapter (Chapter III), we have seen that clique networks of timed probabilistic protocols are a powerful model leading to undecidability of the qualitative parameterized reachability problems. However, taking inspiration from a well-known infinite state model combining non-determinism and probabilities, namely lossy channel systems [BS03, BBS07, BS13], we regained decidability when considering networks with probabilistic disappearance and creation of processes *i.e.* networks for which the size evolves randomly over time. However, the decidability comes at the price of high complexity. In order to achieve better complexity bounds, this chapter is devoted to the study of selective broadcast networks. Indeed, the complexity for the reachability problem is much lower (PTIME) in reconfigurable broadcast networks. Moreover, instead of applying well structured transition systems techniques which have a high complexity, in this chapter we adapt a technique (see for instance [CdAFL09]) for the verification of finite probabilistic models, which consists in translating a Markov decision process into a 2-player game, for which the existence of a winning strategy is equivalent to the reachability questions in the MDP. However, due to the unknown number of participants in the parameterized verification problem, we have an infinite model, hence we cannot apply directly this solution.

In order to solve this problem, we introduce selective broadcast of parity protocols, which are games on a network composed of an unknown number of processes that all run the same finite state parity protocol. The novelty is that there are now two players and a parity objective. The states of the parity protocols are thus partitioned into player 1 and player 2 states. Moreover, a parity function assigns to each transition of the protocol a given parity. The aim for player 1 is to achieve an execution in which the maximal parity seen infinitely often is even, the aim for player 2 is to make it odd. Since the second player is introduced in order to simulate probabilistic choices, it is less powerful than player 1. Indeed, in selective broadcast of parity protocols, the first player is the one deciding which process plays, which processes receive the messages and the action to play when the selected process is in a state belonging to player 1. However, the second player can only choose the action to play when the first player has chosen a process in a state belonging to player 2. In this setting, we show that one can decide in co-NP whether player 1 has a strategy that fulfills a parity objective for any choice made by player 2. Moreover, we provide a translation, at the protocol level, from the parameterized qualitative reachability problems in selective broadcast networks of probabilistic protocols to the parameterized game problem in selective broadcast networks of parity protocols.

In addition to their usefulness to solve the qualitative probabilistic problems, we believe that selective broadcast of parity protocols are an interesting tool on their own. Indeed, this model of distributed games, as well as the techniques used to solve the game, could be of use in different settings of parameterized systems with many identical processes.

The chapter is organized as follow. First, in Section 2, we introduce the model of selective broadcast networks of probabilistic protocols which are a probabilistic extension of the reconfigurable broadcast protocols [DSTZ12]. We define as well the parameterized qualitative state reachability that we study in this chapter. An example of these

problems asks whether there exist a network size and a scheduler resolving the non-determinism that allows to reach almost surely a configuration in which at least one process is in a given control state. In addition to this particular problem, we also study all the variants by comparing to both thresholds 0 and 1, and considering existential and universal quantifiers over the schedulers.

Our proof technique relies on the transformation into distributed games. We therefore introduce in Section 3 the new model of selective broadcast networks of parity protocols. This model is a 2-player game described in a parameterized way. A parametric number of processes run a finite state parity game protocol for which the states are split in two parts: states of player 1 and states of player 2. In this game, at each turn, player 1 has the role of choosing which process will play and the set of its neighbors. The action to be performed is then let to the player to whom the state of the selected process belongs. These choices for both players are called strategies, and we say that a strategy is winning for player 1 if for all strategies of player 2, the maximal parity seen infinitely often on the transition of the resulting run is even. We thus define the parameterized game problem as whether there exists a network size such that player 1 has a winning strategy.

We present, in Section 3.3, an algorithm that allows one to solve the parameterized game problems. This result is obtained thanks to a key argument stating that one can consider only very simple strategies of player 2 in order to decide whether player 1 has a winning strategy. These simple strategies for player 2 are called state-based and consist of strategies that always choose the same edges at the protocol level independently of the whole network configuration, or history. The algorithm then consists in guessing such a strategy, candidate to be winning for player 2, and checking whether there exists an infinite path of even parity. This last part is done thanks to a counting abstraction, translating the problem into a parameterized VASS for which finding a loop with positive effect is known to be solvable in polynomial time [KS88]. We thus obtain an NP algorithm for the parameterized game problem.

Finally, Section 4 is dedicated to present several reductions of the probabilistic parameterized problems to parameterized game problems allowing us to derive decidability of these first problems. We also give lower bounds on the complexity of the probabilistic parameterized problems matching the upper bounds of the parameterized game problems. This is performed by reduction from UNSAT, the unsatisfiability problem of formula in conjunctive normal form. We thus obtain that all these parameterized problems, both probabilistic and game, are NP-complete.

2 Selective broadcast networks of probabilistic protocols

This section is devoted to the introduction of probabilistic protocols and to the semantics we attach to selective broadcast networks of such protocols. A selective broadcast network of probabilistic protocols is a network in which the number of participants is fixed but unknown, and in which the participants follow the same behaviour described as a probabilistic protocol. Each participant can interact with the others via broad-

cast and reception of messages. Contrary to clique networks studied in the previous chapter, these broadcasts do not necessarily reach all the participants but rather a non-deterministically selected subset of participants, hence the name of *selective* networks.

2.1 Probabilistic protocols

In order to model randomized algorithms, we now introduce probabilistic protocols. This model is a probabilistic extension of the broadcast protocol model studied in [DSZ11b, DSZ11a, DSZ12] or, equivalently, a restriction of the probabilistic timed protocol model defined in Definition 2.1 without the clock variable, or equivalently, where the guards are always true. We give here an other definition in order to emphasize this difference and to simplify the notations of this chapter.

Definition 2.1 (Probabilistic protocol) *A* probabilistic protocol *is a tuple* $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \Sigma, \Delta)$ *where:*

- Q *is a finite set of states partitioned into probabilistic states* $\mathrm{Q}^{(p)}$ *and non-deterministic states* $\mathrm{Q}^{(n)}$*,*
- $\mathsf{q}_0 \in \mathrm{Q}^{(n)}$ *is the initial control state,*
- Σ *is a finite alphabet of messages,*
- Δ *is the edge relation, partitioned into:*
  - *internal actions:* $\Delta_i \subseteq (\mathrm{Q}^{(n)} \times \{\varepsilon\} \times \mathrm{Q})$*,*
  - *communications:* $\Delta_c \subseteq (\mathrm{Q}^{(n)} \times \{!!m, ??m \mid m \in \Sigma\} \times \mathrm{Q}^{(n)})$*, that can be broadcasts* $(!!m)$ *or receptions* $(??m)$
  - *probabilistic transitions:* $\Delta_p : \mathrm{Q}^{(p)} \to \mathit{Dist}\left(\mathrm{Q}^{(n)}\right)$ *that, for all* $\mathsf{q} \in \mathrm{Q}^{(p)}$*, associates a unique distribution.*

We refer to broadcasts and internal actions as active actions since they can be initiated by a process, in opposition to receptions that are only triggered when another process sends a message.

As for probabilistic timed protocols, broadcasts and receptions are non-deterministic and only internal actions can lead to probabilistic behaviors. Again, this is not a real restriction since probabilistic choices for broadcasts and receptions can be encoded thanks to additional states and internal actions (see Remark 2.1). Also, simplifying definitions (with the same idea as Remark 2.2), we consider that there is (at least) one reception of every message from every state, *i.e.* one could say that the protocols are "input complete". Again, this is not a real restriction since, in the semantics defined in the following section, we consider non-blocking broadcasts. We can hence add a self loop receiving a message for all the unspecified receptions.

Example 2.1 *A toy example of a probabilistic protocol is given in Figure 2.1. For simplicity, the self loops for the receptions are omitted. For example, in state* q_2 *there is a self loop* $(\mathsf{q}_2, ??a, \mathsf{q}_2)$ *that is not represented.*

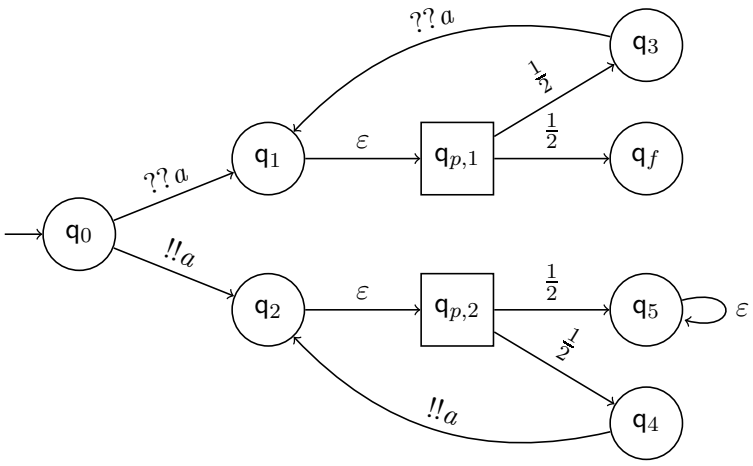


Figure 2.1: Simple example of probabilistic protocol.

The initial state of this protocol is q_0*. From here, there is a broadcast transition* $(\mathsf{q}_0, !!a, \mathsf{q}_2) \in \Delta_c$ *going to state* q_2 *by broadcasting message* a*. Upon reception of message* a *in state* q_0*, there is a reception transition* $(\mathsf{q}_0, ??a, \mathsf{q}_1) \in \Delta_c$ *that leads to* q_1*.*

In state q_1*, there is an internal transition* $(\mathsf{q}_1, \varepsilon, \mathsf{q}_{p1}) \in \Delta_i$ *that leads to* $\mathsf{q}_{p,1}$*. The distribution associated to* $\mathsf{q}_{p,1}$ *gives, with probability* $\frac{1}{2}$*,* q_f*, and with probability* $\frac{1}{2}$*,* q_3*.*

2.2 Semantics of selective broadcast networks

We now introduce selective broadcast networks. In contrast to clique networks (defined in Section 2.2), the broadcast may not reach all the participants. Indeed the sets of participants receiving the messages are chosen non-deterministically.

Definition 2.2 (Selective broadcast network of probabilistic protocol) *A selective broadcast network* $S(\mathcal{P}^N)$ *is composed of* $N \in \mathbb{N}$ *copies, called* processes*, of a probabilistic protocol* $\mathcal{P}$*.*

In a selective broadcast network $S(\mathcal{P}^N)$, as in a static clique network defined in Section 2.2, the number of processes is fixed to N along all computations. However, the communication topology is no longer a clique but evolves over the computations. More precisely, at each broadcast, a set of processes is selected as destination of this broadcast and only them receive the message.

Example 2.2 *The intuitive semantics of a selective broadcast network composed of processes running the protocol given in Figure 2.1 is the following.*

Each process starts in state q_0*. From here, a process is selected to perform an active action (either an internal transition or a broadcast). In state* q_0*, the only possible active action is a broadcast of* a*, leading to state* q_2*. While executing a broadcast, a set of processes is selected. This set contains all the processes that receive the broadcast*

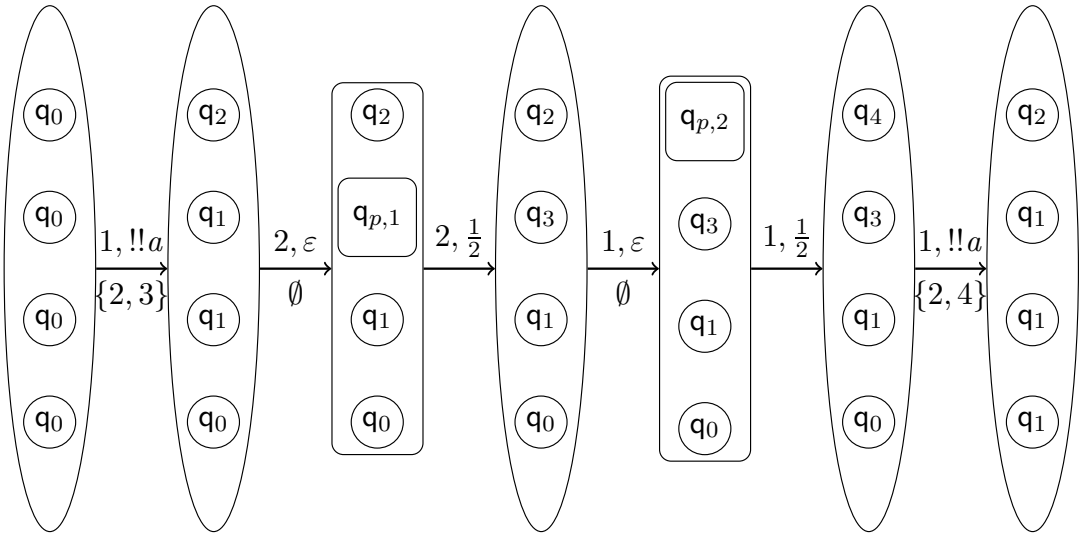


Figure 2.2: An execution of a selective broadcast network of 4 processes running the protocol represented in Figure 2.1.

message. In this case, all the processes in the reception set receive message a by taking the transition $(\mathsf{q}_0, ??a, \mathsf{q}_1)$ *and thus move to state* q_1.

In the case of an internal action, for example from state q_1*, taking the transition* $(\mathsf{q}_1, \varepsilon, \mathsf{q}_{p,1})$ *only affects the process performing it. Then, in state* q_{p1}*, the process moves to* q_3 *with probability* $\frac{1}{2}$ *and moves to* q_f *with probability* $\frac{1}{2}$.

An example of an execution is given in Figure 2.2. In this execution, first process 1 *is selected to perform a broadcast of a by moving to state* q_2*. For this broadcast, the selected reception set consists in processes* 2 *and* 3*, only them receive the message and move to* q_1*. Process* 4 *is not concerned by the broadcast and thus stays in state* q_0*. After that, process* 2 *performs an internal action leading to* $\mathsf{q}_{p,1}$*. Since it is a probabilistic state, it is directly followed by a probabilistic transition leading to* q_3 *with probability* $\frac{1}{2}$*. Next, process* 1 *performs an internal action and moves to* $\mathsf{q}_{p,2}$ *and then to* q_4 *with probability* $\frac{1}{2}$*. Notice that for internal actions, the reception set does not matter (here it is chosen to be the empty set but any other set would give the same result). Last, process* 1 *performs a broadcast of a and moves to* q_2*, with reception set* $\{2, 4\}$*, hence processes* 2 *and* 4 *receive the message a and move to* q_1.

Since we have to keep track of which processes are in the reception set, we now consider that processes have a unique identifier. However, theses identifiers are not manipulated during the computation, they are only used to specify the processes performing the action as well as the processes receiving the messages.

We define a *configuration* as, either a vector of control states $\gamma \in \mathrm{Q}^N$ of size N, or as a pair $(\gamma, p_{id}) \in \mathrm{Q}^N \times [1 \dots N]$, denoting that the process with index p_{id} has to perform the next action.

The semantics of a selective broadcast network $S(\mathcal{P}^N)$ is the Markov decision process $\mathscr{S}(\mathcal{P}^N) = (\Gamma, \Gamma^{(n)}, \Gamma^{(p)}, \gamma_0, T^{(n)}, T^{(p)})$, where:

- $\Gamma \subseteq \mathrm{Q}^N \cup (\mathrm{Q}^N \times [1 \dots N])$ is the set of configurations partitioned into non-deterministic configurations $\Gamma^{(n)}$ and probabilistic configurations $\Gamma^{(p)}$;
- $\Gamma^{(n)} = {\mathrm{Q}^{(n)}}^N \cup ({\mathrm{Q}^{(n)}}^N \times [1 \dots N])$ is the set of non-deterministic configurations;
- $\Gamma^{(p)} = \{(\gamma, p_{id}) \in \mathrm{Q}^N \times [1 \dots N]) \mid \gamma[p_{id}] \in \mathrm{Q}^{(p)}\}$ is the set of probabilistic configurations, *i.e.* configurations in which the designate process is in a probabilistic state;
- $\gamma_0 \in \Gamma^{(n)}$ is the initial configuration, defined by for all $p_{id} \in [1 \dots N]$, $\gamma_0[p_{id}] = \mathsf{q}_0$;
- $T^{(p)}$ is the transition probability function naturally lifted from Δ_p, *i.e.* if $\Delta_p(\mathsf{q}) = d$ and $(\gamma, p_{id}) \in \Gamma^{(p)}$ with $\gamma[p_{id}] = \mathsf{q}$ then $T^{(p)}((\gamma, p_{id})) = d'$ with $d'(\gamma') = 0$ if there exists $p_{id} \in [1 \dots N] \setminus \{p_{id}\}$ such that $\gamma[p'_{id}] \neq \gamma'[p'_{id}]$ and $d'(\gamma') = d(\gamma'[p_{id}])$ otherwise;
- $T^{(n)} \subseteq (\Gamma^{(n)} \times [1 \dots N] \times \Gamma^{(n)}) \cup (\Gamma^{(n)} \times \Delta \times 2^{[1 \dots N] \setminus \{p_{id}\}} \times \Gamma)$ is the non-deterministic transition relation defined as follow:

  Choice of a process $(\gamma, p_{id}, (\gamma, p_{id})) \in T^{(n)}$ if there exists an active action from $\gamma[p_{id}]$

  Internal action $((\gamma, p_{id}), \delta, R, \gamma') \in T^{(n)}$ if $\delta = (\gamma[p_{id}], \varepsilon, \gamma'[p_{id}])$ and $\gamma'[p_{id}] \in \mathrm{Q}^{(n)}$ or $((\gamma, p_{id}), \delta, R, (\gamma', p_{id})) \in T^{(n)}$ if $\delta = (\gamma[p_{id}], \varepsilon, \gamma'[p_{id}])$ and $\gamma'[p_{id}] \in \mathrm{Q}^{(p)}$ where, in both cases, γ' is such that for all $p'_{id} \in [1 \dots N] \setminus \{p_{id}\}$, $\gamma'[p'_{id}] = \gamma[p'_{id}]$.

  Communication $((\gamma, p_{id}), \delta, R, \gamma') \in T^{(n)}$ if $\delta = (\gamma[p_{id}], !!m, \gamma'[p_{id}])$ and for all $p^R_{id} \in R$ $(\gamma[p^R_{id}], ??m, \gamma'[p^R_{id}]) \in \Delta$ and for all $p'_{id} \in [1 \dots N] \setminus (\{p_{id}\} \cup R)$, $\gamma'[p'_{id}] = \gamma[p'_{id}]$.

We denote $\gamma \to \gamma'$ for $(\gamma, \gamma') \in T^{(n)}$ or if $T^{(p)}(\gamma)(\gamma') > 0$.

A finite execution ρ is a finite sequence of configurations $\gamma_1 \dots \gamma_n$ such that for all $i \in [1 \dots n-1]$ $\gamma_i \to \gamma_{i+1}$. An infinite (or maximal) execution is an infinite sequence of configurations such that all its prefixes are finite executions. We denote by $\mathcal{E}$ the set of all (finite and infinite) executions.

Remark 2.1 (Non-blocking execution) *In this chapter, we assume that all maximal executions are infinite,* i.e. *that from all configurations γ there exists a configuration γ' such that $\gamma \to \gamma'$. This restriction will be explained in Section 3.3.1 for the proof of Lemma 3.1. Notice that the example given in Figure 2.1 satisfies this assumption. Indeed, for any execution, there is at least a process in the bottom part that can cycle on the states q_2, $\mathsf{q}_{p,2}$ and q_4 or loop in state q_5.*

2.3 Parameterized probabilistic verification problems

In order to define the relevant parameterized qualitative reachability problems, we first define strategies for selective broadcast networks that will be in charge of resolving nondeterminism. Hence strategies choose the process performing an action, the performed action, as well as the reception set associated with this action.

Definition 2.3 (Strategy) *A* strategy *for a selective broadcast network is a scheduler for the Markov decision process $S(\mathcal{P}^N)$ representing the semantics. Formally, a strategy is a function $\sigma : \mathcal{E} \to [1 \dots N] \cup \Delta \times 2^{[1 \dots N]}$, specifying for each execution either the process selected to play the next action or the action to be performed together with the reception set. Moreover, a strategy is such that for all executions $\rho = \gamma_1 \dots \gamma_n$, there exists a configuration γ_{n+1} such that $\gamma_n \xrightarrow{\sigma(\rho)} \gamma_{n+1}$.*

The problems we propose to investigate are qualitative questions, where we will compare to 0 or 1 the probability of reaching a particular state in a network built over a probabilistic protocol. Given a probabilistic protocol $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \Sigma, \Delta)$ and a state $\mathsf{q}_f \in \mathrm{Q}$, we denote by $\Diamond \mathsf{q}_f$ the set of all maximal paths of $S(\mathcal{P}^N)$ of the form $\gamma_0 \gamma_1 \cdots$ such that there exists $i \in \mathbb{N}$ verifying $\gamma_i[p_{id}] = \mathsf{q}_f$ for some process p_{id}, i.e. the set of paths which eventually reach a configuration where a process is in state q_f.

We are now ready to provide the definition of the different qualitative reachability problems studied in this chapter. Given a quantifier $\exists\forall \in \{\exists, \forall\}$, a qualitative probability threshold $p \in \{0, 1\}$ and a comparison operator $\sim \in \{<, =, >\}$, let $REACH^{\exists\forall}_{\sim p}(\mathscr{S})$ be the following decision problem:

Input:
A probabilistic protocol $\mathcal{P}$, and a control state $\mathsf{q}_f \in \mathrm{Q}$.
Question:
Does there exist a network size $N \in \mathbb{N}$ such that $\exists\forall\sigma,\ \mathbb{P}_\sigma(S(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) \sim p$

Remark that these decision problems are parameterized by the network size. Assuming a fixed initial configuration, all problems boil down to the analysis of a finite Markov decision process, and are decidable in PTIME [KNPS08, BK08] (see Section I.3).

3 Selective broadcast networks of parity protocols

In order to solve the parameterized qualitative reachability problems for selective broadcast networks $REACH^{\exists\forall}_{\sim p}(\mathscr{S})$ defined in the previous section, we use a polynomial transformation to selective broadcast game networks of parity protocols. These protocols are a 2-player extension of the broadcast protocols. This section is devoted to the introduction of parity protocols, and to the semantics we attach to selective broadcast game networks of parity protocols.

3.1 Parity protocols

Parity protocols are a two player extension of broadcast protocols (see Section I.3.3). Equivalently, they are defined as probabilistic protocols, except that there are no probability but a second player who controls a subset of states (previously probabilistic states). From such states, the outcome is no longer given by a probabilistic distribution but is chosen by player 2 among the possible internal actions.

Definition 3.1 (Parity protocol) *A* parity protocol *is a tuple* $\mathcal{G} = (\mathrm{L}, \mathrm{L}^{(1)}, \mathrm{L}^{(2)}, \mathsf{l}_0, \Sigma, \Delta, \mathrm{col}, \mathrm{safe})$ *where:*

- L *is a finite set of control states partitioned into* $\mathrm{L}^{(1)}$ *states of player 1 and* $\mathrm{L}^{(2)}$ *states of player 2;*
- $\mathsf{l}_0 \in \mathrm{L}^{(1)}$ *is the initial control state;*
- Σ *is a finite message alphabet;*
- Δ *is the transition relation partitioned into:*
  - *internal actions for player 1:* $\Delta \subseteq \mathrm{L}^{(1)} \times \{\varepsilon\} \times \mathrm{L}$,
  - *internal actions for player 2:* $\Delta \subseteq \mathrm{L}^{(2)} \times \{\varepsilon\} \times \mathrm{L}$,
  - *broadcasts and receptions:* $\Delta \subseteq \mathrm{L}^{(1)} \times \{!!m, ??m \mid m \in \Sigma\} \times \mathrm{L}$;
- $\mathrm{col} : \Delta \to \mathbb{N}$ *is the coloring function, assigning a* color *(or* parity*);*
- $\mathrm{safe} \subseteq \Delta$ *is a set of safe edges.*

As before, we refer to broadcast and internal actions as *active actions*. Moreover, we denote by $\mathit{ActStates} = \{\mathsf{l} \in \mathrm{L} \mid \exists \mathsf{l}' \in \mathrm{L}, (\mathsf{l}, \varepsilon, \mathsf{l}') \in \Delta \ \texttt{or}\ \exists m \in \Sigma, (\mathsf{l}, !!m, \mathsf{l}') \in \Delta\}$ the set of all states that are sources of at least one active action.

Notice that the protocols are equipped by a coloring function that colors the edges, and by a subset of safe edges. Intuitively, the unsafe edges are edges that player 1 must avoid to win the safety game. Additionally, for player 1 to win the parity game, the maximal color taken infinitely often should be even. The semantics is detailed in Section 3.2.

Example 3.1 *An example of a parity protocol is given in Figure 3.3. In this example (as in the rest of this chapter), the states of player 1 are circles and those of player 2 are squares; the color of each edge is given separated of the action by a colon; and the unsafe edges are dotted.*

The initial state of this protocol is l_0*, and belongs to player 1. From here, there is a broadcast of message* a*, leading to the state* l_2 *also belonging to player* 1*. This edge has color* 0 *and is a safe edge. Formally,* $\delta = (\mathsf{l}_0, !!a, \mathsf{l}_2) \in \Delta \cap \mathrm{safe}$*, and* $\mathrm{col}(\delta) = 0$.

In state l_2^2*, player* 2 *can choose between two internal transitions, an unsafe edge* $(\mathsf{l}_2^2, \varepsilon, \mathsf{l}_5)$ *of parity 0, and a safe edge* $(\mathsf{l}_2^2, \varepsilon, \mathsf{l}_4)$ *of parity* 1.

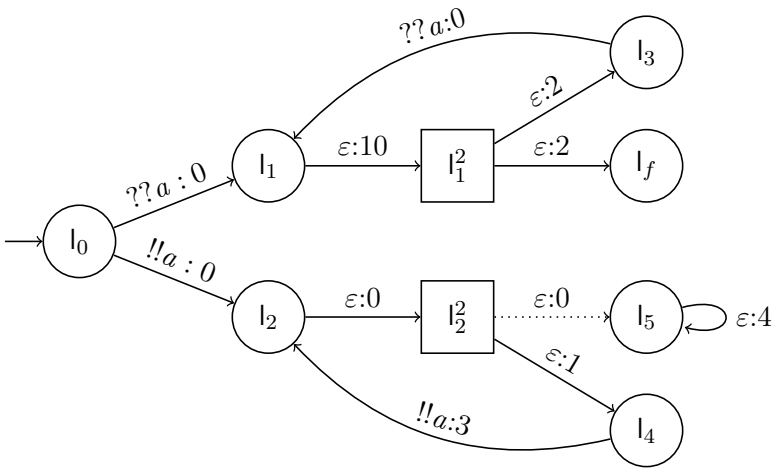


Figure 3.3: Simple example of a parity protocol.

3.2 Semantics of selective broadcast game networks of parity protocols

Parity protocols form the basic (finite) ingredient of an infinite number of 2-player games that are formed by networks of processes running the same parity protocol. Here again, similar to all models of networks of many identical processes, the infiniteness comes from the unbounded size of the networks. In this subsection, we define the semantics that we attach to such networks as well as the parametric safety and parity winning conditions on the games.

Definition 3.2 (Selective broadcast game networks of parity protocol) *A selective broadcast game network $SG(\mathcal{G}^N)$ is composed of $N \in \mathbb{N}$ copies, called* processes, *of a parity protocol $\mathcal{G}$.*

Example 3.2 *An example of a play is given in Figure 3.4. In this example, the choices made by player* 1 *are represented in blue and the choices made by player* 2 *in red. First, player* 1 *chooses that process* 1 *should perform an action. Since process* 1 *is in state $\mathsf{l}_0 \in \mathrm{L}^{(1)}$ that belongs to player* 1*, player* 1 *also chooses the action to take (here the broadcast of message a with reception set $\{2,3\}$); it also chooses the reception edges to be taken by processes 2 and 3 (even though there are no choice here).*
Next, player 1 *decides that process* 2 *should do an action. Again, since process 2 is in a state of player* 1*, the action to take is also chosen by player* 1*, and similarly for the next step in which process* 1 *takes an internal transition.*
In the fourth step of the play, player 1 *chooses that it is the turn of process* 2 *to play again. Since process* 2 *is in state $\mathsf{l}_1^2 \in \mathrm{L}^{(2)}$, player* 2 *chooses the transition to be taken by process 2, here it moves to l_3. Similarly, process* 1 *moves to state l_4. Lastly, player* 1 *chooses that process 1 broadcasts message a to process 2.*

Notice that this run is safe since it does not take any unsafe transitions. Since the last and second configurations are the same, we can consider the infinite play that loops

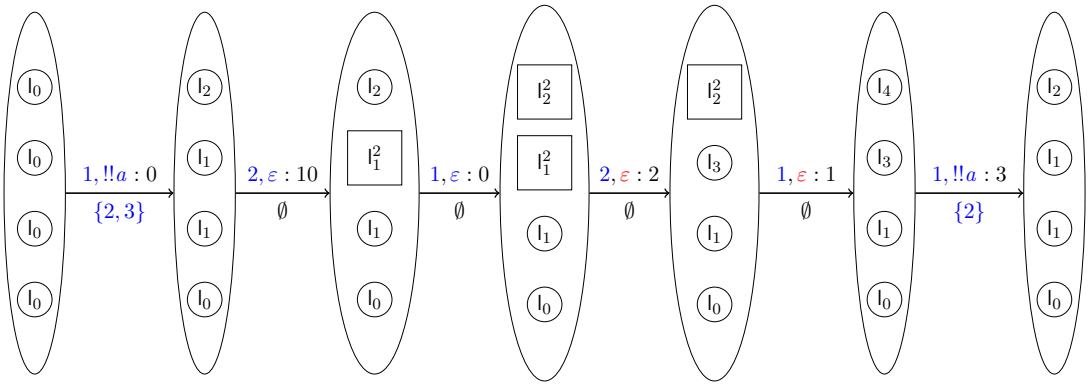


Figure 3.4: Example of a play with 4 processes running the parity protocol given in Figure 3.3.

from the last configuration to the second and where both players keep making the same decisions. Naturally, this infinite play is also safe, and it contains infinitely often the colors 10, 0, 2, 1 *and* 3. *Since the maximal parity seen infinitely often is even (here 10) this infinite play is winning for player 1.*

The semantics associated with a selective broadcast game network $SG(\mathcal{G}^N)$ is given in terms of a 2-player game whose definition is similar to Markov decision process associated with selective broadcast network of probabilistic protocol. Recall that one can see a Markov decision process as a $1-\frac{1}{2}$ player game. Here also, player 1 has the ability to choose a node which will perform an action, and according to the control state labelling this node, either player 1 or player 2 will then perform the next move. Note that the roles of player 1 and player 2 are not symmetric, because player 1 only, by definition of Δ, can initiate a communication, whereas player 2 may only perform internal actions. Also, player 1 chooses the reception set in case of a communication.

Let $\mathcal{G} = (\mathrm{L}, \mathrm{L}^{(1)}, \mathrm{L}^{(2)}, \mathsf{l}_0, \Sigma, \Delta, \mathrm{col}, \mathrm{safe})$ be a parity protocol. We define a *configuration* of the selective broadcast game network $SG(\mathcal{G}^N)$ as a vector of size N, $\lambda \in \mathrm{L}^N$ of control states, or as a pair $(\lambda, p_{id}) \in \mathrm{L}^N \times [1 \dots N]$, denoting that the process with index p_{id} has been chosen to perform the next action.

Player 1 owns configurations where no process is selected to perform an action, and, for $i \in \{1,2\}$, player i owns configurations (λ, p_{id}) where the selected process p_{id} is in a player i control state. Formally:

$$\Lambda_{\mathcal{G}}^{(1)} = \mathrm{L}^N \cup \{(\lambda, p_{id}) \in \mathrm{L}^N \times [1 \dots N] \mid \lambda[p_{id}] \in \mathrm{L}^{(1)}\}$$

and

$$\Lambda_{\mathcal{G}}^{(2)} = \{(\lambda, p_{id}) \in \mathrm{L}^N \times [1 \dots N] \mid \lambda[p_{id}] \in \mathrm{L}^{(2)}\}.$$

The semantics of the network built over $\mathcal{G}$ is then given in terms of the 2-player game $\mathscr{SG}(\mathcal{G}^N) = (\Lambda_{\mathcal{G}}, \Lambda_{\mathcal{G}}^{(1)}, \Lambda_{\mathcal{G}}^{(2)}, T_{\mathcal{G}}, \mathrm{col}_{\mathcal{G}}, \mathrm{safe}_{\mathcal{G}})$ where $T_{\mathcal{G}} \subseteq \Lambda_{\mathcal{G}} \times \Lambda_{\mathcal{G}}$ is defined using

reconfiguration and process choices for player 1 and internal and communication rules as the one defined in the case of probabilistic protocols (see page 99). The coloring function $\mathrm{col}_{\mathcal{G}} : T_{\mathcal{G}} \to \mathbb{N}$ and the safe set of transitions $\mathrm{safe}_{\mathcal{G}} \subseteq T_{\mathcal{G}}$ are lifted from their analog at the protocol level, col and safe. More precisely, the color associated with an internal action is lifted at the level of configurations, and in case of communication, the color of a communication transition between configurations is defined as the color of the corresponding broadcast transition (colors of reception edges are ignored). Moreover, a transition at the level of configurations is declared safe if and only if all individual transitions that compose it are safe. Finally, we will say that a configuration $\lambda \in \mathrm{L}^N$ is initial if $\lambda[p_{id}] = \mathsf{l}_0$ for all $p_{id} \in [1 \dots N]$.

Formally: $T_{\mathcal{G}} \subseteq \Lambda_{\mathcal{G}} \times \Lambda_{\mathcal{G}}$, $\mathrm{col}_{\mathcal{G}} : T_{\mathcal{G}} \to \mathbb{N}$ and $\mathrm{safe}_{\mathcal{G}} \subseteq T_{\mathcal{G}}$ are defined as follows:

Choice of a process $(\lambda, p_{id}, (\lambda, p_{id})) \in T_{\mathcal{G}}$ if $\lambda[p_{id}] \in \mathit{ActStates}$;
in this case, the transition is safe and of color 0 *i.e.* $\mathrm{col}_{\mathcal{G}}((\lambda, p_{id}, (\lambda, p_{id}))) = 0$ and $(\lambda, p_{id}, (\lambda, p_{id})) \in \mathrm{safe}_{\mathcal{G}}$.

Internal action $((\lambda, p_{id}), \delta, R, \lambda') \in T_{\mathcal{G}}$ if $\delta = (\lambda[p_{id}], \varepsilon, \lambda'[p_{id}])$ and γ' is such that for all $p'_{id} \in [1 \dots N] \setminus \{p_{id}\}$, $\gamma'[p'_{id}] = \gamma[p'_{id}]$;
in this case, $\mathrm{col}_{\mathcal{G}}(((\lambda, p_{id}), \delta, R, \lambda')) = \mathrm{col}(\delta)$ and $(\lambda, p_{id}, (\lambda, p_{id})) \in \mathrm{safe}_{\mathcal{G}}$ if and only if $\delta \in \mathrm{safe}$.

Communication $((\lambda, p_{id}), \delta, R, \lambda') \in T_{\mathcal{G}}$ if $\delta = (\lambda[p_{id}], !!m, \lambda'[p_{id}])$ and for all $p^R_{id} \in R$ $(\lambda[p^R_{id}], ??m, \lambda'[p^R_{id}]) \in \Delta$ and for all $p'_{id} \in [1 \dots N] \setminus (\{p_{id}\} \cup R)$, $\lambda'[p'_{id}] = \lambda[p'_{id}]$;
in this case, $\mathrm{col}_{\mathcal{G}}(((\lambda, p_{id}), \delta, R, \lambda')) = \mathrm{col}(\delta)$ and $((\lambda, p_{id}), \delta, R, \lambda') \in \mathrm{safe}_{\mathcal{G}}$ if $\delta \in \mathrm{safe}$ and for all $p^R_{id} \in R$, $(\lambda[p^R_{id}], ??m, \lambda'[p^R_{id}]) \in \mathrm{safe}$.

A *finite path* ρ is a finite sequence of configurations $\lambda_0\lambda_1 \cdots \lambda_n \in \Lambda^*_{\mathcal{G}}$ such that $(\lambda_i, \lambda_{i+1}) \in T_{\mathcal{G}}$ for all $0 \leq i < n$. Such a path is said to start at configuration λ_0. An *infinite path* is an infinite sequence $\rho \in \Lambda^\omega_{\mathcal{G}}$ such that any finite prefix of ρ is a finite path. *Maximal paths* in $\mathscr{SG}(\mathcal{G}^N)$ are infinite paths. As for selective broadcast networks (see Remark 2.1), we assume that there is no finite maximal path, *i.e.* there is no paths ending in a configuration λ such that there is no configuration λ' with $(\lambda, \lambda') \in T_{\mathcal{G}}$. This restriction will be explained in Section 3.3.1 for the proof of Lemma 3.1.

A *strategy* for player 1 dictates its choices in configurations of $\Lambda^{(1)}$. More precisely, a strategy for player 1 in the game $\mathscr{SG}(\mathcal{G}^N) = (\Lambda_{\mathcal{G}}, \Lambda^{(1)}_{\mathcal{G}}, \Lambda^{(2)}_{\mathcal{G}}, T_{\mathcal{G}}, \mathrm{col}_{\mathcal{G}}, \mathrm{safe}_{\mathcal{G}})$ is a function $\alpha : \Lambda^*_{\mathcal{G}}\Lambda^{(1)}_{\mathcal{G}} \to \Lambda_{\mathcal{G}}$ such that for every finite path ρ and $\lambda \in \Lambda^{(1)}_{\mathcal{G}}$, $(\lambda, \alpha(\rho\lambda)) \in T_{\mathcal{G}}$. Strategies $\beta : \Lambda^*_{\mathcal{G}}\Lambda^{(2)}_{\mathcal{G}} \to \Lambda_{\mathcal{G}}$ for player 2 are defined symmetrically, and we write $S^{(1)}$ and $S^{(2)}$ for the set of strategies for each player. A *strategy profile* is a pair of strategies, one for each player. Given a strategy profile (α, β) and a network size $N \in \mathbb{N}$, the game $\mathscr{SG}(\mathcal{G}^N)$ gives rise to the following maximal path, also referred to as the *play*, $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) = \lambda_0\lambda_1 \cdots$ such that for all $i \in \mathbb{N}$, if $\lambda_i \in \Lambda^{(1)}_{\mathcal{G}}$ then $\lambda_{i+1} = \alpha(\lambda_0 \dots \lambda_i)$, otherwise $\lambda_{i+1} = \beta(\lambda_0 \dots \lambda_i)$.

The winning condition for player 1 is a subset of plays $\mathit{Win} \subseteq \Lambda^\omega_{\mathcal{G}}$. In this chapter, we characterize winning conditions through safety, parity objectives and combinations

of these two objectives, respectively denoted by Win_s, Win_p and Win_{sp}, and defined as follows:

$$Win_s = \{\rho \in \Lambda^\omega_{\mathcal{G}} \mid \forall 0 \leq i < |\rho| - 1.(\rho(i), \rho(i+1)) \in \text{safe}\}$$
$$Win_p = \{\rho \in \Lambda^\omega_{\mathcal{G}} \mid \max\{n \in \mathbb{N} \mid \forall i \geq 0.\exists j \geq i.\text{col}((\rho(j), \rho(j+1))) = n\} \text{ is even}\}$$
$$Win_{sp} = (Win_p \cap Win_s)$$

The safety objective contains all infinite paths that use only edges in safe. The parity objective contains all infinite paths for which the maximum color visited infinitely often is even. Finally, the safety-parity objective contains the infinite paths which respect both the parity and the safety objectives.

Note that in the context of games played over a finite graph, the safety-parity objective can easily be turned into a parity objective, by removing the unsafe edges and by adding a self-loop of even parity on deadlock states. However, in our case, this transformation is difficult because removing unsafe edges may introduce deadlock configurations (which are forbidden) and since the number of processes is not known *a priori*, adding self-loops only on this deadlock configurations may be difficult.

Finally, we say that a play ρ is *winning for player 1* for an objective $Win \subseteq \Lambda^\omega_{\mathcal{G}}$ if $\rho \in Win$, in the other case it is winning for player 2. Last, as usual, a strategy α for player 1 is a *winning strategy* if for every strategy β of player 2, the play $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta)$ is winning for player 1.

Given a parity protocol $\mathcal{G}$ and a winning condition Win, the game problem is defined as follows:

Input:
A parity protocol $\mathcal{G}$, and a winning condition $Win \in \{Win_s, Win_p, Win_{sp}\}$.
Question:
Does there exist a network size $N \in \mathbb{N}$ such that player 1 has a winning strategy α in the game $\mathscr{SG}(\mathcal{G}^N)$?

3.3 Resolution of the game

3.3.1 Restricting the strategies of player 2

In order to solve the game problem for parity protocols, we first show that we can restrict the strategies of player 2 to what we call state based strategies. These state based strategies always choose from a given control state the same successor, independently of the rest of the configuration and of the game history.

We consider a parity protocol $\mathcal{G} = (\mathrm{L}, \mathrm{L}^{(1)}, \mathrm{L}^{(2)}, \mathsf{l}_0, \Sigma, \Delta, \text{col}, \text{safe})$. We begin by defining state based strategies for player 2 in $\mathscr{SG}(\mathcal{G}^N)$. A *state based behavior* for player 2 in $\mathscr{SG}(\mathcal{G}^N)$ is a function $b : (\mathrm{L}^{(2)} \cap \mathit{ActStates}) \to \Delta$ such that for every $\mathsf{l} \in \mathrm{L}^{(2)} \cap \mathit{ActStates}$, $b(\mathsf{l}) \in \{(\mathsf{l}, \varepsilon, \mathsf{l}') \mid (\mathsf{l}, \varepsilon, \mathsf{l}') \in \Delta\}$. Such a state based behavior induces a *state based strategy* β_b for player 2 in $\mathscr{SG}(\mathcal{G}^N)$ defined as follows: let ρ be a finite path in $\Lambda^*_{\mathcal{G}}$ and $(\lambda, p_{id}) \in \Lambda^{(2)}_{\mathcal{G}}$, we have $\beta_b(\rho\lambda) = \lambda'$ where λ' is the unique configuration obtained from λ by applying, according to the definition of $\mathscr{SG}(\mathcal{G}^N)$,

the rule corresponding to $b(\mathsf{l})$ (*i.e.* the internal action initiated from process p_{id}). We denote by $S^{(2)}_{sb}$ the set of state based strategies for player 2. Note that $\mathcal{G}$ contains a finite number of states and edges, so that the set $S^{(2)}_{sb}$ is finite and of cardinality bounded by $|\Delta|$.

The next proposition shows that we can restrict strategies of player 2 to state based ones in order to solve the game problem for $\mathcal{G}$ when considering safety, parity and safety/parity winning objectives.

Proposition 3.1 *For $Win \in \{Win_s, Win_p, Win_{sp}\}$, we have*

$$\exists N \in \mathbb{N},\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$
$$\Longleftrightarrow$$
$$\forall \beta \in S^{(2)}_{sb}.\ \exists N \in \mathbb{N},\ \exists \alpha \in S^{(1)}, \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win\ .$$

The proof of this proposition is split into two parts. First we establish (Lemma 3.1) that the existence of a winning strategy against each state based strategy of player 2 implies the existence of a winning strategy against any strategy of player 2. This proof shares some similarities with the one showing that memoryless strategies are sufficient for player 2 in energy parity games [CD12]. It consists of an induction on the number of states of player 2 in the parity protocol. In the induction step, a configuration is split into several sub-configurations (one for each state based strategy of player 2) and player 1 navigates among the sub-configurations each time player 2 changes strategy. For instance, if player 2 has two choices, say left and right, then at the beginning player 1 plays in the "left" sub-configuration and in case player 2 chooses right rather than left, then the associated process is moved to the "right"-sub-configuration and the game proceeds in this sub-configuration. Intuitively, we show that if player 1 wins against the stubborn strategy which chooses always left and against the one which chooses always right, then it wins against any strategy of player 2.

In the second part (Lemma 3.2), we show that we can build a unique strategy of player 1 that wins against all state based strategies of player 2, assuming we are given winning strategies of player 1 for each state based strategy of player 2. The idea here is to guess the state based strategy of player 2 , and then play the corresponding winning strategy.

Let us start with the first part of the proof, and the following lemma:

Lemma 3.1 *For $Win \in \{Win_s, Win_p, Win_{sp}\}$, we have:*

$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$
$$\Longleftrightarrow$$
$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)}_{sb},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$

Proof Only the right-to-left implication deserves a proof, since $S^{(2)}_{sb} \subseteq S^{(2)}$. We assume that there exist a number of processes $N \in \mathbb{N}$ and a strategy α for player 1 such that for all state based strategies β of player 2, $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$. The proof is done by induction on $|\mathrm{L}^{(2)}|$, the number of states of player 2 in the parity protocol.

If $\mathrm{L}^{(2)} = \emptyset$, player 2 makes no decision in the game, hence, the set of state based strategies agrees with the set of all strategies and they are empty. Therefore, the base case is obvious.

Assume now that the property holds for $|\mathrm{L}^{(2)}| < n$, and let us consider a parity protocol $\mathcal{G}$ with $|\mathrm{L}^{(2)}| = n$. We suppose that $\mathrm{L}^{(2)} = \{\mathsf{l}_1, \cdots, \mathsf{l}_n\}$. We first rule out the case where in l_n a single internal transition is enabled. In that case, player 2 has no real choice in l_n, and l_n can as well belong to $\mathrm{L}^{(1)}$, so that the induction hypothesis applies. Without loss of generality, we consider that two transitions, $\delta_\ell = (\mathsf{l}_n, \varepsilon, \mathsf{l}_\ell)$ and $\delta_r = (\mathsf{l}_n, \varepsilon, \mathsf{l}_r)$, are enabled in l_n. From $\mathcal{G}$, we derive two variants of the parity protocol $\mathcal{G}_\ell$ and $\mathcal{G}_r$, where only the left transition δ_ℓ (resp. the right transition δ_r) is present, and in which l_n belongs to $\mathrm{L}^{(1)}$. Any state based strategy of player 2 in the game $\mathscr{SG}(\mathcal{G}_\ell^N)$ is also a state based strategy in $\mathscr{SG}(\mathcal{G}^N)$, and similarly for $\mathscr{SG}(\mathcal{G}_r^N)$. Therefore, the strategy α wins against all local strategies in both $\mathscr{SG}(\mathcal{G}_l^N)$ and $\mathscr{SG}(\mathcal{G}_r^N)$.

Because the number of player 2 states in $\mathcal{G}_\ell$ [resp. $\mathcal{G}_r$] is at most $n-1$, the induction hypothesis applies, and there exist a number of processes N_ℓ [resp. N_r] and a strategy α_ℓ [resp. α_r] for player 1 such that α_ℓ [resp. α_r] wins against all strategies of player 2 in $\mathscr{SG}(\mathcal{G}_\ell^{N_\ell})$ [resp. $\mathscr{SG}(\mathcal{G}_r^{N_r})$]. We now explain how to define a winning strategy α for player 1 in the game $\mathscr{SG}(\mathcal{G}^{N_\ell+N_r})$, using α_ℓ and α_r.

First, we will refer to the first N_ℓ processes of a configuration as the "left" part and to the N_r others as the "right" part. Now, we explain how α is defined, the idea being that this strategy simulates α_ℓ on the "left" part of the configuration and α_r on the "right" part of the configuration. Also, α sometimes exchanges processes from both parts: for instance, a process in state l_n present in the "left" part and from which player 2 will choose to perform δ_r. Note that moving such a process is possible because the actions of player 2 are only internal actions (labeled with ε) and consequently they only change the states of a single node.

The global strategy α starts by playing as α_r on the processes in the "right" part, until the decision in α_r is to choose a process p_{id}^r in state l_n, meaning that the next move for player 2 will be to decide which transition to perform between δ_r and δ_ℓ. Note that, if this never happens, the play is clearly winning for player 1 since α_r is winning in $\mathscr{SG}(\mathcal{G}_r^{N_r})$. Just before choosing p_{id}^r, the simulation of α_r is suspended, and α changes mode to play as α_ℓ in the "left" part. Similarly as above, if α_ℓ never dictates to choose a process p_{id}^ℓ in state l_n, then α sticks to α_ℓ, and the play is winning.

Otherwise, player 1 has to choose one process (either p_{id}^r or p_{id}^ℓ) in state l_n. Assume it does so, in the "left" part, choosing p_{id}^ℓ. Assuming further that the next move of player 2 corresponds to choosing transition δ_ℓ, then α continues to simulate α_ℓ on the processes of the "left" part. Else, if player 2 chooses to play transition δ_r, the processes p_{id}^l and p_{id}^r are exchanged (remember that p_{id}^r and p_{id}^ℓ where both in state l_n just before the choice of player 2) and α changes mode and now simulates α_r on the right part. Note that now the right part is composed of the previous N_r processes, from which p_{id}^r was removed, and p_{id}^ℓ added.

If we consider now a strategy β for player 2, the play $\rho(\mathscr{SG}(\mathcal{G}^{N_\ell+N_r}), \alpha, \beta)$ can have the three following forms where we add a superscript (ℓ) or (r) to denote for each state whether the game is played as in $\mathscr{SG}(\mathcal{G}_\ell^{N_\ell})$ or in $\mathscr{SG}(\mathcal{G}_r^{N_r})$:

1. $\lambda_0'^{(r)} \dots \lambda_i^{(\ell)} \dots \lambda_j^{(r)} \lambda_{j+1}^{(r)} \lambda_{j+2}^{(r)} \dots$ in which after some point α plays only as α_r in $\mathscr{SG}(\mathcal{G}_r^{N_r})$;
2. $\lambda_0'^{(r)} \dots \lambda_i^{(\ell)} \dots \lambda_j^{(\ell)} \lambda_{j+1}^{(\ell)} \lambda_{j+2}^{(\ell)} \dots$ in which after some point α plays only as α_ℓ in $\mathscr{SG}(\mathcal{G}_\ell^{N_\ell})$;
3. $\lambda_0'^{(r)} \dots \lambda_i^{(\ell)} \lambda_{i+1}^{(\ell)} \lambda_{i+2}^{(\ell)} \dots \lambda_j^{(r)} \lambda_{j+1}^{(r)} \lambda_{j+2}^{(r)} \dots \lambda_k^{(\ell)} \lambda_{k+1}^{(\ell)} \lambda_{k+2}^{(\ell)} \dots$ in which, infinitely often, α plays alternatively as α_r in $\mathscr{SG}(\mathcal{G}_r^{N_r})$ and as α_ℓ in $\mathscr{SG}(\mathcal{G}_\ell^{N_\ell})$.

Notice that this is only possible since we assumed all plays to be infinite. Without this assumption, let us consider the right play to be finite. While α is playing as α_r in the right part, it will eventually reach a deadlock configuration; the strategy α will thus be forced to play again in the left part, but the configuration in the left part may not correspond to a configuration reachable with the strategy α_ℓ, since a process in state l_ℓ is missing.

From the fact that α_r and α_ℓ are winning strategies in their respective game, it is obvious that in the two first cases $\rho(\mathscr{SG}(\mathcal{G}^{N_\ell+N_r}), \alpha, \beta)$ is a winning play. For the third case, from how we build our strategies α, it is clear that the play obtained from $\rho(\mathscr{SG}(\mathcal{G}^{N_\ell+N_r}), \alpha, \beta)$, by considering only the left sub-configurations and removing the configurations with superscripts (r), corresponds to a play in $\mathscr{SG}(\mathcal{G}_\ell^{N_\ell})$ which respects a strategy profile (α_ℓ, β') for some strategy β' of player 2, and consequently the play is winning. Symmetrically, the play obtained by removing the left sub-configurations and the configurations with superscripts (ℓ) is winning in $\mathscr{SG}(\mathcal{G}_r^{N_r})$. Consequently the "composition" of these two plays is necessarily winning: for the safety objective, it never uses "unsafe" actions and for the parity objective, the maximal color seen infinitely often is either in the (ℓ) part or in the (r) part and in both cases it is even. This shows that $\rho(\mathscr{SG}(\mathcal{G}^{N_\ell+N_r}), \alpha, \beta)$ is a winning play for player 1. □

In order to complete the proof of Proposition 3.1, we still need to prove:

Lemma 3.2

$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S_{sb}^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$
$$\Longleftrightarrow$$
$$\forall \beta \in S_{sb}^{(2)}.\ \exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}, \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$

Proof Here again, only the right-to-left implication deserves a proof.

Without loss of generality, when the winning objective is $Win = Win_p$, we assume that safe $= \Delta$.

Assume that for every state based strategy β of player 2, there exist $N_\beta \in \mathbb{N}$ and $\alpha_\beta \in S^{(1)}$ such that $\rho(\mathscr{SG}(\mathcal{G}^{N_\beta}), \alpha_\beta, \beta) \in Win$. The intuition is to define a strategy α for player 1 that guesses which state based strategy player 2 is playing, and plays the adequate counter strategy α_β. When the guess is incorrect, and the error is detected, α starts again on an other part of the network and guesses an other state based strategy for player 2.

In order to define α, we let $S^{(2)}_{sb,ns}$ denote the set of *unsafe* state based strategies for player 2, for which there is no risk for player 1 to guess them, since they fire unsafe transitions. Formally, $S^{(2)}_{sb,ns}$ is the set of strategies β_b induced by the local behaviors b such that $\forall \mathsf{l} \in \mathrm{L}^{(2)}$ if there exists $\delta = (\mathsf{l}, \varepsilon, \mathsf{l}') \in \Delta$ with $\delta \notin$ safe then $b(\mathsf{l}) \notin$ safe. Hence $S^{(2)}_{sb,ns}$ is the set of strategies that always choose an unsafe transition if possible.

We define N as the sum of N_β for all local unsafe strategies $\beta \in S^{(2)}_{sb,ns}$, and for each strategy $\beta \in S^{(2)}_{sb,ns}$ we refer to the N_β corresponding processes as the sub-configuration β.

The strategy α for player 1 is defined as the strategy that starts playing as α_β on the sub-configuration β for some $\beta \in S^{(2)}_{sb,ns}$. When α detects a wrong guess (when the last move was $(\mathsf{l}, \varepsilon, \mathsf{l}')$ whereas β should have done $(\mathsf{l}, \varepsilon, \mathsf{l}'')$), α changes for an other sub-configuration β' in which he did not already play and switches to $\alpha_{\beta'}$.

Notice that when α detects that the current strategy is the wrong one, since the strategy currently played wins against some strategy of $S^{(2)}_{sb,ns}$, necessarily the transition $(\mathsf{l}, \varepsilon, \mathsf{l}')$ is safe. Also notice that before detecting that β is a wrong copy, α is playing a strategy winning for some β' hence never fires an unsafe transition.

Let $\beta \in S^{(2)}_{sb}$ be any state based strategy that player 2 plays against α. Let us consider the play $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta)$. It can be decomposed into $\rho_1.\rho_2$ where ρ_2 is the longest suffix of the play that remains in the same sub-configuration, say β' sub-configuration. Let ρ'_2 be defined as the projection of ρ_2 on the sub-configuration β'. Since α is playing as $\alpha_{\beta'}$ in this suffix, and since it never detects a wrong guess in this suffix, we know that $\rho'_2 = \rho(\mathscr{SG}(\mathcal{G}^{N_{\beta'}}), \alpha_{\beta'}, \beta) = \rho(\mathscr{SG}(\mathcal{G}^{N_{\beta'}}), \alpha_{\beta'}, \beta') \in Win$. Therefore, $\rho_2 \in Win$. Finally, because ρ_1 never uses unsafe transitions, we obtain $\rho_1.\rho_2 \in Win$. □

3.3.2 Solving the game against state based strategies

In this section, we explain how to decide whether there exists a network size such that there exists a strategy for player 1 which is winning against a fixed state based strategy for player 2. We consider a parity protocol $\mathcal{G} = (\mathrm{L}, \mathrm{L}^{(1)}, \mathrm{L}^{(2)}, \mathsf{l}_0, \Sigma, \Delta, \mathrm{col}, \mathrm{safe})$ and a local behavior b. From $\mathcal{G}$ and b, we build a parity protocol $\mathcal{G}' = (\mathrm{L}, \mathrm{L}, \emptyset, \mathsf{l}_0, \Sigma, \Delta', \mathrm{col}', \mathrm{safe}')$ by removing the choices of player 2 not corresponding to b and by merging states of player 1 and states of player 2; this protocol is formally defined as follows: $\Delta' \subseteq \Delta$ and $(\mathsf{l}, \alpha, \mathsf{l}') \in \Delta'$ if and only if $\mathsf{l} \in \mathrm{L}^{(1)}$ and $(\mathsf{l}, \alpha, \mathsf{l}') \in \Delta$, or, $\mathsf{l} \in \mathrm{L}^{(2)} \cap \mathit{ActStates}$ and $b(\mathsf{l}) = (\mathsf{l}, \alpha, \mathsf{l}')$, furthermore col' is the restriction of col to Δ' and $\mathrm{safe}' = \Delta' \cap \mathrm{safe}$.

Example 3.3 *This construction is illustrated in Figure 3.5, on the example given in Figure 3.3 with the local behavior that from state* l^2_1 *chooses the transition leading to* l_3 *and from state* l^2_2 *chooses the unsafe transition leading to* l_5.

The following lemma states the relation between $\mathcal{G}$ and $\mathcal{G}'$.

Lemma 3.3 *For* $Win \in \{Win_s, Win_p, Win_{sp}\}$*, there exists a path* ρ *in* $\mathscr{SG}(\mathcal{G}'^N)$*, for some network size* $N \in \mathbb{N}$*, such that* $\rho \in Win$ *if and only if* $\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}, \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta_b) \in Win$.

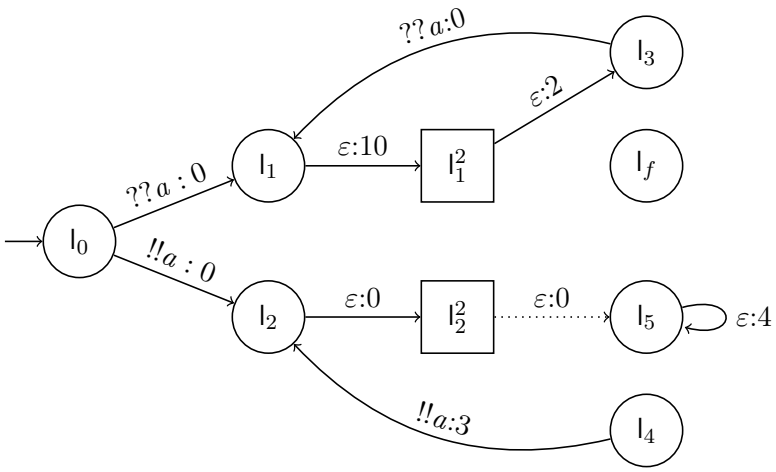


Figure 3.5: Example of the parity protocol obtained when considering a local behavior on the protocol illustrated in Figure 3.3.

Solving the game thus amounts to decide the following property: whether there exist $N \in \mathbb{N}$ and an infinite path in $Win_p \cap Win_s$ on $\mathscr{SG}(\mathcal{G}'^N)$. The remaining of this section is devoted to the resolution of this question: we provide an algorithm to decide whether there exist $N \in \mathbb{N}$ and a winning path in $\mathscr{SG}(\mathcal{G}'^N)$.

3.3.3 Existence of an infinite winning path

We now consider the safety/parity winning condition and show how to decide in polynomial time whether there exists an infinite path starting from some initial configuration and belonging to $Win_p \cap Win_s$. Notice that considering only safety/parity condition is not a restriction since: if the winning condition was initially only parity, one can consider the same parity protocol with all edges safe, and for the safety condition one can consider the same protocol with all parities set to 0.

The brief idea to decide in polynomial time whether there exists a winning infinite path starting from some initial configuration, is to first remove from the broadcast protocol $\mathcal{G}$ all unsafe edges. Then, we compute in polynomial time the set $Occur(\mathcal{G})$ of reachable control states, using the techniques from [DSTZ12, Algorithm 1]. Moreover, as recalled in Lemma 2.1 section 2, there must exist a reachable configuration $\lambda \in \Lambda_{\mathcal{G}}$ with as many processes as desired in each of the reachable control states. We finally look for the existence of a loop from λ that respects the parity condition. This last step relies on a counting abstraction that translates the game $\mathscr{SG}(\mathcal{G}^N)$ into a Vector Addition System with States (VASS, see Section 4.3), and reduces to the existence of a cycle in this VASS with positive effect on each component, which is decidable in polynomial time [KS88].

Let us now see in details how, given a local behavior for player 2, to decide whether there exists an infinite play satisfying the winning conditions.

Lemma 3.4 *Let* $\mathcal{G} = (\mathrm{L}, \mathrm{L}^{(1)}, \mathrm{L}^{(2)}, \mathsf{l}_0, \Sigma, \Delta, \mathrm{col}, \mathrm{safe})$ *be a parity protocol with all states belonging to player* 1, i.e. *such that* $\mathrm{L}^{(1)} = \mathrm{L}$ *and* $\mathrm{L}^{(2)} = \emptyset$. *Then, one can decide in polynomial time whether there exist* $N \in \mathbb{N}$ *and an infinite path in* $\mathscr{SG}(\mathcal{G}^N)$ *belonging to* $Win_p \cap Win_s$.

First remove all unsafe edges from $\mathcal{G}$ in order to obtain a broadcast protocol $\mathcal{G}' = (\mathrm{L}, \mathsf{l}_0, \Sigma, \Delta', \mathrm{col}')$, where col' is the restriction of col to Δ'. Note that we leave the safe set in $\mathcal{G}'$ unspecified, meaning that all transitions are considered as safe. The existence of an infinite path in $\mathscr{SG}(\mathcal{G}^N)$ that belongs to $Win_p \cap Win_s$ is equivalent to the existence of a cycle in $\mathscr{SG}(\mathcal{G}'^N)$ from a reachable configuration such that the maximum color along this cycle is even.

To decide the existence of such a cycle, we use a counting abstraction which consists in abstracting away the processes of each configuration and remembering only for each control state l of $\mathcal{G}'$ how many processes are in state l. We therefore translate the selective broadcast game networks obtained from $\mathcal{G}'$ into a Vector Addition System with States.

As introduced in Section 4.3, a VASS of dimension n with parities is defined as a tuple $\mathcal{V} = (\mathrm{S}, \mathrm{E}, \mathrm{col})$ where S is a finite set of control states, $\mathrm{E} \subseteq \mathrm{S} \times \mathbb{N}^n \times \mathrm{S}$ is the transition relation over n-tuples of integers and $\mathrm{col} : \mathrm{E} \to \mathbb{N}$ is the coloring function. To the VASS $\mathcal{V}$ is associated the transition system $T_{\mathcal{V}} = (\mathrm{S} \times \mathbb{N}^n, \hookrightarrow, \mathrm{col})$ where $\mathrm{S} \times \mathbb{N}^n$ is the set of configurations, $\hookrightarrow \subseteq (\mathrm{S} \times \mathbb{N}^n) \times (\mathrm{S} \times \mathbb{N}^n)$ is the transition relation and $\mathrm{col} : \hookrightarrow \to \mathbb{N}$ is the coloring function, defined as follows: $(\mathsf{s}, v) \hookrightarrow (\mathsf{s}', v')$ and $\mathrm{col}((\mathsf{s}, v) \hookrightarrow (\mathsf{s}', v')) = c$ if and only if there exists a transition $(\mathsf{s}, \mathbf{v}, \mathsf{s}') \in \mathrm{E}$ such that $v' = v + \mathbf{v}$ and $\mathrm{col}((\mathsf{s}, \mathbf{v}, \mathsf{s}')) = c$. Note that the vectors in configurations of $T_{\mathcal{V}}$ are tuples of naturals, so that a transition can only be fired if the resulting configuration ensures non-negativity of all components. An *infinite run* of $\mathcal{V}$ starting from a configuration (s_0, v_0) is an infinite sequence of the form $(\mathsf{s}_0, v_0) \hookrightarrow (\mathsf{s}_1, v_1) \hookrightarrow (\mathsf{s}_2, v_2) \dots$ and we say that it is *valid* with respect to col if it satisfies the parity condition, *i.e.* if $\max\{c \in \mathbb{N} \mid \forall i \geq 0\ \exists j \geq i \text{ s.t. } \mathrm{col}((\mathsf{s}_j, v_j) \hookrightarrow (\mathsf{s}_{j+1}, v_{j+1}))) = c\}$ is even.

We now explain how to build a VASS $\mathcal{V}_{\mathcal{G}'}$ with parities from the broadcast protocol $\mathcal{G}' = (\mathrm{L}, \mathsf{l}_0, \Sigma, \Delta', \mathrm{col}')$, such that there exist, in $\mathcal{V}_{\mathcal{G}'}$, a configuration (s_0, v_0) and a valid infinite run starting from this configuration if and only if there exist $N \in \mathbb{N}$ and an infinite path ρ in $\mathscr{SG}(\mathcal{G}'^N)$ satisfying $\rho \in Win_p$. Let us write $Occur(\mathcal{G}') = \{\mathsf{l}_1, \dots, \mathsf{l}_n\}$ for the set of reachable states in $\mathcal{G}'$. From $\mathcal{G}'$, we define a VASS $\mathcal{V}_{\mathcal{G}'} = (\mathrm{S}_{\mathcal{G}'}, \mathrm{E}_{\mathcal{G}'}, \mathrm{col}'')$ of dimension $2n$ as follows:

- $\mathrm{S}_{\mathcal{G}'} = \{\mathsf{s}_0\} \cup \{\mathsf{s}^1_\delta, \mathsf{s}^2_\delta \mid \delta \in \Delta' \text{ is of the form } (\mathsf{l}_i, !!a, \mathsf{l}_j) \text{ or } (\mathsf{l}_i, \varepsilon, \mathsf{l}_j)\}$;
- $\mathrm{E}_{\mathcal{G}'}$ is the smallest relation such that for every edge $\delta = (\mathsf{l}_i, \alpha, \mathsf{l}_j) \in \Delta'$:
  - if $\alpha = \varepsilon$ or $\alpha = !!m$, then $\{(\mathsf{s}_0, \mathbf{v}, \mathsf{s}^1_\delta), (\mathsf{s}^1_\delta, \mathbf{0}, \mathsf{s}^2_\delta), (\mathsf{s}^2_\delta, \mathbf{v}', \mathsf{s}_0)\} \subseteq \mathrm{E}_{\mathcal{G}''}$ with:
    - $\mathbf{v}[i] = -1$ and $\mathbf{v}[k] = 0$ for every $k \neq i$;
    - $\mathbf{v}'[j] = 1$ and $\mathbf{v}[k] = 0$ for every $k \neq j$;

    and $\mathrm{col}''(\mathsf{s}_0, \mathbf{v}, \mathsf{s}^1_\delta) = \mathrm{col}''(\mathsf{s}^1_\delta, \mathbf{0}, \mathsf{s}^2_\delta) = \mathrm{col}''(\mathsf{s}^2_\delta, \mathbf{v}', \mathsf{s}_0) = \mathrm{col}'(\delta)$;

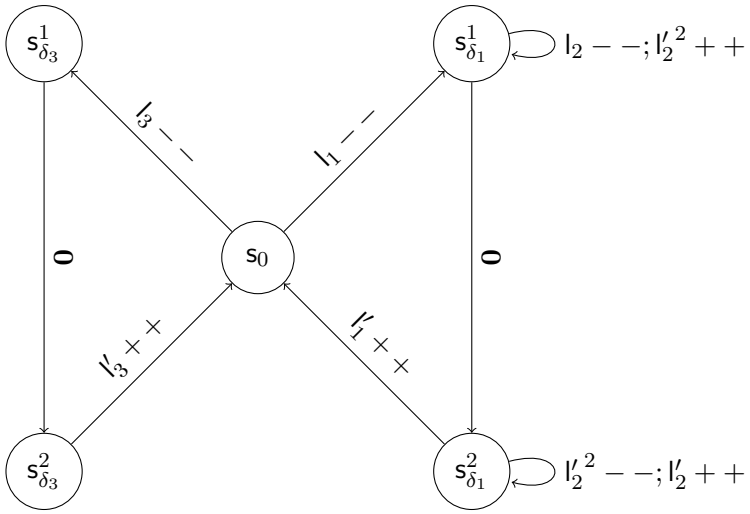


Figure 3.6: Example of a VASS obtained with the construction.

– if $\alpha = ??m$, then for each edge $\delta' = (\mathsf{l}_k, !!a, \mathsf{l}_\ell) \in \Delta'$, we have $\{(\mathsf{s}^1_{\delta'}, \mathbf{v}, \mathsf{s}^1_{\delta'}), (\mathsf{s}^2_{\delta'}, \mathbf{v}', \mathsf{s}^2_{\delta'})\} \subseteq \mathrm{E}_{\mathcal{G}'}$ with:
  * $\mathbf{v}[i] = -1$, $\mathbf{v}[n+j] = 1$ and $\mathbf{v}[k] = 0$ for every $k \neq i$, $k \neq n+j$;
  * $\mathbf{v}'[j] = 1$, $\mathbf{v}'[n+j] = -1$ and $\mathbf{v}[k] = 0$ for every $k \neq j$, $k \neq n+j$;

  and $\mathrm{col}''(\mathsf{s}^1_{\delta'}, \mathbf{v}, \mathsf{s}^1_{\delta'}) = \mathrm{col}''(\mathsf{s}^2_{\delta'}, \mathbf{v}', \mathsf{s}^2_{\delta'}) = 0$.

Example 3.4 *An example of this construction is given in Figure 3.6. This example illustrates the construction on a protocol composed of one broadcast transition* $\delta_1 = (\mathsf{l}_1, !!m, \mathsf{l}'_1)$*, one reception* $\delta_2 = (\mathsf{l}_2, ??m, \mathsf{l}'_2)$ *and an internal transition* $\delta_3 = (\mathsf{l}_3, \varepsilon, \mathsf{l}'_3)$*.*

For simplicity $\mathsf{l}++$ *(resp.* $\mathsf{l}--$*) represents that the dimension corresponding to* l *is incremented by one (resp. decremented). Also* l^2 *corresponds to the second dimension* $i+n$ *if dimension* i *corresponds to* l*.*

In this example, one cycle on the left part, going from s_0 *to* s_0 *through* $\mathsf{s}^1_{\delta_3}$ *and* $\mathsf{s}^2_{\delta_3}$*, has as effect to decrement by one the dimension corresponding to* l_3 *and increment by one the dimension corresponding to* l'_3*. This cycle represents the fact that one process took the transition* δ_3 *and thus moved from* l_3 *to* l'_3*.*

With the same idea, on the right part, going from s_0 *to* s_0 *through* $\mathsf{s}^1_{\delta_1}$ *and* $\mathsf{s}^2_{\delta_1}$ *represents the fact that one process took the transition* δ_1 *and thus moved from* l_1 *to* l'_1*. Moreover, since this transition is a broadcast of message* m*, for each reception of message* m *(here only* δ_2*) there is a self loop on* $\mathsf{s}^1_{\delta_1}$ *that decrements the origin of the reception and increments its destination. However, since we want to forbid the same process to receive multiple times the same message, we use an intermediary dimension (*${\mathsf{l}'_2}^2$ *in the example, and* $n+i$ *in the definition) that allows to ensure only one reception of a message by process.*

Intuitively, in a configuration of $\mathcal{V}_{\mathcal{G}'}$, each vector value represents the number of nodes in a state. To simulate a broadcast $\delta = (\mathsf{l}_k, !!a, \mathsf{l}_\ell)$, one first decreases by one the compo-

nent corresponding to l_k when taking the transition from s_0 to s^1_δ. Then, corresponding reception rules of the form $(\mathsf{l}_i, ??a, \mathsf{l}_j)$ are simulated by decrementing by one the component corresponding to l_i and incrementing the l_j-component; precisely, this is done using the $n+j$-th component. Last, when the simulation of receptions is over, one increments by one the component corresponding to l_ℓ, thanks to the transition from l^2_δ to l_0.

Note that two dimensions per state are needed, otherwise we could simulate a sequence of receptions by the same node. Moreover, the value of component $n+j$ (for $1 \leq j \leq n$) is not necessarily reset to 0 after the simulation of a communication. This is not a problem since it would just represent processes that are not present in any reception set.

The next lemma formalizes the properties of the above construction. Its proof is a direct consequence of the semantics of the game $\mathscr{SG}(\mathcal{G}'^N)$ and of the definition of $\mathcal{V}_{\mathcal{G}'}$.

Lemma 3.5 *Let λ be a configuration of $\mathscr{SG}(\mathcal{G}'^N)$ such that for all $p_{id} \in [1 \dots N]$, $\lambda[p_{id}] \in Occur(\mathcal{G}')$ and $\mathbf{v} \in \mathbb{N}^{2n}$ such that $\mathbf{v}[i] = |\{p_{id} \in [1 \dots N] \mid \lambda[p_{id}] = \mathsf{l}_i\}|$ for every $i \in \{1, \dots, n\}$ and $\mathbf{v}[i] = 0$ for every $i \in \{n+1, \dots, 2n\}$. Then, there exists an infinite path in $\mathscr{SG}(\mathcal{G}'^N)$ starting from λ and belonging to Win_p if and only if there exists an infinite run in $\mathcal{V}_{\mathcal{G}'}$ starting from $(\mathsf{s}_0, \mathbf{v})$ and valid with respect to* col''.

By definition of $\mathcal{V}_{\mathcal{G}'}$, there exists a vector $\mathbf{v} \in \mathbb{N}^{2n}$ with $\mathbf{v}[i] = 0$ for all $i \geq n+1$ such that there exists an infinite run in $\mathcal{V}_{\mathcal{G}''}$ starting from $(\mathsf{s}_0, \mathbf{v})$ and valid with respect to col'' if and only if there exists a cycle $(\mathsf{s}_0, \mathbf{v}_0, \mathsf{s}_1)(\mathsf{s}_1, \mathbf{v}_1, \mathsf{s}_2) \dots (\mathsf{s}_k, \mathbf{v}_k, \mathsf{s}_0)$ in $\mathcal{V}_{\mathcal{G}'}$ such that $\sum_{j=0}^{k} \mathbf{v}_j \geq \mathbf{0}$ and $\max\{\mathrm{col}''(\mathsf{s}_j, \mathbf{v}_j, \mathsf{s}_{j+1}) \mid j \in \{0, \dots, k\}\}$ is even.

As stated by the following lemma, checking the existence of such a cycle can be done in polynomial time, using a known algorithm to detect, in a graph labeled with vector of integers, a cycle whose accumulated sum is greater than $\mathbf{0}$ [KS88].

Lemma 3.6 *Given $\mathcal{V}_{\mathcal{G}'} = (\mathrm{S}_{\mathcal{G}'}, \mathrm{E}_{\mathcal{G}'}, \mathrm{col}'')$ the VASS associated with $\mathcal{G}'$, one can decide in polynomial time whether there exists a finite path $(\mathsf{s}_0, \mathbf{v}_0, \mathsf{s}_1) \dots (\mathsf{s}_k, \mathbf{v}_k, \mathsf{s}_{k+1})$ in $\mathcal{V}_{\mathcal{G}'}$ with $\mathsf{s}_{k+1} = \mathsf{s}_0$ and such that $\sum_{i=0}^{k} \mathbf{v}_i = \mathbf{0}$ and $\max\{\mathrm{col}''(\mathsf{s}_i, \mathbf{v}_i, \mathsf{s}_{i+1}) \mid i \in [0 \dots k]\}$ is even.*

Proof First note that, by construction of $\mathcal{V}_{\mathcal{G}'}$, the existence of a cycle of positive accumulated sum is equivalent to the existence of a cycle of accumulated sum equal to $\mathbf{0}$ and starting from s_0. Indeed, the sum of all components is constant and no null cycle can loop only on states of the form s^1_δ or s^2_δ since simple loops on these states decrease a vector component. Solving a system of linear inequalities, one can decide in polynomial time the existence of null cycles starting in s_0 [KS88]. The catch is that such a cycle may not satisfy the parity condition (*i.e.*, that the maximum color visited along the cycle is even).

For every even color c that appears on some transition of the VASS $\mathcal{V}_{\mathcal{G}'}$, we explain how to decide the existence of a non-negative cycle starting in s_0 and with maximum color c. First, we remove from $\mathcal{V}_{\mathcal{G}'}$ all transitions labeled with a color $c' > c$. Second, we

augment the VASS with a $2n+1$-th dimension, and modify the transitions as follows: for every edge $\delta = (\mathsf{l}_i, !!a, \mathsf{l}_j)$ or $\delta = (\mathsf{l}_i, \varepsilon, \mathsf{l}_j)$ in $\mathcal{G}'$, if $\mathrm{col}'(\delta) \neq c$, we transform the corresponding transition $(\mathsf{s}_0, \mathbf{v}, \mathsf{s}^1_\delta)$ by additionally decrementing the component $2n+1$ by 1; otherwise, if $\mathrm{col}'(\delta) = c$, we add a self loop on the corresponding state s^1_δ that increments the component $2n+1$ by 1. After these modifications, the only possibility for a null cycle is to take at least one transition colored by c.

Applying the polynomial time algorithm by Kosaraju and Sullivan [KS88] a linear number of times (for each even color c), we obtain a polynomial time decision procedure for our problem. □

If there exists a non-negative cycle from $(\mathsf{s}_0, \mathbf{v})$ satisfying the parity condition, then the same cycle can be fired infinitely often from any $(\mathsf{s}_0, \mathbf{v}')$ such that $\mathbf{v}'[i] \geq \mathbf{v}[i]$ for every $i \in \{1, \ldots, n\}$, and $\mathbf{v}'[i] = 0$ for every $i \in \{n+1, \ldots, 2n\}$.

Let us put all arguments together in order to complete the proof of Lemma 3.4. From Lemma 2.1, we can compute in polynomial time $Occur(\mathcal{G}')$, and know that for every $k \in \mathbb{N}$ there exists a reachable configuration for which each state of $Occur(\mathcal{G}')$ labels at least k vertices. Then, using Lemmas 3.5 and 3.6, we can decide in polynomial time whether from one of these configurations there exists an infinite path in $\mathscr{SG}(\mathcal{G}'^N)$ satisfying the parity condition. Since $\mathcal{G}'$ only has safe edges, we obtain the decidability in polynomial time of the existence of infinite path in $\mathscr{SG}(\mathcal{G}^N)$ that belongs to $Win_p \cap Win_s$.

3.3.4 Solving parity networks

Back to our parity networks, given a state based strategy $\beta_b \in S^{(2)}_{sb}$, by Lemma 3.3 and Lemma 3.4, we obtain a polynomial time algorithm to decide whether there exist a network size $N \in \mathbb{N}$ and a strategy $\alpha \in S^{(1)}$ for player 1 such that $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta_b) \in Win$ for $Win \in \{Win_s, Win_p, Win_{sp}\}$. Since the number of state based strategies is finite, this gives us a non deterministic algorithm to solve whether $\exists \beta \in S^{(2)}_{sb}.\ \forall N \in \mathbb{N}.\ \forall \alpha \in S^{(1)}, \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \notin Win$ with $Win \in \{Win_s, Win_p, Win_{sp}\}$. The algorithm consists in guessing a state based strategy β for player 2 and checking, thanks to Lemma 3.3 and Lemma 3.4, in polynomial time, that it is indeed a strategy such that $\forall N \in \mathbb{N}.\ \forall \alpha \in S^{(1)}, \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \notin Win$ with $Win \in \{Win_s, Win_p, Win_{sp}\}$.

Moreover, Proposition 3.1 tells us that to solve the game problems we need to check the absence of such strategy for player 2. Hence, we proved the main result of this section.

Theorem 3.1 *For safety, parity and safety-parity objectives, the game problem for parity protocol is decidable in co-NP.*

3.3.5 Restriction to urgent strategies

Another interesting result on selective game networks is that, player 1 does not loose power by making player 2 make its choices as soon as possible. This is intuitive since the sooner player 2 has to make its choice the less information he has and thus it is

more difficult for him to be harmful. This result will be useful in the next section to reduce the parameterized reachability problems to the game problem.

First, we define urgent strategies for player 1: $\alpha \in S^{(1)}$ is *urgent* if for every play $\rho\lambda$ with $\lambda \in \mathrm{Q}^N$ there exists $p_{id} \in [1 \dots N]$ such that $\lambda[p_{id}] \in \mathrm{L}^{(2)}$, then $\alpha(\rho\lambda) = (\lambda, p_{id})$. In words, the urgent strategies are those where player 1 activates the processes in states belonging to player 2 as soon as possible. The set of urgent strategies for player 1 is denoted $S_u^{(1)}$. Note that if α_u is urgent, and $\rho\lambda$ respects α_u, we have $|\{p_{id} \in [1 \dots N] \mid \lambda[p_{id}] \in \mathrm{L}^{(2)}\}| \leq 1$, that is, at most one process is in a state of player 2.

We prove that there is a winning strategy for player 1 if and only if there is an urgent one.

Lemma 3.7

$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$
$$\Longleftrightarrow$$
$$\exists N \in \mathbb{N}.\ \exists \alpha \in S_u^{(1)},\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$$

Proof Only the right-to-left implication deserves a proof, since $S_u^{(1)} \subseteq S^{(1)}$.

We assume that there exist a network size $N \in \mathbb{N}$ and a strategy for player 1 $\alpha \in S^{(1)}$ such that $\forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win$. Without loss of generality, we can assume that α is memoryless. Indeed, given the size of the network $N \in \mathbb{N}$, the game is finite, hence there is a winning strategy if and only if there is a memoryless winning strategy.

Intuitively, the urgent strategy α' plays similarly to α but delays the moment the processes enter in state of L_2 until α would have played them. Strategy α' is then urgent, but has memory.

More formally, on the one hand, in λ, if α would have moved a process to a state $\mathsf{l} \in \mathrm{L}^{(2)}$ then for the paths ρ that α' identifies to λ, α' remembers that this process should be in l. On the other hand, in λ, if α would have chosen a process in a state $\mathsf{l}' \in \mathrm{L}^{(2)}$ to play, then for the paths ρ that α' identifies to λ, α' first moves the process to l' and then chooses it immediately to play.

Let β be a strategy for player 2. With a play $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha', \beta)$, we can associate the play $\tilde{\rho}(\mathscr{SG}(\mathcal{G}^N), \alpha', \beta)$ of configurations as stored in the memory of α'. Notice that $\tilde{\rho}(\mathscr{SG}(\mathcal{G}^N), \alpha', \beta)$ corresponds to a play for α, and hence satisfies the winning condition Win. Moreover, the transitions visited along that play are the same as $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha', \beta)$. Hence, the urgent strategy α' built above is winning: for every strategy β for player 2, $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha', \beta) \in Win$. □

4 Solving probabilities with games

In this section, we solve the qualitative reachability problems for probabilistic reconfigurable broadcast networks. The most involved case is $REACH^{\exists}_{=1}(\mathscr{S})$ for which we reduce to games on parity protocols with a parity winning condition.

4.1 Decidability using monotonicity

The decidability and complexity of the first cases are established directly, without reducing to games on parity protocols. First, let us establish a monotonicity property: intuitively, with more processes, the probability to reach the target can only increase.

Lemma 4.1 $\forall \sigma,\ \forall N \in \mathbb{N},\ \forall N' \geq N,\ \exists \sigma',\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathtt{q}_f) \leq \mathbb{P}_{\sigma'}(\mathscr{S}(\mathcal{P}^{N'}) \models \Diamond \mathtt{q}_f)$.

Proof Intuitively, with N' processes σ' behaves as σ on the N first processes. The $N' - N$ other processes stay in the initial state. □

Using this monotonicity property, the problems are then reduced to qualitative reachability problems in the finite state MDP for the network with a single process, and thus belong to PTIME.

Theorem 4.1 *$REACH^{\forall}_{=0}(\mathscr{S})$, $REACH^{\forall}_{<1}(\mathscr{S})$, $REACH^{\forall}_{=1}(\mathscr{S})$ and $REACH^{\forall}_{>0}(\mathscr{S})$ are decidable in PTIME.*

Proof Due to Lemma 4.1, it is sufficient to consider the network of size 1. We then solve these decision problems on the corresponding finite-state MDP.

If $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) = 0$, then, trivially, the answer to $REACH^{\forall}_{=0}(\mathscr{S})$ is positive. Otherwise, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) > 0$, and by Lemma 4.1, we obtain that for all $N \geq 1$ there exists a strategy σ' with $\mathbb{P}_{\sigma'}(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathtt{q}_f) \geq \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) > 0$. Therefore, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathtt{q}_f) > 0$, and the answer to $REACH^{\forall}_{=0}(\mathscr{S})$ is negative. Exactly the same reasoning applies to $REACH^{\forall}_{<1}(\mathscr{S})$.

If $\forall \sigma,\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) = 1$, then, the answer to $REACH^{\forall}_{=1}(\mathscr{S})$ is positive. Otherwise, $\exists \sigma,\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) < 1$, and for $N \geq 1$, we consider the class of strategies that use only one process and always choose an empty communication set. For this class of strategies, the behavior of the used process is independent of the other processes, and hence simulates what happens for a single process. Therefore, $\exists \sigma',\ \mathbb{P}_{\sigma'}(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathtt{q}_f) = \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^1) \models \Diamond \mathtt{q}_f) < 1$, and the answer to $REACH^{\forall}_{=1}(\mathscr{S})$ is negative. Exactly the same reasoning applies to $REACH^{\forall}_{>0}(\mathscr{S})$. □

Then, notice that $REACH^{\exists}_{>0}(\mathscr{S})$ is inter-reducible to the reachability problem in *non-probabilistic* reconfigurable broadcast networks, known to be P-complete [DSTZ12].

Theorem 4.2 *$REACH^{\exists}_{>0}(\mathscr{S})$ is in PTIME.*

4.2 Decidability and complexity of $REACH^{\exists}_{=1}(\mathscr{S})$

Let us now discuss the most involved case, $REACH^{\exists}_{=1}(\mathscr{S})$, and show how to reduce it to the game problem for parity protocols with a parity winning condition.

The reduction to the game problem is inspired from the same kind of reduction for finite state systems. Although, given a network size the network is finite, one cannot use directly the reduction used in finite state system to decide the existence

of a network size meeting the desired property. To tackle this problem, we define a reduction at the protocol level, and use the game problem defined in Section 3.2 to solve the parameterized reachability problem.

From $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(p)}, \mathrm{Q}^{(n)}, \mathsf{q}_0, \Sigma, \Delta)$ a probabilistic protocol and $\mathsf{q}_f \in \mathrm{Q}$ a control state, we derive the parity protocol $\mathcal{G} = (\mathrm{L}_\mathcal{G}, \mathrm{L}^{(1)}_\mathcal{G}, \mathrm{L}^{(2)}_\mathcal{G}, \mathsf{l}_{0\mathcal{G}}, \Sigma_\mathcal{G}, \Delta_\mathcal{G}, \mathrm{col}, \mathrm{safe})$ as follows:

- $\mathrm{L}_\mathcal{G} = \mathrm{L}^{(1)}_\mathcal{G} \cup \mathrm{L}^{(2)}_\mathcal{G}$,
- $\mathrm{L}^{(1)}_\mathcal{G} = \mathrm{Q}^{(n)} \cup \big(\mathrm{Q}^{(p)} \times \{1\}\big)$,
- $\mathrm{L}^{(2)}_\mathcal{G} = \mathrm{Q}^{(p)} \times \{2\}$, and
- $\mathsf{l}_{0\mathcal{G}} = \mathsf{q}_0$;
- $\Sigma_\mathcal{G} = \Sigma$;
- $\Delta_\mathcal{G} = \big(\big(\mathrm{Q}^{(n)} \times \{!!m, ??m \mid m \in \Sigma\} \times \mathrm{Q}^{(n)}\big) \cap \Delta\big) \cup \{(\mathsf{q}_f, \varepsilon, \mathsf{q}_f)\}$
  $\cup \{(\mathsf{q}, \varepsilon, (\mathsf{q}', 2)), ((\mathsf{q}', i), \varepsilon, \mathsf{q}'), ((\mathsf{q}, 2), \epsilon, (\mathsf{q}, 1)) \mid (\mathsf{q}, \epsilon, \mathsf{q}') \in \Delta, i \in \{1, 2\}\}$
  $\cup \{((\mathsf{q}, i), \varepsilon, \mathsf{q}') \mid (\mathsf{q}, d) \in \Delta, d(\mathsf{q}') > 0, i \in \{1, 2\}\}$;
- $\mathrm{col}((\mathsf{q}_f, \epsilon, \mathsf{q}_f)) = 2$, $\mathrm{col}(((\mathsf{l}, 2), \epsilon, q') = 2$ and otherwise $\mathrm{col}(\delta) = 1$.

Intuitively, all random choices corresponding to internal actions in $\mathcal{P}$ are replaced in $\mathcal{G}$ with choices for player 2, where either he decides the outcome of the probabilistic choice, or he lets player 1 choose. Only transitions where player 2 makes the decision corresponding to a probabilistic choice and the self loop on the state l_f have parity 2. Figure 4.7 illustrates this reduction on the example probabilistic protocol from Figure 2.1, page 97.

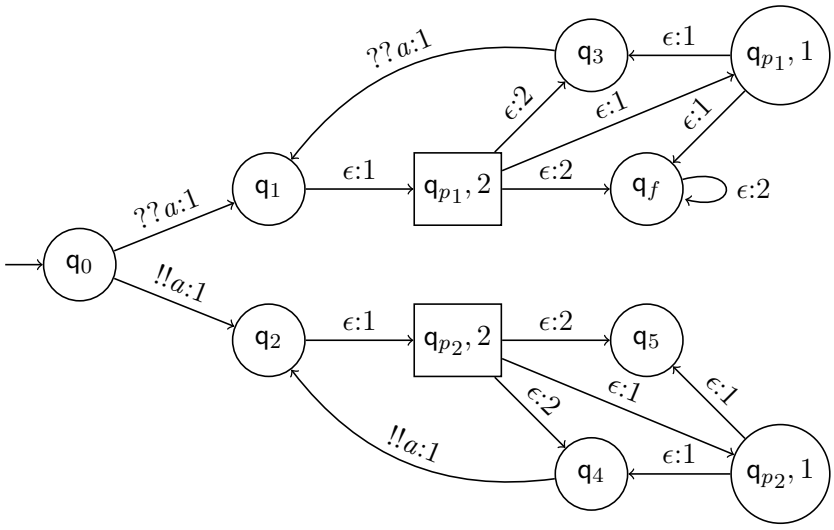


Figure 4.7: Parity protocol for the probabilistic protocol from Figure 2.1.

This construction ensures:

Proposition 4.1 $\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_p$ *if and only if* $\exists N \in \mathbb{N}.\ \exists \sigma \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1$.

Proof The easiest direction is from left-to-right. Assuming that some strategy σ ensures to reach q_f with probability 1, one builds a winning strategy α for the parity objective as follows: when player 2 makes a decision corresponding to a probabilistic choice in $\mathcal{P}$, the strategy chooses to play this probabilistic transition. Now, when player 1 needs to make a decision in some configuration λ where there is a process p_{id} in state $(\mathsf{q}, 1)$, the strategy is to play along a shortest path respecting σ from γ to a configuration containing q_f, where γ is defined as λ but the state of p_{id} is q. Assuming that σ reaches q_f with probability 1, such a path must exist for every reachable configuration in the game. This definition of α ensures to eventually reach q_f under the assumption that player 2, from some point on, always lets player 1 decide in configurations corresponding to probabilistic states of $\mathcal{P}$.

More formally, let us prove the following implication:

$$\begin{array}{c} \exists N \in \mathbb{N}.\ \exists \sigma \in S, \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1 \\ \Downarrow \\ \exists N \in \mathbb{N}.\ \exists \alpha \in S_u^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_p. \end{array}$$

First, remark that we can embed configurations of $\mathscr{S}(\mathcal{P}^N)$ in configurations of $\mathscr{SG}(\mathcal{G}^N)$ by identifying $\mathsf{q} \in \mathrm{Q}^{(p)}$ to $(\mathsf{q}, 2) \in \mathrm{L}_{\mathcal{G}}^{(2)}$, so that $\mathrm{Q}_{\mathcal{P}} \subseteq \mathrm{L}_{\mathcal{G}}$.

Let N and σ be such that $\mathbb{P}(\mathscr{S}(\mathcal{P}^N), \sigma, \Diamond \mathsf{q}_f) = 1$. We can assume, without loss of generality that σ is memoryless, because, given the initial configuration N, the number of configurations is finite.

Now we explain how to define α from σ. For a configuration λ:

- If there exists $p_{id} \in [1 \dots N]$ with $\lambda[p_{id}] = \mathsf{q}_f$, then α plays forever the two actions $\lambda \to (\lambda, p_{id}) \to \lambda$ using the loop on q_f, which has parity 2.
- Otherwise, if there exists $p_{id} \in [1 \dots N]$ with $\lambda[p_{id}] = (\mathsf{q}, 2)$, for some state $\mathsf{q} \in \mathrm{Q}^{(p)}$, then α moves to the configuration where this process is selected, *i.e.* (λ, p_{id}).
- Otherwise, if there are no process $p_{id} \in [1 \dots N]$ in any state $(\mathsf{q}, 1)$ of $\mathrm{Q}^{(p)} \times \{1\}$, λ can be identified to a configuration γ, and α in λ mimics σ in γ.
- Otherwise, there exists $p_{id} \in [1 \dots N]$ with $\lambda[p_{id}] = (\mathsf{q}, 1)$ for some state $\mathsf{q} \in \mathrm{L}$. In this case, we let γ_1 be the configuration corresponding to λ except that p_{id} is in state q. We then pick a shortest path $\gamma_1 \gamma_2 \dots \gamma_n$ that respects σ and reaches q_f, and define $\alpha(\lambda)$ as the configuration corresponding to γ_2.

Strategy α is urgent by definition, and we show now that it is winning. Let β be any strategy for player 2, and consider the play $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta)$. There are two possibilities: either q_f is reached, and then α ensures to loop on q_f with parity 2 so that $\rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_p$; or q_f is not reached. By definition of α, notice that every configuration that can be reached under α and that has no process labeled with a state

$(\mathsf{q},1)$ (for $\mathsf{q} \in \mathrm{Q}^{(p)}$) corresponds to a configuration reachable under σ. Moreover, since $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1$, from every configuration reachable under σ there exists a path to q_f. Second, notice that if, at some point, β always chooses to move processes to states of the form $(\mathsf{q},1)$, then α would ensure to reach q_f by following a shortest path. Hence, infinitely often β chooses not to move the process to $(\mathsf{q},1)$ and this implies that the play has parity 2. Therefore, $\rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_p$.

Let us now briefly explain how the right-to-left implication works. Notice that if player 2 always chooses transitions with parity 1 (thus letting player 1 decide the outcome of probabilistic choices), the only way for player 1 to win is to reach q_f, and from there to use the self loop to ensure the parity condition. As a consequence, from any reachable configuration, the target state q_f must be reachable.
From a winning strategy α, we define a strategy σ that mimics the choices of α on several copies of the network. The difficulty comes from the transformation of choices of player 1 in states of the form $(\mathsf{q},1) \in \mathrm{Q}^{(p)} \times \{1\}$ into probabilistic choices. Indeed, the outcome of these random choices cannot surely match the decision of player 1. The idea is the following: when a probabilistic choice in $\mathcal{P}$ does not agree with the decision of player 1 in $\mathcal{G}$, this "wrong choice" is attributed to player 2. The multiple copies thus account for memories of the "wrong choices", and a process performing such a choice is moved to a copy where the choice was made by player 2. With probability 1, a "good choice" is eventually made, and the 1-1/2 player game can continue in the original copy of the network. Therefore, the play will almost-surely end in a given copy, where player 1 always decides, and thus q_f is reached.

Let us now prove the other direction:

$$\begin{array}{c} \exists N \in \mathbb{N}.\ \exists \alpha \in S_u^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_p \\ \Downarrow \\ \exists N \in \mathbb{N}.\ \exists \sigma \in S, \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N), \Diamond \mathsf{q}_f) = 1\ . \end{array}$$

Let $N \in \mathbb{N}$ be a network size and $\alpha \in S_u^{(1)}$ be an urgent strategy for player 1 such that $\forall \beta \in S^{(2)},\ \rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_p$.

We define N' as $n \cdot N$, where n is the number of words on the alphabet $\mathrm{Q}^{(p)}$, in which each letter appears at most once. We see the network as partitioned, and for each w with $w = w_1 \cdots w_k$ such that for all $i, j \in \{1 \dots k\}$, we have $w_i, w_j \in \mathrm{Q}^{(p)}$ and if $i \neq j$ then $w_i \neq w_j$. We refer to N different processes as the sub-network w. We use the prefix relation to compare two such words, *i.e.* w is smaller than w', written $w \preceq w'$ if w is a prefix of w'. Last, for a state $w \in \mathrm{Q}^{(p)}$, we will say that q belongs to w, written $\mathsf{q} \in w$ if there exists $i \in \{1 \dots k\}$ such that $w_i = \mathsf{q}$.

We now explain how to define the strategy σ that mimics α in every sub-network w and that starts in the sub-network ε. Defining a strategy for the probabilistic network from a strategy for the game network is not straightforward. Indeed in $\mathcal{G}$, every state $\mathsf{q} \in \mathrm{Q}^{(p)}$ is duplicated in $\mathcal{P}$ into $(\mathsf{q},1)$ and $(\mathsf{q},2)$. Hence, we cannot establish a direct correspondence between the configurations of $\mathscr{S}(\mathcal{P}^N)$ and $\mathscr{S}\mathscr{G}(\mathcal{G}^N)$. Instead, we use the memory of σ to make this correspondence feasible. The idea is that σ plays in graph w as if for every $\mathsf{q} \notin w$ player 2 always chooses in the future to move to $(\mathsf{q},1)$, and for $\mathsf{q} \in w$ as if player 2 always chooses to not move to $(\mathsf{q},1)$.

Let us consider a fixed probabilistic state $\mathsf{q} \in \mathrm{Q}^{(p)}$ such that $(\mathsf{q}, \mu) \in \Delta$. We can assume, without loss of generality, that there are only two successors to q, the left one q_l and the right one q_r, and hence that $\mu(\mathsf{q}_r) + \mu(\mathsf{q}_l) = 1$. The built strategy σ has memory. We will explain how to update the memory states. Assume that w is the graph in which σ is currently playing, and that its memory state is m_σ. Two cases arise:

- Assume first that $\mathsf{q} \notin w$. The normal behavior of the strategy σ is to mimic the choice of α on the memory state m_σ. If at some point, α moves a process to the state $(\mathsf{q}, 2)$, then σ moves the corresponding process p_{id}, to state q. The subsequent probabilistic choice then leads to either q_l or q_r. Assume, for example, that it is q_r. In this case, we store in the memory state m_σ that p_{id} should be in state $(\mathsf{q}, 1)$ and the computation continues.

  After some time, α may decide to move this process out of $(\mathsf{q}, 1)$. Two cases are possible:

  - The easy case is when the strategy dictates to move to q_r. Since p_{id} is already in q_r, nothing needs to be done, and the computation resumes.
  - Otherwise, the process should have been moved to q_l. This somehow contradicts the result of the probabilistic choice that already happened in the probabilistic protocol. In this case, m_σ remembers that the graph w needs a process in state q_l. Also, the strategy σ changes its working graph to the graph w', where w' is a shortest word such that $w.\mathsf{q} \preceq w'$ and such that the graph w' does not already need a process in any state.

- Assume now that $\mathsf{q} \in w$. As in the other case, σ plays as α would have played on the state m_σ. If at some point, α moves a process to the state $(\mathsf{q}, 2)$, then σ moves the corresponding process p_{id} to state q. The subsequent probabilistic choice then leads to either q_l or q_r. Assume for example that it is q_r.

  If there exists $w' \preceq w$ such that w' needs a process in state q_r (remark that if such a w' exists, it is unique), then σ changes its working graph to w'. Also, σ switches the process labeled by q_r in w and the one "in need" in w'. Doing so, the configuration in w' corresponds exactly to a configuration reachable by α, and the configuration in w corresponds to a configuration where player 2 has chosen q_l instead of q_r.

  If there is no such $w' \preceq w$ and if w' needs a process in state q_r (*e.g.* they may all need a process in state q_l), then σ continues to mimic α in w assuming player 2 chooses to move directly to q_r.

First, notice that in every graph w, σ plays according to α. Second, notice that if σ is playing in graph w then, for every $\mathsf{q} \in w$, there must exist $w' \preceq w$ such that the graph w' needs either q_r or q_l. Last, notice that for every play, there exists a smallest graph w such that the play visits infinitely often w. Indeed, from w, to reach another graph w' with neither $w \preceq w'$ nor $w' \preceq w$, the play necessarily visits a graph w'' such that $w'' \preceq w$ and $w'' \preceq w'$.

We now show that σ, as defined above, is winning, that is: $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1$. Towards a contradiction, suppose that there exists a set A of plays respecting σ, such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A) > 0$ and $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A \cap \Diamond \mathsf{q}_f) = 0$. Since there are only finitely many graphs, we can assume without loss of generality that all plays of A share the same smallest graph w visited infinitely often.

For $\mathsf{q} \in w$, let $A'_\mathsf{q} \subseteq A$ be the subset of plays in which state q is visited infinitely often. Every play of A'_q is composed of a finite prefix, followed by infinitely many probabilistic choices in q that always have the same outcome. Indeed, otherwise σ would have moved to a prefix of w that needs q_r or q_l. Therefore, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A'_\mathsf{q}) = 0$, and letting $A' = \cup_{\mathsf{q} \in w} A'_\mathsf{q}$, we conclude $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A') = 0$.

Consider now a play ρ in $A \setminus A'$. After a finite prefix, ρ never visits any state of w. Hence, in the graph w, the strategy σ plays as α when player 2 always moves to the states $\mathrm{Q}^{(p)} \times \{1\}$. For such choices of player 2, the only way to win is to reach q_f, and then to achieve the parity objective by looping on q_f. Since α is winning, we deduce that ρ eventually reaches q_f. Hence we have: $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models (A \setminus A') \cap \Diamond \mathsf{q}_f) = \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A \setminus A')$, and since $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A) = \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models (A \setminus A')) + \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A')$ we obtain $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A \cap \Diamond \mathsf{q}_f) > 0$. This contradicts the definition of A. Therefore, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1$. □

We now provide the precise complexity of parameterized almost sure reachability in probabilistic selective networks.

Theorem 4.3 *$REACH^{\exists}_{=1}(\mathscr{S})$ is co-*NP*-complete.*

Proof The co-NP membership is a consequence of Proposition 4.1 and Theorem 3.1. Indeed, Proposition 4.1 allows us to reduce the problem $REACH^{\exists}_{=1}(\mathscr{S})$ to the game problem that is shown to be in co-NP in Theorem 3.1. We now establish the matching lower-bound by reducing the unsatisfiability problem of a formula in conjunctive normal form, to $REACH^{\exists}_{=1}(\mathscr{S})$. From φ, a formula in conjunctive normal form, we define a probabilistic protocol $\mathcal{P}_\varphi$ and a control state q_f such that φ is unsatisfiable if and only if there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N_\varphi) \models \Diamond \mathsf{q}_f) = 1$.

We provide here the construction on an example in Figure 4.8. For simplicity, the initial state q_0 of the probabilistic protocol is duplicated in the picture, and the internal self loops on the states are not represented. The idea is to generate a random assignment v of the variables (using the gadgets represented on the bottom of the figure). Assuming φ unsatisfiable, v will necessarily violate a clause of φ. Choosing this clause in the above part of the protocol then allows to reach state r_1, and from there to reach q_f with probability half. Iterating this process, the target is almost-surely reached. The converse implication relies on the fact that if φ is satisfiable, there is a positive probability to generate a valuation satisfying it, and then not to be able to reach r_1, a necessary condition to reach q_f. Therefore, the maximum probability to reach the target is smaller than 1 in this case.

We now provide the general definition of the reduction. Let V be a set of variables, and $\varphi = \bigwedge_{0 \leq i \leq l} x_{i,1} \vee x_{i,2} \vee x_{i,3}$ a formula in conjunctive normal form, where each literal

$x_{i,j}$ is either a variable $x \in V$ or its negation $\overline{x}$. From φ, we define the probabilistic protocol $\mathcal{P}_\varphi = (\mathrm{Q}, \mathrm{Q}^{(n)}, \mathrm{Q}^{(p)}, \mathsf{q}_0, \Sigma, \Delta)$, where:

- $\mathrm{Q}^{(n)} = \{x_{i,j} \mid 0 \leq i \leq l,\ 1 \leq j \leq 3\} \cup \{\mathsf{q}_0, r_1, \mathsf{q}_f\} \cup \{x_1, \overline{x_1}, x_2, \overline{x_2} \mid x \in V\}$
- $\mathrm{Q}^{(p)} = V \cup \{r_P\}$;
- $\Sigma = \{x, \overline{x} \mid x \in V\} \cup \{ok\}$;
- $\Delta = \{(x_{i,j}, ??\overline{x}_{i,j+1}, x_{i,j+1}) \mid 0 \leq i \leq l,\ 1 \leq j \leq 3\}$
  $\cup\, \{(\mathsf{q}_0, \varepsilon, x_{i,0}), (x_{i,3}, !!ok, r_1) \mid 0 \leq i \leq l\}$
  $\cup\, \{(\mathsf{q}_0, \varepsilon, x), (x_1, !!x, x_2), (\overline{x_1}, !!\overline{x}, \overline{x_2}), (x_2, ??ok, \mathsf{q}_0), (\overline{x_2}, ??ok, \mathsf{q}_0) \mid x \in V\}$
  $\cup\, \{(r_1, \varepsilon, r_P)\} \cup \{(\mathsf{q}, \varepsilon, \mathsf{q}) \mid \mathsf{q} \in \mathrm{Q}^{(n)}\}$;
- for every $x \in V$, with $(x, d) \in \Delta$ $d(x_1) = d(\overline{x_1}) = \frac{1}{2}$,
  and with $(r_P, d') \in \Delta$ $d'(\mathsf{q}_f) = d'(\mathsf{q}_0) = \frac{1}{2}$.

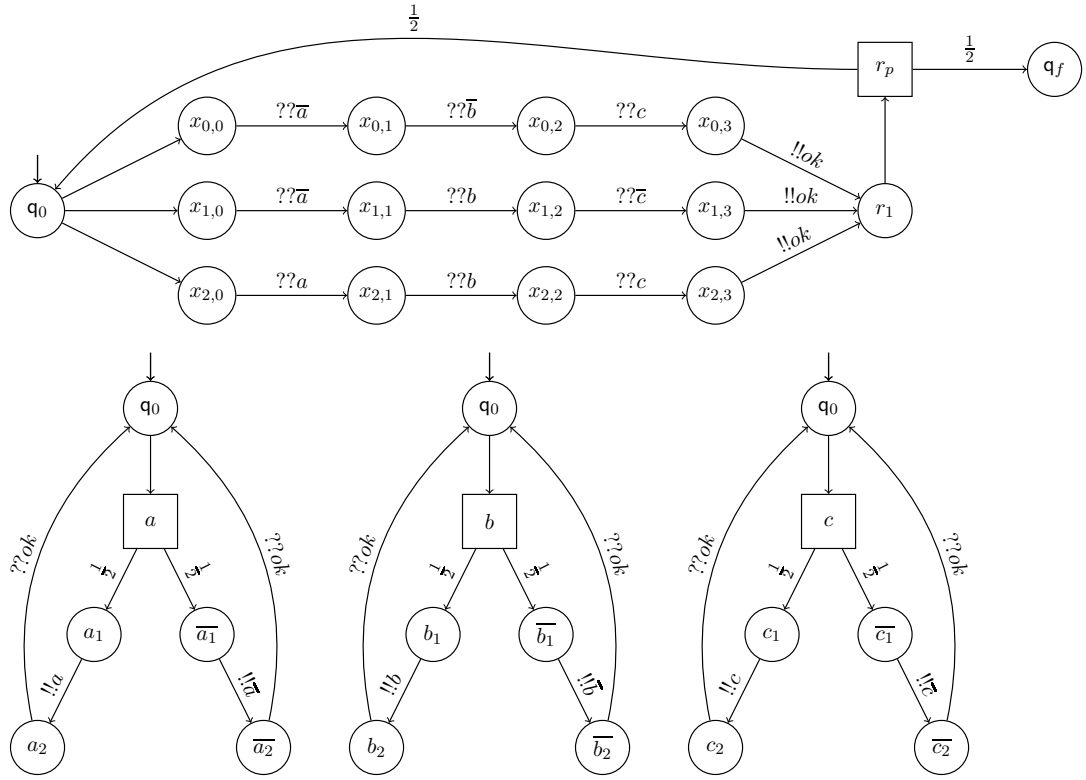


Figure 4.8: Probabilistic protocol for the formula $\varphi = (a \vee b \vee \overline{c}) \wedge (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee \overline{c})$.

Let us now prove that φ is unsatisfiable if and only if there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}_\varphi^N) \models \Diamond \mathsf{q}_f) = 1$.

Assume first that φ is unsatisfiable. Without loss of generality, we assume that each clause in φ does not contain two literals with the same variable ($x \vee \overline{x}$ can be replaced with tt, and $x \vee x$ with x). We set $N = |V| + 1$, and we define a strategy σ for $\mathscr{S}(\mathcal{P}^N)$ as follows. From the initial configuration, and for each variable $x \in V$, σ decides to perform the transition $(\mathsf{q}_0, \varepsilon, x)$ followed by the corresponding probabilistic transition, thus moves one process to either state x or state $\overline{x}$. These processes are called *variable processes* and at this step, their position defines a valuation $v : V \to \{0, 1\}$. Alternatively, v can be seen as a set of variables or negated variables that fulfill the valuation. Since φ is unsatisfiable, there exists a clause i such that for every literal $x_{i,j}$, $x_{i,j} \notin v$. At this stage, σ decides to move the remaining process, called the *formula process*, along the transition $(\mathsf{q}_0, \varepsilon, x_{i,0})$ corresponding to choosing the clause which is violated by v. Next, the variable processes in state $\overline{x_{i,1}}, \overline{x_{i,2}}$ and $\overline{x_{i,3}}$ broadcast in turn to the formula process, so that the latter moves to state $x_{i,3}$. The other variable processes broadcast to no one. Then, the formula process broadcasts to all variable processes the *ok*-message: the $N - 1$ variable processes are back to their initial state q_0. During this broadcast, the formula process reaches state r_P, where a probabilistic choice happens. With probability half, the state q_f is reached, and with the remaining probability the formula process is back in the initial state q_0, so that all processes are back in the initial configuration N. In this first round, σ thus ensures to reach q_f with probability $\frac{1}{2}$. Iterating this over and over, σ ensures to reach q_f almost-surely: $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 1$.

Assume now that φ is satisfiable, and let $v : V \to \{0, 1\}$ be a satisfying assignment of the variables. We fix σ an arbitrary strategy and N an arbitrary network size. A probabilistic choice from state x is said *consistent* with v if it sets x to its truth value in v. We aim at showing that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f)$ is bounded away from 1 by some factor independent of σ. We partition the set of paths E into incompatible events:

$$E = A_0 \sqcup A_1 \sqcup \cdots \sqcup A_N \sqcup A_{N+1} \sqcup A'_1 \sqcup \cdots \sqcup A'_N, \text{ where}$$

- for $0 \leq k \leq N$, A_k is the set of paths that contain exactly k probabilistic choices, and all of them are consistent with v;
- A_{N+1} is the set of paths that contain at least $N + 1$ probabilistic choices, and the first $N + 1$ are consistent with v;
- for $1 \leq k \leq N$, A'_k is the set of paths that contain at least k probabilistic choices, the first $k - 1$ are consistent with v and the k-th is not.

Clearly enough, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A'_k) \leq \frac{1}{2^k}$, for every $1 \leq k \leq N$ since for each variable, there is probability $\frac{1}{2}$ for the probabilistic choice to be consistent with v. Therefore, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \sqcup_{k=1}^N A'_k) \leq 1 - \frac{1}{2^N}$. Observe also that A_{N+1} is empty. Indeed, if the first probabilistic choices are consistent with v, then the formula processes, if any, stay stuck in the initial state q_0 and the variable processes cannot receive an *ok* message to get back to q_0. Therefore, starting with N processes, only N consecutive probabilistic choices consistent with v are possible, and A_{N+1} is empty. We thus derive a bound

for the remaining paths in the partition: $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \sqcup_{k=0}^{N} A_k) \geq \frac{1}{2^N}$. It remains to prove that all paths in the set A_k end in a deadlock before reaching q_f. Indeed, if the k first probabilistic choices for variable processes are consistent with v, then the formula processes cannot progress further than states $x_{i,2}$'s. As a consequence, the formula processes cannot move back to q_0, and they get stuck. Therefore, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \neg\Diamond\mathsf{q}_f) \geq \frac{1}{2^N}$, or equivalently, $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond\mathsf{q}_f) \leq 1 - \frac{1}{2^N}$. Because this holds for every strategy, we finally obtain $\forall\sigma\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond\mathsf{q}_f) < 1$. □

4.3 Decidability and complexity of $REACH^{\exists}_{=0}(\mathscr{S})$

Similarly to $REACH^{\exists}_{=1}(\mathscr{S})$, in order to solve $REACH^{\exists}_{=0}(\mathscr{S})$, we reduce it to the game problem for parity protocols. However, since we want here to avoid almost surely the target state we use a safety winning condition instead of a parity winning condition.

From $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(n)}, \mathrm{Q}^{(p)}, \mathsf{q}_0, \Sigma, \Delta)$ a probabilistic protocol and $\mathsf{q}_f \in \mathrm{Q}$ a control state, we derive the parity protocol $\mathcal{G} = (\mathrm{L}_\mathcal{G}, \mathrm{L}^{(1)}_\mathcal{G}, \mathrm{L}^{(2)}_\mathcal{G}, \mathsf{l}_{0\mathcal{G}}, \Sigma_\mathcal{G}, \Delta_\mathcal{G}, \mathrm{col}, \mathrm{safe})$ as follows:

- $\mathrm{L}_\mathcal{G} = \mathrm{L}$, $\mathrm{L}^{(1)}_\mathcal{G} = \mathrm{L}^{(1)}$, $\mathrm{L}^{(2)}_\mathcal{G} = \mathrm{Q}^{(p)}$, and $\mathsf{l}_{0\mathcal{G}} = \mathsf{q}_0$;
- $\Sigma_\mathcal{G} = \Sigma$;
- $\Delta_\mathcal{G} = \Delta \cup \{(\mathsf{q}, \varepsilon, \mathsf{q}') \mid (\mathsf{q}, d) \in \Delta,\ d(\mathsf{q}') > 0\}$;
- $\mathrm{safe} = \Delta \setminus \{\delta \mid \delta \in \mathrm{L} \times \big(\{!!m, ??m \mid m \in \Sigma\} \cup \{\varepsilon\}\big) \times \{\mathsf{q}_f\}\}$.

Notice that we will only need to consider safety objective, hence the coloring function col does not need to be defined. Intuitively, all random choices in $\mathcal{P}$ are replaced in $\mathcal{G}$ with choices for player 2. Figure 4.9 illustrates this reduction on the probabilistic protocol example from Figure 2.1. As usual, the dotted arrow represents the unsafe edge.

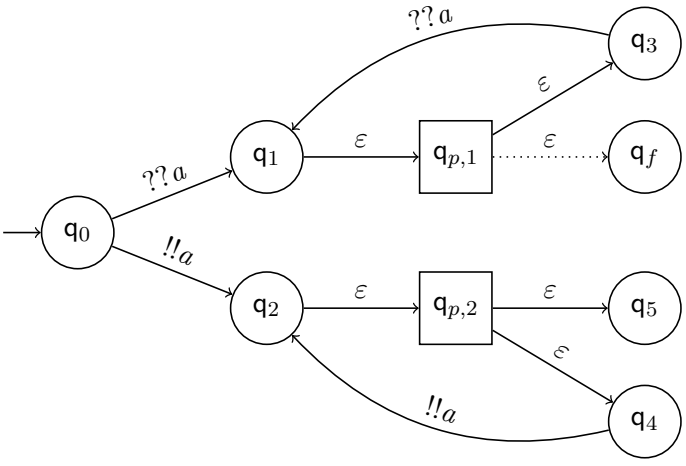


Figure 4.9: Parity protocol for the probabilistic protocol from Figure 2.1.

This construction gives us a reduction of $REACH^{\exists}_{=0}(\mathscr{S})$ to the game problem for parity protocols. More formally, this construction ensures the following equivalence:

Proposition 4.2 $\exists N \in \mathbb{N}\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_s$ *if and only if* $\exists N \in \mathbb{N}.\ \exists \sigma\ \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N), \Diamond \mathsf{q}_f) = 0$.

Proof Transitions with target q_f are the only ones that do not belong to the safe set safe. Hence, a winning strategy in $\mathscr{SG}(\mathcal{G}^N)$ for the safety objective must ensure that q_f is avoided. Similarly, in order to reach q_f with probability 0 in $\mathscr{S}(\mathcal{P}^N)$, a strategy also has to ensure that q_f is avoided. Since the parity protocol $\mathcal{G}_\mathcal{P}$ and the probabilistic protocol $\mathcal{P}$ have the same set of states, there is a direct correspondence between the configurations γ of $\mathscr{S}(\mathcal{P}^N)$ and the configurations λ of $\mathscr{SG}(\mathcal{G}^N_\mathcal{P})$. Hence there is a direct correspondence between strategies on $\mathscr{S}(\mathcal{P}^N)$ and urgent strategies for player 1. Indeed, urgent strategies for player 1 lead to configurations with at most one process in the states of player 2, exactly like strategies in $\mathscr{S}(\mathcal{P}^N)$ that lead to configurations with at most one process in probabilistic sates.

For a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 0$, for every reachable configuration λ we know that $\mathsf{q}_f \notin \lambda$. For the same network size N, the strategy α_σ that corresponds to σ never uses a transition leading to q_f nor does it visit a state $\mathsf{q} \in \mathrm{L}^{(2)}$ that may lead to q_f. Therefore, α_σ ensures the safety objective from λ_0.

For a network size $N \in \mathbb{N}$ and a strategy α such that $\forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_s$, for every reachable configuration λ, we know that $\mathsf{q}_f \notin \lambda$, since the safety objective prevents using action leading to q_f. Hence the strategy σ_α corresponding to α never reaches a configuration containing q_f. Thus, $\mathbb{P}_{\sigma_\alpha}(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) = 0$.

This shows the correctness of the reduction. □

From Proposition 4.2 and Theorem 3.1, we deduce the decidability of the problem $REACH^{\exists}_{=0}(\mathscr{S})$ and establish a matching complexity lower-bound.

Theorem 4.4 $REACH^{\exists}_{=0}(\mathscr{S})$ *is co-NP-complete.*

Proof The co-NP membership is a consequence of Proposition 4.2 and Theorem 3.1. Indeed, Proposition 4.2 allows us to reduce the problem $REACH^{\exists}_{=0}(\mathscr{S})$ to the game problem that is shown to be in co-NP in Theorem 3.1. We now establish the matching lower-bound. To establish the co-NP-hardness, we reduce the unsatisfiability problem to $REACH^{\exists}_{=0}(\mathscr{S})$ as we did for $REACH^{\exists}_{=1}(\mathscr{S})$. From φ, a formula in conjunctive normal form, we define a probabilistic protocol $\mathcal{P}'_\varphi$ and a control state ℓ such that φ is unsatisfiable if and only if there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}({\mathcal{P}'_\varphi}^N) \models \Diamond \ell) = 0$.

The protocol $\mathcal{P}'_\varphi$ is a modification of protocol $\mathcal{P}_\varphi$ defined in the proof of Theorem 4.3 (represented in Figure 4.8). From any state q, of $\mathcal{P}_\varphi$, we remove the internal self loop, and add an internal transition $(\mathsf{q}, \varepsilon, \ell)$ to a new state ℓ, and we add the internal transitions $(\mathsf{q}_f, \varepsilon, \mathsf{q}_f)$ and $(\ell, \varepsilon, \ell)$ that loop on q_f and ℓ.

With these modifications, the only way to avoid reaching ℓ for an infinite execution is either to reach q_f and then loop there forever or to cycle forever in the original states of $\mathcal{P}_\varphi$. However, we have seen in the proof of Theorem 4.3, that the probability to obtain infinite executions in $\mathcal{P}_\varphi$ is 0. Thus, there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond\ell) = 0$ if and only if $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond\mathsf{q}_f) = 1$. According to Theorem 4.3, this only happens if and only if φ is unsatisfiable.

□

4.4 Decidability and complexity of $REACH^{\exists}_{<1}(\mathscr{S})$

For $REACH^{\exists}_{<1}(\mathscr{S})$, we reduce to a game problem for parity protocols with a combination of safety and parity winning conditions. From $\mathcal{P} = (\mathrm{Q}, \mathrm{Q}^{(n)}, \mathrm{Q}^{(p)}, \mathsf{q}_0, \Sigma, \Delta)$, a probabilistic protocol, and $\mathsf{q}_f \in \mathrm{Q}$ a control state, we define the parity protocol $\mathcal{G} = (\mathrm{L}_\mathcal{G}, \mathrm{L}^{(1)}_\mathcal{G}, \mathrm{L}^{(2)}_\mathcal{G}, \mathsf{l}_{0\mathcal{G}}, \Sigma_\mathcal{G}, \Delta_\mathcal{G}, \mathrm{col}, \mathrm{safe})$ as:

- $\mathrm{L}^{(1)}_\mathcal{G} = \big(\mathrm{L}^{(1)} \times \{1,2\}\big) \cup \big(\mathrm{Q}^{(p)} \times \{1\}\big)$, $\mathrm{L}^{(2)}_\mathcal{G} = \mathrm{Q}^{(p)} \times \{2\}$, and $\mathsf{l}_{0\mathcal{G}} = (\mathsf{q}_0, 1)$;
- $\Sigma_\mathcal{G} = \Sigma$;
- $\Delta_\mathcal{G} = \{((\mathsf{q}, i), \_, (\mathsf{q}', i)) \mid (\mathsf{q}, \_, \mathsf{q}') \in \Delta, i \in \{1,2\}\}$
  $\cup\ \{((\mathsf{q}, i), \varepsilon, (\mathsf{q}', i)) \mid (\mathsf{q}, d) \in \Delta,\ d(\mathsf{q}') > 0, i \in \{1,2\}\} \cup\ \{((\mathsf{q}, 1), \varepsilon, (\mathsf{q}, 2)) \mid \mathsf{q} \in \mathrm{Q}\}$;
- $\mathrm{col}((\mathsf{q}, i), \_, (\mathsf{q}', i'))) = 0$ if $i' = 2$, and 1 otherwise;
- $\mathrm{safe} = \Delta \setminus \{\delta \mid \delta \in \mathrm{L} \times \big(\{!!m, ??m \mid m \in \Sigma\} \cup \{\varepsilon\}\big) \times \{\mathsf{q}_f\}\}$.

Intuitively, $\mathcal{G}$ consists of two copies of $\mathcal{P}$. In the first copy, all random choices are replaced with choices of player 1, whereas in the second copy they are replaced with choices of player 2. Also, at any time, one can move from the first to the second copy. Figure 4.10 illustrates this reduction on the example probabilistic protocol from Figure 2.1. As usual, the dotted arrows represent unsafe edges. The second copy is represented symmetrically to the first copy in order to clarify the drawing.

This construction gives us a reduction of $REACH^{\exists}_{<1}(\mathscr{S})$ to the game problem for parity protocols with a safety/parity winning condition. More formally, this construction ensures the following equivalence:

Proposition 4.3 *$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{SG}(\mathcal{G}^N), \alpha, \beta) \in Win_{sp}$ if and only if $\exists N \in \mathbb{N}.\ \exists \sigma \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N), \Diamond \mathsf{l}_f) < 1$.*

Proof Since the only unsafe edges are the ones leading to copies of q_f, the safety parity winning condition forces a winning strategy to avoid q_f. Moreover, the parity condition ensures that only a finite number of "probabilistic" choices (*i.e.* choices that correspond to probabilistic choices in $\mathcal{P}$) are made by player 1. On the one hand, the existence of a winning strategy implies that with a fixed finite number of probabilistic choices for player 1, q_f can be avoided. On the other hand, any strategy for which the probability to reach q_f is less than one, guarantees that after a finite prefix (hence after finitely

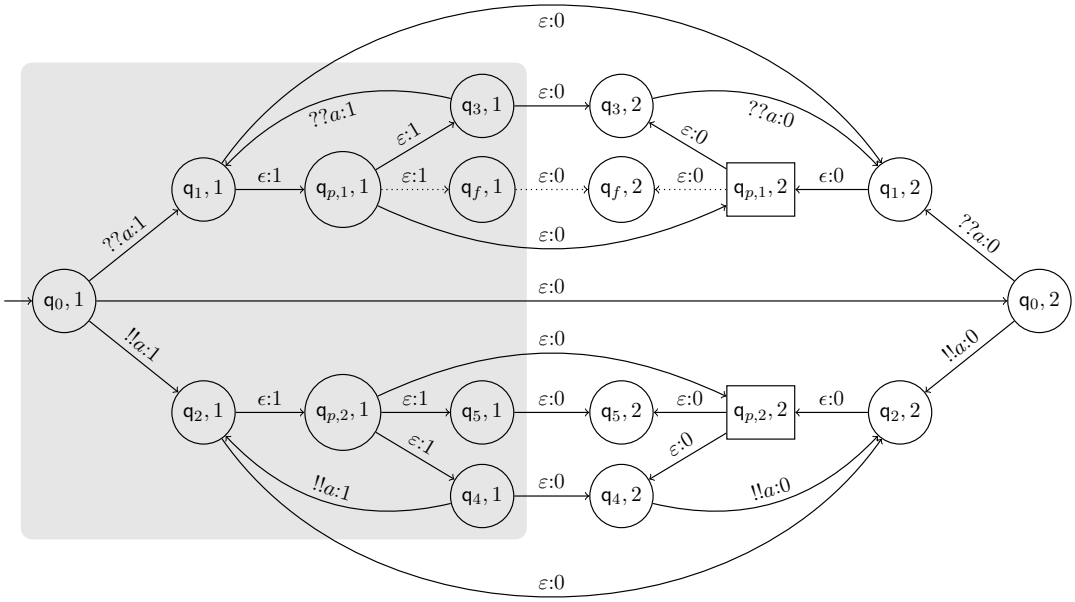


Figure 4.10: Parity protocol for the probabilistic protocol from Figure 2.1.

many probabilistic choices) q_f is avoided with probability 1 (thus whatever the other probabilistic choices).

We start by proving:

$$\exists N \in \mathbb{N}.\ \exists \sigma \in S, \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) < 1$$
$$\Downarrow$$
$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_{sp}.$$

Let $N \in \mathbb{N}$ be a network size and σ be a strategy such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) < 1$. Hence, there exists a finite path ρ that respects σ and such that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models Cyl(\rho) \cap \Diamond \mathsf{q}_f) = 0$, where $Cyl(\rho)$ is the cylinder of paths starting with prefix ρ.

We define α as the strategy, for the same network size N, that plays the finite prefix ρ with all processes in the first copy, and after ρ, moves them all to the second copy and then plays according to σ (as in the proof of Proposition 4.2).

For any finite play respecting α, the safety objective is satisfied because we never use a transition that enters $(\mathsf{q}_f, 1)$ during the prefix ρ, and because the same arguments as in the proof of Proposition 4.2 apply for the suffix in the second copy. Moreover, any infinite play respecting α only stays for a finite number of steps in the first copy and then ends in the second copy. Hence the play satisfies the parity objective. Altogether, α ensures the combined safety/parity objective.

We now prove:

$$\exists N \in \mathbb{N}.\ \exists \alpha \in S^{(1)}.\ \forall \beta \in S^{(2)},\ \rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_{sp}$$
$$\Downarrow$$
$$\exists N \in \mathbb{N}.\ \exists \sigma \in S, \mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) < 1.$$

Let $N \in \mathbb{N}$ be a network size and α be a strategy for player 1 such that $\forall \beta \in S^{(2)},\ \rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta) \in Win_{sp}$.

We define N as λ_0 but in which the labels $(\mathsf{q}_0, 1)$ are replaced with q_0. Also, if we merge states $(\mathsf{q}, 1)$ and $(\mathsf{q}, 2)$, a configuration λ of $\mathscr{S}\mathscr{G}(\mathcal{G}^N)$ can be seen as a configuration of $\mathscr{S}(\mathcal{P}^N)$. We define σ as the strategy that mimics α on these merged configurations, and that skips the actions leading from the first copy to the second one (since they have no meaning in $\mathcal{P}$).

Let A be the set of all plays that correspond to a play $\rho(\mathscr{S}\mathscr{G}(\mathcal{G}^N), \alpha, \beta)$ where we look at merged configurations and in which we removed the actions leading from copy 1 to copy 2. Since α is winning for the safety objective, we know that $\mathbb{P}(\mathscr{S}(\mathcal{P}^N), \sigma, A \cap \Diamond \mathsf{q}_f) = 0$. Moreover, since α ensures also the parity objective, we obtain that α only "chooses" a finite number of probabilistic choices (the choices made in the first copy). Hence there is a positive probability that these probabilistic choices are all correct in $\mathscr{S}(\mathcal{P}^N)$ under σ, hence $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A) > 0$.

The combination of $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A \cap \Diamond \mathsf{q}_f) = 0$ and $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models A) > 0$ yields that $\mathbb{P}_\sigma(\mathscr{S}(\mathcal{P}^N) \models \Diamond \mathsf{q}_f) < 1$. □

From Proposition 4.3 and Theorem 3.1, we deduce the decidability of the problem $REACH^{\exists}_{<1}(\mathscr{S})$ and establish a matching lower-bound.

Theorem 4.5 *$REACH^{\exists}_{<1}(\mathscr{S})$ is co-NP-complete.*

Proof The co-NP membership is a consequence of Proposition 4.3 and Theorem 3.1. Indeed, Proposition 4.3 allows us to reduce the problem $REACH^{\exists}_{=0}(\mathscr{S})$ to the game problem with safety/parity condition that is shown to be in co-NP in Theorem 3.1.

We now establish the matching lower-bound. To establish the co-NP-hardness we reduce the unsatisfiability problem to $REACH^{\exists}_{<1}(\mathscr{S})$ as we did for $REACH^{\exists}_{=1}(\mathscr{S})$ and $REACH^{\exists}_{=0}(\mathscr{S})$ (see proofs of Theorems 4.3 and 4.4). From φ, a formula in conjunctive normal form, we define a probabilistic protocol $\mathcal{P}''_\varphi$ and a control state ℓ such that φ is unsatisfiable if and only if there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}({\mathcal{P}''}^N_\varphi) \models \Diamond\ell) < 1$.

The protocol $\mathcal{P}''_\varphi$ is defined as an extension of protocol $\mathcal{P}_\varphi$ defined in the proof of Theorem 4.3 (represented in Figure 4.8) except that we remove the state q_f and r_p and consequently from r_1 the internal transition now goes to q_0. Additionally, from any state q of $\mathcal{P}_\varphi$, we remove the internal self loop and replace it with an internal transition $(\mathsf{q}, \varepsilon, \ell)$ to a new state ℓ.

With these modifications, the only way to avoid reaching ℓ for a maximal execution is to loop forever in the original states of $\mathcal{P}_\varphi$. However, we have seen in the proof of Theorem 4.3, that if the formula is satisfiable, for every strategy, the network will eventually get stuck in the original states of $\mathcal{P}_\varphi$ and thus one process will have to move to ℓ. In the other case, when φ is not satisfiable, it is possible to loop forever in the states of $\mathcal{P}_\varphi$ with probability 1 and thus to avoid ℓ. Therefore, there exist a network size $N \in \mathbb{N}$ and a strategy σ such that $\mathbb{P}_\sigma(\mathscr{S}({\mathcal{P}''}^N_\phi) \models \Diamond\ell) < 1$ if and only if φ is unsatisfiable. □

5 Conclusion

In this chapter, we studied a probabilistic extension of the reconfigurable broadcast protocols studied in [DSTZ12]. These networks consist of a parametric number of processes running the same finite state protocol. The processes are able to communicate with each other via selective broadcast of messages. In opposition to the clique networks, studied in Chapter III, the messages do not reach all the processes but only a subset of processes, chosen in a non-deterministic way. In addition to these non-deterministic behaviors, the protocols are equipped with probabilistic internal transitions that allow processes to change state according to fixed probabilistic distributions. These probabilistic transitions are useful to model uncertainty on the outcome of an action. They also can be used to model randomized distributed algorithms that use randomness to break the symmetry between the processes.

On these networks, we studied the parameterized qualitative reachability problems which consist in determining whether there exist a network size and a scheduler, resolving the non-determinism, that allow to reach almost surely a configuration in which at least one process is in a given state. We studied all the variants of this question by investigating both qualitative thresholds 0 and 1 and all the comparison operators.

Moreover, we also studied the universal questions which ask whether there exists a network size for which all strategies reach target configurations within the given threshold.

First, we used a key monotonicity result stating that adding processes in the network may only increase the probability to reach the target. Indeed, thanks to reconfigurations one can always leave apart the additional processes and simulate a smaller network. This result allows us to boil down all the universal parameterized questions to the same questions in a network with a single process. Indeed, intuitively either the property is true for the network of size one and thus the parameterized problem is solved. Otherwise, since there are strategies that ignore the additional processes, if it does not hold for the network of size one it will not hold for bigger networks.

In order to solve the other problems, we introduced selective broadcast networks of parity protocols which are parameterized distributed games. In these games, the role of player 1 and player 2 differ since player 1 only is able to decide which process will play next and its neighbors set. Player 2 only chooses the action to perform when a process, in a state belonging to player 2, was selected to play. This asymmetry, together with the fact that player 1 can partition the network, allowed us to prove that player 2 has a counter strategy for the parity or safety parity winning condition if and only if it has a very simple state-based strategy. The parameterized game problem thus boils down to checking whether there exist a network size and a strategy for player 1 winning against all state-based strategies for player 2. Moreover, we provided a reduction to parameterized VASS [KS88] allowing us to, given a state-based strategy for player 2, decide in polynomial time whether player 1 has a winning strategy. We thus obtained a co-NP algorithm consisting in guessing a counter local strategy for player 2 and checking whether it is indeed a counter strategy.

Finally we provided reductions from the parameterized probabilistic problems to parameterized game problems by modeling the probabilistic distributions by choices of player 2. We gave a polynomial reduction for each case, tuning the parity on the transition in order to reflect the specificity of each case. For example, to solve the almost sure reachability problem, the reduction consists in letting player 2 perform a finite number of choices, by setting the parity of these transitions even, and then checking whether player 1 has a strategy to reach the target by allowing him to perform the other choices but with odd parity. We also proved the co-NP hardness of these problems thanks to a reduction from the unsatisfiability problem of conjunctive normal form formulas. Notice that these hardness results entail the hardness of the game problems as well. We thus obtain that the game problems as well as the reachability problems are co-NP-complete.

We believe that distributed games are an interesting approach for the many identical processes setting and it would be interesting to study them further. In particular, one assumption made in this chapter is that there is no deadlock configuration. It would be interesting to see if our results still hold when this hypothesis is relaxed. Moreover, since in finite state systems the translation from Markov decision process to parity games allows to solve problems much harder than reachability, it would not be surprising if that would also be the case here. Our proof of decidability for the game problem crucially exploits the difference on the powers of player 1 and player 2. Even though giving to

player 2 the possibility to perform broadcasts and choose processes will certainly lead to undecidability, it would be interesting to see to what extent we can grant power to player 2 while retaining decidability. An other possibility is to investigate other communication means for distributed game networks, such as broadcasts or shared memories.

For the probabilistic parameterized problems, we only considered qualitative properties in this chapter. We are currently investigating the quantitative problems. It seems that, for some cases, the monotonicity of the networks allows us to reduce the quantitative problems to the qualitative ones. In the cases where the monotony does not hold, we believe that all the problems are monotonic in the number of processes for 'large enough' networks. In order to confirm or invalidate this intuition we are implementing, with the help of Aminatou Mohamadou, a master student in internship, a prototype tool in C. This tool allows us to build networks by making products of protocols for different sizes and check the desired properties thanks to the model checker Prism [KNP11].

Chapter V

Local strategies

1 Introduction

In this thesis, we consider parametric models with an unknown number of identical processes. This is a possible approach to tame distributed systems in which all processes share the same code. However, the solutions given until now all rely on centralized strategies to choose the actions of the processes with a full knowledge of the network. Due to the non-determinism, in the description of the protocol, it may happen that two processes behave differently, even if they have the same information on what has happened so far in an execution. To forbid such non-truly distributed behaviors, in this chapter, we constrain processes to take the same decisions in case they fired the same sequence of transitions so far.

To concentrate on this aspect of true distributivity, in this chapter we consider untimed and non-probabilistic networks composed of an arbitrary number of components, or processes, all running the same protocol. These protocols are finite state machines with three kinds of transitions: internal actions that affect only the process performing it, broadcasts that send messages to the other processes, and receptions that allow processes to receive the messages sent by broadcast. In this setting, the past of a process is the sequence of transitions it has taken so far. The *local strategies* ensure that processes with the same past take the same decision. As an example, from the initial configuration, hence with an empty past, if a process chooses to perform an internal action then all the processes with an empty past should perform the same internal action unless they get in the mean-time additional information by receiving a message.

We study the parameterized reachability and synchronization problems in broadcast protocol networks restricted to *local strategies*. The first problem, reachability, asks whether there exist a network size and a local strategy such that a configuration with at least one process in a given state is reached. The later problem, synchronization, asks to reach a configuration with all processes gathered in a given set of states. We consider two different settings for the networks: first, reconfigurable broadcast networks for which the messages reach only a subset of the processes chosen non-deterministically. In these networks, without the restriction to local strategies, the parameterized reachability

problem is known to be in PTIME [DSTZ12], and the synchronization problem has the same complexity (see Chapter II 2). The second networks we consider are clique broadcast networks, which are a restriction of reconfigurable networks in which the messages reach all processes every time.

Interestingly, the notably difficult distributed controller synthesis problem [PR90] is relatively close to the problem of existence of a local strategy. Indeed, a local strategy corresponds to a local controller for the processes executing the protocol and whose role is to resolve the non-deterministic choices. Local strategies are of interest to implement distributed algorithms. Indeed, non-determinism is hard to implement in real life, thus one can see the local strategy as an implementable deterministic version of a non-deterministic specification for a distributed algorithm.

The chapter is organized as follows. First, in Section 2 we recall the model of reconfigurable broadcast networks (see Chapter II 2) also called selective broadcast networks. We then introduce the notion of local strategies and local executions, as well as the problems studied, which are parameterized control state reachability and parameterized control state synchronization.

In Section 3.1, we show that the reachability and synchronization problems under local strategies in reconfigurable broadcast networks are NP-complete. To obtain the upper bound, we prove that local strategies can be succinctly represented by a finite tree of polynomial size in the size of the input protocol. This result is particularly interesting because deciding the existence of a local strategy is intrinsically difficult. Indeed, even with a fixed number of processes, the locality constraint cannot be simply tested on the induced transition system, and *a priori* local strategies may need unbounded memory. From our decidability proofs, we derive an upper bound on the memory needed to implement local strategies. We also give cutoffs, *i.e.* upper bounds on the minimal number of processes needed to reach or synchronize in target states.

Lastly, in Section 4, we show the two problems to be undecidable when the communication topology is a clique, that is when the broadcast of a message reaches all the processes in the network. Moreover, the undecidability proof of the target problem applies even if the locality assumption is dropped. However, the reachability problem under local strategies in cliques is decidable (yet non-primitive recursive) for complete protocols, *i.e.* when receptions are always possible from every state. Hence when the processes are input-complete, and when all messages reach all processes, the parameterized reachability problem with locality assumption is decidable.

2 Networks of reconfigurable broadcast protocols

2.1 Syntax and semantics

In this chapter, following the seminal approach by Delzanno et al. [DSZ10, DSZ11a, DSTZ12], we assume that each process in the network executes the same broadcast protocol given by a (non-deterministic) finite state machine where the actions are of three kinds: broadcast of a message m (denoted by $!!m$), reception of a message m (denoted by $??m$) and internal action (denoted by ε). We recall here the definition of a

broadcast protocol (see Chapter I Section 2):

Definition 2.1 *A* broadcast protocol *is a tuple* $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ *with:*

- Q *a finite set of control states;*
- $\mathsf{q}_0 \in \mathrm{Q}$ *the initial control state;*
- Σ *a finite message alphabet and*
- $\Delta \subseteq \mathrm{Q} \times (\{!!m, ??m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times \mathrm{Q}$ *a finite set of edges.*

We denote by $Act(\mathsf{q})$ the set of actions containing broadcasts and internal actions, called *active actions*, of $\mathcal{P}$ that start from state q. Formally, $Act(\mathsf{q}) = \{(\mathsf{q}, \varepsilon, \mathsf{q}') \in \Delta \mid \mathsf{q}' \in \mathrm{Q}\} \cup \{(\mathsf{q}, !!m, \mathsf{q}') \in \Delta \mid \mathsf{q}' \in \mathrm{Q}, m \in \Sigma\}$. Furthermore, for each message $m \in \Sigma$, we denote by $Recept_m(\mathsf{q})$ the set $\{(\mathsf{q}, ??m, \mathsf{q}') \in \Delta \mid \mathsf{q}' \in \mathrm{Q}\}$ containing the edges that start in state q and can be taken on reception of message m. We say that a broadcast protocol is *complete* if for every state $\mathsf{q} \in \mathrm{Q}$ and every message $m \in \Sigma$, $Recept_m(\mathsf{q}) \neq \emptyset$. Contrary to the previous chapters where the completeness of the protocols was assumed for simplicity of notations, in this chapter we will see that whether protocols are complete or not may change the decidability status of the problems we consider (see Section 4).

Example 2.1 *An example of a broadcast protocol is given in Figure 2.1. The initial state is* q_0*. There is a broadcast of message* m *from this state leading to* q_1 *and a reception of this message leading to* q_3*,* i.e. $(\mathsf{q}_0, !!m, \mathsf{q}_1) \in \Delta$ *and* $(\mathsf{q}_0, ??m, \mathsf{q}_3) \in \Delta$*. Notice that this protocol is not complete since there is no reception of message* m *in state* $\mathsf{q}_{\mathfrak{T}}$*.*

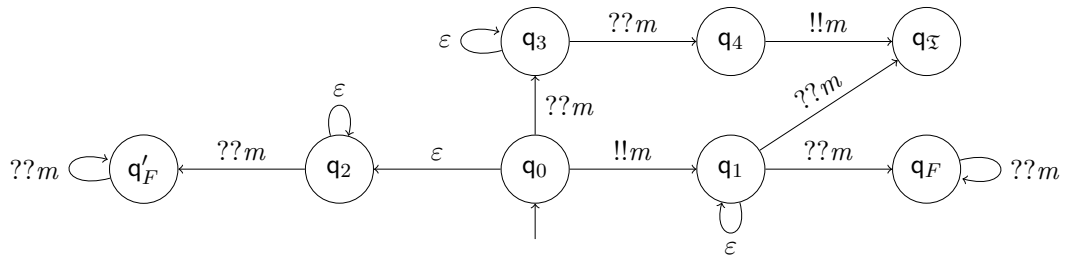


Figure 2.1: Example of a broadcast protocol.

Definition 2.2 (Selective broadcast network) *A* selective broadcast network *is composed of an arbitrary number of copies, called processes, of a broadcast protocol* $\mathcal{P}$*.*

We now define the semantics associated with such a network. It is common to represent the network topology by an undirected graph describing the communication links [DSTZ12]. Since the topology may change at any time (such an operation is called reconfiguration), we decide here, as in the Chapter IV, to simplify the

notations by specifying, for each broadcast, a set of possible receivers that is chosen non-deterministically. The semantics of a network built over a broadcast protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ is given by a transition system $\mathcal{T}_\mathcal{P} = (\Gamma, \Gamma_0, \rightarrow)$ where $\Gamma = \mathcal{V}_\mathrm{Q}$ is the set of configurations (represented by vectors over Q); $\Gamma_0 = \mathcal{V}_{\{\mathsf{q}_0\}}$ is the set of initial configurations and $\rightarrow \subseteq \Gamma \times \mathbb{N} \times \Delta \times 2^\mathbb{N} \times \Gamma$ is the transition relation defined as follows: $(\gamma, p_{id}, \delta, R, \gamma') \in \rightarrow$ (also denoted by $\gamma \xrightarrow{p_{id},\delta,R} \gamma'$) if and only if $|\gamma| = |\gamma'|$ and $p_{id} \in [1..|\gamma|]$ and $R \subseteq [1..|\gamma|] \setminus \{p_{id}\}$ and one of the following conditions holds:

> **Internal action:** $\delta = (\gamma[p_{id}], \varepsilon, \gamma'[p_{id}])$ and $\gamma'[p'_{id}] = \gamma[p'_{id}]$ for all $p'_{id} \in [1..|\gamma|] \setminus \{p_{id}\}$;
> *process p_{id} performs an internal action.*

> **Communication:** $\delta = (\gamma[p_{id}], !!m, \gamma'[p_{id}])$ and $(\gamma[p'_{id}], ??m, \gamma'[p'_{id}]) \in \Delta$ for all $p'_{id} \in R$ such that $Recept_m(\gamma[p'_{id}]) \neq \emptyset$, and $\gamma'[p''_{id}] = \gamma[p''_{id}]$ for all $p''_{id} \in [1..|\gamma|] \setminus (R \cup \{p_{id}\})$ and for all $p''_{id} \in R$ such that $Recept_m(\gamma[p''_{id}]) = \emptyset$;
> *process p_{id} broadcasts message m to all the processes in the reception set R (note that the message m is ignored by processes not ready to receive m).*

Obviously, when an internal action is performed, the reception set R is not taken into account. We point out the fact that the hypothesis $|\gamma| = |\gamma'|$ implies that the number of processes remains constant during an execution. Contrary to Chapter III, Section 2.3 there is no creation or deletion of processes. Yet, $\mathcal{T}_\mathcal{P}$ is an infinite state transition system since the number of possible initial configurations is infinite. An *execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_\mathcal{P}$ of the form $\theta = \gamma_0 \xrightarrow{p_{id0},\delta_0,R_0} \gamma_1 \dots \xrightarrow{p_{id\ell},\delta_\ell,R_\ell} \gamma_{\ell+1}$ and we denote by $\Theta[\mathcal{P}]$ (or simply Θ when $\mathcal{P}$ is clear from context) the set of all executions of $\mathcal{P}$. Furthermore, we use $nbproc(\theta) = |\gamma_0|$ to represent the number of processes involved in the execution θ.

2.2 Restricting executions to local strategies and clique executions

Our goal is to analyze executions of broadcast protocols under *local strategies*, where each process performs the same choices of edges according to its past history (*i.e.* according to the edges of the protocol it has fired so far).

A *finite path* in $\mathcal{P}$ is either the empty path, denoted by ϵ, or a non-empty finite sequence of edges $\delta_0 \cdots \delta_\ell$ such that δ_0 starts in q_0 and for all $i \in [1..\ell]$, δ_i starts in the state in which δ_{i-1} ends. For convenience, we say that ϵ ends in state q_0. We write $\mathsf{Path}(\mathcal{P})$ for the set of all finite paths in $\mathcal{P}$.

For an execution $\theta \in \Theta[\mathcal{P}]$, we define, for every $p_{id} \in [1..nbproc(\theta)]$, the *past* of process p_{id} in θ (also referred to as its *history*), written $\pi_{p_{id}}(\theta)$, as the finite path in $\mathcal{P}$ that stores the sequence of edges of $\mathcal{P}$ taken by p_{id} along θ. Formally, for an execution $\theta \in \Theta[\mathcal{P}]$, we inductively define for all $p_{id} \in [1..nbproc(\theta)]$ the *past*, or *history*, of process p_{id} in θ, written $\pi_{p_{id}}(\theta)$, as follows: $\pi_{p_{id}}(\gamma_0) = \epsilon$ and for $\rho = \gamma_0 \xrightarrow{p_{id0},\delta_0,R_0} \gamma_1 \dots \xrightarrow{p_{id_{n-1}},\delta_{n-1},R_{n-1}} \gamma_n$, $\pi_{p_{id}}(\rho \xrightarrow{p_{id_n},\delta_n,R_n} \gamma_{n+1})$ is equal to:

- $\pi_{p_{id}}(\rho)\cdot\delta_n$ if $p_{id_n} = p_{id}$; the past is augmented by the active action if it is performed by p_{id}.
- $\pi_{p_{id}}(\rho) \cdot \delta$ if $p_{id} \in R_n$, $\delta = (\gamma_n[p_{id}], ??m, \gamma_{n+1}[p_{id}]) \in Recept_m(\gamma_n[p_{id}])$ and $\delta_n = (\gamma_n[p_{id_n}], !!m, \gamma_{n+1}[p_{id_n}])$; in the case of a broadcast received by p_{id}, the past is augmented by the reception performed by p_{id}.
- $\pi_{p_{id}}(\rho)$ if $p_{id} \in R_n$, $\delta_n = (\gamma_n[p_{id_n}], !!m, \gamma_{n+1}[p_{id_n}])$ and $Recept_m(\gamma_n[p_{id}]) = \emptyset$; in the case of a broadcast of a message that p_{id} cannot receive, its past does not change.
- $\pi_{p_{id}}(\rho)$ otherwise. In the case where p_{id} is not involved, its past does not change.

Note that by definition of the transition relation $\rightarrow$ of $\mathcal{T}_{\mathcal{P}}$, for every execution θ and every $p_{id} \in [1..nbproc(\theta)]$, the past of process p_{id} in θ is a finite path in $\mathcal{P}$.

We can now define local strategies which allow us to focus on the executions in which each process performs the same choices according to its past.

Definition 2.3 (Local strategy) *A* local strategy σ *for* $\mathcal{P}$ *is a pair* (σ_a, σ_r) *of functions specifying, given a past, the next active action to be taken, and the reception edge to choose in case of a broadcast, respectively.*

Formally $\sigma_a : \mathsf{Path}(\mathcal{P}) \rightarrow \mathrm{Q} \times (\{!!m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times \mathrm{Q}$ *satisfies, for every* $\rho \in \mathsf{Path}(\mathcal{P})$ *ending in* $\mathsf{q} \in \mathrm{Q}$*, either* $Act(\mathsf{q}) = \emptyset$ *or* $\sigma_a(\rho) \in Act(\mathsf{q})$*. Moreover,* $\sigma_r : \mathsf{Path}(\mathcal{P}) \times \Sigma \rightarrow \mathrm{Q} \times \{??m \mid m \in \Sigma\} \times \mathrm{Q}$ *satisfies, for every* $\rho \in \mathsf{Path}(\mathcal{P})$ *ending in* $\mathsf{q} \in \mathrm{Q}$ *and every* $m \in \Sigma$*, either* $Recept_m(\mathsf{q}) = \emptyset$ *or* $\sigma_r(\rho, m) \in Recept_m(\mathsf{q})$*.*

Since our aim is to analyze executions where each process behaves according to the same local strategy, we now provide the formal definition of such executions. Given a local strategy $\sigma = (\sigma_a, \sigma_r)$, we say that a path $\delta_0 \cdots \delta_\ell$ *respects* σ if for all $i \in [0..\ell - 1]$, we have $\delta_{i+1} = \sigma_a(\delta_0 \dots \delta_i)$ or $\delta_{i+1} = \sigma_r(\delta_0 \cdots \delta_i, m)$ for some $m \in \Sigma$. Following this, an execution θ respects σ if for all processes $p_{id} \in [1..nbproc(\theta)]$, we have that $\pi_{p_{id}}(\theta)$ respects σ (*i.e.* each process behaves as dictated by σ). Finally, we define $\Theta_{\mathcal{L}} \subseteq \Theta$ as the set of *local executions* (also called local semantics), which are executions θ respecting a local strategy.

We also consider another set of executions where we assume that every message is broadcast to all the processes of the network (apart from the emitter). Formally, an execution $\theta = \gamma_0 \xrightarrow{p_{id_0}, \delta_0, R_0} \dots \xrightarrow{p_{id_\ell}, \delta_\ell, R_\ell} \gamma_{\ell+1}$ is said to be a *clique execution* if $R_k = [1, \dots, nbproc(\theta)] \setminus \{p_{id_k}\}$ for every $k \in [0..\ell]$. We denote by $\Theta_{\mathcal{C}}$ the set of clique executions (also called clique semantics). Note that clique executions of broadcast networks have been studied in [DSZ11a] and that such networks correspond to broadcast protocols with no *rendez-vous* [EFM99]. We will also consider combination of these 2 restrictions hence the intersection of these subsets of executions and write $\Theta_{\mathcal{LC}}$ for the set $\Theta_{\mathcal{L}} \cap \Theta_{\mathcal{C}}$ of clique executions which respect a local strategy.

Example 2.2 *To illustrate the notions of local strategies and clique executions, we provide an example of a broadcast protocol in Figure 2.1. On this protocol, no clique*

execution can reach state q_F*: as soon as a process in* q_0 *sends message* m*, all the other processes in* q_0 *receive this message, and move to* q_3 *because of the clique topology. An example of a clique execution is:* $(\mathsf{q}_0, \mathsf{q}_0, \mathsf{q}_0, \mathsf{q}_0) \to (\mathsf{q}_1, \mathsf{q}_3, \mathsf{q}_3, \mathsf{q}_3)$ *(where we omit the labels over* $\to$*). However, there exists a local execution reaching* q_F*:* $(\mathsf{q}_0, \mathsf{q}_0) \to (\mathsf{q}_1, \mathsf{q}_0) \to (\mathsf{q}_F, \mathsf{q}_1)$*. This execution respects a local strategy. Indeed, from* q_0 *with empty past, the first process chooses the edge broadcasting* m *with empty reception set, and in the next step, the second process, also with empty past, performs the same action, broadcasting the message* m *to the first process. On the other hand, no local strategy permits to reach* q'_F*. Indeed, intuitively, to reach* q'_F*, in state* q_0 *one process with empty past needs to go to* q_1 *and another one to* q_2*, which is forbidden by locality. Finally,* $(\mathsf{q}_0, \mathsf{q}_0, \mathsf{q}_0) \to (\mathsf{q}_1, \mathsf{q}_0, \mathsf{q}_3) \to (\mathsf{q}_1, \mathsf{q}_1, \mathsf{q}_4) \to (\mathsf{q}_\mathfrak{T}, \mathsf{q}_\mathfrak{T}, \mathsf{q}_\mathfrak{T})$ *is a local execution that synchronizes all processes in the target set* $\mathfrak{T} = \{\mathsf{q}_\mathfrak{T}\}$*.*

2.3 Verification problems

In this work, we study the parameterized verification of the reachability and synchronization properties for broadcast protocols restricted to local strategies. The first one asks whether there exists an execution respecting some local strategy that eventually reaches a configuration where a given control state appears. The latter problem seeks for an execution respecting some local strategy ending in a configuration where all the control states belong to a given target set. We consider several variants of these problems depending on whether we restrict to clique executions or not and whether we consider complete protocols or not.

For an execution $\theta = \gamma_0 \xrightarrow{p_{id_0}, \delta_0, R_0} \gamma_1 \dots \xrightarrow{p_{id_\ell}, \delta_\ell, R_\ell} \gamma_{\ell+1}$, we denote by $\mathsf{End}(\theta) = \{\gamma_{\ell+1}[p_{id}] \mid p_{id} \in [1..nbproc(\theta)]\}$ the set of states that appear in the last configuration of θ. Reach$[\mathcal{S}]$, the parameterized reachability problem for executions restricted to a class $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$, is defined as follows:

Reach$[\mathcal{S}]$
Input: A broadcast protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ and a control state $\mathsf{q}_F \in \mathrm{Q}$.
Output: Does there exist an execution $\theta \in \Theta_\mathcal{S}$ such that $\mathsf{q}_F \in \mathsf{End}(\theta)$?

In previous works, the parameterized reachability problem has been studied without the restriction to local strategies. In particular the reachability problem on unconstrained executions is in PTIME [DSTZ12] and Reach$[\mathcal{C}]$ is decidable and Non-Primitive Recursive (NPR) [DSZ11a, EFM99] (actually it is Ackermann-complete [SS13]).

Similarly, Synch$[\mathcal{S}]$, the parameterized synchronization problem for executions restricted to the class $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$ is defined as follows:

Synch$[\mathcal{S}]$
Input: A broadcast protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ and a set of control states $\mathfrak{T} \subseteq \mathrm{Q}$.
Output: Does there exist an execution $\theta \in \Theta_\mathcal{S}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$?

It has been shown that a generalization of the synchronization problem, without restriction to local strategies, can be solved in NP [DSTZ12] (as it is recalled in Section 2).

In this work, we focus on executions under local strategies and we obtain the results presented in the following table:

REACH[$\mathcal{L}$]	REACH[$\mathcal{LC}$]	SYNCH[$\mathcal{L}$]	SYNCH[$\mathcal{LC}$]
NP-complete [Thm. 3.2]	Undecidable [Thm. 4.1] Decidable and NPR for complete protocols [Thm. 4.2]	NP-complete [Thm. 3.3]	Undecidable [Thm. 4.1]

Most of the problems listed in the above table are monotonic: if, in a network of a given size, an execution satisfying the reachability or synchronization property exists, then, in any bigger network, there also exists an execution satisfying the same property. More precisely:

Proposition 2.1 *Let θ be an execution. For every $N \geq nbproc(\theta)$:*

- *if $\theta \in \Theta_{\mathcal{L}}$ there exists θ' in $\Theta_{\mathcal{L}}$ such that $nbproc(\theta') = N$ and $\mathsf{End}(\theta) = \mathsf{End}(\theta')$.*
- *if $\theta \in \Theta_{\mathcal{LC}}$, there exists θ' in $\Theta_{\mathcal{LC}}$ such that $nbproc(\theta') = N$ and $\mathsf{End}(\theta) \subseteq \mathsf{End}(\theta')$.*

Proof We first prove that, given a local execution $\theta \in \Theta_{\mathcal{L}}$, there exists another local execution $\theta' \in \Theta_{\mathcal{L}}$ such that $nbproc(\theta') = nbproc(\theta) + 1$ and $\mathsf{End}(\theta) = \mathsf{End}(\theta')$. The proof is by induction on the length of θ. The idea is to add in θ' a process denoted $p_{id_{add}}$ that behaves exactly as the first process (with process identifier 1) of θ and such that all other processes behave in θ' as in θ. Formally, we define inductively a function $copycat(\theta)$, such that $copycat(\gamma_0) = \gamma_0'$ with $|\gamma_0'| = |\gamma_0| + 1$ and

$$copycat(\theta \xrightarrow{p_{id},\delta,R} \gamma) = \begin{cases} copycat(\theta) \xrightarrow{p_{id},\delta,R} \gamma' & \text{if } p_{id} \neq 1 \text{ and } 1 \notin R \\ copycat(\theta) \xrightarrow{p_{id},\delta,R\cup\{p_{id_{add}}\}} \gamma' & \text{if } p_{id} \neq 1 \text{ and } 1 \in R \\ copycat(\theta) \xrightarrow{p_{id},\delta,R} \gamma_{int} \xrightarrow{p_{id_{add}},\delta,\emptyset} \gamma' & \text{if } p_{id} = 1 \end{cases}$$

$$\text{with } \gamma'[p_{id_{add}}] = \gamma[1] \text{ and } \forall p'_{id} \in [1..nbproc(\theta)],\ \gamma'[p'_{id}] = \gamma[p'_{id}].$$

Intuitively, if the transition did not affect the first process in θ, the exact same transition is fired in θ', and it affects neither process 1 nor process $p_{id_{add}}$. Otherwise, in case process 1 receives a message, $p_{id_{add}}$ performs exactly the same reception (as specified by the reception set and the condition on γ'). Finally, if process 1 performs an active action in θ, $p_{id_{add}}$ also performs that active action, yet the associated reception set is empty, so that execution θ can continue on the original processes.

Clearly enough, for any local execution $\theta \in \Theta_{\mathcal{L}}$, $copycat(\theta)$ is also a local execution. At any time processes 1 and $p_{id_{add}}$ share the same past and behave similarly. Moreover this execution satisfies $nbproc(copycat(\theta)) = nbproc(\theta) + 1$ and $\mathsf{End}(\theta) = \mathsf{End}(copycat(\theta))$. Applying iteratively the function $copycat$, one obtains a local execution θ' with arbitrarily many processes and such that $\mathsf{End}(\theta') = \mathsf{End}(\theta)$. This shows the first item of Proposition 2.1.

We now restrict to local clique executions, and similarly to the previous case, prove that given a local clique execution $\theta \in \Theta_{\mathcal{LC}}$ there exists $\theta' \in \Theta_{\mathcal{LC}}$ such that $nbproc(\theta') =$

$nbproc(\theta) + 1$ and $\mathsf{End}(\theta) \subseteq \mathsf{End}(\theta')$. The proof is easier than for the first item since we only require an inclusion of the set of states appearing in the last configuration. It suffices to add a new process $p_{id_{add}}$ that does not perform any active action, so that θ can be mimicked exactly, yet on a larger number of processes. Formally, given a strategy σ and an execution θ following σ, we define inductively the function $passiv(\theta)$, such that $passiv(\gamma_0) = \gamma'_0$ with $|\gamma'_0| = |\gamma_0| + 1$ and $passiv(\theta \xrightarrow{p_{id},\delta,R} \gamma) = passiv(\theta) \xrightarrow{p_{id},\delta,R\cup\{p_{id_{add}}\}} \gamma'$ where $\forall p'_{id} \in [1..nbproc(\theta)],\ \gamma'[p'_{id}] = \gamma[p'_{id}]$ and in the case where δ is a broadcast of $m \in \Sigma$, we ask that $p_{id_{add}}$ follows the local strategy: $\sigma_r(\pi_{p_{id_{add}}}(passiv(\theta)), m) = (\mathsf{dest}(\pi_{p_{id_{add}}}(passiv(\theta))), ??m, \gamma'[p_{id_{add}}])$.

Clearly enough for any local clique execution $\theta \in \Theta_{\mathcal{LC}}$, $passiv(\theta)$ is also a local clique execution since $p_{id_{add}}$ does not perform any active action and is present in all reception sets. Moreover, it satisfies $nbproc(passiv(\theta)) = nbproc(\theta) + 1$ and $\mathsf{End}(\theta) \subseteq \mathsf{End}(passiv(\theta))$. Applying iteratively the function $passiv$, one obtains a local execution θ' with arbitrarily many processes and such that $\mathsf{End}(\theta) \subseteq \mathsf{End}(\theta')$.

Note that in the case of a clique topology, one cannot preserve the set $\mathsf{End}(\theta)$ in general while increasing the number of processes, because processes are bound to receive all messages. Consider as an example the simple protocol composed only of two transitions $(\mathsf{q}_0, !!m, \mathsf{q}_1)$ and $(\mathsf{q}_0, ??m, \mathsf{q}_2)$. It admits a local clique execution $\theta \in \Theta_{\mathcal{LC}}$ with a single process such that $\mathsf{End}(\theta) = \{\mathsf{q}_1\}$, yet any local clique execution θ' with at least two processes satisfies $\mathsf{End}(\theta') = \{\mathsf{q}_1, \mathsf{q}_2\}$ or $\mathsf{End}(\theta') = \{\mathsf{q}_0\}$. □

These monotonicity properties allow us to look for cutoffs, *i.e.* minimal number of processes such that a local execution with a given property exists. Indeed, if we can effectively compute a network size such that the property holds, using Proposition 2.1, one obtains that for any bigger network the property holds. We will see that we can compute an upper-bound on these cutoffs for the problems Reach[$\mathcal{L}$] (Proposition 3.3) and Synch[$\mathcal{L}$] (Theorem 3.3.2). Moreover, it is shown that these cutoffs are polynomial in the size of the protocol. For the combined case of locality and clique restricted to complete protocols (*i.e.* Reach[$\mathcal{LC}$]), given that the complexity is Ackerman-hard, such an upper-bound would be non-primitive recursive and thus would not be of any practical use.

3 Solving verification problems for local executions

We begin with studying the parameterized reachability and synchronization problems under local executions, *i.e.* we seek for a local strategy ensuring either to reach a specific control state, or to reach a configuration in which all the control states belong to a given set.

3.1 Solving Reach[$\mathcal{L}$]

To obtain an NP-algorithm for Reach[$\mathcal{L}$], we prove that there exists a local strategy to reach a specific control state if and only if there is a local strategy which can be represented thanks to a finite tree of polynomial size. The idea behind such a tree

is that the paths in the tree represent relevant past histories and the edges outgoing a specific node represent the decisions of the local strategy. The algorithm will then consist in guessing such finite tree of polynomial size and verifying that it satisfies some conditions needed to reach the specified control state. This can be done in polynomial time, so that we obtain an NP-algorithm.

3.1.1 Representing strategies with trees

We now define our tree representation of strategies called strategy patterns, which are standard labeled trees with labels on the edges. Intuitively, a strategy pattern defines, for some of the paths in the associated protocol, the active action and reception edges to take.

Let us first provide some formal definitions with relation to labeled trees used to represent strategy patterns.

Definition 3.1 (Labeled tree) *A* labeled tree *is a finite graph* $T = (N, n_0, E, \Upsilon, \mathsf{lab})$ *where* N *is a finite set of nodes,* $n_0 \in N$ *is called the* root *of* T *and* $E \subseteq N \times N$ *is the edge relation which satisfies the following conditions for all* $n \in N$*:* $(n, n) \notin E$*;* $(n, n_0) \notin E$*; if* $n \neq n_0$ *then there exists a unique* $n' \in N$ *such that* $(n', n) \in E$*; moreover,* $\mathsf{lab} : E \to \Upsilon$ *is an edge-labeling function.*

For each edge $e = (n, n')$, we use the following notations: $\mathsf{src}(e) = n$ to denote the source and $\mathsf{dest}(e) = n'$ the destination of e. A path in the tree is then either the empty path ϵ or a finite sequence of edges $e_1 \cdots e_\ell$ such that $\mathsf{src}(e_1) = n_0$ and $\mathsf{dest}(e_i) = \mathsf{src}(e_{i+1})$ for all $i \in [1..\ell - 1]$. A node n' is said to be the descendant of a node n if there exist a non-empty path $e_1 \cdots e_\ell$ and $i, j \in [1..\ell]$ such that $i \leq j$ and $\mathsf{src}(e_i) = n$ and $\mathsf{dest}(e_j) = n'$. We denote by $desc(T, n)$ the set of descendants of a node n in T.

The *subtree* at node $n \in N$ of T, denoted by $\mathsf{Sub}(T, n)$ is the tree $(desc(T, n) \cup \{n\}, n, E', \Upsilon, \mathsf{lab}')$ such that $E' = E \cap ((desc(T, n) \cup \{n\}) \times desc(T, n))$ and lab' is the restriction of lab to E'. For a node $n \in N$ such that $n \neq n_0$, we use $\mathsf{pred}(n)$ to represent the unique edge $e \in E$ such that $\mathsf{dest}(e) = n$. Finally, we define the size of T, denoted by $|T|$, as the number of its nodes.

We are now ready to define strategy patterns, which consist in finite trees labeled by transitions of a broadcast protocol. Alternatively, one can see a strategy pattern as a finite unfolding, with constraints, of a broadcast protocol $\mathcal{P}$.

Definition 3.2 (Strategy pattern) *A* strategy pattern *for a broadcast protocol* $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ *is a labeled tree* $T = (N, n_0, E, \Delta, \mathsf{lab})$ *such that, if* $e_1 \cdots e_\ell$ *is a path in* T*, then* $\mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell) \in \mathsf{Path}(\mathcal{P})$*, and for every node* $n \in N$ *and every message* $m \in \Sigma$*, we have:*

- *there is at most one edge* $e = (n, n') \in E$ *such that* $\mathsf{lab}(e)$ *is an active action;*
- *there is at most one edge* $e = (n, n') \in E$ *such that* $\mathsf{lab}(e)$ *is a reception of* m*.*

Since all labels of edges outgoing a node share a common source state (due to the hypothesis on labeling of paths), the labeling function lab can be consistently extended to nodes by letting $\mathsf{lab}(n_0) = \mathsf{q}_0$ and $\mathsf{lab}(n) = \mathsf{q}$ as soon as $(n', n) \in E$ and $\mathsf{lab}((n', n)) = (\mathsf{q}', a, \mathsf{q})$.

Given a strategy pattern $T = (N, n_0, E, \Delta, \mathsf{lab})$ for a broadcast protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$, let us define the *history* function $h : N \to \mathsf{Path}(\mathcal{P})$ that associates with each node of the strategy pattern the path in $\mathcal{P}$ it represents. Formally, for every $n \in N$ writing $e_1 \cdots e_\ell$ for the path in T with $\mathsf{dest}(e_\ell) = n$ then $h(n) = \mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell)$.

Example 3.1 *A strategy pattern is represented in Figure 3.2, for the broadcast protocol from Fig. 2.1.*

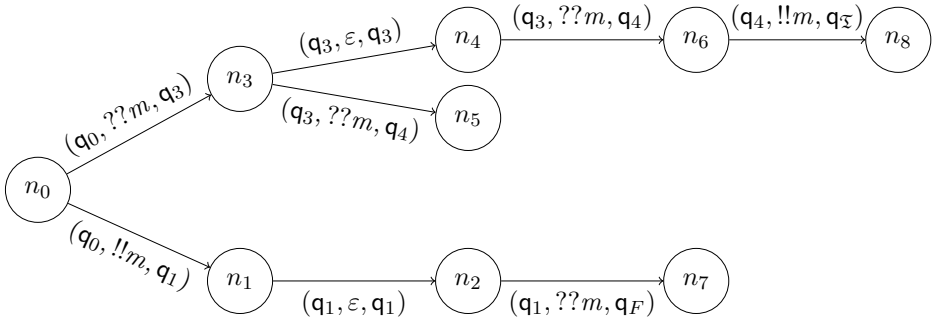


Figure 3.2: A strategy pattern for the broadcast protocol depicted in Figure 2.1.

In this example, $\mathsf{lab}(n_5) = \mathsf{q}_4$ *since the edge entering* n_5 *is labeled by a transition with destination* q_4*. Moreover, the history* $h(n_6)$ *of node* n_6 *is composed of the sequence of transitions:* $(\mathsf{q}_0, ??m\mathsf{q}_3)(\mathsf{q}_3, \varepsilon, \mathsf{q}_3)(\mathsf{q}_3, ??m, \mathsf{q}_4)$*.*

This example illustrates that strategy patterns somehow correspond to under-specified local strategies. For example, this pattern corresponds to a local strategy $\sigma = (\sigma_a, \sigma_r)$ *such that* $\sigma_r((\mathsf{q}_0, ??m\mathsf{q}_3)(\mathsf{q}_3, \varepsilon, \mathsf{q}_3), m) = (\mathsf{q}_3, ??m, \mathsf{q}_4)$ *which is represented by the label on the edge* (n_4, n_6)*.*

However, strategies patterns are under-specified. Indeed, from node n_1 *(labeled by* q_1*) no reception of message* m *is specified, and from node* n_5 *(labeled by* q_4*) no reception and no active action are specified. This pattern thus corresponds to many different local strategies that can take different decisions from node* n_1 *or* n_5*.*

More generally, given $\mathcal{P}$ a broadcast protocol, and T a strategy pattern for $\mathcal{P}$ with edge-labeling function lab, a local strategy $\sigma = (\sigma_a, \sigma_r)$ for $\mathcal{P}$ is said to *follow* T if for every path $e_1 \cdots e_\ell$ in T, the path $\rho = \mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell)$ in $\mathcal{P}$ respects σ. Notice that any strategy pattern admits at least one local strategy that follows it (actually, in general, it admits several). Reciprocally, for any local strategy there exists at least one (several in general) strategy pattern representing it.

3.1.2 Reasoning on strategy patterns

We now show that one can test directly on a strategy pattern whether the local strategies following it can yield an execution reaching a specific control state.

Definition 3.3 (Admissible strategy pattern) *An* admissible strategy pattern *for* $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ *is a pair* $(T, \prec)$ *where* $T = (N, n_0, E, \Delta, \mathsf{lab})$ *is a strategy pattern for* $\mathcal{P}$ *and* $\prec \subseteq N \times N$ *is a strict total order on the nodes of* T *such that:*

(1) for all $(n, n') \in E$ *we have* $n \prec n'$*;*

(2) for all $e = (n, n') \in E$*, if* $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ *for some* $m \in \Sigma$*, then there exists and edge* $e_1 = (n_1, n_1') \in E$ *such that* $n_1' \prec n'$ *and* $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n_1'))$*.*

In words, in this definition, (1) states that $\prec$ respects the natural order on the tree and (2) that every node corresponding to a reception of m should be preceded by a node corresponding to a broadcast of m.

Example 3.2 *The example of strategy pattern on Fig. 3.2 is admissible with the order* $n_i \prec n_j$ *if* $i < j$*. On the contrary, for any order including* $n_3 \prec n_1$ *it is not admissible. Indeed, the destination of a broadcast of* m *should precede* n_3*.*

We now give the relation between admissible strategy patterns and local strategies, but first we establish the complexity of checking whether a pattern is admissible.

Lemma 3.1 *Given a strategy pattern* $T = (N, n_0, E, \Delta, \mathsf{lab})$ *for a broadcast protocol* $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ *and a strict total order* $\prec \subseteq N \times N$*, checking whether* $(T, \prec)$ *is admissible can be done in polynomial time.*

Proof The proof is by induction on the size of the T.

For the base case, we consider the only possible pattern with a single node: $T = (\{n_0\}, n_0, \emptyset, \Delta, \mathsf{lab})$. The only strict total order is the trivial ordering $\prec = \emptyset$. The two conditions of the definition are trivially respected because the edge relation is empty.

We now assume that for all strategy patterns of size K and for all strict total orders on the nodes, we can check in polynomial time whether $(T, \prec)$ is admissible. We will now prove that this property still holds for the strategy patterns of size $K + 1$.
Let $T = (N, n_0, E, \Delta, \mathsf{lab})$ be a strategy pattern of size $K+1$ and $\prec \subseteq N \times N$ be a strict total order on the nodes. Let n be the maximal node with respect to $\prec$. First, we can check in polynomial time if there exists a node $n' \in N$ such that $(n, n') \in E$. If such a node exists then $(T, \prec)$ is not admissible (contradiction with condition *(1)*). Otherwise, let T' be the pattern in which we remove the node n and its associated edges and $\prec'$ the total order $\prec$ without the node n. Formally, $T' = (N \setminus \{n\}, n_0, E \setminus \{\mathsf{pred}(n)\}, \Delta, \mathsf{lab}')$ and $\prec' = \prec \setminus \{(n', n) \mid n' \in N\}$. By induction hypothesis, we can check in polynomial time whether $(T', \prec')$ is admissible. If it is not admissible, then one of the conditions is violated and would also be violated for $(T, \prec)$. Otherwise, the last thing to check in

the case where $\mathsf{lab}(\mathsf{pred}(n))$ is a reception of a message m, is whether there exists an edge $e_1 = (n_1, n_1') \in E$ such that $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n_1'))$, otherwise *(2)* is not satisfied. □

Our objective is to show that admissible strategy patterns are necessary and sufficient to represent the sets of states that can be reached under local strategies. To do so we introduce the following: given an execution $\theta \in \Theta[\mathcal{P}]$, process p_{id} is said to be in node n if $h(n) = \pi_{p_{id}}(\theta)$.

We now prove the following lemma, which states that for all strategy patterns and all integers M, there exists a local execution that puts at least M processes in each node of the pattern. Formally:

Lemma 3.2 *Given an admissible strategy pattern $(T, \prec)$, for all $M \in \mathbb{N} \setminus \{0\}$ and for all strategies σ that follows T, there exists an execution $\theta \in \Theta[\mathcal{P}]$ that respects σ and such that for all nodes n of T:*

$$|\{p_{id} \in [0 \dots nbproc(\theta)] \mid \pi_{p_{id}}(\theta) = h(n)\}| \geq M$$

Proof The proof is by induction on the size of the strategy pattern T.

For the base case, we consider the only possible pattern with a single node: $T = (\{n_0\}, n_0, \emptyset, \Delta, \mathsf{lab})$ with trivial ordering $\prec = \emptyset$. Any local strategy σ follows T. For any $M \in \mathbb{N} \setminus \{0\}$, the execution consisting only of the initial configuration $\gamma_0 = \{\mathsf{q}_0\}^M$ respects any local strategy σ and in the last configuration, there are exactly M processes in node n_0.

We now assume that the property holds for all the admissible strategy patterns of size K and we will prove that it holds for the admissible strategy patterns of size $K+1$. Let $(T, \prec)$ be an admissible strategy pattern of size $K+1$ with $T = (N, n_0, E, \Delta, \mathsf{lab})$. Let σ be a strategy following T and $M \in \mathbb{N} \setminus \{0\}$. We denote by $n \in N$ the maximal node according to the total order $\prec$. Note that n is necessarily a leaf thanks to condition *(1)* on admissible strategy patterns. We denote by $(T', \prec')$ the admissible strategy pattern obtained from $(T, \prec)$ by removing the leaf n and its preceding edge $\mathsf{pred}(n) = (n', n)$.

First, note that σ also follows T'. By induction hypothesis applied to T' and $M' = 2M + 1$, there exists an execution θ such that θ respects σ and such that there are at least $2M + 1$ processes in each node of T' in the last configuration of θ.

Let us now explain how θ can be extended depending on the type of the label of the deleted edge $\delta = \mathsf{lab}(\mathsf{pred}(n))$:

- If δ is an active action, either internal $(\mathsf{q}, \varepsilon, \mathsf{q}')$ or a broadcast $(\mathsf{q}, !!m, \mathsf{q}')$, then we know that there are $2M + 1$ processes in n'; hence we extend θ by choosing M processes among those processes to perform the active action δ with an empty reception set at each step;

- If $\delta = (\mathsf{q}, ??m, \mathsf{q}')$, then we know that since $(T, \prec)$ is an admissible strategy pattern, there exists an edge $e_1 = (n_1, n_1')$ in T such that $n_1' \prec n$ and $\mathsf{lab}(e_1) = (\mathsf{q}_1, !!m, \mathsf{q}_1')$. Furthermore, e_1 belongs also to T', hence there are $2M+1$ processes

in node n' and $2M+1$ processes in node n_1. We extend θ by choosing one process to perform the broadcast of message m from n_1 and the associated reception set consists of M processes with history $h(n')$. This results in sending this M processes in node n.

In all the cases, the obtained execution θ' respects σ and there are at least M processes in each node of the pattern in the last configuration of θ'. □

In order to state the relation between admissible strategy patterns and local strategies, we define $\mathsf{lab}(T) = \{\mathsf{lab}(n) \mid n \in N\}$ as the set of control states labeling nodes of T and $\mathsf{Reach}(\theta) = \{\gamma_i[p_{id}] \mid i \in [0..\ell+1] \text{ and } p_{id} \in [1..nbproc(\theta)]\}$ as the set of states that appear along an execution $\theta = \gamma_0 \to \cdots \to \gamma_{\ell+1}$. The next proposition tells us that admissible strategy patterns are necessary and sufficient to represent the sets of states that can be reached under local strategies.

Proposition 3.1 *For all* $\mathrm{Q}' \subseteq \mathrm{Q}$*, there exists an admissible strategy pattern* $(T, \prec)$ *such that* $\mathsf{lab}(T) = \mathrm{Q}'$ *if and only if there exist a local strategy* σ *and an execution* θ *such that* θ *respects* σ *and* $\mathrm{Q}' = \mathsf{Reach}(\theta)$*. Furthermore,* σ *follows* T*.*

Proof The first direction is a direct consequence of Lemma 3.2, taking *e.g.* $M = 1$.

To prove the second direction we suppose that there exist a local strategy σ and an execution θ such that θ respects σ and $\mathrm{Q}' = \mathsf{Reach}(\theta)$. We let $\theta = \gamma_0 \xrightarrow{p_{id_0},\delta_0,R_0} \ldots \xrightarrow{p_{id_\ell},\delta_\ell,R_\ell} \gamma_{\ell+1}$. We will explain how to build an admissible strategy pattern $(T, \prec)$ such that $\mathsf{lab}(T) = \mathrm{Q}'$ from this execution. In the sequel, for every $i \in [1..\ell+1]$, we denote by θ_i the prefix execution $\gamma_0 \xrightarrow{p_{id_0},\delta_0,R_0} \ldots \xrightarrow{p_{id_{i-1}},\delta_{i-1},R_{i-1}} \gamma_i$ consisting of the i first transitions in θ.

We provide now the definition of a function *admtree* which, given a prefix θ_i of the execution θ, returns an admissible strategy pattern $(T, \prec)$ that satisfies that for all $p_{id} \in [1 \ldots nbproc(\theta)]$ there exists a node n such that $h(n) = \pi_{p_{id}}(\theta_i)$. The idea is to build an admissible strategy pattern where the labeled paths characterize all possible pasts of the different processes involved in θ.

We proceed inductively as follows: $admtree(\gamma_0) = ((\{n_0\}, n_0, \emptyset, \Delta, \mathsf{lab}), \prec)$ with $\prec = \emptyset$ and $\mathsf{lab}(n_0) = \mathsf{q}_0$. For all $i \in [1..\ell+1]$, if $admtree(\theta_{i-1}) = (T, \prec)$ with $T = (N, n_0, E, \Delta, \mathsf{lab})$ then $admtree(\theta_i) = (T', \prec')$ where $T' = (N', n_0, E', \Delta, \mathsf{lab}')$ is obtained by completing T according to the following case analysis:

- if $\delta_{i-1} = (\gamma_{i-1}[p_{id_{i-1}}], \varepsilon, \gamma_i[p_{id_{i-1}}])$ is an internal action and there does not exist a node n in T such that $\pi_{p_{id_{i-1}}}(\theta_i) = h(n)$, then let n' be the node in T such that $\pi_{p_{id_{i-1}}}(\theta_{i-1}) = h(n')$ (such a node necessarily exists by definition of *admtree*). In that case, we add a new node n to T and we define $\mathsf{lab}'(n', n) = \delta_{i-1}$ and $\prec'$ is obtained from $\prec$ by defining n as the new maximal node.

- if $\delta_{i-1} = (\gamma_{i-1}[p_{id_{i-1}}], !!m, \gamma_i[p_{id_{i-1}}])$, then,

- first, if there does not exist a node n in T such that $\pi_{p_{id}i-1}(\theta_i) = h(n)$, then let n' be the node in T such that $\pi_{p_{id}i-1}(\theta_{i-1}) = h(n')$ (such a node necessarily exists by definition of *admtree*). In that case, we add a new node n to T and we define $\mathsf{lab}'(n', n) = \delta_{i-1}$ and $\prec'$ is obtained from $\prec$ by defining n as the new maximal node.
- afterwards for every $p_{id} \in R_{i-1}$ such that there does not exist a node n in T verifying $\pi_{p_{id}}(\theta_i) = h(n)$, let n' be the node in T such that $\pi_{p_{id}i-1}(\theta_{i-1}) = h(n')$. Then, we add a new node n to N', and lab' is extended such that $\mathsf{lab}'(n', n) = (\gamma_{i-1}[p_{id}], ??m, \gamma_i[p_{id}])$ and we extend $\prec'$ such that n is the new maximal of the order $\prec'$. Note that, in that case, it is important that the destination node of the broadcast is smaller (with relation to $\prec'$) to the destination nodes of the performed receptions, but the order between these latter nodes is not relevant.

Note that if $\mathit{admtree}(\theta) = (T, \prec)$, then T is indeed a strategy pattern. The reason is that θ respects the local strategy σ, hence each path in $\mathcal{P}$ is associated via σ to a unique active action and a unique possible reception per message m. Furthermore, the fact that $\mathit{admtree}(\theta)$ is admissible follows directly from the inductive definition of the order. In fact, condition *(1)* of admissible strategy patterns is verified since we add each time maximal nodes at the end of existing paths, and condition *(2)* is verified because each destination node of a reception is larger according to $\prec$ than a destination node of a matching broadcast. Finally, $\mathsf{lab}(T) = \mathrm{Q}'$ since the labels of the nodes in T correspond exactly to all the control states seen in θ. It is furthermore clear by construction that σ follows T, given that T is built using the choices given by σ in the execution θ. □

3.1.3 Minimizing admissible strategy patterns

In the previous section we have shown that strategy patterns are an adequate tool to talk about local executions. Indeed it was shown in Proposition 3.1 that admissible strategy patterns are necessary and sufficient to represent the sets of states that can be reached under local strategies. This section is dedicated to show that, in fact, strategy patterns are also an efficient tool. Indeed, we show that strategy patterns can be reduced to a polynomial size. Hence, it is enough to focus on polynomial size strategy patterns to have an answer on the parameterized reachability problems.

For $(T, \prec)$ an admissible strategy pattern, we denote by $\mathsf{last}(T, \prec)$ the maximal node w.r.t. $\prec$ and we say that $(T, \prec)$ is q_F*-admissible* if $\mathsf{lab}(\mathsf{last}(T, \prec)) = \mathsf{q}_F$. We now show that there exist polynomial size witnesses of q_F-admissible strategy patterns. The idea is to keep only relevant edges that either lead to a node labeled by q_F or that permit the broadcast of a new message *i.e.* a message that does not appear in the pattern so far. Intuitively, a *minimal* strategy pattern guarantees that (1) there is a unique node labeled with q_F, (2) in every subtree there is either a node labeled by q_F or a broadcast of a new message, and (3) a path starting and ending in two different nodes labeled by the same state cannot be compressed without losing a new broadcast or a path towards q_F. By compressing, we mean here replacing the first node on the path by the last one.

We now formalize these ideas. Given a q_F-admissible pattern $(T, \prec)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$, we denote by $\mathsf{NewBroad}(T, \prec) \subseteq N \setminus \{n_0\}$ the set of nodes that are sources of a new message broadcast. Formally, $n \in \mathsf{NewBroad}(T, \prec)$ if and only if $\mathsf{lab}(\mathsf{pred}(n)) = (\mathsf{q}, !!m, \mathsf{q}')$ and for all $n' \in N \setminus \{n_0, n\}$ such that $\mathsf{lab}(\mathsf{pred}(n')) = (\mathsf{q}'', !!m, \mathsf{q}''')$, we have $n \prec n'$. Also, we denote by $\mathsf{Imp}(T, \prec)$ the set of "important" nodes corresponding to $\mathsf{NewBroad}(T, \prec) \cup \{\mathsf{last}(T, \prec)\}$, *i.e.* the new broadcast and the last node labeled by q_F.

Definition 3.4 (Minimal q_F-admissible strategy patterns) *We say that a q_F-admissible strategy pattern $(T, \prec)$, where $T = (N, n_0, E, \Delta, \mathsf{lab})$, is* minimal *if the following conditions are fulfilled:*

- ***(a)*** *for all $n \in N$, if $\mathsf{lab}(n) = \mathsf{q}_F$ then $n = \mathsf{last}(T, \prec)$;*
- ***(b)*** *for all $n \in N$, if $\mathsf{Sub}(T, n) = (N', n, E', \Delta, \mathsf{lab}')$ then $N' \cap \mathsf{Imp}(T, \prec) \neq \emptyset$;*
- ***(c)*** *for all pairs of different nodes $n', n'' \in N$ such that $\mathsf{lab}(n') = \mathsf{lab}(n'')$ and $n' \neq n''$, if $\mathsf{Sub}(T, n') = (N', n', E', \Delta, \mathsf{lab}')$ and $\mathsf{Sub}(T, n'') = (N'', n'', E'', \Delta, \mathsf{lab}'')$ then $(N' \setminus \{n'\}) \cap \mathsf{Imp}(T, \prec) \neq N'' \cap \mathsf{Imp}(T, \prec)$.*

Intuitively, condition **(a)** states that there is a unique node labeled with q_F and it is the last according to $\prec$; condition **(b)** expresses that in every subtree there should be a node labeled by q_F or a new broadcast message; and condition **(c)** ensures that if two different subtrees have their root labeled by the same state, then there should be at least a new broadcast or the last state present in one of the subtrees and not in the other one. The intuition for this last condition is basically that if there is a smaller subtree included in a subtree such that all the important nodes are in the smaller subtree, we can replace the bigger subtree by the smaller one without loosing anything.

Lemma 3.3 *If there exists a q_F-admissible strategy pattern for $\mathcal{P}$, then there exists a minimal one.*

Proof Let $(T, \prec)$ with $T = (N, n_0, E, \Delta, \mathsf{lab})$ be a q_F-admissible strategy pattern and assume that $(T, \prec)$ is not minimal.

First, we suppose that there exists a node $n \in N$ such that $\mathsf{lab}(n) = \mathsf{q}_F$ and $n \neq \mathsf{last}(T, \prec)$. Then, let $n' \in N$ be the minimal node labeled by q_F. Formally n' is such that $\mathsf{lab}(n') = \mathsf{q}_F$ and for every node $n \in N \setminus \{n'\}$ verifying $\mathsf{lab}(n) = \mathsf{q}_F$, we have $n' \prec n$. In that case, we remove from T all the nodes n such that $n' \prec n$ and $n \neq n'$ and their associated edges . We obtain a q_F-admissible strategy pattern for $\mathcal{P}$. Indeed, thanks to the condition *(1)* respected by $\prec$, we know that the transformation preserves the tree structure and furthermore, since we only remove nodes bigger than n, we know that condition *(2)* of $\prec$ is still satisfied. Finally, it is clear that the obtained admissible strategy pattern satisfies condition **(a)**.

Now we assume that condition **(a)** is satisfied by $(T, \prec)$ and we suppose that there exists a node $n \in N$ such that $\mathsf{Sub}(T, n) = (N', n, E', \Delta, \mathsf{lab}')$ and $N' \cap \mathsf{Imp}(T, \prec) = \emptyset$. The transformation to get a pattern satisfying condition **(b)** is easy: for every node n

such that $\mathsf{Sub}(T, n) = (N', n, E', \Delta, \mathsf{lab}')$ and $N' \cap \mathsf{Imp}(T, \prec) = \emptyset$, we remove from T all nodes in N' and their associated edges leading to that nodes. Since we remove subtrees, the obtained structure is still a strategy pattern. Also since the important nodes consists in new broadcasts and the last node, $\mathsf{Imp}(T, \prec) = \mathsf{NewBroad}(T, \prec) \cup \{\mathsf{last}(T, \prec)\}$, this allows us to deduce that this strategy pattern with the restriction of $\prec$ to the remaining nodes is still a q_F-admissible strategy pattern. Indeed, condition *(1)* for admissibility is trivially satisfied. Moreover, since we keep for each message m the minimal node associated with the broadcast of this message, then condition *(2)* is also satisfied. Note finally that the obtained admissible strategy pattern satisfies the conditions **(a)** and **(b)**.

We assume now that conditions **(a)** and **(b)** are satisfied by $(T, \prec)$ and that the condition **(c)** is not satisfied. Until condition **(c)** is satisfied, we perform the following operations: first assume there are two nodes $n', n'' \in N$ such that $\mathsf{lab}(n') = \mathsf{lab}(n'')$ and $n' \neq n''$ with $\mathsf{Sub}(T, n') = (N', n', E', \Delta, \mathsf{lab}')$ and $\mathsf{Sub}(T, n'') = (N'', n'', E'', \Delta, \mathsf{lab}'')$ and $N' \cap \mathsf{Imp}(T, \prec) = N'' \cap \mathsf{Imp}(T, \prec)$. Then necessarily either $N' \subset N''$ or $N'' \subset N'$ because they are subtrees rooted at some node. Suppose, without loss of generality, that the second case holds, *i.e.*, that $\mathsf{Sub}(T, n'')$ is a subtree of $\mathsf{Sub}(T, n')$. Since $N' \cap \mathsf{Imp}(T, \prec) = N'' \cap \mathsf{Imp}(T, \prec)$, important nodes matter (as we have seen with the previous case) and $\mathsf{lab}(n') = \mathsf{lab}(n'')$, we can replace in T the subtree $\mathsf{Sub}(T, n')$ by its subtree $\mathsf{Sub}(T, n'')$ (and doing so, remove from T the nodes in $N' \setminus N''$). The obtained structure is still a q_F-admissible strategy pattern. In fact it is a strategy tree pattern because $\mathsf{lab}(n') = \mathsf{lab}(n'')$, and it is still q_F-admissible because $\mathsf{Sub}(T, n'')$ is a subtree of $\mathsf{Sub}(T, n')$ and because we did not remove any important node. Repeating this operation allows us to finally get a q_F-admissible strategy pattern which respects conditions **(a)**, **(b)** and **(c)** and hence which is minimal. □

Lemma 3.3 allows us to seek only for minimal q_F-admissible strategy patterns. We show in the following proposition that the size of such a minimal pattern is at most polynomial in the size of the protocol.

Proposition 3.2 *If there exists a q_F-admissible strategy pattern for $\mathcal{P}$, then there is one of size at most $(2|\Sigma| + 1) \cdot (|\mathrm{Q}| - 1)$ and of height at most $(|\Sigma| + 1) \cdot |\mathrm{Q}|$.*

Proof Given a strategy pattern T, we call *intersection node*, a node n in T from which at least two actions are defined (either two different receptions or a reception and an active action). We gather under the term *noticeable* nodes, the nodes that are important nodes or intersection nodes.

Thanks to Lemma 3.3, to establish Proposition 3.2, it suffices to bound the size and height of minimal q_F-admissible strategy patterns. Let $(T, \prec)$ be a minimal q_F-admissible strategy pattern for $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$, where $T = (N, n_0, E, \Delta, \mathsf{lab})$. First, from condition **(b)**, there are at most $|\mathsf{Imp}(T, \prec)| - 1 \leq |\Sigma|$ intersection nodes. Otherwise, there would be a subtree that does not contain any important node. Moreover, there are at most $|\Sigma| + 1$ important nodes: $|\Sigma|$ for the messages, and 1 for q_F. Therefore, there are at most $2|\Sigma| + 1$ noticeable nodes. Second, from conditions **(a)** and

(c), we know that there are no more than $|\mathrm{Q}| - 2$ nodes between two noticeable nodes. Otherwise, there would be two nodes with the same label that share the same set of important nodes, or there would be a node labeled with q_F. We thus derive the desired bound on the size of minimal q_F-admissible strategy patterns.

We also obtain a bound on the height of minimal q_F-admissible strategy patterns: in the worst case, all important nodes belong to the same branch of T. We conclude by recalling that between two important nodes there are at most $|\mathrm{Q}| - 2$ nodes, and that the number of important nodes is bounded by $|\Sigma| + 1$ (one per message type, plus the final state). □

By Proposition 3.1, there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $\mathsf{q}_F \in \mathsf{Reach}(\theta)$ if and only if there exists a q_F-admissible strategy pattern. Thanks to Proposition 3.2, it suffices to look only for q_F-admissible strategy patterns of polynomial size in the size of the broadcast protocol. A non-deterministic polynomial time algorithm for Reach[$\mathcal{L}$] then consists in guessing a strategy pattern of polynomial size and an order, and then verifying whether it is q_F-admissible.

Theorem 3.1 Reach[$\mathcal{L}$] *is in NP.*

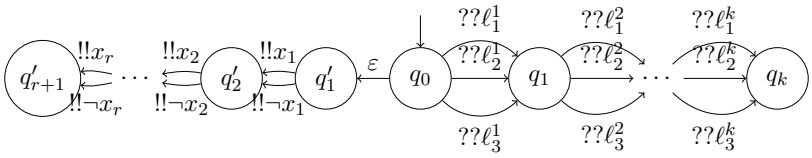


Figure 3.3: Encoding a 3-SAT formula into a broadcast protocol.

By reducing 3-SAT, one can furthermore prove Reach[$\mathcal{L}$] to be NP-hard. Let $\phi = \bigwedge_{1 \leq i \leq k} (\ell_1^i \vee \ell_2^i \vee \ell_3^i)$ be a 3-SAT formula such that $\ell_j^i \in \{x_1, \neg x_1, \ldots, x_r, \neg x_r\}$ for all $i \in [\![1..k]\!]$ and $j \in \{1, 2, 3\}$. We build from ϕ the broadcast protocol $\mathcal{P}$ depicted on Figure 3.3. Under this construction, ϕ is satisfiable if and only if there is an execution $\theta \in \Theta_{\mathcal{L}}$ such that $\mathsf{q}_k \in \mathsf{Reach}(\theta)$. The local strategy hypothesis ensures that even if several processes broadcast a message corresponding to the same variable, all of them must take the same decision so that there cannot be any execution during which both x_i and $\neg x_i$ are broadcast. It is then clear that control state q_k can be reached if and only if each clause is satisfied by the set of broadcast messages. Together with Theorem 3.1, we obtain the precise complexity of Reach[$\mathcal{L}$].

Theorem 3.2 Reach[$\mathcal{L}$] *is NP-complete.*

We can furthermore provide bounds on the minimal number of processes and on the memory needed to implement local strategies. Given a q_F-admissible strategy pattern, one can define an execution following the pattern such that each reception edge of the pattern is taken exactly once and active actions may be taken multiple times but in a row. Such an execution needs at most one process per reception edge. Together with

the bound on the size of the minimal strategy patterns (see Proposition 3.2), this yields a cutoff property on the minimal size of network to reach the final state. Moreover, the past history of every process in this execution is bounded by the depth of the tree, hence we obtain an upper bound on the size of the memory needed by each process for $\textsc{Reach}[\mathcal{L}]$.

Proposition 3.3 *If there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $\mathsf{q}_F \in \mathsf{Reach}(\theta)$, then there exists an execution $\theta' \in \Theta_{\mathcal{L}}$ such that $\mathsf{q}_F \in \mathsf{Reach}(\theta')$ and $nbproc(\theta') \leq (2|\Sigma|+1) \cdot (|\mathrm{Q}|-1)$ and $|\pi_{p_{id}}(\theta')| \leq (|\Sigma|+1) \cdot |\mathrm{Q}|$ for every $p_{id} \in [1..nbproc(\theta')]$.*

The proof of Proposition 3.3 calls for the introduction of the following notations. Given an execution θ, a process p_{id} and a path $\rho \in \mathsf{Path}(\mathcal{P})$, we consider the execution built from θ in which there is an additional process $p_{id\,add}$ that behaves exactly as process p_{id} until its history is ρ. Formally, $follow(\theta, p_{id}, \rho)$ is defined inductively: $follow(\gamma_0, p_{id}, \rho) = \gamma_0'$ with $|\gamma_0'| = |\gamma_0| + 1$ and if $\pi_{p_{id\,add}}(follow(\theta, p_{id}, \rho)) = \rho$ then $follow(\theta \xrightarrow{p'_{id},\delta,R} \gamma, p_{id}, \rho) = follow(\theta, p_{id}, \rho) \xrightarrow{p'_{id},\delta,R} \gamma'$ where $\gamma'(p''_{id}) = \gamma(p''_{id})$ for every $p''_{id} \neq p_{id\,add}$, otherwise

$$follow(\theta \xrightarrow{p'_{id},\delta,R} \gamma, p_{id}, \rho) = \begin{cases} \theta' \xrightarrow{p'_{id},\delta,R} \gamma' & \text{if } p'_{id} \neq p_{id} \text{ and } p_{id} \notin R \\ \theta' \xrightarrow{p'_{id},\delta,R\cup\{p_{id\,add}\}} \gamma' & \text{if } p'_{id} \neq p_{id} \text{ and } p_{id} \in R \\ \theta' \xrightarrow{p_{id},\delta,R} \gamma_{int} \xrightarrow{p_{id\,add},\delta,\emptyset} \gamma' & \text{if } p_{id} = p'_{id} \end{cases}$$

with $\theta' = follow(\theta, p_{id}, \rho)$, $\gamma'(p_{id\,add}) = \gamma(p_{id})$ and $\forall p''_{id} \neq p_{id\,add}$, $\gamma'(p''_{id}) = \gamma(p''_{id})$.

Notice that *follow* refines *copycat* defined page 139, by limiting the copycat behavior to a given history.

Intuitively, in case $p_{id\,add}$ already reached its destination (*i.e.* has history ρ), the execution continues exactly as θ. Otherwise, in case of a broadcast by another process than p_{id}, the new process $p_{id\,add}$ behaves as p_{id}, *i.e.* receives the message in θ' if and only if p_{id} receives it in θ. Last, if p_{id} is responsible for the active action, $p_{id\,add}$ performs exactly the same active action, yet the associated reception set is empty, so that execution θ can continue on the original processes. Note that in all cases, the inductive definition ensures that θ' exists and is unique, so that $follow(\theta, p_{id}, \rho)$ is well defined. Moreover, assuming θ is a local execution, since $p_{id\,add}$ "mimics" the process p_{id}, $follow(\theta, p_{id}, \rho)$ is a local execution too, and $\pi_{p_{id\,add}}(follow(\theta, p_{id}, \rho)) = \pi_{p_{id}}(\theta)$ or $\pi_{p_{id\,add}}(follow(\theta, p_{id}, \rho)) = \rho$ (and then $\pi_{p_{id\,add}}(follow(\theta, p_{id}, \rho))$ is a prefix of $\pi_{p_{id}}(\theta)$).

Let $T = (N, n_0, E, \Delta, \mathsf{lab})$ be a strategy pattern. Given a subset of nodes $N_1 \subseteq N$, we define the restriction of T to the predecessors of nodes in N_1; formally $T_{\downarrow N_1} = (N', n_0, E', \Delta, \mathsf{lab})$ where $N' = \{n \in N \mid \exists n' \in desc(n, T) \cap N_1\}$ and $E' = E \cap (N' \times N')$. Assuming T is equipped with an order $\prec$, for any $k \in [1..|\mathsf{Imp}(T, \prec)|]$ the set of the k first important nodes with respect to $\prec$ is denoted $\mathsf{Imp}(T, \prec, k)$. Formally, we have $\mathsf{Imp}(T, \prec, k) \subseteq \mathsf{Imp}(T, \prec)$ and $|\mathsf{Imp}(T, \prec, k)| = k$ and if $n \in \mathsf{Imp}(T, \prec, k)$ and $n' \in \mathsf{Imp}(T, \prec)$ with $n' \prec n$ then $n' \in \mathsf{Imp}(T, \prec, k)$. The following lemma bounds the number of processes

needed to reach a configuration with processes in each of the nodes of $\mathsf{Imp}(T, \prec, k)$, by the number of predecessors of these nodes.

Recall that $h(n) \in \mathsf{Path}(\mathcal{P})$ is the labeling of the path in T from the root to node n.

Lemma 3.4 *Let $(T, \prec)$ be an admissible strategy pattern and $k \in [1..|\mathsf{Imp}(T, \prec)|]$. Then there exists an execution $\theta \in \Theta_{\mathcal{L}}$ with the following properties:*

1. $\{\pi_{p_{id}}(\theta) \mid p_{id} \in [1..nbproc(\theta)]\} = \{h(n) \mid n \in \mathsf{Imp}(T, \prec, k)\}$
2. $nbproc(\theta) \leq |T_{\downarrow \mathsf{Imp}(T, \prec, k)}|$.

Proof The proof is by induction on k. For the base case $k = 1$, by definition of admissibility, the first important node is reachable only via active actions. Therefore, we can define θ as the execution with a single process p_{id} such that $\pi_{p_{id}}(\theta) = h(n_1)$ where n_1 is the first important node.

Assume now that the lemma holds for $k-1$. Let $\theta_{k-1} \in \Theta_{\mathcal{L}}$ be an execution such that $\{\pi_{p_{id}}(\theta_{k-1}) \mid p_{id} \in [1..nbproc(\theta_{k-1})]\} = \{h(n) \mid n \in \mathsf{Imp}(T, \prec, k-1)\}$ and $nbproc(\theta_{k-1}) \leq |T_{\downarrow \mathsf{Imp}(T, \prec, k-1)}|$. Intuitively, in order to build θ_k that proves the induction step from θ_{k-1}, one additional process will follow some process of θ_{k-1} until it meets some node n on the way to n_k the k-th important node of T. Then the additional process $p_{id\,add}$ will aim at reaching n_k from n. To do so, since there might be reception steps between n and n_k, some more processes that will broadcast the corresponding messages are needed. This will work smoothly by using other additional processes, since the needed broadcast precisely leads to important nodes in $\mathsf{Imp}(T, \prec, k-1)$. These additional processes will be defined thanks to an auxiliary function fill that we define now.

We consider a local execution θ such that for every node $n \in \mathsf{Imp}(T, \prec, k-1)$, there exists a process $p_{id} \in [1..nbproc(\theta)]$ with $\pi_{p_{id}}(\theta) = h(n)$. Moreover, for $p_{id} \in [1..nbproc(\theta)]$ and for a sequence of edges $e_1 \dots e_l$ in T such that $\pi_{p_{id}}(\theta) = h(\mathsf{src}(e_1))$, we define a function $\mathsf{fill}(\theta, e_1 \dots e_l, p_{id})$ inductively as follows: $\mathsf{fill}(\theta, \epsilon, p_{id}) = \theta$ and otherwise

- if $\mathsf{lab}(e_1) = \delta$ is an active action, we let $\mathsf{fill}(\theta, e_1 \dots e_l, p_{id}) = \mathsf{fill}(\theta', e_2 \dots e_l, p_{id})$ with $\theta' = \theta \xrightarrow{p_{id}, \delta, \emptyset} \gamma$; *i.e.* the process p_{id} performs the active action with an empty reception set. Notice that $nbproc(\theta') = nbproc(\theta)$, and for every $p'_{id} \in [1..nbproc(\theta)] \setminus \{p_{id}\}, \pi_{p'_{id}}(\theta') = \pi_{p'_{id}}(\theta)$.
- if $\mathsf{lab}(e_1) = \delta$ is a reception of message m, we consider $e_m = (n_1, n_2)$ such that $\mathsf{lab}(e_m)$ is a broadcast of m and such that $n_2 \in \mathsf{Imp}(T, \prec, k-1)$. Such an edge exists because T is admissible. By assumption on θ, there exists $p'_{id} \in [1..nbproc(\theta)]$ such that $\pi_{p'_{id}}(\theta) = h(n_2)$. We consider the execution $follow(\theta, p'_{id}, h(n_1))$, and write p_{id_m} for the additional process in that execution compared to θ. Then, we let $\theta' = follow(\theta, p'_{id}, h(n_1)) \xrightarrow{p_{id_m}, \mathsf{lab}(e_m), \{p_{id}\}} \gamma$, and finally $\mathsf{fill}(\theta, e_1 \dots e_l, p_{id}) = \mathsf{fill}(\theta', e_2 \dots e_l, p_{id})$ *i.e.* the additional process p_{id_m} broadcasts message m to

p_{id} only. Notice that $nbproc(\theta') = nbproc(\theta) + 1$, $\pi_{p_{id_m}}(\theta) = h(n_2)$, for every $p'_{id} \in [1..nbproc(\theta_1)] \setminus \{p_{id}\}$, $\pi_{p'_{id}}(\theta') = \pi_{p'_{id}}(\theta)$, and $n_2 \in \mathsf{Imp}(T, \prec, k-1)$.

To conclude the induction step of the proof, we now explain how to obtain θ_k applying fill to θ_{k-1}. Let n_k be the k-th important node in T with respect to $\prec$. Let n be the last node appearing on the path to n_k and such that n is visited by some process p_{id1} along θ_{k-1}. We write $e_1 \dots e_l$ for the sequence of edges between n and n_k and consider $\theta' = follow(\theta_{k-1}, p_{id1}, h(n))$ the execution in which an additional process goes to node n, and we denote by p'_{id} this process. The situation is illustrated on Figure 3.4 where important nodes are circled.

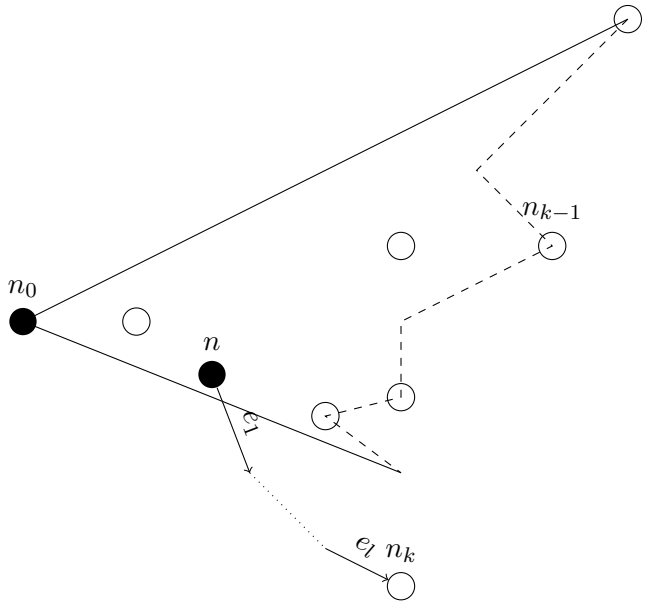


Figure 3.4: Illustration for the construction of θ_k from θ_{k-1}.

By induction hypothesis on θ_{k-1}, we know that for every node $n \in \mathsf{Imp}(T, \prec, k-1)$, there exists $p_{id} \in [1..nbproc(\theta_{k-1})]$ such that $\pi_{p_{id}}(\theta_{k-1}) = h(n)$. This also holds for the execution θ' extended with the additional process p'_{id}. As a consequence, we can apply fill to θ' for the sequence of edges $e_1 \dots e_l$ and the process p'_{id}. The resulting execution $\theta'' = \mathsf{fill}(\theta', e_1 \dots e_l, p'_{id})$ satisfies that for every $n \in \mathsf{Imp}(T, \prec, k-1)$ there exists $p_{id} \in [1..nbproc(\theta'')]$ with $\pi_{p_{id}}(\theta'') = h(n)$, and since the process p_{id} fired the sequence of edges $e_1 \dots e_l$, θ'' moreover satisfies $\pi_{p_{id}}(\theta'') = h(n_k)$. By definition of fill, all additional processes end in important nodes, so that $\{\pi_{p_{id}}(\theta'') \mid p_{id} \in [1..nbproc(\theta'')]\} = \{h(n) \mid n \in \mathsf{Imp}(T, \prec, k)\}$. Last, applying fill adds at most one process per edge of the sequence $e_1 \dots e_l$ from n to the k-th important node n_k. As a consequence, we obtain the following bound on the number of processes: $nbproc(\theta'') \leq |T_{\downarrow \mathsf{Imp}(T,\prec,k-1)}| + l = |T_{\downarrow \mathsf{Imp}(T,\prec,k)}|$. □

Lemma 3.4 combined to Proposition 3.2 implies Proposition 3.3, therefore we were able to show a bound on the minimal number of processes and on the memory needed to implement local strategies.

3.2 Solving Synch[$\mathcal{L}$]

Admissible strategy patterns can also be used to obtain an NP-algorithm for Synch[$\mathcal{L}$]. As we have seen, given an admissible strategy pattern, one can build an execution where the processes visit all the control states present in the pattern. When considering the synchronization problem, one also needs to ensure that the processes can afterwards be directed to the target set. To guarantee this, it is possible to extend admissible strategy patterns with another order on the nodes which ensures that (a) from any node there exists a path leading to the target set and (b) whenever on this path a reception is performed, the corresponding message can be broadcast by a process that will only later on be able to reach the target.

We formalize this idea now.

Definition 3.5 ($\mathfrak{T}$-coadmissible strategy patterns) *For $\mathfrak{T} \subseteq \mathrm{Q}$ a set of states, a* $\mathfrak{T}$-coadmissible strategy pattern *for $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ is a pair $(T, \lhd)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$ is a strategy pattern for $\mathcal{P}$ and $\lhd \subseteq N \times N$ is a strict total order on the nodes T such that, for every node $n \in N$ with $\mathsf{lab}(n) \notin \mathfrak{T}$, there exists an edge $e = (n, n') \in E$ with $n \lhd n'$, and either:*

- $\mathsf{lab}(e) = (\mathsf{lab}(n), \varepsilon, \mathsf{lab}(n'))$ *or,*
- $\mathsf{lab}(e) = (\mathsf{lab}(n), !!m, \mathsf{lab}(n'))$ *for some $m \in \Sigma$ or,*
- $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ *for some $m \in \Sigma$ and there exists an edge $e_1 = (n_1, n_1') \in E$ such that $n \lhd n_1$, $n \lhd n_1'$ and $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n_1'))$.*

Intuitively, the order $\lhd$ in a $\mathfrak{T}$-coadmissible strategy pattern corresponds to the order in which processes must move along the tree towards the target; the conditions express that any node with label not in $\mathfrak{T}$ has an outgoing edge that is feasible. In particular, a reception of m is only feasible before all edges carrying the corresponding broadcast are disabled.

When convenient, in order to manipulate the order $\lhd$ more easily, we will equivalently use an injective rank function $rk_{\lhd} : N \to \mathbb{Z}$ such that, for every pair of nodes (n, n'), we have $rk_{\lhd}(n) < rk_{\lhd}(n')$ if and only if $n \lhd n'$.

A strategy pattern T equipped with two orderings $\prec$ and $\lhd$ is said to be $\mathfrak{T}$-*biadmissible* whenever $(T, \prec)$ is admissible and $(T, \lhd)$ is $\mathfrak{T}$-coadmissible.

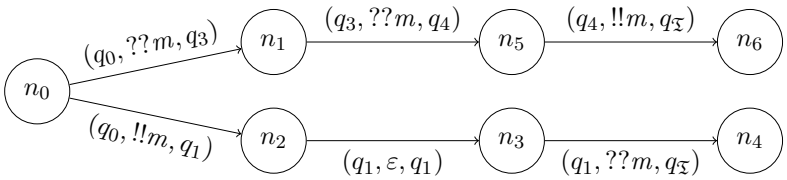


Figure 3.5: A $\mathfrak{T}$-coadmissible strategy pattern on the example protocol of Figure 2.1.

Example 3.3 *To illustrate the notion of $\mathfrak{T}$-coadmissible patterns, we give in Figure 3.5 an example pattern, that, equipped with the natural order $n_i \lhd n_j$ if and only if $i <$*

j, is $\mathfrak{T}$-coadmissible for $\mathfrak{T} = \{q_{\mathfrak{T}}\}$. Indeed, all leaves are labeled with a target state, and the broadcast edge $n_5 \xrightarrow{(q_4,!!m,q_{\mathfrak{T}})} n_6$ allows all processes to take the corresponding reception edges. This $\mathfrak{T}$-coadmissible pattern is in particular obtained from the execution $(q_0,q_0,q_0) \to (q_1,q_3,q_0) \to (q_1,q_3,q_0) \to (q_{\mathfrak{T}},q_4,q_1) \to (q_{\mathfrak{T}},q_4,q_1) \to (q_{\mathfrak{T}},q_{\mathfrak{T}},q_{\mathfrak{T}})$. Notice that $\lhd$ is not an admissible order, because $n_1 \lhd n_2$, and in an admissible order, a broadcast must precede its receptions. However there are admissible orders for this pattern, for example the order $n_0 \prec n_2 \prec n_3 \prec n_4 \prec n_1 \prec n_5 \prec n_6$. Under these definitions, $(T,\prec,\lhd)$ is $\mathfrak{T}$-biadmissible.

Similarly to Lemma 3.1 for admissible patterns, we can establish the following:

Lemma 3.5 *Given a strategy pattern $T = (N,n_0,E,\Delta,\mathsf{lab})$ for a broadcast protocol $\mathcal{P}$, a set of states $\mathfrak{T}$ and a strict total order $\lhd \subseteq N \times N$, checking whether $(T,\lhd)$ is $\mathfrak{T}$-coadmissible can be done in polynomial time.*

As for Reach[$\mathcal{L}$], one can show that polynomial size witnesses of $\mathfrak{T}$-*biadmissible* strategy patterns exist, yielding an NP-algorithm for Synch[$\mathcal{L}$]. Also, the size of minimal $\mathfrak{T}$-*biadmissible* strategy patterns gives here also a cutoff on the number of processes needed to satisfy the target objective, as well as an upper bound on the memory size.

3.3 Link between biadmissibility and local executions

The relation between biadmissible strategy patterns and local strategies satisfying a target objective is stated in the next proposition.

Proposition 3.4 *There exists a $\mathfrak{T}$-biadmissible pattern $(T,\prec,\lhd)$ if and only if there exist a local strategy σ and an execution θ such that θ respects σ and $\mathsf{End}(\theta) \subseteq \mathfrak{T}$; furthermore σ follows T.*

Proof First, we suppose that there exist a local strategy σ and an execution θ such that θ respects σ and $\mathsf{End}(\theta) \subseteq \mathfrak{T}$. We write $\theta = \gamma_0 \xrightarrow{p_{id0},\delta_0,R_0} \dots \xrightarrow{p_{id\ell},\delta_\ell,R_\ell} \gamma_{\ell+1}$. Let us define a function *coadmorder* which, given a prefix of the execution θ, returns an order $\lhd$ on the nodes of $admtree(\theta)$, where *admtree* is the function returning a strategy pattern and an admissible order corresponding to a local execution (See Proof of Proposition 3.1 for the formal definition). The idea of the order $coadmorder(\theta)$ is that if $n \lhd n'$ then, the last time some processes are in node n, during θ, happens before the last time some processes are in n'. Let $admtree(\theta) = (T,\prec)$ with $T = (N,n_0,E,\Delta,\mathsf{lab})$. The coadmissible order is defined inductively as follows, denoting θ_i the prefix of θ of size i : $coadmorder(\gamma_0) = \emptyset$, and for all $i \in [1..\ell+1]$, if $coadmorder(\theta_{i-1}) = \lhd_{i-1}$, $coadmorder(\theta_i) = \lhd_i$ is obtained by completing $\lhd_{i-1}$ according to the following case analysis:

- if $\delta_{i-1} = (\gamma_{i-1}[p_{id_{i-1}}],\varepsilon,\gamma_i[p_{id_{i-1}}])$, by definition of *admtree*, there exist two nodes n and n' in T such that $\pi_{p_{id_{i-1}}}(\theta_{i-1}) = h(n)$ and $\pi_{p_{id_{i-1}}}(\theta_i) = h(n')$. We obtain $coadmorder(\theta_i)$ by defining n as the second maximal node and n' as the maximal node for $\lhd_i$. Formally, $coadmorder(\theta_i) = \lhd_{i-1} \setminus (\{n,n'\} \times N \cup N \times \{n,n'\}) \cup \{(n_1,n) \mid n_1 \in N \setminus \{n'\}\} \cup \{(n_1,n') \mid n_1 \in N\}$.

- if $\delta_{i-1} = (\gamma_{i-1}[p_{id\,i-1}], !!m, \gamma_i[p_{id\,i-1}])$, then,
  - first, for every $p_{id} \in R_{i-1}$, by definition of *admtree*, there exist two nodes n_{id} and n'_{id} in T such that $\pi_{p_{id}}(\theta_{i-1}) = h(n_{id})$ and $\pi_{p_{id}}(\theta_i) = h(n'_{id})$. We obtain $\mathit{coadmorder}(\theta_i)$ by defining n_{id} as the second maximal node and n'_{id} as the maximal node for $\lhd_i$. Formally, $\mathit{coadmorder}(\theta_i) = \lhd_{i-1} \setminus (\{n_{id}, n'_{id}\} \times N \cup N \times \{n_{id}, n'_{id}\}) \cup \{(n_1, n_{id}) \mid n_1 \in N \setminus \{n'_{id}\}\} \cup \{(n_1, n'_{id}) \mid n_1 \in N\}$.
  - afterwards, we treat the process p_{id} responsible for the broadcast as in the case of the internal transition. That is: by definition of *admtree*, there exist two nodes n and n' in T such that $\pi_{p_{id}}(\theta_{i-1}) = h(n)$ and $\pi_{p_{id}}(\theta_i) = h(n')$. We obtain $\mathit{coadmorder}(\theta_i)$ by defining n as the second maximal node and n' as the maximal node. Formally, $\mathit{coadmorder}(\theta_i) = \lhd_{i-1} \setminus (\{n, n'\} \times N \cup N \times \{n, n'\}) \cup \{(n_1, n) \mid n_1 \in N \setminus \{n'\}\} \cup \{(n_1, n') \mid n_1 \in N\}$.

The fact that $(T, \mathit{coadmorder}(\theta))$ is $\mathfrak{T}$-coadmissible follows directly from the inductive definition of the order. In fact, the condition is verified since each time a transition is taken in the execution we make sure that it is possible to take it according to the order. And, since $\mathsf{End}(\theta) \subseteq \mathfrak{T}$, from all states that do not belong to $\mathfrak{T}$ there must be a position in the execution from which any state out of $\mathfrak{T}$ does no longer appear in the execution, yielding a desired outgoing edge.

We now show the other implication: if there exists a $\mathfrak{T}$-biadmissible strategy pattern $(T, \prec, \lhd)$ then there exists a local strategy σ following T and there exists an execution θ that respects σ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$. In order to relate more precisely coadmissible strategy patterns to local strategies, we define a partition of the nodes of T according to the position with respect to a given n: $\mathit{part}(T, \lhd, n) = (S, G)$ with $S = \{n' \mid n' \lhd n\}$ the set of all nodes smaller than n and $G = \{n' \mid n \lhd n'\} \cup \{n\}$ the set of all nodes greater than n. Our proof is then based on the following technical lemma.

Lemma 3.6 *Let $(T, \lhd)$ be a coadmissible strategy pattern, n a node of T with $\mathsf{lab}(n) \notin \mathfrak{T}$ and θ an execution such that, writing $(S, G) = \mathit{part}(T, \lhd, n)$ and $M = |G|$, there are more than M processes in each node of G at the end of θ. Then there exists an execution θ' that extends θ and such that the number of processes in each node of S does not change, the number of processes in each node of G is at least $M - 1$ and no processes are in node n anymore.*

Proof By definition of coadmissibility, there exists $e = (n, n') \in E$ such that $n \lhd n'$ and either:

- $\mathsf{lab}(e) = (\mathsf{lab}(n), \varepsilon, \mathsf{lab}(n'))$ or, $\mathsf{lab}(e) = (\mathsf{lab}(n), !!m, \mathsf{lab}(n'))$ or,
- $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ and there exists an edge $e_1 = (n_1, n'_1) \in E$ such that $n \lhd n_1$, $n \lhd n'_1$ and $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n'_1))$.

In the first case, one can extend θ into an execution θ' by considering all the processes such that $\pi_{p_{id}}(\theta) = h(n)$ and let each of them perform the active action $\mathsf{lab}(e)$ with an empty reception set. In the second case, one builds θ' by considering one process such

that $\pi_{p_{id}}(\theta) = h(n_1)$ and let it perform the broadcast of m with as reception set the set of all processes with $\pi_{p_{id}}(\theta) = h(n)$. In both cases, the processes in nodes of S are not concerned. At most one process that was in a node of G moved to an other node of G and all the processes in node n moved to a node of G yielding the desired properties on θ'. □

To conclude the proof of Proposition 3.4, we observe that given a $\mathfrak{T}$-biadmissible strategy pattern, by Lemma 3.2, there exists an execution with an arbitrary number of processes in each node at the end of the execution, we can thus consider an execution that has as many processes per node as the number of nodes. From this execution, applying Lemma 3.6 to every node by increasing order for $\lhd$, we obtain an execution for which the last configuration satisfies that all the processes are in a state belonging to $\mathfrak{T}$. □

3.4 Minimizing biadmissible strategy patterns

As for reachability, the size of $\mathfrak{T}$-biadmissible strategy patterns can be minimized by keeping only relevant edges that permit a broadcast of either a new message or the last message of this type. More formally, given a $\mathfrak{T}$-biadmissible pattern $(T, \prec, \lhd)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$, we denote as we did before for admissible patterns, $\mathsf{NewBroad}(T, \prec) \subseteq N \setminus \{n_0\}$ the set of nodes that are new broadcasts, that is, $n \in \mathsf{NewBroad}(T, \prec)$ if and only if $\mathsf{lab}(\mathsf{pred}(n)) = (\mathsf{q}, !!m, \mathsf{q}')$ and for all $n' \in N \setminus \{n_0\}$ such that $\mathsf{lab}(\mathsf{pred}(n')) = (\mathsf{q}'', !!m, \mathsf{q}''')$, we have $n \prec n'$. We further denote by $\mathsf{LastBroad}(T, \lhd) \subseteq N \setminus \{n_0\}$ the set of nodes that are last broadcasts, that is, $n \in \mathsf{LastBroad}(T, \lhd)$ if and only if $\mathsf{pred}(n) = (n', n) = e$, $n' \lhd n$, $\mathsf{lab}(e) = (\mathsf{q}, !!m, \mathsf{q}')$ and for all $e' = (n_1, n_1') \in E$ such that $n_1 \lhd n_1'$ and $\mathsf{lab}(e') = (\mathsf{q}'', !!m, \mathsf{q}''')$, we have $n_1 \lhd n'$. Finally, the set of important nodes is defined as $\mathsf{Imp}(T, \prec, \lhd) = \mathsf{NewBroad}(T, \prec) \cup \mathsf{LastBroad}(T, \lhd)$, *i.e.* it consists of the new broadcasts and the last broadcasts.

Definition 3.6 (Minimal $\mathfrak{T}$-biadmissible strategy pattern) *A strategy pattern $(T, \prec, \lhd)$, where $T = (N, n_0, E, \Delta, \mathsf{lab})$, is said to be a $\mathfrak{T}$-biadmissible* minimal *strategy pattern if it is $\mathfrak{T}$-biadmissible and the following conditions are fulfilled:*

- ***(a)*** *for every $n \in N$, if $\mathsf{lab}(n) \in \mathfrak{T}$ and $\mathsf{Sub}(T, n) = (N', n, E', \Delta, \mathsf{lab}')$ with $N' \cap \mathsf{Imp}(T, \prec, \lhd) \subseteq \{n\}$ then $N' = \{n\}$;*
- ***(b)*** *for every $n \in N$ and every $n_1, n_2 \in N$ with $(n, n_1) \in E$ and $(n, n_2) \in E$, if $\mathsf{Sub}(T, n_1) = (N_1, n_1, E_1, \Delta, \mathsf{lab}_1)$ and $\mathsf{Sub}(T, n_2) = (N_2, n_2, E_2, \Delta, \mathsf{lab}_2)$ then, either both $N_1 \cap \mathsf{Imp}(T, \prec, \lhd)$ and $N_2 \cap \mathsf{Imp}(T, \prec, \lhd)$ are not empty, or $n_1 = n_2$;*
- ***(c)*** *for every $n_1, n_2, n_3 \in N$ pairwise different such that $\mathsf{lab}(n_1) = \mathsf{lab}(n_2) = \mathsf{lab}(n_3)$, if for $i \in \{1, 2, 3\}$, $\mathsf{Sub}(T, n_i) = (N_i, n_i, E_i, \Delta, \mathsf{lab}_i)$, $n_3 \in N_2$ and $n_2 \in N_1$, then there exist $i, j \in \{1, 2, 3\}$ such that $N_i \cap \mathsf{Imp}(T, \prec, \lhd) \neq N_j \cap \mathsf{Imp}(T, \prec, \lhd)$.*

Intuitively, these conditions state that **(a)** the branches of the tree end at the first target node not followed by any important node, **(b)** for any branching, there cannot be two subtrees without important nodes and, **(c)** if three different subtrees have their root labeled by the same state, then there should be at least one important node in one of the subtrees and not in the other ones. The reason is the same as for admissible pattern *i.e.* otherwise we can replace a bigger subtree by a smaller one.

Proposition 3.5 *If there exists a $\mathfrak{T}$-biadmissible strategy pattern for $\mathcal{P}$, then there exists a minimal one.*

Proof Let $(T, \prec, \lhd)$ with $T = (N, n_0, E, \Delta, \mathsf{lab})$ be a $\mathfrak{T}$-biadmissible strategy pattern and assume that $(T, \prec, \lhd)$ is not minimal.

First, we suppose that there exists a set of nodes $S \subseteq N$ such that for every node $n \in S$, $\mathsf{lab}(n) \in \mathfrak{T}$ and $\mathsf{Sub}(T, n) = (N', n, E', \Delta, \mathsf{lab}')$ with $N' \cap \mathsf{Imp}(T, \prec, \lhd) \subseteq \{n\}$. For each $n \in S$, we remove from T all the nodes in its subtree (*i.e.* every $n' \in N' \setminus \{n\}$), except n, and their associated edges. The resulting object is a $\mathfrak{T}$-biadmissible strategy pattern for $\mathcal{P}$. Indeed thanks to condition (1) of admissibility[1] on $(T, \prec)$, such an operation preserves the tree structure. Moreover, since we only remove nodes bigger than nodes of S with respect to $\prec$, we know that condition (2) of $\prec$ is also preserved. Finally, since no important node was removed, and since the pattern was pruned at a node labeled by a state in $\mathfrak{T}$, we deduce that the conditions for coadmissibility[2] are still respected. In the end, the obtained strategy pattern is admissible, co-admissible and verifies condition **(a)** of minimality.

Now, we assume that condition **(a)** is satisfied by $(T, \prec, \lhd)$ and we suppose that there exists a node $n \in N$ such that there are descendants $n_1 \neq n_2 \in N$ with $(n, n_1) \in E$, $(n, n_2) \in E$, $\mathsf{Sub}(T, n_1) = (N_1, n_1, E_1, \Delta, \mathsf{lab}_1)$, $\mathsf{Sub}(T, n_2) = (N_2, n_2, E_2, \Delta, \mathsf{lab}_2)$ and with $N_1 \cap \mathsf{Imp}(T, \prec, \lhd) = \emptyset$, and $N_2 \cap \mathsf{Imp}(T, \prec, \lhd) = \emptyset$. Getting a pattern that satisfies in addition condition **(b)** is easy: for every such node n we remove from T the nodes in N_1 if $n_1 \lhd n_2$ (symmetrically those of N_2 in case $n_2 \lhd n_1$) and their associated edges. Here again, since we remove entire subtrees, the resulting structure is still a strategy pattern. Furthermore, since between n_1 and n_2, the maximal node w.r.t. $\lhd$ was kept, the condition for coadmissibility concerning node n still holds. Also, since no important node was removed, the pattern is still biadmissible. Finally the obtained admissible strategy pattern thus satisfies conditions **(a)** and **(b)**.

Last, we assume that conditions **(a)** and **(b)** are satisfied by $(T, \prec, \lhd)$ whereas condition **(c)** is not satisfied. Until condition **(c)** is satisfied, we perform the following operations. Suppose that there are three pairwise distinct nodes $n_1, n_2, n_3 \in N$ such that $\mathsf{lab}(n_1) = \mathsf{lab}(n_2) = \mathsf{lab}(n_3)$, for $i \in \{1, 2, 3\}$, $\mathsf{Sub}(T, n_i) = (N_i, n_i, E_i, \Delta, \mathsf{lab}_i)$ and $n_3 \in N_2$ and $n_2 \in N_1$ and such that $N_1 \cap \mathsf{Imp}(T, \prec) = N_2 \cap \mathsf{Imp}(T, \prec) = N_3 \cap \mathsf{Imp}(T, \prec)$. We proceed by case inspection:

- First, assume that $n_1 \lhd n_2$. Since $N_1 \cap \mathsf{Imp}(T, \prec, \lhd) = N_2 \cap \mathsf{Imp}(T, \prec, \lhd)$, important nodes matter, and $\mathsf{lab}(n_1) = \mathsf{lab}(n_2)$, we can replace in T the subtree $\mathsf{Sub}(T, n_1)$

[1] See Definition 3.3, on page 143.
[2] See Definition 3.5, on page 153.

by its subtree $\mathsf{Sub}(T, n_2)$, and doing so, remove from T the nodes in $N_1 \setminus N_2$. The resulting object is still a $\mathfrak{T}$-biadmissible strategy pattern because no important node was removed, and the predecessor of n_1 is now connected to a bigger node, with respect to $\lhd$. Hence it still satisfies the coadmissibility condition.
The cases $n_2 \lhd n_3$ and $n_1 \lhd n_3$ are treated similarly.

- Assume now the hardest situation: $n_3 \lhd n_2 \lhd n_1$. Note that $N_1 \cap \mathsf{Imp}(T, \prec, \lhd) = N_2 \cap \mathsf{Imp}(T, \prec, \lhd) = N_3 \cap \mathsf{Imp}(T, \prec, \lhd)$ hence that $N_1 \setminus N_3 \cap \mathsf{Imp}(T, \prec, \lhd) = \emptyset$. Let $n'_1, \ldots, n'_k \in N$ be the nodes on the path from n_1 to n_2 (both included), defined formally as the nodes such that $n'_1 = n_1$, $n'_k = n_2$ and $\forall i \in [1..k-1], (n'_i, n'_{i+1}) \in E$. Since $n_2 \lhd n_1$ we know that there exist $n' = n'_i$ such that $n'_{i+1} \lhd n'_i$, we denote $\mathsf{Sub}(T, n') = (N', n', E', \Delta, \mathsf{lab}')$ for clarity. We now modify $\lhd$ into an order $\lhd'$ such that $n_2 \lhd' n_3$ and $(T, \lhd')$ is $\mathfrak{T}$-coadmissible. The idea is that we can decrease the rank of all the nodes in $(N' \setminus N_2) \cup \{n_2\}$ by the same value without falsifying the coadmissibility property (in fact none of these nodes are important nodes). Formally, letting $B = \min_{n \in N}\{rk(n)\} - \max_{n \in N}\{rk(n)\} - 1$ we define $\lhd'$ as the order associated with the following rank function: for every node $n \in N \setminus ((N' \setminus N_2) \cup \{n_2\})$, $rk_{\lhd'}(n) = rk_{\lhd}(n)$ and for all the other nodes $n \in (N' \setminus N_2) \cup \{n_2\}$, $rk_{\lhd'}(n) = rk_{\lhd}(n) + B$. Let us argue that $(T, \lhd')$ is $\mathfrak{T}$-coadmissible. Indeed, the only edges that could cause a problem (to maintain the existence of an edge $e = (n, n') \in E$ with $n \lhd n'$ for each n such that $\mathsf{lab}(n) \notin \mathfrak{T}$) are of two forms: first, (n'_{i+1}, n'_i) since the rank of n'_i was decreased, but since $n'_{i+1} \lhd n'_i$ we are safe; second, the edges leaving n_2 but since the rank of n_2 was decreased and not the one of its successor we are also safe. Finally the rank of the important nodes is left unchanged. As a consequence, $(T, \lhd')$ is $\mathfrak{T}$-coadmissible and it satisfies $n_2 \lhd' n_3$, so that we can apply the transformation described for the first case.

Repeating these operations allows us to finally get a $\mathfrak{T}$-biadmissible strategy pattern which respects conditions **(a)**, **(b)** and **(c)**, and hence which is minimal. □

We now give a bound on the size of minimal $\mathfrak{T}$-coadmissible strategy patterns.

Lemma 3.7 *If there exists a $\mathfrak{T}$-biadmissible strategy pattern for $\mathcal{P}$, then there is one of size at most $16|\Sigma| \cdot |\mathrm{Q}| \cdot (|\mathrm{Q}| - |\mathfrak{T}| + 1)$ and of height at most $4|\Sigma| \cdot |\mathrm{Q}| + 2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$.*

Proof We proceed in two steps: in the first step, we bound the number of nodes that precede important nodes, and in the second step we bound the number of nodes that do not lead to important nodes, but can be useful to converge towards the target.

The first step is very similar to the proof of the bound given for admissible trees. Let $(T, \prec, \lhd)$ with $T = (N, n_0, E, \Delta, \mathsf{lab})$ be a $\mathfrak{T}$-biadmissible strategy pattern for $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$. By Proposition 3.5, we can assume $(T, \prec, \lhd)$ to be minimal. We consider $T' = (N', n_0, E', \Delta, \mathsf{lab}') = T_{\downarrow \mathsf{Imp}(T, \prec, \lhd)}$ the pattern restricted to predecessors of important nodes. Recall that important nodes are the new broadcasts and last broadcasts. Pattern T' contains at most $|\mathsf{Imp}(T, \prec, \lhd)| - 1 \leq 2|\Sigma|$ intersection nodes. One

can thus bound the number of noticeable nodes (recall that noticeable nodes gather intersection nodes and important nodes) by $2|\Sigma|+2|\Sigma| = 4|\Sigma|$. Moreover, from condition **(c)** of minimality, we deduce that there are no more than $2|\mathrm{Q}|-1$ nodes between two noticeable nodes. Otherwise there would be three nodes with the same label that share the same set of important nodes. This implies that $|N'| \leq 4|\Sigma|(2|\mathrm{Q}|-1)+4|\Sigma| = 8|\Sigma||\mathrm{Q}|$, and concludes the first step.

For the second step, from condition **(b)** of minimality, from every node $n' \in N'$, there can be a node $n_2 \in N$ with $(n', n_2) \in E$ and such that $\mathsf{Sub}(T, n_2) = (N_2, n_2, E_2, \Delta, \mathsf{lab}_2)$ and $N_2 \cap \mathsf{Imp}(T, \prec, \lhd) = \emptyset$ and $\mathsf{Sub}(T, n_2)$ has a unique branch (by condition **(b)**). Yet, conditions **(a)** and **(c)** ensure in that case that $|N_2| \leq 2(|\mathrm{Q}| - |\mathfrak{T}|)+1$. Otherwise there would be three nodes with the same label (which is impossible by condition **(c)**) or a node with a label in target that is not a leaf (which is impossible by condition **(a)**).

To conclude, we deduce that for each node $n' \in N'$, there is at most a unique node $n_2 \in N$ with $(n', n_2) \in E$ such that $\mathsf{Sub}(T, n_2) = (N_2, n_2, E_2, \Delta, \mathsf{lab}_2)$, $N_2 \cap \mathsf{Imp}(T, \prec, \lhd) = \emptyset$, $N_2 \cap N' = \emptyset$ and $|N_2| \leq 2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$. Since $|N'| \leq 8|\Sigma||\mathrm{Q}|$, we deduce that the size of a minimal $\mathfrak{T}$-biadmissible strategy pattern for $\mathcal{P}$ is at most $16|\Sigma| \cdot |\mathrm{Q}| \cdot (|\mathrm{Q}| - |\mathfrak{T}| + 1)$.

For what concerns the height, the worst case is that a unique branch containing all the important nodes (separated by at most $2|\mathrm{Q}|$ nodes, thanks to **(c)**) and finishing with $2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$ nodes to reach the target. Since the number of important nodes is bounded by twice the number of letters in Σ, we obtain that the height is bounded by $4|\Sigma| \cdot |\mathrm{Q}| + 2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$. □

Theorem 3.3 Synch*[$\mathcal{L}$] is NP-complete.*

Proof Using Proposition 3.4, we deduce that there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$ if and only if there exists a $\mathfrak{T}$-biadmissible strategy pattern. Lemma 3.7 allows us to look only for $\mathfrak{T}$-biadmissible strategy patterns whose size is polynomial in the size of the broadcast protocol $\mathcal{P}$. Thus, we deduce a non-deterministic polynomial time algorithm which consists in guessing a strategy pattern (equipped with two orders) of polynomial size and then verifying whether it is $\mathfrak{T}$-biadmissible (this can be done in polynomial time thanks to Lemma 3.5). This proves that Synch[$\mathcal{L}$] is in NP. Moreover, since Synch[$\mathcal{L}$] is harder than Reach[$\mathcal{L}$] (as we will discuss in Remark 3.1) and Reach[$\mathcal{L}$] is NP-hard by Theorem 3.2, we establish that Synch[$\mathcal{L}$] is NP-complete. □

We now give a bound on the minimal number of processes needed to implement a local strategy gathering all the processes in target states.

Theorem 3.4 *If there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$, then there exists an execution $\theta' \in \Theta_{\mathcal{L}}$ such that $\mathsf{End}(\theta') \subseteq \mathfrak{T}$ and $nbproc(\theta') \leq 16|\Sigma| \cdot |\mathrm{Q}| + 4|\Sigma| \cdot (|\mathrm{Q}| - |\mathfrak{T}| + 1)$ and $|\pi_{p_{id}}(\theta')| \leq 4|\Sigma| \cdot |\mathrm{Q}| + 2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$ for every $p_{id} \leq nbproc(\theta')$.*

Proof We consider a minimal $\mathfrak{T}$-biadmissible strategy pattern $(T, \prec, \lhd)$. The cutoff on the number of processes is proved by constructing an execution that has two phases.

The first phase consists in filling all the important nodes (and only them) with at least one process. This can be done with less processes than $|T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}|$ using the same techniques as the ones in the proof of Lemma 3.4 and using broadcasts that lead to nodes of $\mathsf{NewBroad}(T,\prec,\lhd)$). The minimality of T implies an upper bound of $8|\Sigma||Q|$ on the number of processes needed (see the proof of Lemma 3.7 to get the bound on the size of $T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}$) .

The second phase consists in emptying the nodes towards target nodes. This is also done as in the proof of Lemma 3.4 but this time using broadcasts that lead to nodes of $\mathsf{LastBroad}(T,\prec,\lhd)$ and in the order defined by $\lhd$ rather than $\prec$. Since at the end of the first phase, all the processes are in important nodes, we only need to consider the pattern restricted to $T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}$ plus at most one branch per important node that does not contain an important node and leads to a target node. For $k = |\mathsf{Imp}(T,\prec,\lhd)|$, we let $\theta_{imp} \in \Theta_{\mathcal{L}}$ be an execution obtained following the same techniques as for the proof of Lemma 3.4 and which respects that at the end all the processes are in important nodes (at least one process in each of them) and the number of processes needed is bound by the number of nodes in $T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}$, as it is summed up by the following properties:

1. $\{\pi_{p_{id}}(\theta_{imp}) \mid p_{id} \in [1..nbproc(\theta_{imp})]\} = \{h(n) \mid n \in \mathsf{Imp}(T,\prec,\lhd)\}$
2. $nbproc(\theta_{imp}) \leq |T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}|$

We define inductively the function empt that, given a node $n \in N$ and an execution such that there is a process in all the important nodes greater than n w.r.t. $\lhd$, extends the execution emptying all the non target nodes in the order given by $\lhd$. Formally, $\mathsf{empt}(n,\theta) = \theta$ if n is the maximal node. Otherwise, letting n' for the successor of n w.r.t. $\lhd$, we define $\mathsf{empt}(n,\theta)$ by:

- $\mathsf{empt}(n',\theta)$ if $\{p_{id} \mid \pi_{p_{id}}(\theta) = h(n)\} = \emptyset$; if there are no processes in n we continue with the next node.
- $\mathsf{empt}(n',\theta)$ if $\mathsf{lab}(n) \in \mathfrak{T}$; if node n is labeled by a state of $\mathfrak{T}$ we leave processes in node n unchanged and we continue with the next node.
- $\mathsf{empt}(n',\theta')$ if $\{p_{id} \mid \pi_{p_{id}}(\theta) = h(n)\} = \{p_{id_1},\ldots,p_{id_k}\}$ and there exists an edge $e = (n,n_1)$ with $n \lhd n_1$ such that e is labeled with an active action $\delta = \mathsf{lab}(e)$ and where $\theta' = \theta \xrightarrow{p_{id_1},\delta,\emptyset} \ldots \xrightarrow{p_{id_k},\delta,\emptyset} \gamma$. In other words, if there is an edge labeled by an active action outgoing of n, all the processes in n perform this action with an empty reception set.
- $\mathsf{empt}(n',\theta')$ if there exists an edge $e = (n,n')$ with $n \lhd n'$ such that e is labeled with a reception of message m, $\delta = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$. We consider $e_m = (n_1,n_2)$ such that $\mathsf{lab}(e_m)$ is a broadcast of m and such that $n_2 \in \mathsf{Imp}(T,\prec,\lhd)$ with $n \lhd n_2$. Such an edge exists because T is coadmissible. By assumption on θ, there exists $p'_{id} \in [1..nbproc(\theta)]$ such that $\pi_{p'_{id}}(\theta) = h(n_2)$. We consider the execution $follow(\theta, p'_{id}, h(n_1))$, and write p_{id_m} for the additional process in that execution compared to θ. Then, denoting $P = \{p_{id} \mid \pi_{p_{id}}(\theta) = h(n)\}$ the set of all processes

p_{id} in n, we let $\theta' = \mathit{follow}(\theta, p'_{id}, h(n_1)) \xrightarrow{p_{id\,m},\mathsf{lab}(e),P} \gamma$, *i.e.* the additional process $p_{id\,m}$ broadcasts message m only to the processes of P, and they move along edge e. Here, $\pi_{p_{id\,m}}(\theta) = h(n_2)$.

By definition of the function empt, the number of processes is incremented by 1 only in the last case.

Applying iteratively empt on θ_{imp} starting from the minimal node (with respect to $\lhd$), one obtains a local execution in which all the processes end in a target state. An additional process is added (through the function *follow*) only in the case of a reception, and at most one process is added per edge labeled by a reception. Moreover, in θ_{imp} all the processes are in important nodes and while applying empt, the processes remain together, hence the execution $\theta = \mathsf{empt}(n_0, \theta_{imp})$ visits only $|\mathsf{Imp}(T, \prec, \lhd)|$ branches that do not belong to $T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}$. As a consequence, we can bound the number of processes by $\mathit{nbproc}(\theta) \leq \mathit{nbproc}(\theta_{imp}) + |T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}| + |\mathsf{Imp}(T, \prec, \lhd)|(2(|\mathrm{Q}| - |\mathfrak{T}|) + 1) \leq 16|\Sigma| \cdot |\mathrm{Q}| + 4|\Sigma|(|\mathrm{Q}| - |\mathfrak{T}| + 1)$. In fact, the term $|T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}|$ is the number of additive broadcasts that needs to be performed to bring all the nodes out of $T_{\downarrow \mathsf{Imp}(T,\prec,\lhd)}$. And then for each of the $|\mathsf{Imp}(T, \prec, \lhd)|$ branches, there might be at most $2(|\mathrm{Q}| - |\mathfrak{T}|) + 1$ necessary broadcasts to bring the processes at the end of the branch.

To conclude, as in the case of Reach[$\mathcal{L}$], the upper bound on the past of each process trivially coincides with the upper bound on the height of T. □

Remark 3.1 *The NP-hardness derives from the fact that the target problem is harder than the reachability problem. To reduce* Reach*[$\mathcal{L}$] to* Synch*[$\mathcal{L}$], one can add the broadcast of a new message from q_F, and its reception from any state to q_F.*

Another consequence of this simple reduction is that Synch*[$\mathcal{L}$] in NP yields another proof that* Reach*[$\mathcal{L}$] is in NP. Yet, the two proofs of NP-membership allowed us to give an incremental presentation, starting with admissible strategy patterns, and proceeding with co-admissible strategy patterns.*

4 Cliques and local strategies

4.1 Undecidability of Reach[$\mathcal{LC}$] and Synch[$\mathcal{LC}$]

Reach[$\mathcal{LC}$] and Synch[$\mathcal{LC}$] happen to be undecidable and for the latter, even in the case of complete protocols. The proofs of these two results are based on a reduction from the halting problem of a two-counter Minsky machine (a finite program equipped with two integer variables which can be incremented, decremented and tested to zero, see Chapter I.4.4). The main idea consists in both cases in isolating some processes to simulate the behavior of the machine while the other processes encode the values of the counters.

Thanks to the clique semantics it is possible to isolate one process. This is achieved by setting the first transition to be the broadcast of a message *start* whose reception makes all the other processes change their state. Hence, thanks to the clique semantics, there is only one process that sends the message *start*, such process, called the *controller*,

will be in charge of simulating the instructions of the Minsky machine. The clique semantics is also used to correctly simulate the increment and decrement of counters. The value of the counter is represented by the number of processes in state 1. For instance to increment a counter, the controller asks whether a process simulating the counter can be moved from state 0 to state 1 and if it is possible, relying on the clique topology only one such process changes its state. In fact, all the processes will receive the request, but the first one answering it, will force the other processes to come back to their original state, ensuring that only one process will move from state 0 to 1.

The main difficulty is that broadcast protocols (even under the clique semantics) cannot test the absence of processes in a certain state (which would be needed to simulate a test to 0 of one of the counters). Here is how we overcome this issue for SYNCH[$\mathcal{LC}$]: the controller, when simulating a zero-test, sends all the processes with value 1 into a sink error state and the target problem allows to check for the reachability of a configuration with no process in this error state (and thus to test whether the controller has 'cheated', *i.e.* has taken a zero-test transition whereas the value of the associated counter was not 0). We point out that in this case, restricting to local executions is not necessary, we get in fact as well that SYNCH[$\mathcal{C}$] is undecidable.

For REACH[$\mathcal{LC}$], the reduction is more tricky since we cannot rely on a target set of states to check that zero-tests were faithfully simulated. Here in fact we will use two controllers. Basically, before sending a *start* message, some processes will be able to go to a waiting state (thanks to an internal transition) from which they can become controller and in which they will not receive any messages (this is where the protocol needs to be incomplete). Then we will use the locality hypothesis to ensure that two different controllers will simulate, exactly in the same way, the run of the Minsky machine and with exactly the same number of processes encoding the counters. Restricting to local strategies guarantees the two runs to be identical, and the correctness derives from the fact that if in the first simulation the controller 'cheats' while performing a zero-test (and sending as before some processes encoding a counter value into a sink state), then in the second simulation, the number of processes encoding the counters will be smaller (due to the processes blocked in the sink state), so that the simulation will fail (because there will not be enough processes to simulate faithfully the counter values).

Theorem 4.1 REACH*[$\mathcal{LC}$] is undecidable and* SYNCH*[$\mathcal{LC}$] restricted to complete protocol is undecidable.*

Proof We begin by recalling the definition of Minsky machines [Min67] (see Section 4.4). A *deterministic Minsky machine* $\mathcal{M}$ manipulates two integer variables c_1 and c_2, which are called counters, and it is composed of a finite set of instructions. Each of the instructions is either of the form (1) $L : c_i := c_i + 1$; `goto` L' or (2) L : `if` $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L'', where $i \in \{1, 2\}$ and L, L', L'' are labels of instructions. Furthermore there is a special instruction labeled L_F, from which nothing can be done. The halting problem then asks whether the execution starting from L_0 with both counters equal to 0 reaches L_F. Without loss of generality, we

can assume that when the machine reaches L_F, the values of the two counters are equal to 0.

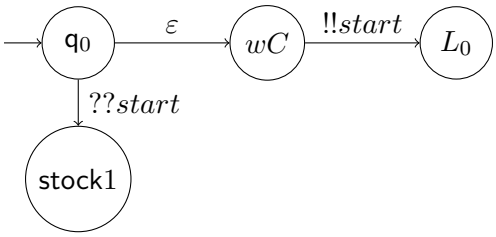


Figure 4.6: Initialization phase for REACH[$\mathcal{LC}$].

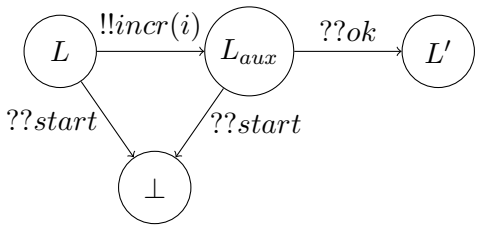


Figure 4.7: Encoding an increment $L : c_i := c_i + 1;$ `goto` L'.

We begin by proving that REACH[$\mathcal{LC}$] is undecidable. For this, we encode a Minsky machine in the broadcast protocol $\mathcal{P}$ given in several parts in the Figures 4.6, 4.7, 4.8, 4.9 and 4.10. The protocol $\mathcal{P}$ is built so as to simulate twice the run of the Minsky machine. In order to do so, in $\mathcal{P}$, (at least) two processes will decide the sequence of instructions of the Minsky machine (represented on Figs. 4.7 and 4.8). The remaining processes will encode the values of the counters (see Fig. 4.9): precisely, the number of processes in the state 1_i will represent the value of counter c_i.

Here are some key points on how this protocol is working along a local clique execution:

- During the initialization phase (see Figure 4.6), using the internal actions, some processes may stay in q_0, some other may move to a waiting state wC (standing for waiting controller) where they will wait to become the next controller.

- As soon as a process in wC moves to L_0, it broadcasts message *start* to processes in q_0 and the simulation properly begins. Note that after this step no process can be in state q_0, the clique semantics guarantees that they moved to stock1. Yet there can be some processes in wC.

- Intuitively the process in L_0 will simulate the sequence of instructions of the Minsky machine while the processes in stock1 will be used to encode the counter

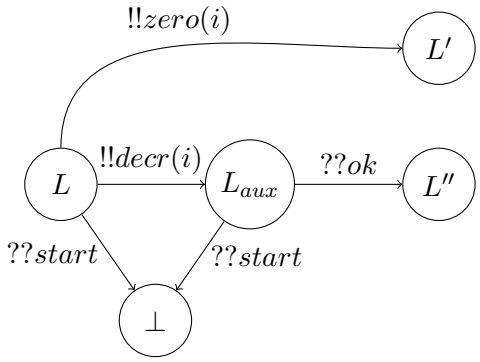


Figure 4.8: Encoding a test-to-zero L : `if` $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L''.

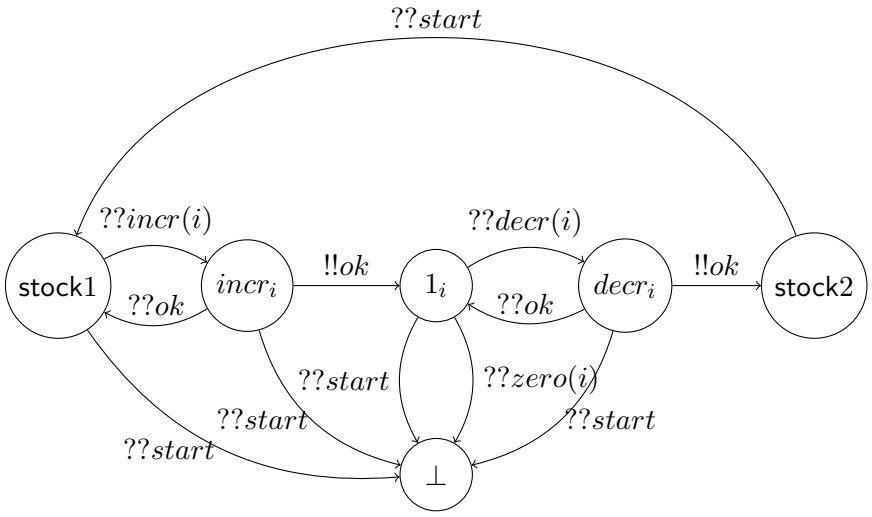


Figure 4.9: Encoding counter c_i.

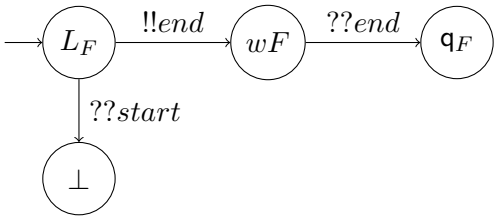


Figure 4.10: Ending phase for Reach[$\mathcal{LC}$].

values in the first simulation of the run. Later, another process from state wC will be used to perform the simulation a second time.

- Let us first comment on the following. What happens if a process in wC moves to L_0 while another process is already acting as the controller of the counter machine (*i.e.* is in state L or L_{aux} of the Figs. 4.7 or 4.8)? Then the first controller will receive the message *start* and move to state $\perp$ which is a deadlock (as shown in the above mentioned figures). The processes simulating the counters will all move either to $\perp$ or to stock1 (see Fig. 4.9). As a consequence, a new simulation starts and it cannot interfere with the previous one that has been stopped.

- Then let us explain how the simulation of each action works precisely:

  Incrementing a counter. To simulate an increment instruction of the form $L : c_i := c_i + 1$; `goto` L', the controller behaves as represented in Fig. 4.7. It broadcasts the message $incr(i)$, which is received by all the processes in state stock1 which all move to $incr_i$ (see Fig. 4.9). Then it waits to receive an acknowledgement message ok, this message is broadcast by one process in $incr_i$; as a consequence, the controller moves to state L' and exactly one process moves to 1_i (the one which performed the broadcast of ok), whereas all the other processes in $incr_i$ move back to stock1.

  Decrementing a counter. The decrement of a counter (see Fig. 4.8) is pretty similar to the increment, and only one process will move from 1_i to stock2, the pool of processes for the second simulation.

  Zero testing. When the controller mimics a zero test decrement instruction *i.e.* an instruction of the shape L : `if` $c_i = 0$ `then goto` L' `else` $c_i := c_i - 1$; `goto` L'', it has no way to know whether some processes are in state 1_i or not. However, it can choose to broadcast $zero(i)$, even if the counter value is not 0, *i.e.* if there are some processes in 1_i. A consequence of this is that all the processes in 1_i are sent to $\perp$. This represents the fact that if the process performs the broadcast of $zero(i)$ while some processes are in 1_i, it *cheats*: it assumes that the counter value is 0 although it is not the case.

- Observing the behavior of the processes simulating the counters, one sees that for each increment-decrement pair, exactly one counter process is sent from stock1 to stock2 and possibly some processes are lost (*i.e.* moved to $\perp$) when the controller cheats.

- What happens at the end of a simulation of the two-counter machine run? When a process controller reaches L_F, it may move to state wF (standing for waiting final) (see Fig. 4.10).

- A new simulation can then begin with a process moving from wC to L_0 and broadcasting *start*. When the message *start* is sent (remember that the first simulation has to be finished otherwise, it will correspond to a new first simulation

as explained previously), all the processes in stock1 are moved to $\perp$ and all the processes in stock2 are moved to stock1. Hence there are at most as many processes in stock1 as the number of decrements during the first simulation.
Because we restrict to local executions, the new controller will perform exactly the same choices as the previous one, hence mimicking exactly the same run. Consequently there are two options:

1. Either the first controller has cheated at some point, then the second one will be eventually stuck, because there will not be enough processes in stock1 to answer an increment request;
2. Or the first controller did not cheat, which means that the simulation was correct, and the second simulation will then also be correct. The second controller will reach L_F, then it will reach wF broadcasting *end* and so the first controller will move from wF to q_F (in case the first controller was not in wF, another simulation has to be performed).

Under this construction, the Minksy machine halts if and only if there is an execution $\theta \in \Theta_{\mathcal{LC}}$ such that $\mathsf{q}_F \in \mathsf{End}(\theta)$. This proves that Reach[$\mathcal{L}$] is undecidable.

Remark 4.1 *Note that this proof heavily relies on three features: (1) the execution is local, (2) the execution is a clique execution and (3) the protocols can be incomplete. Indeed, restricting to clique executions allows one to distinguish a controller from processes encoding counters, and to be sure that exactly one process answers to increment/decrement requests. Second, restricting to local executions ensures that the second sequence of instructions exactly repeats the first one. Last, we use the fact that the protocol is not complete to ensure that some processes can stay in wC and that all the processes arriving for the first time in L_0 share the same history.*

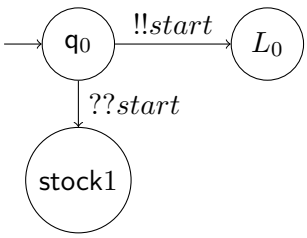


Figure 4.11: Initialization phase for Synch[$\mathcal{LC}$].

To prove that Synch[$\mathcal{LC}$] restricted to complete protocol is undecidable, we use the same kind of reasoning and we reuse Fig. 4.7, 4.8 and 4.9 for the simulation of the actions and of the counters. The initialization phase differs and is represented in Fig. 4.11. Note that in that case, a single controller process will reach L_0 and we will guarantee that it does not cheat by defining as target set of states $\mathfrak{T} = \{\mathsf{stock1}, \mathsf{stock2}, L_F\}$. Following the same argumentation as the one before, one can show that the target set of states

allows us to ensure that the unique controller process (which reaches L_F) did not cheat, otherwise there will be some processes in $\bot$. Hence for the broadcast protocol $\mathcal{P}$ we build here, we can show that there exists an execution $\theta \in \Theta_{\mathcal{LC}}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$ if and only if the Minsky machine halts. Note that the protocol we describe is not actually complete but it will not harm the reduction to complement it by adding an edge to $\bot$ for each unspecified reception.

□

Remark 4.2 *In contrast to the undecidability proof for* Reach*[$\mathcal{LC}$] in which the run of the Minsky machine is simulated twice in a row, restricting to local executions is not necessary here, so that we can show that* Synch*[$\mathcal{C}$] is undecidable.*

The undecidability proof for Reach[$\mathcal{LC}$] strongly relies on the protocol being incomplete. Indeed, in the absence of specified receptions, the processes ignore broadcast messages and keep the same history, thus allowing to perform twice the same simulation of the run. In contrast, for complete protocols, all the processes are aware of all broadcast messages, therefore one cannot force the two runs to be identical. In fact, the reachability problem is decidable for complete protocols, as we shall see in the next section.

4.2 Decidability of Reach[$\mathcal{LC}$] for complete protocols

To prove the decidability of Reach[$\mathcal{LC}$] for complete protocols, we abstract the behavior of a protocol under local clique semantics by counting the possible number of different histories in each control state.

We identify two cases when the history of processes can differ (under local clique semantics): (1) When a process p_{id} performs a broadcast, its history is unique for ever (since all the other processes must receive the emitted message); (2) A set of processes sharing the same history can be split when some of them perform a sequence of internal actions and the others perform only a prefix of that sequence.

From a complete broadcast protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ we build an abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}} = (\Lambda, \lambda_0, \Rightarrow)$ where configurations count the number of different histories in each control state. More precisely the set of abstract configurations is $\Lambda = fake\mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\} \times \{\varepsilon, !!\}$. Abstract configurations are thus pairs where the first element is a multiset and the second element is a flag in $\{\varepsilon, !!\}$. The latter indicates the type of the next actions to be simulated (sequence of internal actions or broadcast): it prevents to simulate consecutively two incoherent sequences of internal actions (with respect to the local strategy hypothesis). For the former, an element $(q, \mathbf{s}, !!_{ok})$ in the multiset represents a single process (flag $\mathbf{s}$) in state q with a unique history which is allowed to perform a broadcast (flag $!!_{ok}$). An element $(q, \mathbf{m}, !!_{no})$ represents many processes (flag $\mathbf{m}$) in state q, all sharing the same unique history and none of them is allowed to perform a broadcast (flag $!!_{no}$). The initial abstract configuration λ_0 is then $(\langle(\mathsf{q}_0, \mathbf{m}, !!_{ok})\rangle, \varepsilon)$. In the sequel we will write $\mathbf{HM}$ for the set $fake\mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$ of history multisets, so that $\Lambda = \mathbf{HM} \times \{\varepsilon, !!\}$, and typical elements of $\mathbf{HM}$ are denoted $\mathbb{M}$, $\mathbb{M}'$, etc.

In order to provide the definition of the abstract transition relation $\Rightarrow$, we need to introduce new notions, and notations. An ε-path ρ in $\mathcal{P}$ from q to q' is either the empty path (and in that case $\mathsf{q} = \mathsf{q}'$) or it is a non-empty finite path $\delta_0 \cdots \delta_n$ that starts in q, ends in q' and such that all the δ_i's are internal transitions.

An ε-path ρ in $\mathcal{P}$ is said to be a *prefix* of an ε-path ρ' if $\rho \neq \rho'$ and either ρ is the empty path or $\rho = \delta_0 \cdots \delta_n$ and $\rho' = \delta_0 \cdots \delta_n \delta_{n+1} \ldots \delta_{n+m}$ for some $m > 0$. Since we will handle multisets, let us give some convenient notations. Given E a set, and $\mathbb{M}$ a multiset over E, we write $\mathbb{M}(e)$ for the number of occurrences of element $e \in E$ in $\mathbb{M}$. Moreover, $card(\mathbb{M})$ stands for the cardinality of $\mathbb{M}$: $card(\mathbb{M}) = \sum_{e \in E} \mathbb{M}(e)$. Last, we will write $\oplus$ for the addition on multisets: $\mathbb{M} \oplus \mathbb{M}'$ is such that for all $e \in E$, $(\mathbb{M} \oplus \mathbb{M}')(e) = \mathbb{M}(e) + \mathbb{M}'(e)$.

The abstract transition relation $\Rightarrow \in \Lambda \times \Lambda$ is composed of two transitions relations: one simulates the broadcast of messages and the other one sequences of internal transitions. This will guarantee an alternation between abstract configurations flagged with ε and the ones flagged with !!. Let us first define $\Rightarrow_{!!} \subseteq (\mathbf{HM} \times \{!!\}) \times (\mathbf{HM} \times \{\varepsilon\})$ which simulates a broadcast. We have $(\mathbb{M}, !!) \Rightarrow_{!!} (\mathbb{M}', \varepsilon)$ if and only if there exist $(\mathsf{q}_1, !!m, \mathsf{q}_2) \in \Delta$ and $\mathit{fl}_1 \in \{\mathbf{s}, \mathbf{m}\}$ such that

1. $\mathbb{M}(\mathsf{q}_1, \mathit{fl}_1, !!_{ok}) > 0$;
2. there exists a family of functions G indexed by $(\mathsf{q}, \mathit{fl}, b) \in \mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $G_{(\mathsf{q},\mathit{fl},b)} : [1..\mathbb{M}(\mathsf{q}, \mathit{fl}, b)] \to \mathbf{HM}$, and:

$$\mathbb{M}' = \langle \mathsf{q}_2, \mathbf{s}, !!_{ok} \rangle \oplus \bigoplus_{\{(\mathsf{q},\mathit{fl},b) | \mathbb{M}(\mathsf{q},\mathit{fl},b) \neq 0\}} \bigoplus_{i \in [1..\mathbb{M}(\mathsf{q},\mathit{fl},b)]} G_{(\mathsf{q},\mathit{fl},b)}(i)$$

   and such that for each $(\mathsf{q}, \mathit{fl}, b)$ verifying $\mathbb{M}(\mathsf{q}, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(\mathsf{q}, \mathit{fl}, b)]$, the following conditions are satisfied:

   (a) if $\mathit{fl}_1 = \mathbf{s}$, $card(G_{(\mathsf{q}_1,\mathit{fl}_1,!!_{ok})}(1)) = 0$ and if $\mathit{fl}_1 = \mathbf{m}$, then there exists $\mathsf{q}' \in \mathrm{Q}$ such that $G_{(\mathsf{q}_1,\mathit{fl}_1,!!_{ok})}(1) = \langle (\mathsf{q}', \mathit{fl}_1, !!_{ok}) \rangle$ and such that $(\mathsf{q}, ??m, \mathsf{q}') \in \Delta$;

   (b) if $(\mathsf{q}, \mathit{fl}, b) \neq (\mathsf{q}_1, \mathit{fl}_1, !!_{ok})$ or $i \neq 1$, then there exists $\mathsf{q}' \in \mathrm{Q}$ such that $G_{(\mathsf{q},\mathit{fl},b)}(i) = \langle (\mathsf{q}', \mathit{fl}, !!_{ok}) \rangle$ and such that $(\mathsf{q}, ??m, \mathsf{q}') \in \Delta$.

Intuitively to provide the broadcast, we need to find a process which is 'allowed' to perform a broadcast and which is hence associated with an element $(\mathsf{q}_1, \mathit{fl}_1, !!_{ok})$ in $\mathbb{M}$. The transition $(\mathsf{q}_1, !!m, \mathsf{q}_2)$ tells us which broadcast is simulated. Then the functions $G_{(\mathsf{q},\mathit{fl},b)}$ associate with each element of the multiset $\mathbb{M}$ of the form $(\mathsf{q}, \mathit{fl}, b)$ a single element which can be reached thanks to a reception of the message m. Of course this might not hold for an element of the shape $(\mathsf{q}_1, \mathbf{s}, !!_{ok})$ if it is the one chosen to do the broadcast since it represents a single process, and hence this element moves to q_2. Note however that if $\mathit{fl}_1 = \mathbf{m}$, then $(\mathsf{q}_1, \mathbf{m}, !!_{ok})$ represents many processes, hence the one which performs the broadcast is isolated, but the many other ones have to be treated for reception of the message. Note also that we use here the fact that since an element $(\mathsf{q}, \mathbf{m}, b)$ represents many processes with the same history, all these processes will behave the same way on reception of the message m.

We now define $\Rightarrow_\varepsilon \subseteq (\mathbf{HM} \times \{\varepsilon\}) \times (\mathbf{HM} \times \{!!\})$ which simulates the firing of sequences of ε-transitions. We have $(\mathbb{M}, \varepsilon) \Rightarrow_\varepsilon (\mathbb{M}', !!)$ if and only if there exists a family of functions F indexed by $(\mathsf{q}, \mathit{fl}, b) \in \mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $F_{(\mathsf{q},\mathit{fl},b)} : [1..\mathbb{M}(\mathsf{q}, \mathit{fl}, b)] \to \mathbf{HM}$, and

$$\mathbb{M}' = \bigoplus_{\{(\mathsf{q},\mathit{fl},b) | \mathbb{M}(\mathsf{q},\mathit{fl},b) \neq 0\}} \; \bigoplus_{i \in [1..\mathbb{M}(\mathsf{q},\mathit{fl},b)]} F_{(\mathsf{q},\mathit{fl},b)}(i)$$

and such that for each $(\mathsf{q}, \mathit{fl}, b)$ verifying $\mathbb{M}(\mathsf{q}, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(\mathsf{q}, \mathit{fl}, b)]$, we have:

1. $\mathit{card}(F_{(\mathsf{q},\mathit{fl},b)}(i)) \geq 1$ and if $\mathit{fl} = \mathbf{s}$, $\mathit{card}(F_{(\mathsf{q},\mathit{fl},b)}(i)) = 1$;
2. if $F_{(\mathsf{q},\mathit{fl},b)}(i)(\mathsf{q}', \mathit{fl}', b') \neq 0$, then $\mathit{fl}' = \mathit{fl}$;
3. there exists a pair $(\mathsf{q}_{!!}, \mathit{fl}_{!!}) \in \mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\}$ such that:
   - $F_{(\mathsf{q},\mathit{fl},b)}(i)(\mathsf{q}_{!!}, \mathit{fl}_{!!}, !!_{ok}) = 1$,
   - for all $(\mathsf{q}', \mathit{fl}') \neq (\mathsf{q}_{!!}, \mathit{fl}_{!!})$ $F_{(\mathsf{q},\mathit{fl},b)}(i)(\mathsf{q}', \mathit{fl}', !!_{ok}) = 0$,
   - there exists a ε-path $\rho_{!!}$ from q to $\mathsf{q}_{!!}$;
4. for all $(\mathsf{q}', \mathit{fl}')$ such that $F_{(\mathsf{q},\mathit{fl},b)}(i)(\mathsf{q}', \mathit{fl}', !!_{no}) = k > 0$, there exist k different ε-paths (strict) prefix of $\rho_{!!}$ from q to q'.

Intuitively the functions $F_{(\mathsf{q},\mathit{fl},b)}$ associate with each element $(\mathsf{q}, \mathit{fl}, b)$ of the multiset $\mathbb{M}$ a set of elements that can be reached via internal transitions. We recall that each such element represents a set (or a singleton if $\mathit{fl} = \mathbf{s}$) of processes sharing the same history. Condition 1. states that if there are multiple processes ($\mathit{fl} = \mathbf{m}$) then they can be matched to more states in the protocol, but if it is single ($\mathit{fl} = \mathbf{s}$) it should be matched by an unique state. Condition 2. expresses that if an element in $\mathbb{M}$ represents many processes, then all its images represent as well many processes. Conditions 3. and 4. deal with the locality assumption. Precisely, condition 3. states that among all the elements of $\mathbb{M}'$ associated with an element of $\mathbb{M}$, one and only one should be at the end of a ε-path, and only one process associated with this element will be allowed to perform a broadcast. This justifies the use of the flag $!!_{ok}$. Last, condition 4. concerns all the other elements associated to this element of $\mathbb{M}$: their flag is set to $!!_{no}$ (they cannot perform a broadcast, because the local strategy will force them to take an internal transition), and their state should be on the previously mentioned ε-path.

As announced, we define the abstract transitive relation by $\Rightarrow = \Rightarrow_\varepsilon \cup \Rightarrow_{!!}$. Note that by definition we have a strict alternation of transitions of the type $\Rightarrow_\varepsilon$ and of the type $\Rightarrow_{!!}$. An *abstract local clique execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ of the shape $\xi = \lambda_0 \Rightarrow \lambda_1 \cdots \Rightarrow \lambda_{\ell+1}$. As for concrete executions, if $\lambda_{\ell+1} = (\mathbb{M}_{\ell+1}, t_{\ell+1})$ we denote by $\mathsf{End}(\xi) = \{\mathsf{q} \mid \exists \mathit{fl} \in \{\mathbf{m}, \mathbf{s}\}.\exists b \in \{!!_{ok}, !!_{no}\}.\mathbb{M}_{\ell+1}(\mathsf{q}, \mathit{fl}, b) > 0\}$ the set of states that appear in the end configuration of ξ.

As an example, a possible abstract execution of the broadcast protocol from Figure 2.1 is: $(\langle(\mathsf{q}_0, \mathbf{m}, !!_{ok})\rangle, \varepsilon) \Rightarrow (\langle(\mathsf{q}_0, \mathbf{m}, !!_{no}), (\mathsf{q}_2, \mathbf{m}, !!_{no}), (\mathsf{q}_2, \mathbf{m}, !!_{ok})\rangle, !!)$. This single-step execution represents the fact that among the processes in q_0, some processes will

take an internal action to q_2 and loop there with another internal action (they are represented by the element $(\mathsf{q}_2, \mathbf{m}, !!_{ok})$), others will only move to q_2 taking a single internal action (they are represented by $(\mathsf{q}_2, \mathbf{m}, !!_{no})$), and finally some processes will stay in q_0 (they are represented by $(\mathsf{q}_0, \mathbf{m}, !!_{no})$); note that these processes cannot perform a broadcast, because due to the local strategy hypothesis, they committed to firing the internal action leading to q_2.

Another example of an abstract execution is: $(\langle(\mathsf{q}_0, \mathbf{m}, !!_{ok})\rangle, \varepsilon) \Rightarrow (\langle(\mathsf{q}_0, \mathbf{m}, !!_{ok})\rangle, !!)$ $\Rightarrow (\langle(\mathsf{q}_1, \mathbf{s}, !!_{ok}), (\mathsf{q}_3, \mathbf{m}, !!_{ok})\rangle, \varepsilon) \Rightarrow (\langle(\mathsf{q}_1, \mathbf{s}, !!_{ok}), (\mathsf{q}_3, \mathbf{m}, !!_{no}), (\mathsf{q}_3, \mathbf{m}, !!_{ok})\rangle, \varepsilon)$. Here in the first step, no process performs internal actions, in the second step one of the processes in q_0 broadcasts m, moves to q_1 and we know that no other process will ever share the same history, it is hence represented by $(\mathsf{q}_1, \mathbf{s}, !!_{ok})$; then all the other processes with the same history represented by $(\mathsf{q}_0, \mathbf{m}, !!_{ok})$ must receive m and move to q_3, they are hence represented by $(\mathsf{q}_3, \mathbf{m}, !!_{ok})$. The last step represents the fact that some processes perform the internal action loop on q_3.

The definition of the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ ensures a correspondence between abstract local clique executions and local clique executions in $\mathcal{P}$. Formally:

Lemma 4.1 *Let $\mathsf{q}_F \in \mathrm{Q}$. There exists an abstract local clique execution ξ of $\mathcal{P}$ such that $\mathsf{q}_F \in \mathsf{End}(\xi)$ if and only if there exists a local clique execution $\theta \in \Theta_{\mathcal{LC}}$ such that $\mathsf{q}_F \in \mathsf{End}(\theta)$.*

Given the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, in order to show that Reach$[\mathcal{LC}]$ is decidable, we then rely on the theory of well-structured transition systems [ACJT00, FS01]. Indeed, the natural order on abstract configurations is a well-quasi-order compatible with the transition relation $\Rightarrow$ of $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ (bigger abstract configurations simulate smaller ones) and one can compute predecessors of upward-closed sets of configurations.

Formally, we define a natural order $\preceq$ on the set of abstract configurations Λ as follows: $(\mathbb{M}, t) \preceq (\mathbb{M}', t')$ if and only if $t = t'$ and $\mathbb{M}(\mathsf{q}, \mathit{fl}, b) \leq \mathbb{M}'(\mathsf{q}, \mathit{fl}, b)$ for all $(\mathsf{q}, \mathit{fl}, b)$.

Since the set $\mathrm{Q} \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$ is finite and since the set $\{\varepsilon, !!\}$ is also finite, then Dickson's lemma allows us to say that $(\Lambda, \preceq)$ is a well-quasi-order (wqo). From this we know that for any infinite sequence $(\lambda_i)_{i \in \mathbb{N}} \in \Lambda^{\mathbb{N}}$, there exists $i < j$ such that $\lambda_i \preceq \lambda_j$.

For a set $S \subseteq \Lambda$, we denote by $\uparrow S$ its *upward-closure* (with respect to $\preceq$) defined by $\uparrow S = \{\lambda' \mid \exists \lambda \in S \text{ s.t. } \lambda \preceq \lambda'\}$. A set $S \subseteq \Lambda$ is said *upward-closed* if $S = \uparrow S$. Since $(\Lambda, \preceq)$ is a wqo, for each upward-closed set $S \subseteq \Lambda$ there exists a finite basis $\{b_0, \ldots, b_k\} \subseteq S$ such that $S = \uparrow \{b_0, \ldots, b_k\}$. This provides a way to finitely represent infinite subsets of Λ. We will now show that the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}} = (\Lambda, \Lambda_0, \Rightarrow)$ equipped with the wqo $\preceq$ is a well-structured transition system [ACJT00, FS01] and that one can effectively compute the predecessors of upward-closed sets.

The following monotonicity lemma is immediate given the definition of the transition relation $\Rightarrow$.

Lemma 4.2 (Monotonicity lemma) *Given $\lambda_1, \lambda_2, \lambda_1' \in \Lambda$ such that $\lambda_1 \Rightarrow \lambda_2$ and $\lambda_1 \preceq \lambda_1'$, there exists $\lambda_2' \in \Lambda$ such that $\lambda_1' \Rightarrow \lambda_2'$ and $\lambda_2 \preceq \lambda_2'$.*

Now we define an operator which allows to compute a finite basis for the set of one-step predecessors of an upward-closed set. For a set $S \subseteq \Lambda$, we define $Pre(S) = \{\lambda \in \Lambda \mid \exists \lambda' \in S \text{ s.t. } \lambda \Rightarrow \lambda'\}$. We will now see that given an abstract configuration λ, we can compute (using the definition of $\Rightarrow$) a finite basis of the set $Pre(\uparrow \{\lambda\})$. Let $\lambda = (\mathbb{M}, t)$ be an abstract configuration in λ. We define $p(\lambda)$ as follows:

- if $t =!!$, then $p(\lambda) = \{\lambda' \mid \lambda' \Rightarrow_{\varepsilon} \lambda\}$.
- if $t = \varepsilon$, then

$$p(\lambda) = \{\lambda' \mid \lambda' \Rightarrow_{!!} \lambda\} \cup$$
$$\bigcup_{(\mathsf{q}_1, !!m, \mathsf{q}_2) \in \Delta} \{\lambda' \mid \lambda' \Rightarrow_{!!} (\mathbb{M} \oplus \langle (\mathsf{q}_2, \mathbf{s}, !!_{ok}) \rangle, \varepsilon)\} \cup$$
$$\bigcup_{(\mathsf{q}_1, !!m, \mathsf{q}_2), (\mathsf{q}_1, ??m, \mathsf{q}_3) \in \Delta} \{\lambda' \mid \lambda' \Rightarrow_{!!} (\mathbb{M} \oplus \langle (\mathsf{q}_2, \mathbf{s}, !!_{ok}), (\mathsf{q}_3, \mathbf{m}, !!_{ok}) \rangle, \varepsilon)\}$$

Let us explain the second part of this definition. Since our aim is to compute a basis of $Pre(\uparrow \{\lambda\})$, in order to compute the predecessors of $\uparrow \{\lambda\}$, we need to take into account for the transition relation $\Rightarrow_{!!}$, that the element that witnesses the broadcast might not be in λ but in a configuration belonging to its upward closure. This is the reason why we include the sets $\bigcup_{(\mathsf{q}_1, !!m, \mathsf{q}_2) \in \Delta} \{\lambda' \mid \lambda' \Rightarrow_{!!} (\mathbb{M} \oplus \langle (\mathsf{q}_2, \mathbf{s}, !!_{ok}) \rangle, \varepsilon)\}$ and $\bigcup_{(\mathsf{q}_1, !!m, \mathsf{q}_2), (\mathsf{q}_1, ??m, \mathsf{q}_3) \in \Delta} \{\lambda' \mid \lambda' \Rightarrow_{!!} (\mathbb{M} \oplus \langle (\mathsf{q}_2, \mathbf{s}, !!_{ok}), (\mathsf{q}_3, \mathbf{m}, !!_{ok}) \rangle, \varepsilon)\}$. This kind of assumptions to compute the predecessor basis of an upward closed set is similar to the one proposed in [EFM99] to solve Reach[$\mathcal{C}$]. Note however that, for the transition relation $\Rightarrow_{\varepsilon}$, such trick is not necessary. Note also that given a configuration $\lambda \in \Lambda$, $p(\lambda)$ is finite since it contains abstract configurations, where the cardinality of the multiset is at most the cardinal of the multiset of λ plus 2. Using the definition of $\Rightarrow$, one obtains the following lemma.

Lemma 4.3 *For all $\lambda \in \Lambda$, $p(\lambda)$ is finite, effectively computable and $\uparrow p(\lambda) = Pre(\uparrow \{\lambda\})$.*

We consider the following upward closed set: $F = \bigcup_{(q_F, fl, b)} \uparrow \{(\langle (q_F, fl, b) \rangle, \varepsilon)\} \cup \uparrow \{(\langle (q_F, fl, b) \rangle, !!)\}$. Using the methodology presented in [ACJT00, FS01], thanks to Lemmas 4.3 and 4.2, we know it is possible to compute a finite basis for the set $Pre^*(F) = \{\lambda \in \Lambda \mid \exists \lambda' \in F \text{ s.t. } \lambda \Rightarrow^* \lambda'\}$. Hence we can decide whether there exists an abstract local clique execution ξ of $\mathcal{P}$ such that $\mathsf{q}_F \in \mathsf{End}(\xi)$: it suffices to test whether $\lambda_0 \in pre^*(F)$. Applying Lemma 4.1, we conclude that Reach[$\mathcal{LC}$] is decidable.

Theorem 4.2 *The problem Reach[$\mathcal{LC}$] restricted to complete protocols is decidable and non-primitive recursive.*

Proof We also show that Reach[$\mathcal{LC}$] is non-primitive recursive thanks to a PTIME reduction from Reach[$\mathcal{C}$] (which is Ackermann-complete [SS13]) to Reach[$\mathcal{LC}$]. We exploit the fact that the only difference between the semantics $\mathcal{C}$ and $\mathcal{LC}$ is that in the latter, processes with the same history take the same decision. We simulate this in $\mathcal{C}$ with a gadget which assigns a different history to each individual process at the

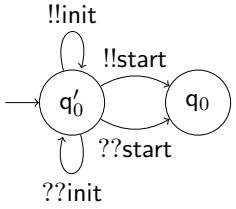


Figure 4.12: Initialization gadget.

beginning of the protocol, hence making the reachability problem for $\mathcal{C}$ equivalent to the one with $\mathcal{LC}$ semantics.

Intuitively the only difference between the semantics $\mathcal{C}$ and $\mathcal{LC}$ is that within $\mathcal{LC}$, processes with the same history take the same decisions. However, with the simple initialization gadget, represented in Fig. 4.12, one can assign a different history to each individual process, before they actually start the protocol. Indeed, in a clique topology, if a process performs a broadcast, it has a unique history forever. Hence to have k processes in q_0 with a different history, it suffices to perform k broadcasts of init and then one broadcast of start making all the processes move to q_0 thanks to the clique topology.

This provides a PTIME reduction from $\textsc{Reach}[\mathcal{C}]$ to $\textsc{Reach}[\mathcal{LC}]$. Note that this reduction also works for the target problem. Let us formally define the construction illustrated in Fig. 4.12. Given a protocol $\mathcal{P} = (\mathrm{Q}, \mathsf{q}_0, \Sigma, \Delta)$ we let $\mathcal{P}' = (\mathrm{Q}', \mathsf{q}'_0, \Sigma', \Delta')$ with

- $\mathrm{Q}' = \mathrm{Q} \cup \{\mathsf{q}'_0\}$;
- $\Sigma' = \Sigma \cup \{\mathsf{init}, \mathsf{start}\}$;
- $\Delta' = \Delta \cup \{(\mathsf{q}'_0, !!\mathsf{init}, \mathsf{q}'_0), (\mathsf{q}'_0, ??\mathsf{init}, \mathsf{q}'_0), (\mathsf{q}'_0, !!\mathsf{start}, \mathsf{q}_0), (\mathsf{q}'_0, ??\mathsf{start}, \mathsf{q}_0)\}$

We now show that reaching q_F under clique semantics in $\mathcal{P}$ is equivalent to reaching q_F under local strategies and clique semantics in $\mathcal{P}'$. Formally, we prove that there exists an execution $\theta \in \Theta_{\mathcal{C}}[\mathcal{P}]$ such that $\mathsf{q}_F \in \mathsf{Reach}(\theta)$ if and only if there exists an execution $\theta' \in \Theta_{\mathcal{LC}}[\mathcal{P}']$ such that $\mathsf{q}_F \in \mathsf{Reach}(\theta')$.

($\Leftarrow$) Let $\theta' \in \Theta_{\mathcal{LC}}[\mathcal{P}']$ be a local clique execution of $\mathcal{P}'$ such that $\mathsf{q}_f \in \mathsf{Reach}(\theta')$. From the definition of $\mathcal{P}'$, this execution must start with a series of broadcasts of init followed by the broadcast of start. We define θ as the suffix of execution θ' after the broadcast of start. Notice that $\theta \in \Theta_{\mathcal{C}}[\mathcal{P}]$ is a clique execution of $\mathcal{P}$, and that $\mathsf{q}_f \in \mathsf{Reach}(\theta') = \mathsf{Reach}(\theta) \cup \{\mathsf{q}'_0\}$.

($\Rightarrow$) Let $\theta \in \Theta_{\mathcal{C}}[\mathcal{P}]$ be a clique execution of $\mathcal{P}$ such that $\mathsf{q}_f \in \mathsf{Reach}(\theta)$. In the local clique semantics for protocol $\mathcal{P}'$ we define the following execution θ'. First, each process broadcasts the message init in turn, then one process broadcasts start. Recall that, in local clique executions, when a process performs a broadcast, its history becomes unique forever. Therefore, at this stage, each of the processes has its own history. Moreover, by definition of $\mathcal{P}'$, all the processes are in state q_0. Hence, from then on, one can

reproduce execution θ. This execution θ' is thus a local clique execution in $\mathcal{P}'$ such that $\mathsf{q}_f \in \mathsf{Reach}(\theta')$. □

5 Conclusion

We investigated reconfigurable broadcast networks under the new hypothesis of local strategies. The local strategies ensure that two processes with the same knowledge perform the same actions. We have shown that the parameterized reachability question, under the local strategy assumption, is NP-complete. The proof is based on a new tool, strategy patterns, that allows to give finite representations of local strategies. We also introduce admissible strategy patterns that are patterns with ordered nodes. If a node is smaller than an other one, it means that the smaller node can be visited before by an execution respecting a local strategy. Moreover, we have then shown that such patterns can be reduced to polynomial size admissible patterns while keeping the same set of reachable states. This yields a polynomial time algorithm for the parameterized reachability question consisting in guessing a strategy pattern of polynomial size, and checking whether it is admissible, and if it contains the target state. The NP-hardness of this problem is shown by reduction of 3-SAT.

We also considered the parameterized synchronization problem in reconfigurable broadcast networks under local strategies. Using again the idea of admissible strategy patterns, we introduced co-admissible patterns which are equipped with a second ordering on the nodes representing the order in which the nodes can be emptied in a local execution. Strategy patterns with these two orders are called bi-admissible if they are admissible and co-admissible. We were able to show that polynomial size witnesses are sufficient for bi-admissible strategy patterns, yielding an NP algorithm. The synchronization problem is in fact NP-complete, since it is harder than the reachability problem.

In addition to these decidability results, the bound on the size of the admissible and bi-admissible strategy patterns allowed us to derive polynomial bounds on the minimal number of processes required to fulfill the reachability and synchronization problems. Since all the runs of a reconfigurable networks can be simulated by a bigger network by ignoring the additional processes, we therefore obtain a cut-off on the number of processes required to fulfill the reachability and synchronization problems.

In addition to reconfigurable broadcast protocol networks, we also studied clique networks in which the messages are received by all the processes. Interestingly, the parameterized reachability problem which is decidable and non-primitive recursive in clique protocols, becomes undecidable with the assumption of local strategies. This is shown thanks to a reduction from the halting problem for 2-counter machines [Min67]. In this proof, we use the clique topology and the parametric number of processes to implement the counters, but only that is not enough to implement a test to zero since nothing can force a process to notify that the counter is not zero. However, relying on non-deterministic guesses for zero tests that discard the processes if the value of the counter was not zero, and using the locality assumption in order to run twice

the 'same' simulation of the 2-counter machine, we can force, by using only the non-discarded process for the second run, the simulation to be correct. Thus, we proved the undecidability of the parameterized reachability problem with locality assumption. Notice that, when considering the synchronization problem, one can directly ensure that the simulation is correct by forbidding discarded processes in the final configuration. We thus obtain the undecidability of the synchronization problem in clique networks already without the locality assumption.

However, notice that the trick of running twice the simulation is only possible if some process can wait in a particular state without getting any new information, hence without receiving any messages. We thus investigated a restriction on protocols to input complete protocols, in which all messages can be received from any state. This way, we cannot ensure to simulate twice the counter machine exactly in the same way. In fact under this additional assumption, the parameterized reachability question becomes decidable and non-primitive recursive. The decidability proof is based on well structured transition systems for which the reachability problem can be solved by a backward computation [ACJT96, AJ01, FS01].

We strongly believe that one could reuse the idea of strategy patterns in order to check more difficult properties like repeated reachability. This could be done by adding a loop at the end of the branches of the patterns. More generally, it would be interesting to see if we can combine local strategies and the game networks presented in the previous chapter (see Chapter IV). This is a promising way to tackle the probabilistic parameterized problems restricted to local strategies.

Conclusion

Summary

In this thesis, we investigated the verification of networks in which the number of processes is a parameter. We considered the parameterized reachability question asking whether one can reach a configuration in which at least one process is in a given state, and the parameterized synchronization problem, asking whether one can reach a configuration where all processes are in a given set of states.

Chapter III In Chapter III, we gave a timed and probabilistic extension of broadcast networks studied in [DSZ11a]. We have shown in Theorems 3.1 and 3.2 that some reachability problems are decidable in clique networks of probabilistic timed protocols. The proofs rely on a monotonicity result allowing to focus on a network of size one and to the non-probabilistic case, which is known to be decidable. However, all the other cases are undecidable, see Theorems 3.3, 3.4, and 3.5 for reachability and Theorem 3.6 for synchronization.

Then we considered dynamic networks, *i.e.* networks in which the number of processes evolves along the computation according to probabilistic distributions. We have shown that the qualitative problems are decidable for dynamic networks thanks to four key properties: 1. one can abstract the time by an appropriate region abstraction (Proposition 4.2); 2. the region abstraction enjoys the finite attractor property (Proposition 4.3); 3. there exists a well-quasi-order on region-configurations (Proposition 4.4); 4. the predecessor operator is effectively computable and preserves upward closure (Proposition 4.5). With these properties, we can reuse techniques developed for non-deterministic probabilistic lossy channel systems [BBS06b] to show the decidability of the parameterized qualitative reachability and synchronization problems. Notice that the termination of the decision procedure is obtained via the classical backward computation on well-structured transition systems and is thus of high complexity. Yet, we also provided a reduction from lossy channel systems to dynamic networks to show that these problems are indeed non primitive recursive, see Theorem 4.7.

As future work it would be interesting to investigate the open problems left in this chapter. In particular, the quantitative analysis in dynamic clique networks is still open. There is hope for the decidability there since we have seen that the qualitative problems are decidable, and even more that we can compute the set of all region-configurations satisfying a qualitative property. In this direction, the only non-parameterized result

of this thesis is Theorem 4.5, giving an approximation procedure for the minimal probability to reach a state for a given fixed number of processes. This approximation is computed thanks to an adaptation of the approximation scheme for fully probabilistic lossy channel systems [IN97, Rab03, ABRS05]. This extension to non-determinism is non trivial since we possibly have to deal with an unbounded number of schedulers, whereas in the previous works the models were fully probabilistic. This approximation result is interesting in itself since it relies only on general properties, such as the effectiveness of the predecessor operator, the effectiveness of computing the set of states from where we can avoid the target with probability one, and the finite attractor. In other words, it can *a priori* be generalized to a larger class of models, in particular to probabilistic lossy channel systems with insertions.

The implementation, currently under work, of the algorithms given in this chapter would allow to give to the parameterized reachability verification problem a practical interest, in addition to a theoretical interest. Indeed, the complexity of this problem is high, however since it allows to give guaranties for networks of all possible sizes, the complexity is still better than the approach consisting of testing all possible sizes. Moreover, the high complexity is a worst case scenario, so most likely the implementation would be faster, and case studies would be manageable.

Chapter IV In Chapter IV, we considered an other topology for the networks in which the messages do not reach all the processes but only a subset of processes chosen in a non-deterministic way. This model of selective broadcast networks of probabilistic protocols are a probabilistic extension of the reconfigurable broadcast protocol studied in [DSTZ12]. In this chapter, we restricted our investigation to qualitative reachability problems.

As a first result, we showed in Theorem 4.1 that some parameterized reachability problems in selective broadcast networks of probabilistic protocols are decidable. We again use a key monotonicity result stating that adding processes in the network may only increase the probability to reach a target state since, thanks to reconfigurations one can always leave apart the additional processes and simulate a smaller network. This result allows us to reduce to networks composed of a single process.

In order to solve the other problems, we introduced selective broadcast networks of parity protocols which are parameterized distributed games. In these games, we proved that player 2 has a counter-strategy for the parity or safety parity winning condition if and only if there exists a really simple state-based counter-strategy, see Proposition 3.1. The parameterized game problem thus boils down to checking whether there exist a network size and a strategy for player 1 winning against all state-based strategies for player 2. Moreover, we provided a reduction to parameterized VASS [KS88] allowing us to, given a state-based strategy for player 2, decide in polynomial time whether player 1 has a winning strategy. We thus obtained a co-NP algorithm (see Theorem 3.1) consisting in guessing a counter state-based strategy for player 2 and checking whether it is really a counter-strategy.

Finally we provided reductions of the parameterized probabilistic problems to parameterized game problems by modeling the probabilistic choices by choices of player 2.

We provided a polynomial reduction for each case, tuning the parity on the transition in order to reflect the specificity of each case. For example, to solve the almost sure reachability problem, the reduction consists in letting player 2 perform a finite number of choices, by setting the parity of these transitions even, and then checking whether player 1 has a strategy to reach the target by allowing him to perform the other choices but with odd parity. We also proved the co-NP hardness of these problems thanks to reduction of the unsatisfiability problem of conjunctive normal form formulas. These hardness results also give the hardness of the game problem. We thus obtain that the game problem as well as the reachability problems are co-NP-complete (Theorems 4.3, 4.4, and 4.5).

We believe that distributed games are an interesting approach for the many identical process setting and it would be interesting to study them further. In particular, one assumption made in this chapter is that there is no deadlock configuration. It would be interesting to see if the decidability and complexity results still hold without this assumption. Moreover, since in finite state systems the translation from probabilities to parity allows to solve problems much harder than reachability, it would not be surprising if that would be the case here. Our proof of decidability for the game problem relies on the difference on the power of player 1 and player 2. Even though giving to player 2 the possibility to perform broadcasts and choose processes will certainly lead to undecidability, it would be interesting to see to what extent we can extend the power of player 2 while keeping decidability.

For the probabilistic parameterized problems, we only considered qualitative properties. We are currently investigating the quantitative problems and it seems that, for some cases, the monotonicity of the networks allows us to reduce the quantitative problems to the qualitative ones. We believe that this monotonicity may hold for the other problems as well but for 'large enough' networks. In order to validate this intuition we are implementing a tool in C that allows us to build networks and check the desired properties thanks to the model checker Prism [KNP11].

Chapter V In Chapter V, we investigated reconfigurable broadcast networks under the new hypothesis of local strategies. The local strategies ensure that two processes with the same knowledge perform the same actions. We have shown that the parameterized reachability question, under the local strategy assumption, is NP-complete. The proof is based on a new tool, admissible strategy patterns, that allows to give a finite representation of local strategies. We have shown that such patterns can be reduced to polynomial size admissible patterns while keeping the same set of reachable states (see Proposition 3.2). This yields a polynomial algorithm for the parameterized reachability question consisting in guessing a witness strategy pattern of polynomial size and checking whether it allows to reach the target state. The NP-hardness of this problem is shown by reduction of 3-SAT (Theorem 3.2).

We also considered the parameterized synchronization problem in reconfigurable broadcast networks. Using again the idea of admissible strategy patterns, we introduced bi-admissible patterns. We were able to show that polynomial size witnesses are also enough for bi-admissible strategy patterns to decide the synchronization problem, thus

yielding an NP algorithm. The synchronization problem is in fact NP-complete, since it is harder than the reachability problem, see Theorem 3.3.

In addition to this decidability result, the bound on the size of the admissible and bi-admissible strategy patterns allowed us to derive a polynomial bound on the minimal number of processes required to fulfill the reachability and synchronization problems. Moreover, since all the runs of a reconfigurable network can be simulated by a bigger network by ignoring the additional processes, we obtain a cut-off on the number of processes required to fulfill the reachability and synchronization problems.

In addition to reconfigurable broadcast protocol networks, we also studied clique networks in which the messages are received by all the processes. Interestingly, the parameterized reachability problem, which is decidable and NPR in clique protocols, becomes undecidable with the assumption of local strategy, see Theorem 4.1. However, considering a restriction on protocols, namely input-completeness, requiring that all messages can be received from any state, we show in Theorem 4.2 the parameterized reachability problem to be decidable and non-primitive recursive. The proof of decidability is based on a counting abstraction shown to be a well structured transition system.

We strongly believe that one could reuse the idea of strategy patterns in order to check more difficult properties like repeated reachability. This could be done by adding a loop at the leaves of the patterns. More generally, it would be interesting to see if we can combine local strategies and the game networks presented in Chapter IV. This is a promising way to tackle the probabilistic parameterized problems restricted to local strategies.

Future works

In addition to the possible future works already presented we propose here some leads that may be of interest to continue this work.

Communication topology In this thesis, we studied selective broadcast networks where the set of receivers of a message is chosen non-deterministically. It would be interesting to study these networks under a new assumption stating for example that once a link is used it stays active for some time. Such restrictions could be investigated by, for example, modeling the topology by a graph and only allowing rewriting of the graph topology according to fixed rules. An other possibility is to consider probabilistic failures of the links, as well as probabilistic creations of the links modeling failures and maintenance of the links. The shape of the topology is of major importance for the decidability and complexity status of parameterized verification, see *e.g.* [DSZ11a]. It would be interesting to investigate other topologies than cliques and reconfigurable broadcast with probabilistic protocols to see the computational power that probabilities bring.

An other interesting extension would be to consider protocols with registers as it was done in [DST13]. Indeed it is common for peer to peer applications to maintain a

table containing a subset of the other participants addresses. To verify such networks it is thus useful to consider processes with identities and registers allowing to manipulate those identities. In [DST13] it is shown that with two registers and two fields in the messages allowing to transmit data, the parameterized reachability problem is undecidable. Indeed, one can use the two fields in the messages to specify the sender and receiver and thus link processes in an unbounded tape. However, limiting the field in the messages to only one leads to decidability. An other possibility is to consider communicating register automata [AAKR15] which are communicating automata with a finite set of registers storing ids and allowed to create new processes. The processes can communicate with processes whose ids they knows and the messages are stored in unbounded FIFO buffers. In [AAKR15], the reachability problem was shown to be undecidable. However, restricting to bounded FIFO buffers and considering that the simple paths in the underlying comunication topology are bounded by a constant, one can regain decidability with the assumption that any link can be disconnected non-deterministically in any state. Instead of considering the disconnection of processes as non-deterministic it would be interesting to see them as random failure of the registers, and thus extend the model of communicating register automata with probabilities.

Fairness In this thesis, we always considered strategies that choose in a non-deterministic way the next process to perform an action. However, this leads to solutions where some processes are totally left apart, which is not really realistic. An interesting work to do in the future is to consider restrictions on the way the next process to play is chosen. A first approach could be to look at fair schedulers where each process plays infinitely often in all infinite executions. Note that fair strategies do not really make sense for the problem we presented since we looked only at finite properties, however this would be interesting for more elaborate properties such as repeated reachability. With the same idea, since we looked at probabilistic models, it would be interesting to look at probabilistic choices for the next process to play, and to investigate the decidability and complexity status of the parameterized problems when considering that the processes are picked uniformly at random. In multithreaded programs Round Robin schedulers, where each process performs an action one after the other in a given order, are often used. Such Round Robin schedulers would certainly lead to undecidability in a clique topology since it would allow to implement a test to zero by using one round to check whether there are some processes in a given state or not. However, it would be interesting to see the decidability and complexity status of the parameterized problems in reconfigurable networks under this assumption. Indeed, since processes are forced to play at each round, the key monotonicity property would no longer hold.

Unifying model for parameterized networks The many identical processes field has been studied under many different settings in the past years. There are works considering different topologies such as clique bounded graphs [DSZ11a], trees [AAR13], and different communication means such as broadcast communication [EFM99], token-passing [CTTV04, AJKR14], message passing [BGS14] or shared memory [EGM13]. However, these works lack of an unifying model or at least meta model that could

subsume all the present works in order to have a clear view of the power of parameterized systems with many identical processes. Indeed some of the proofs given for different models share some similarities, as an example the proof of decidability of reachability for shared memory processes and reconfigurable broadcast networks both rely on the fact that once a state is reached it can be reached by any number of processes. This property seems to be a key property for decidability and low complexity of the parameterized verification. It would be nice to have an overview of such kind of properties that could somehow unify the field. A first step was made by Esparza in its survey [Esp14] by sorting models by whether they can produce a leader or not and whether every process must listen or not. It would be interesting to have a broader classification including all the existing parameterized networks.

Implementation and case studies An other need in the field is tool support. Indeed, there are lots of different works on the theoretical part, however there is a lack of actual case studies. This is partly due to the fact that currently the algorithms used in distributed systems are proved by hand and thus are set in classes of models that are too powerful for automatic verification. A way to tackle this problem is to consider approximations. However, the flaw of approximations is that you cannot give guarantees on the behaviors of the systems.

Synthesis of distributed algorithm An other approach is to reverse the problem and consider synthesis. Instead of trying to certify systems already implemented that lead to complex models, the synthesis automatically builds algorithms that achieve some given properties. Maybe an other way to approach the problem would be to enumerate the possible models and check them for given properties until one is found that satisfies the properties. Indeed, the tools available for developers and conceptors of distributed algorithms lead to man readable solution but, most of the time, that cannot be automatically verified. Maybe, enumeration of models would lead to less 'human friendly', but automatically certified to be correct, solutions.

Bibliography

[AAKR15] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmet Kara, and Othmane Rezine. Verification of buffered dynamic register automata. In *NETYS 2015, May 11–13, Agadir, Morocco*. Springer Berlin/Heidelberg, 2015.

[AaP] Aapal website. `http://lit2.ulb.ac.be/aapal/`.

[AAR13] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Othmane Rezine. Verification of directed acyclic ad hoc networks. In *FMOODS/FORTE'13*, volume 7892 of *LNCS*, pages 193–208. Springer, 2013.

[ABRS05] Parosh Aziz Abdulla, Nathalie Bertrand, Alexander Rabinovich, and Philippe Schnoebelen. Verification of probabilistic systems with faulty communication. *Information and Computation*, 202(2):141–165, 2005.

[ACJT96] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE Computer Society, 1996.

[ACJT00] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Automata, languages and programming*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[AdFE15] Luca Aceto and David de Frutos Escrig. Verification of population protocols. In *26th International Conference on Concurrency Theory (CONCUR 2015)*, volume 42, pages 470–482. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[ADM04] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 345–354. IEEE, 2004.

[ADR+11] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *Formal Modeling and Analysis of Timed Systems*, pages 256–270. Springer, 2011.

[AHM07] Parosh Aziz Abdulla, Noomene Ben Henda, and Richard Mayr. Decisive Markov chains. *Logical Methods in Computer Science*, 3(4), 2007.

[AHV93] Rajeev Alur, Thomas A Henzinger, and Moshe Y Vardi. Parametric real-time reasoning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 592–601. ACM, 1993.

[AJ93] Parosh Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 160–170. IEEE, 1993.

[AJ01] Parosh Aziz Abdulla and Bengt Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science*, 256(1):145–167, 2001.

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.

[AJKR14] Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281, 2014.

[AK86] Krzysztof R Apt and Dexter C Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.

[BBS06a] Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen. A note on the attractor-property of infinite-state Markov chains. *Information Processing Letters*, 97(2):58–63, 2006.

[BBS06b] Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen. Symbolic verification of communicating systems with probabilistic message losses: liveness and fairness. In *Formal Techniques for Networked and Distributed Systems-FORTE 2006*, pages 212–227. Springer, 2006.

[BBS07] Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Transactions on Computational Logic*, 9(1), 2007.

[BF13] Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *FSTTCS*, volume 13, pages 501–513, 2013.

[BFS14] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *Foundations of Software Science and Computation Structures*, pages 134–148. Springer, 2014.

[BFS15] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Distributed local strategies in broadcast networks. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 44–57, 2015.

[BGS14] Benedikt Bollig, Paul Gastin, and Jana Schubert. Parameterized verification of communicating automata under context bounds. In *RP'14*, volume 8762 of *LNCS*, pages 45–57, 2014.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BKS05] Tomás Brázdil, Antonín Kucera, and Oldrich Strazovský. On the decidability of temporal properties of probabilistic pushdown automata. In *STACS'05*, volume 3404 of *LNCS*, pages 145–157. Springer, 2005.

[BS03] Nathalie Bertrand and Philippe Schnoebelen. Model checking lossy channels systems is probably decidable. In *Foundations of Software Science and Computation Structures*, pages 120–135. Springer, 2003.

[BS13] Nathalie Bertrand and Philippe Schnoebelen. Computable fixpoints in well-structured symbolic model checking. *Formal Methods in System Design*, 2013. To appear.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.

[CD12] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012.

[CdAFL09] Krishnendu Chatterjee, Luca de Alfaro, Marco Faella, and Axel Legay. Qualitative logics and equivalences for probabilistic systems. *Logical Methods in Computer Science*, 5(2), 2009.

[CS08] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *LICS*, volume 8, pages 205–216, 2008.

[CTTV04] Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *CONCUR'04*, volume 3170 of *LNCS*, pages 276–291, 2004.

[Daw05] Conrado Daws. Symbolic and parametric model checking of discrete-time markov chains. In *Theoretical Aspects of Computing-ICTAC 2004*, pages 280–294. Springer, 2005.

[DJJ+15] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. Prophesy: A probabilistic parameter synthesis tool. In *Computer Aided Verification*, pages 214–231. Springer, 2015.

[DST13] Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of broadcast networks of register automata. In *Reachability Problems*, pages 109–121. Springer, 2013.

[DSTZ12] Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR 2010-Concurrency Theory*, pages 313–327. Springer, 2010.

[DSZ11a] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Proc. 14th Int. Conference on Foundations of Software Science and Computational Structures (FoSSaCS'11)*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

[DSZ11b] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of safety properties in ad hoc network protocols. In *Proc. 1st Int. Workshop on Process Algebra and Coordination (PACO'11)*, volume 60 of *EPTCS*, pages 56–65, 2011.

[DSZ12] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Verification of ad hoc networks with node and communication failures. In *Proc. joint 14th IFIP WG 6.1 Int. Conference and 32nd IFIP WG 6.1 Int. Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE'12)*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.

[EFM99] Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Computer Society, 1999.

[EGM13] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140, 2013.

[Esp14] Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[EY05] Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. In *ICALP'05*, volume 3580 of *LNCS*, pages 891–903. Springer, 2005.

[Fru06] Matthias Fruth. Probabilistic model checking of contention resolution in the ieee 802.15. 4 low-rate wireless personal area network protocol. In *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, pages 290–297. IEEE, 2006.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[GS92] Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

[Hag11] Matthew Hague. Parameterised pushdown systems with non-atomic writes. *arXiv preprint arXiv:1109.6264*, 2011.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183–220, 2002.

[IN97] Purush Iyer and Murali Narasimha. Probabilistic lossy channel systems. In *TAPSOFT'97: Theory and Practice of Software Development*, pages 667–681. Springer, 1997.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.

[KNPS08] Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, chapter Verification of Real-Time Probabilistic Systems, pages 249–288. John Wiley & Sons, 2008.

[KS88] S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In *STOC'88*, pages 398–406. ACM, 1988.

[KSK66] John G Kemeny, J Laurie Snell, and Anthony W Knapp. Denumerable markov chains. the university series in higher mathematics, 1966.

[LPY97] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[Min67] Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS'90*, pages 746–757. IEEE Computer Society, 1990.

[Rab03] Alexander Rabinovich. Quantitative analysis of probabilistic lossy channel systems. In *Automata, Languages and Programming*, pages 1008–1021. Springer, 2003.

[RL94] Michael O Rabin and Daniel Lehmann. The advantages of free choice: A symmetric and fully distributed solution for the dining philosophers problem. In *A classical mind*, pages 333–352. Prentice Hall International (UK) Ltd., 1994.

[Spe] ZigBee Specification. v1. 0: Zigbee specification (2005). *San Ramon, CA, USA: ZigBee Alliance*.

[SS13] Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *CONCUR'13*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013.

[Var85] Moshe Y Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 327–338. IEEE, 1985.

List of Figures

Résumé

Ce travail s'inscrit dans le cadre de la vérification formelle de programmes. La vérification de modèle permet de s'assurer qu'une propriété est vérifiée par le modèle du système. Cette thèse étudie la vérification paramétrée de réseaux composés d'un nombre non borné de processus identiques où le nombre de processus est considéré comme un paramètre.

- Concernant les réseaux de protocoles probabilistes temporisés nous montrons que les problèmes de l'accessibilité et de synchronisation sont indécidables pour des topologies de communication en cliques. Cependant, en considérant des pertes et créations probabiliste de processus ces problèmes deviennent décidables.
- Pour ce qui est des réseaux dans lequel les messages n'atteignent qu'une sous partie des composants choisie de manière non-déterministe, nous prouvons que le problème de l'accessibilité paramétrée est décidable grâce à une réduction à un nouveau modèle de jeux à deux joueurs distribué pour lequel nous montrons que l'on peut décider de l'existence d'une stratégie gagnante en co-NP.
- Finalement, nous considérons des stratégies locales qui permettent d'assurer que les processus effectuent leurs choix non-déterministes uniquement par rapport a leur connaissance locale du système. Sous cette hypothèse de stratégies locales, nous prouvons que les problèmes de l'accessibilité et de synchronisation paramétrées sont NP-complet.

Abstract

This thesis deals with formal verification of distributed systems. Model checking is a technique for verifying that the model of a system under study fulfills a given property. This PhD investigates the parameterized verification of networks composed of many identical processes for which the number of processes is the parameter.

- Considering networks of probabilistic timed protocols, we show that the parameterized reachability and synchronization problems are undecidable when the communication topology is a clique. However, assuming probabilistic creation and deletion of processes, the problems become decidable.
- Regarding selective networks, where the messages only reach a subset of the components, we show decidability of the parameterized reachability problem thanks to reduction to a new model of distributed two-player games for which we prove decidability in co-NP of the game problem.
- Finally, we consider local strategies that enforce all processes to resolve the nondeterminism only according to their own local knowledge. Under this assumption of local strategy, we were able to show that the parameterized reachability and synchronization problems are NP-complete.