COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Nicolas WALDBURGER

Parameterized verification of distributed shared-memory systems

Thèse présentée et soutenue à Rennes, le 11 décembre 2024
Unité de recherche : IRISA

Rapporteurs avant soutenance :

Javier ESPARZA Professeur, Technische Universität München
Antonín KUČERA Professeur, Masaryk University, Brno

Composition du Jury :

Président :	Igor WALUKIEWICZ	Directeur de recherche, CNRS, LaBRI, Université de Bordeaux
Examinatrice :	Patricia BOUYER-DECITRE	Directrice de recherche, CNRS, LMF, ENS Paris-Saclay
Examinateur :	Javier ESPARZA	Professeur, Technische Universität München
Examinateur :	Antonín KUČERA	Professeur, Masaryk University, Brno
Examinatrice :	Nathalie SZNAJDER	Maîtresse de conférence, Sorbonne Université
Examinateur :	Igor WALUKIEWICZ	Directeur de recherche, CNRS, LaBRI, Université de Bordeaux
Directrice de thèse :	Nathalie BERTRAND	Directrice de recherche, INRIA, Université de Rennes
Co-directeur de thèse :	Nicolas MARKEY	Directeur de recherche, CNRS, IRISA, Université de Rennes
Encadrant :	Ocan SANKUR	Chargé de recherche, CNRS, IRISA, Université de Rennes

TABLE OF CONTENTS

REMERCIEMENTS

Scientific Acknowledgments

I start by thanking my supervisors, Nathalie Bertrand, Nicolas Markey and Ocan Sankur, for providing me with a promising PhD topic and for the numerous scientific discussions and advice. I thank my reviewers, Javier Esparza and Antonin Kučera, for reading my thesis and for their kind feedback. I also the other members of my jury: Patricia Bouyer, Nathalie Sznajder and Igor Walukiewicz. I thank all my co-authors. In particular, I thank Corto Mascle and Chana Weil-Kennedy for the numerous discussions and works. I thank Pierre Ganty for welcoming me for a research stay in Madrid. I thank Mahsa Shirmohammadi for the work on one-counter automata and for all the scientific advice. I thank Sylvain Schmitz for the discussions related to various topics related to well-quasi-orders. I thank Arnaud Sangnier for encouraging me to work with Corto and Lucie and for pointing us to broadcast networks of register automata. I thank Mathias Rousset, Frédéric Cérou and Bruno Sericola for the discussions on stochastic processes and on random walks. I thank the Autobóz participants for their inputs, and in particular Petra Wolf for the $W[2]$-hardness. Finally, I thank Tali Sznajder and Benedikt Bollig for taking part in my CSI.

Remerciements Personnels

Mes premiers remerciements vont à mes encadrant·es, Nathalie, Nicolas et Ocan. Je vous remercie pour votre soutien le long de ces trois ans. Vous m'avez initié au monde de la recherche, me permettant d'y appréhender les codes et d'y trouver petit à petit ma place. Vous avez été tous les trois à l'écoute, toujours prêt·es à trouver un créneau pour une réunion mais également à me laisser voler de mes propres ailes, ce qui m'a permis de rendre ma thèse plus variée et plus enrichissante. Nico et Ocan, puisque vous êtes vous-même partis battre des ailes en dehors de la recherche académique, je vous souhaite un bon vol.

J'ai été heureux de faire ces trois ans au sein de l'équipe SUMO, devenue DEVINE. Merci à mon fidèle co-bureau Aymeric pour ta bonne humeur quotidienne et pour les conversations absurdes dont tu as le secret et qui laissent généralement Nathalie dans une certaine perplexité amusée (de plus, tu es indéniablement le roi). Merci à Emily pour tes efforts pour m'accueillir

dans le labo et à Rennes (les mochis c'est plus joli en bleu). Merci aux autres doctorants passés et présents de l'équipe : Léo, Suman, Abdul, Bastien, Antoine, Mathieu, Victorien et Luca. Merci aux deux post-docs du plat pays, Aline et Gaëtan, qui viennent nous envahir de septante, de tantôt et de bonne humeur. Du côté des permanents, je commence par remercier Thierry et Nathalie pour la joie que vous mettez aux pauses café et pour l'invention de ce génial nom d'équipe (devine à quoi je fais référence ?). Merci aux autres permanents : Loïc H., Uli, et les anciens partis vers d'autres horizons comme Nicolas M. et Ocan. J'en profite pour remercier Nicolas M., Nathalie et Aymeric d'avoir supporté à plusieurs reprises mon incapacité chronique à me replacer au padel. Merci Loïc G. et Julie, vous avez bien dynamisé l'ambiance de l'équipe en quelques mois. Merci Laurence pour ta gentillesse et pour toute l'aide que tu m'as apporté dans mes démarches administratives.

J'ai tressé pendant ces trois ans des liens hors de l'équipe DEVINE. Je commence par remercier mes plus fidèles camarades de route : Jean-Loup et Théo. Sans vous, ces trois années auraient pu être un peu trop solitaires. Merci à tous les deux pour tous les moments de détente, les déjeuners à Matcha, les pauses dans l'annexe de la cafet pour pouvoir discrètement râler sur nos collègues (en toute bienveillance, bien entendu) et tout le reste. Une dédicace spéciale aux digressions interminables de Jean-Loup (les marseillais le détestent) et aux phrases magiquement inhabituelles de Théo (oui je sais, tu fais pas exprès mais ne t'inquiète pas on t'aime pour ça). Merci aux autres membres du platypus et plus spécifiquement à Santiago, Victoire, Vincent (vos départs ont porté plusieurs coups durs à l'ambiance du couloir d'à côté) et Pierre (laisse-moi gagner au badminton un jour stp). Merci à Mathias pour les déjeuners à raconter des bêtises et à se marrer (on t'a déjà dit que tu ris fort ?). Merci à Simon pour les parties de The Crew et les diatribes écolo, et merci à Alan pour les midis jeux. Merci à Dylan que je croise toujours avec plaisir (bon départ de Zombieland). Merci aussi à tous les collègues loin de Rennes qui sont devenu des amis : Corto, Lucie, Chana... I also thank Henry for all the good times in the UK and elsewhere. I thank all the IMDEA crew who welcomed me so well in Madrid: Marga, Alberto, Mario, Gaspard and all the others. The atmosphere at IMDEA was simply amazing and it was hard to leave.

Je n'oublie pas bien entendu de remercier celles et ceux, ami·es et familles, qui ont été là pour moi pendant ces trois ans. Je commence par Alice et Roland, qui m'ont toléré pendant deux ans de cohabitation, au travers des huiles parfumées et des quinquennats de salut public, et avec qui j'ai partagé tant de bons moments (au fait, on dit *roulante*). Merci et tous les autres copain·es loin de Rennes mais que je vois toujours volontiers, notamment du Bad17 et de Montautre. En particulier, merci à Rémi pour les belles vacances et pour l'accueil à Bordeaux. Merci à

Arthur pour les randos et autres traquenards (notamment au Thabor, j'étais 100% là). Merci à Joseph pour les beaux séjours dans les Alpes. Je voudrais citer également Margot, Hector, Eve, Clémence, Nathan et Milan. Merci aux ami·es du MPRI et de l'IRIF: Klara, Octave, Victor et les autres. Merci aussi à toutes celles et ceux dont j'ai croisé la route pendant ces trois ans à Rennes. Merci aux badistes, qui sont trop nombreux pour être tous cités ici. Je dois tout de même mentionner Alice B., non seulement pour les séances au bad mais aussi les séances de jeux et les diverses excursions (les fraises de tes parents sont très bonnes). Merci à Lise pour toutes les séances de mots croisés, de *So Clover* (je commence à croire que tu aimes bien ce jeu) et plus récemment de Geoguessr. Je conclus ces remerciements avec ma famille. Merci à mes oncles et tantes, cousins et cousines et à mes grands-mères. Merci à mon frère et à ma sœur, Paul et Sixtine: c'est toujours si facile de passer des moments en or avec vous. Merci Papa, toi qui m'as toujours aidé et encouragé et qui sais être là pour m'aider quand j'en ai besoin. Enfin, merci Maman, toi qui es toujours présente même à distance, et sur qui je peux toujours compter dans les moments difficiles.

VÉRIFICATION PARAMÉTRÉE DE SYSTÈMES DISTRIBUÉS À MÉMOIRE PARTAGÉE

Systèmes Distribués

Les systèmes distribués sont des systèmes informatiques composés de processus qui communiquent et qui travaillent ensemble pour accomplir une tâche commune. Les systèmes distribués sont aujourd'hui omniprésents et essentiels, et la théorie de l'informatique distribuée a de nombreuses applications dans des domaines comme les télécommunications, la robotique, les systèmes de partage de fichiers, les modèles biologiques et les réseaux de capteurs. De nombreux problèmes théoriques se posent dans le cadre de systèmes distribués, notamment sur comment réaliser l'exclusion mutuelle, l'élection d'un meneur, la synchronisation des horloges, la cohérence du cache ou le consensus. Nous renvoyons à [Lyn96; Asp24] pour des introductions détaillées sur la théorie des systèmes distribués et des algorithmes distribués. Dans la théorie des systèmes distribués, le problème du consensus [PSL80] est d'une importance capitale. Dans le problème du consensus, n processus doivent se mettre d'accord sur une valeur d'un ensemble donné. Chaque processus commence avec une valeur initiale appelée préférence et doit finir par décider définitement d'une valeur. Classiquement, le problème considéré est le *consensus binaire* où les valeurs possibles sont 0 et 1. Les algorithmes de consensus doivent satisfaire trois propriétés [PSL80; Asp03] : *accord*, *validité* et *terminaison*. La propriété d'accord exprime que tous les processus qui décident d'une valeur choisissent la même valeur. La validité demande que la valeur décidée corresponde à la préférence initiale d'un processus. La terminaison exprime que tous les processus (non défectueux) finissent par terminer. Le problème du consensus est particulièrement difficile dans le cas des systèmes asynchrones, c'est-à-dire des systèmes sans horloge globale où les processus fonctionnent à des vitesses différentes et doivent communiquer pour se synchroniser. Il existe de nombreux résultats d'impossibilité relatifs au consensus dans les systèmes asynchrones [FLP85; LA87; DDS87] ; en particulier, il n'existe pas d'algorithme déterministe résolvant le consensus asynchrone dans le cas où les processus peuvent crasher de

manière indétectable et où la communication se fait par transmission de messages ou par mémoire partagée (voir [FR03] pour un rapport technique sur les résultats d'impossibilité relatifs au consensus). C'est pourquoi il existe de nombreuses approches différentes qui reposent sur différentes façons de contourner ces résultats d'impossibilité (voir [Asp03, Section 2]). Deux approches du problème du consensus asynchrone nous intéressent particulièrement. La première consiste à utiliser des algorithmes à rondes, où le code d'un processus prend la forme d'une boucle `for` sur une variable k (la *ronde*) allant de 0 à $+\infty$. Dans ce genre d'algorithme, il est commun que les rondes restreignent la communication d'une manière ou d'une autre, par exemple en interdisant aux processus de lire des messages provenant de processus à des rondes différentes, ou en ayant une mémoire partagée propre à chaque ronde dans laquelle seuls les processus de cette ronde peuvent écrire. Une autre approche du consensus est la randomisation : alors que le résultat d'impossibilité de [FLP85] interdit que chaque exécution termine, il peut être contourné en exigeant une terminaison avec probabilité 1 seulement. Plusieurs sources de randomisation peuvent être utilisées, en particulier les lancers de pièces locaux (les processus sont autorisés à effectuer des expériences aléatoires), les lancers de pièce globaux (des expériences aléatoires sont effectuées globalement et les résultats sont visibles par tous les processus) et les planificateurs stochastiques (l'ordre dans lequel les actions se déroulent est décidé de manière aléatoire). L'un des premiers et des plus célèbres algorithmes de consensus, l'algorithme de Ben-Or [Ben83], est à la fois à rondes et randomisé ; il en va de même pour de nombreux autres algorithmes de consensus [BT85; Bra87; AH90; CR93; Asp02; GR07; RS12]. En effet, les rondes sont généralement utiles pour répéter une expérience aléatoire un nombre arbitraire de fois afin d'atteindre une probabilité de terminaison de 1.

Toutefois, la conception d'algorithmes pour les systèmes distribués asynchrones est une tâche difficile. L'asynchronisme fait que le nombre d'entrelacements et donc le nombre de comportements d'un système distribué sont potentiellement très importants. Les preuves à la main ne sont souvent pas suffisantes pour garantir l'exactitude. Pour citer Leslie Lamport [Lam19] :

> L'expérience que nous avons acquise au fil des ans en matière de conception d'algorithmes de synchronisation nous a appris que ce type de preuve n'est pas très fiable. À plusieurs reprises, nous avons "prouvé" l'exactitude des algorithmes de synchronisation pour découvrir par la suite qu'ils étaient incorrects.

Les méthodes formelles sont donc nécessaires pour obtenir des garanties de correction pour les algorithmes distribués.

Méthodes Formelles pour les Algorithmes Distribués

Il existe de nombreuses approches pour appliquer les méthodes formelles aux systèmes distribués. Une famille de techniques consiste à implémenter directement l'algorithme dans un langage qui permet d'écrire une preuve formelle ; cela permet de vérifier non seulement l'algorithme lui-même, mais aussi son implémentation. Entre autres, Verdi [WWPTWEA15] fournit un cadre pour implémenter et vérifier les algorithmes distribués en Coq [tea04] et IronFleet [Haw+17] fournit une méthodologie pour le faire en Dafny [Lei10].

Dans le cadre de cette thèse, nous nous intéressons aux techniques de vérification automatiques, et plus particulièrement au model checking. Le terme *model checking* fait référence aux techniques automatisées qui prennent en entrée un modèle du système considéré et une propriété que le système doit satisfaire et qui vérifient si le modèle satisfait à la propriété. La propriété peut être, par exemple, l'accessibilité d'un état d'erreur donné ou l'absence de blocage. Si le model checker conclut que le modèle ne satisfait pas à la propriété, il renvoie généralement un contre-exemple. Nous nous intéressons aux applications du model checking pour vérifier la conception de systèmes et d'algorithmes distribués. Il faut noter que le model checking ne peut pas vérifier formellement le système, mais seulement un modèle du système. Lorsque nous travaillons sur le model checking, nous supposons implicitement que la traduction du système vers le modèle est correcte. Pour citer Baier et Katoen [BK08] :

> Une méthode de vérification utilisant des techniques basées sur des modèles n'est bonne que si le modèle du système l'est.

Le livre cité ci-dessus, [BK08], constitue une excellente introduction au model checking. Nous désignons ici les techniques de model checking de [BK08] sous le nom de model checking *traditionnel*. Dans le model checking traditionnel, le model checker explore typiquement tous les états du système de manière exhaustive, bien qu'avec des algorithmes et des structures de données appropriées. Parmi les nombreux model checkers existants, l'un des plus célèbres et des plus puissants est Uppaal [LPY97].

Le model checking traditionnel est couramment appliqué aux systèmes distribués, souvent avec succès. Nous avons cité plus haut Leslie Lamport soulignant la difficulté de prouver la correction d'un algorithme distribué. De fait, Lamport considère le model checking comme une solution à ce problème [Lam06] :

> La vérification des modèles d'algorithmes avant leur soumission pour publication doit devenir la norme.

Une façon classique d'appliquer le model checking aux systèmes distribués est de supposer que le système est composé d'un nombre fixe n de processus et que l'espace d'état de chaque

processus est fini. Dans ce cas, le système distribué peut être considéré comme un système avec un nombre d'états finis. L'un des défauts de cette approche est que l'espace d'état subit une augmentation exponentielle en le nombre n de processus, de sorte qu'il peut rapidement devenir très grand. Les model checkers modernes étant très efficaces, cette approche peut néanmoins donner des résultats satisfaisants.

Vérification paramétrée

En plus de l'explosion de l'espace d'état mentionnée ci-dessus, les techniques traditionnelles de model checking présentent une autre limitation importante lorsqu'il s'agit de vérifier des algorithmes distribués. Un algorithme distribué est typiquement conçu avec le nombre n de processus vu comme un paramètre, de sorte qu'il peut être instancié avec n'importe quelle valeur de n. Le model checking traditionnel ne peut prouver la correction que pour des valeurs fixes de n, et ne peut donc pas prouver la correction de l'algorithme en général. Cela a conduit la communauté des méthodes formelles à envisager des systèmes distribués dits paramétrés, dans lesquels le nombre de processus n'est pas fixé à l'avance.

Un modèle est dit *paramétré* lorsqu'une certaine valeur n'est pas fixée à l'avance et est donc considérée comme un paramètre. Un modèle paramétré représente en fait toute une famille de systèmes : un système pour chaque instanciation du paramètre. Dans cette thèse, le paramètre considéré est le nombre n de processus ; nous utilisons le terme *paramétré* pour désigner les modèles et systèmes distribués où le nombre de processus n'est pas fixé à l'avance.

La vérification de modèles paramétrés de systèmes distribués est appelée *vérification paramétrée* (voir [Esp14; BJKKRVW15] pour des rapports techniques sur le sujet). L'intérêt de la vérification paramétrée est triple. Premièrement, elle permet d'éviter l'explosion de l'espace d'état lorsque l'on considère des systèmes de grande taille. Deuxièmement, elle permet de prouver que le système est correct pour chaque valeur de n, ce qui n'est pas possible avec le model checking traditionnel. Troisièmement, considérer le système pour des valeurs arbitrairement grandes de n rend parfois l'analyse plus facile, car cela permet d'exploiter les propriétés de monotonicité du système.

Dans cette thèse, nous nous intéressons à la vérification paramétrée d'un point de vue théorique. Nous essayons de concevoir des modèles paramétrés qui capturent (certaines caractéristiques) des algorithmes distribués de la littérature et d'étudier, dans ces modèles, les propriétés qui sont pertinentes pour ces algorithmes. Nous nous concentrerons principalement sur les modèles d'algorithmes à mémoire partagée. En particulier, nous définirons et étudierons, dans les chapitres 4 et 5, un modèle à mémoire partagée à rondes qui est destiné à capturer les

algorithmes à mémoire partagée à rondes tels que l'algorithme de consensus bruité d'Aspnes [Asp02]. Nous nous intéressons à la décidabilité et aux classes de complexité des problèmes de décision associés : étant donné une instance du modèle, satisfait-elle la propriété ? Un avertissement nécessaire est que cette approche théorique est avant tout motivée par la curiosité intellectuelle ; les résultats obtenus ne se transforment pas nécessairement en des outils de vérification pratiques. Il arrive qu'en déterminant la complexité d'un problème, on obtienne un algorithme efficace. Souvent, cependant, la complexité associée est élevée et suggère que, pour les applications pratiques, il est plus pertinent de considérer des méthodes incomplètes (c'est-à-dire des méthodes dont la terminaison n'est pas théoriquement garantie mais qui donnent de bons résultats sur des instances pratiques).

Contributions et organisation

Chapitre 1 Dans le premier chapitre de cette thèse, nous commençons par quelques préliminaires. Dans la section 1.1, nous introduisons les notions mathématiques standard et leurs notations. Nous introduisons ensuite des concepts de théorie des probabilités qui seront utiles dans le chapitre 5. Dans la section 1.2, nous introduisons les variables aléatoires, que nous utilisons comme briques de base pour nos définitions probabilistes. Dans la section 1.3, nous présentons quelques outils classiques de la théorie des probabilités. Dans la section 1.4, nous donnons une introduction rapide aux marches aléatoires, en particulier en dimension 1. Enfin, dans la section 1.5, nous présentons les beaux préordres et quelques résultats sur le sujet qui seront utiles dans le chapitre 3.

Chapitre 2 L'objet de ce deuxième chapitre est de présenter et d'étudier les systèmes asynchrones à mémoire partagée (ASMS). Dans ce modèle, introduit pour la première fois dans [EGM13], les processus communiquent en lisant et en écrivant dans une mémoire partagée. Dans cette thèse, tous les processus sont supposés être identiques et à ensemble d'états fini. Bien que notre modèle des ASMS soit très similaire à celui de [EGM13; EGM16], il constitue une généralisation à deux égards : dans notre modèle, la mémoire partagée peut être constituée de plusieurs registres, et les registres contiennent initialement une valeur initiale. Dans les modèles paramétrés de systèmes distribués comme les ASMS, les questions étudiées sont typiquement des *problèmes d'accessibilité* de la forme $\exists n, \exists \rho : \gamma_0(n) \xrightarrow{*} \gamma, \mathsf{prop}(\gamma)$, *i.e.*, ils demandent si, pour un certain nombre de processus, il existe une exécution qui atteint une configuration satisfaisant une certaine propriété prop. Ces problèmes sont appelés *paramétrés* en raison de la quantification universelle sur le nombre n de processus. La question la plus simple

est le problème de la couverture, où la propriété est que γ possède au moins un processus dans un état particulier, l'état final q_f; ici, la couverture est notée COVER. COVER dans les ASMS a été étudié dans [EGM13; EGM16] ; plus précisément, le problème étudié dans [EGM13; EGM16] est la négation de COVER, appelée *sûreté*. Dans le problème de la sûreté, on demande qu'il n'y ait pas d'exécution qui mette un processus dans l'état q_f, qui est considéré comme un état d'erreur. Une question plus difficile est le *problème de synchronisation* qui demande si l'on peut atteindre une configuration où *tous* les processus sont dans l'état final q_f; il est dénoté ici TARGET. Nous généralisons COVER et TARGET en un problème d'accessibilité paramétré plus expressif appelé *problème d'accessibilité de présence*. Les propriétés considérées sont des contraintes de présence qui sont des formules exprimant que certains états doivent être peuplés (au moins un processus se trouve dans l'état en question) et que d'autres doivent être vides (aucun processus ne se trouve dans l'état en question). Les définitions formelles du modèle ASMS et des problèmes qui nous intéressent se trouvent dans la section 2.2. Dans la section 2.3, nous présentons la propriété d'imitation : lorsqu'un processus passe de q_1 à q_2, tout autre processus dans q_1 peut aller dans q_2 sans affecter le reste du système. Cette propriété permet de prouver une forme de monotonicité de l'ensemble des configurations accessibles. Grâce à cette propriété, nous définissons une abstraction non comptante qui est correcte et complète pour le problème de l'accessibilité de présence dans la section 2.4. Nous établissons que ce problème est NP-complet dans la section 2.5, et que la NP-dureté tient déjà pour COVER. Cela nous amène à considérer certaines restrictions du modèle dans l'espoir d'obtenir des algorithmes en temps polynomial. En particulier, la NP-dureté de COVER utilise directement les valeurs initiales des registres. Par conséquent, dans la section 2.6, nous étudions le cas *non initialisé* où les registres ne contiennent initialement aucune valeur. Cette restriction rend effectivement COVER soluble en temps polynomial, mais les problèmes plus difficiles tels que TARGET restent NP-durs. Dans la section 2.7, nous considérons une autre restriction où le système n'a qu'un seul registre partagé (comme dans [EGM13]). Inspirés par un algorithme similaire dans le modèle des RBN [Fou15], nous prouvons que, lorsque le système n'a qu'un seul registre, TARGET peut être résolu dans PTIME et qu'il en va de même pour le problème de l'accessibilité de présence en supposant que la contrainte est donnée en forme normale disjonctive. Dans la section 2.8, nous faisons une petite incursion dans le monde de la complexité paramétrée en étudiant la dépendance de nos résultats de complexité par rapport au nombre de registres. Nous prouvons que, avec le nombre de registres comme paramètre de complexité, COVER est FPT mais TARGET est W[2]-dur.

Chapitre 3 Dans le troisième chapitre de cette thèse, nous introduisons un modèle plus abstrait, appelé *système à imitation*, qui est destiné à capturer les modèles qui bénéficient de la propriété dite d'imitation. Ces systèmes sont composés d'un nombre arbitraire de identiques dont l'ensemble d'états est fini. Une configuration du système est donc un multiensemble d'états accompagné d'une valeur provenant d'un ensemble fini. Cette valeur, appelée *point de contrôle*, représente l'état global du système et peut correspondre, *e.g.*, à la valeur de variables partagées ou à l'état d'un meneur. Une transition prend la forme d'un flux de transfert, qui consiste en un point de contrôle de départ, un point de contrôle d'arrivée et une fonction exprimant le nombre de processus pouvant passer d'un état donné à un autre. Nous définissons un cadre mathématique pour les flux de transfert et en particulier un produit compositionel qui décrit les possibilités offertes par plusieurs transitions effectuées à la suite. Dans les flux de transfert, si m processus sont nécessaires pour passer de q_1 à q_2, alors n'importe quel entier dans $[\![m, +\infty[\![$ est autorisé. Cette hypothèse est cruciale et modélise la propriété d'imitation décrite dans le chapitre 2, d'où le nom *système à imitation*. Les systèmes asynchrones à mémoire partagée [EGM13], les réseaux de diffusion reconfigurables [DSTZ12] et les protocoles de population à observation immédiate [ERW19] sont des exemples de modèles de la littérature qui peuvent être encodés dans des systèmes à imitation. La section 3.2 est consacré à la définition des systèmes à imitation, des flux de transfert et aux preuves des propriétés basiques des flux de transfert. Dans la section 3.3, nous fournissons une borne générale sur les systèmes à imitation appelée *borne structurelle*. En particulier, étant donné deux configurations γ_1, γ_2 telles que γ_2 peut être atteint à partir de γ_1, nous bornons le nombre d'étapes nécessaires pour aller de γ_1 à γ_2. En outre, notre résultat exprime que, si γ_1 et γ_2 ont de nombreux processus, on peut trouver γ'_1 et γ'_2 de taille bornée qui sont similaires à γ_1 et γ_2 jusqu'à un certain seuil en le nombre de processus et de telle sorte que γ'_2 peut être atteint à partir de γ'_1. Cette borne structurelle est doublement exponentielle dans le nombre d'états dans la description des processus, mais polynomiale dans le reste de la description du système. Elle est basée sur un borne sur la longueur des chaînes descendantes dans $\mathbb{N}^d$ [LS21; SS24] introduite dans Section 1.5. Dans la section 3.4, nous expliquons ce que cette borne structurelle implique pour les systèmes à imitation. En particulier, elle permet d'obtenir facilement des résultats de décidabilité pour plusieurs problèmes en limitant la taille des configurations à considérer ; c'est par exemple le cas de la vérification LTL. Dans la section 3.5, nous présentons des applications à d'autres modèles, et en particulier aux ASMS du chapitre 2. Étant donné un protocole ASMS, on peut construire un système à imitation dont la sémantique correspond à la sémantique accélérée de l'ASMS, *i.e.*, une étape dans le système à imitation correspond à une transition fixée de l'ASMS appliquée un nombre arbitraire de fois.

Chapitre 4 Dans ce chapitre, nous introduisons les *ASMS à rondes*, un modèle conçu pour capturer les algorithmes de consensus à mémoire partagée à rondes, et en particulier l'algorithme de consensus bruité d'Aspnes [Asp02]. Pour motiver notre modèle, nous présentons cet algorithme en détail dans la section 4.2. Les algorithmes de consensus à rondes sont généralement structurés à l'aide d'une boucle `for` dans laquelle une valeur de ronde k va de 0 à $+\infty$. Dans les ASMS à rondes, chaque ronde possède son propre ensemble de registres, de sorte que le nombre total de registres n'est pas borné. Tous les processus commencent à la ronde 0 et progressent de manière asynchrone en terme de ronde, de sorte qu'il n'y a pas de borne *a priori* dans la différence de rondes entre les processus. Globalement, les informations relatives à un processus donné à un moment donné prennent la forme d'une paire (q, k) où q est l'état et k la ronde. Les processus ne peuvent interagir qu'avec les registres des rondes proches. Le premier problème naturel est à nouveau la couverture où l'on demande d'atteindre une configuration où au moins un processus est dans l'état q_f. Encore une fois, la motivation de ce problème est que q_f modélise un état d'erreur qui doit être évité, de sorte que le système est sûr (pour tout nombre de processus) si la réponse à Cover est négative. Comme dans le chapitre 2, nous voulons considérer des problèmes plus généraux, et nous définissons une généralisation des contraintes de présence du chapitre 2. Dans les contraintes de présence du modèle à rondes, nous autorisons les quantificateurs sur les valeurs des rondes, mais les quantificateurs ne peuvent pas être imbriqués. Le problème d'accessibilité de présence est suffisamment expressif pour exprimer la validité et l'accord des algorithmes de consensus et en particulier de l'algorithme de consensus bruité d'Aspnes [Asp02]. Le modèle des ASMS à rondes et les problèmes associés sont présentés dans la section 4.3. Dans la section 4.4, nous commençons notre analyse du modèle en mettant en évidence des bornes inférieures exponentielles sur le nombre de rondes pertinentes pour Cover. En fait, même le nombre de rondes qui doivent être pris en compte *à un moment donné* peut devoir être exponentiel pour trouver une exécution témoin pour Cover. Par conséquent, un algorithme à espace polynomial pour Cover ne peut pas naïvement deviner l'exécution configuration par configuration. Nous fournissons également une borne inférieure PSPACE pour Cover et donc pour PRP. Dans la section 4.5, nous présentons un algorithme en espace polynomial pour Cover. Cet algorithme est non-déterministe ; il devine l'exécution à l'aide d'un mécanisme de fenêtre coulissante. Cette fenêtre coulissante repose sur la notion d'*empreinte*, qui correspond à la projection d'une exécution sur un ensemble de rondes consécutives. En utilisant la même abstraction non comptante que dans le chapitre 2, nous prouvons que l'empreinte d'une exécution bien choisie sur une petite fenêtre de rondes peut être stockée dans un espace raisonnable, de sorte que l'on peut deviner l'exécution empreinte

par empreinte. Dans la section 4.6, nous étendons cette appartenance à PSPACE au problème PRP. L'algorithme devine encore l'empreinte d'exécution par empreinte, mais il doit effectuer des choix non-déterministes et des calculs supplémentaires pour vérifier que la contrainte de présence est satisfaite. Pour garantir que cet algorithme fonctionne en espace polynomial, nous devons supposer que les constantes entières de l'entrée sont encodées en unaire ; dans la section 4.7, nous prouvons que, si les entiers sont encodés en binaire, alors PRP et même COVER deviennent EXPSPACE-complets.

Chapitre 5 De nombreux algorithmes de consensus à rondes nécessitent un nombre illimité de rondes parce qu'ils sont randomisés et terminent avec probabilité 1 seulement. La randomisation peut avoir lieu à différents niveaux : au niveau du processus (le code exécuté par le processus effectue des tirages au sort) ou au niveau du système. La randomisation au niveau du système peut impliquer des tirages au sort globaux (des tirages au sort dont le résultat est visible par tous) ou provenir du planificateur (l'ordre dans lequel les processus agissent est sélectionné de manière aléatoire). Nous nous intéressons au dernier cas, où le planificateur est stochastique. C'est en effet le type de randomisation utilisé dans l'algorithme de consensus bruité d'Aspnes [Asp02]. Dans la section 5.2, nous introduisons formellement les planificateurs stochastiques pour les ASMS à rondes. Jusqu'à la section 5.6 (incluse), le planificateur stochastique considéré sélectionne simplement le processus suivant qui agit de manière aléatoire uniforme parmi tous les processus, indépendamment du passé. Si le processus sélectionné a plusieurs actions possibles, l'action exécutée est sélectionnée de manière aléatoire uniforme parmi toutes les actions possibles. Dans la section 5.3, nous introduisons les deux propriétés probabilistes qui nous intéressent. Elles sont liées à un état spécial q_f que les processus ne peuvent pas quitter et qui correspond à la terminaison du processus. La *couverture presque sûre* exprime que, avec une probabilité de 1, un processus finit dans l'état final, tandis que la *termination presque sûre* exprime que, avec une probabilité de 1, tous les processus se retrouvent dans l'état final. Cette dernière propriété est particulièrement importante : si l'on considère, par exemple, le protocole codant l'algorithme de consensus bruité d'Aspnes, la terminaison presque sûre de l'algorithme est exprimée comme la terminaison presque sûre de l'ASMS à rondes pour tout n. Dans la section 5.4, nous présentons les résultats de [BMRSS16] relatifs à la couverture presque sûre dans les ASMS sans rondes. Dans [BMRSS16], les auteurs prouvent une équivalence entre la couverture presque sûre et une propriété non probabiliste. Cette équivalence stipule que, pour un n donné, q_f est couvert de façon presque sûre si et seulement s'il peut être couvert à partir de chaque configuration accessible. Cette équivalence est très pratique, c'est pourquoi nous essayons de

suivre la même approche pour les ASMS à rondes. Dans la section 5.5, nous prouvons que, malheureusement, cette équivalence ne tient pas dans notre modèle à rondes. En effet, nous construisons un protocole qui met en évidence des comportements de marche aléatoire et où il y a une probabilité non nulle que les processus s'éloignent les uns des autres, de sorte que q_f n'est pas couvert de façon presque sûre même s'il peut être couvert à partir de n'importe quelle configuration accessible. Avec cet exemple, l'analyse de la couverture presque sûre semble très difficile car les marches aléatoires sont hautement non triviales d'un point de vue mathématique. Nous essayons donc de restreindre le modèle pour éviter les comportements de marche aléatoire. Dans la section 5.6, nous présentons une restriction où les processus doivent incrémenter leurs rondes au même rythme en moyenne. Cependant, nous montrons que cela ne résout pas notre problème et que des comportements de marche aléatoire peuvent toujours se produire. Dans la section 5.7, nous considérons une autre famille de planificateurs, appelée *planificateurs à temps d'attente*. Un planificateur à temps d'attente s'appuie sur un modèle de temps continu ; chaque fois qu'un processus effectue une action, le temps qui s'écoule jusqu'à la prochaine action de ce processus est choisi selon une distribution de probabilité continue. Une fois de plus, ce choix de planificateur n'empêche pas les comportements de marche aléatoire. Cela nous oblige à revoir nos ambitions à la baisse et à revenir à nos motivations. Dans la section 5.8, nous définissons une restriction, appelée *absence d'obstruction presque sûre*, qui limite fortement le pouvoir d'expression des systèmes mais qui est typiquement satisfaite par les protocoles pour les algorithmes de consensus à ronde. Cette restriction est fortement liée à la notion d'absence d'obstruction présente dans certaines travaux d'algorithmique distribués [Asp24, Chapitre 27] mais nous l'étendons aux processus non déterministes. L'absence d'obstruction presque sûre exprime que, dans toute configuration accessible, si tous les processus tombent en panne sauf un, alors le processus restant finit dans l'état final avec probabilité 1. Nous prouvons que l'absence d'obstruction presque sûre implique la terminaison presque sûre quelque soit le nombre de processus. De plus, étant donné un protocole, il est possible de décider en espace polynomial s'il satisfait ou non la propriété d'absence d'obstruction presque sûre.

INTRODUCTION

Distributed systems

Distributed systems are computerized systems composed of communicating processes that work together to perform a common task. Distributed systems are nowadays ubiquitous and essential, and the theory of distributed computing has numerous applications in many different areas such as telecommunications, robotics, file-sharing systems, biological models and sensor networks. Many theoretical problems arise from distributed settings, such as how to perform mutual exclusion, leader election, clock synchronization, cache coherence or consensus. We refer to [Lyn96; Asp24] for detailed introductions to the theory of distributed systems and distributed algorithms. In the theory of distributed systems, the consensus problem [PSL80] is of paramount importance. In the consensus problem, n processes in a distributed system must agree on a value from a given set. Each process starts with some initial value called *preference*, and it must eventually *decide* of a value once and for all. Typically, one considers the restricted problem of *binary consensus* where the set of values is $\{0, 1\}$. Consensus algorithms must satisfy three properties [PSL80; Asp03]: *agreement*, *validity* and *termination*. Agreement requires that all processes that decide of a value choose the same value. Validity asks that the decided value is the initial preference of some process. Termination requires that all (non-faulty) processes eventually terminate. The consensus problem is particularly challenging in the case of asynchronous systems, *i.e.*, systems where there is no global clock, so that processes may all run at different speeds and must communicate to synchronize. There are many impossibility results related to consensus in asynchronous systems [FLP85; LA87; DDS87]; in particular, there is no deterministic algorithm solving asynchronous consensus in the case where processes may fail undetectably and where communication is by message-passing or shared-memory (see [FR03] for a survey on impossibility results for consensus). For this reason, there are many different approaches to the problem relying on different workarounds to these impossibility results (see [Asp03, Section 2]). Two approaches to the asynchronous consensus problem are of particular interest to us. The first one is the use of round-based algorithms, where the code of a process takes the form of a `for` loop over a variable k (the *round*) ranging from 0 to $+\infty$. Oftentimes, the communication is somehow restricted by the round, *e.g.*, by forbidding

processes to read messages from processes in different rounds, or by having a distinct shared memory per round that may only be written to by processes at this round. The other approach is randomization: while the impossibility result from [FLP85] forbids that every execution terminates, it can be circumvented by requiring termination with probability 1 only. Several sources of randomization can be used, and in particular local coin tosses (processes are allowed to perform random experiments), global coin tosses (some random experiments are performed globally and the results are visible to all processes) and stochastic scheduling (the order in which actions take place is decided at random). One of the first and most famous consensus algorithms, Ben-Or's algorithm [Ben83], is both round-based and randomized; the same is true for many other consensus algorithms [BT85; Bra87; AH90; CR93; Asp02; GR07; RS12]. Indeed, rounds are typically useful to repeat a random experiment arbitrarily many times in order to achieve probability 1 of termination.

The design of algorithms for asynchronous distributed systems, however, is a difficult task. Asynchronicity means that the number of interleavings and therefore the number of behaviors of a given distributed system are potentially very large. Hand-written proofs are often not sufficient to guarantee correctness. To quote Leslie Lamport [Lam19]:

> Our experience in years of devising synchronization algorithms has been that this style of proof is quite unreliable. We have on several occasions "proved" the correctness of synchronization algorithms only to discover later that they were incorrect.

Formal methods are needed to obtain guarantees of correctness for distributed algorithms.

Formal Methods for Distributed Algorithms

There are many approaches that apply formal methods to distributed systems. A family of techniques is to directly implement the algorithm in a language that allows to write a formal proof; this allows to verify not only the algorithm itself, but also its implementation. Among others, Verdi [WWPTWEA15] provides a framework to implement and verify distributed algorithms in Coq [tea04] and IronFleet [Haw+17] provides a methodology to do so using Dafny [Lei10].

For this thesis, we are interested in automated verification techniques, and more specifically in model checking. The term *model checking* refers to automated techniques that take as input a model of the system under consideration and a property that the system must satisfy, and that check whether the model satisfies the property. The property can be, *e.g.*, whether a given error state can be reached or whether the system is deadlock-free. If the model checker finds that the model does not satisfy the property, then it typically returns some form of counter-example. We

are interested in applications of model checking to verify the design of distributed systems and algorithms. Note that model checking cannot formally check the system but only a model of the system. When working with model checking, we implicitly assume that the translation from the system to the model is correct. To quote Baier and Katoen [BK08]:

> Any verification using model-based techniques is only as good as the model of the system.

The above-cited book [BK08] constitutes a great introduction to model checking. We will refer to the model-checking techniques from [BK08] as *traditional* model checking. In traditional model checking, the model checker will typically explore all states of the system in a brute-force manner, although using well-chosen algorithms and data structures. Among the many model checkers available, one of the most famous and powerful ones is Uppaal [LPY97].

Traditional model checking is commonly applied to distributed systems, oftentimes with great success. We have quoted above Leslie Lamport highlighting the difficulty to prove correctness of distributed algorithm. In fact, Lamport sees model checking as a solution to this issue [Lam06]:

> Model checking algorithms prior to submitting them for publication should become the norm.

A classic way to apply model checking to distributed systems is to assume that the system is composed of a fixed number n of processes and that the state space of each process is finite. In this case, the distributed system can be seen as a standard, finite-state system. One of the flaws of this approach is that the state space suffers an exponential blowup in the number n of processes, so that it can quickly become big. Because modern model checkers are very efficient, this approach can nonetheless yield satisfying results.

Parameterized Verification

In addition to the state-space explosion mentioned above, traditional model checking techniques present another important limitation when it comes to verification of distributed systems. A distributed algorithm is typically designed with the number n of processes seen as a parameter, so that it can be instantiated with any value of n. Traditional model checking can only prove correctness for fixed values of n, and therefore cannot prove correctness of the algorithm in general. This has lead the formal methods community to consider so-called *parameterized* distributed systems, where the number of processes is not fixed beforehand.

A model is called *parameterized* when some value is not fixed and therefore considered as a parameter. A parameterized model represents, in fact, a whole family of systems: one system

for each instantiation of the parameter. In this thesis, the parameter considered is the number n of processes; we use the word *parameterized* to refer to distributed models and systems where the number of processes in not fixed in advance [1].

Verification of parameterized models of distributed systems is called *parameterized verification*. The interest of parameterized verification is threefold. First, it allows us to avoid the state-space explosion when considering large systems. Second, it might allow us to prove that the system is correct for every value of n, something that is not possible with traditional model checking. Third, considering the system for arbitrarily large values of n sometimes makes the analysis easier, because it allows to exploit monotonicity properties of the system.

In this thesis, we are interested in parameterized verification from a theoretical point of view. We try to design parameterized models that capture (some features of) distributed algorithms from the literature and to study, in these models, properties that are relevant for these algorithms. We will mostly focus on models for shared-memory algorithm. In particular, we will define and study, in Chapter 4 and Chapter 5, a round-based shared-memory model that is meant to capture round-based shared-memory algorithm such as Aspnes' noisy consensus algorithm [Asp02]. We are interesting in the decidability status and complexity classes of the associated decision problems: given an instance of the model, does the instance of the model satisfy the property? A necessary disclaimer is that this theoretical approach is first and foremost motivated by intellectual curiosity; the results obtained does not necessarily translate to practical verification tools. Sometimes, while settling the complexity status of a problem, one actually obtains an efficient algorithm. Oftentimes, however, the associated complexity is high and suggests that, for practical applications, one should instead look for incomplete methods (*i.e.*, methods whose termination is not theoretically guaranteed but that perform well on practical instances).

Literature on Parameterized Verification

The first result on parameterized verification can be traced back to 1986 [AK86], where the authors establish undecidability of parameterized verification in the general case. This negative result is very simple: if the number of processes is unbounded, then one can make each process encode one cell of the tape of a Turing machine. In this undecidability proof, the descriptions of the processes are asymmetric as each process has its own, pre-assigned role. However, the observation was made that systems with many processes typically consist of the

1. There is one short exception to this choice of terminology for the word *parameterized*: in Section 2.8, we perform a study of parameterized complexity with respect to the number of registers, and in this context we use the word *parameterized* in a different sense, which is the one of parameterized complexity (see for example [Cyg+15]).

same code replicated many times. This is the hypothesis made in the seminal work of [GS92], where the authors consider systems composed of a leader and of arbitrarily many identical contributors. In [GS92], the processes are finite-state machines that communicate by *rendez-vous*, *i.e.*, a communication step consists in two processes exchanging information with each other. In this setting, the authors obtain decidability of *coverability* [2] in EXPSPACE, and they even show that the problem can be solved in polynomial time when there is no leader. Since then, many variations of this model have been studied. They differ on the means of communication (broadcast, message passing, shared memory...), the topology of the network (who communicates to whom), the computational power of a given process (finite-state, pushdown machine, ...), the reliability of the processes (non-faulty, faulty behaviors, Byzantine behaviors)... We present here a few interesting models, see [Esp14; BJKKRVW15] for surveys.

One of the means of communication that has received the most attention by the parameterized verification community is the one of broadcast networks [EFM99; FL02; DP08; DSZ10; DSZ11]. In broadcast networks, processes are finite-state machines, the communication is by broadcasting a message containing a symbol from a finite alphabet and the topology is complete: a message sent must be received by everyone. The first positive result on broadcast protocols is that coverability is decidable [EFM99]; this result, like many others in parameterized verification, relies on the theory of well-quasi-orders (see, *e.g.*, [DFGSS17]). More specifically, it relies on a generic techniques from [ACJT96] later abstracted into the convenient framework of well-structured transition systems [FS01]. This techniques gives no complexity result; it was later proved that coverability for broadcast protocols is in fact a very hard problem, as it is Ackermann-complete [DSZ10]. Moreover, if the topology is seen as a parameter and one asks whether the system is safe for every topology, then the problem is undecidable [DSZ11]. This led the community to consider a simpler model, where the topology may reconfigure itself at any time [DSTZ12]. This model is called reconfigurable broadcast networks, and is meant to model networks where the communication is unreliable. Indeed, another way of viewing this model is that any message sent can be received by any subset of the other processes, as if one copy of the message was sent to each process but copies can get lost. The reconfigurable hypothesis drastically reduces the complexity of the model; in particular, coverability becomes solvable in polynomial time [DSTZ12]. Numerous works have followed on the model of reconfigurable broadcast networks, for example by adding probabilities [BFS14], considering local strategies [BFS15], extending the computational power of processes [Bal18], considering the more gen-

2. We call *coverability* the problem of reaching a configuration with a least one process in a control state. This name varies in the literature; it is in particular often called *control-state reachability*.

eral problem of cube reachability [BW21; BGW22], analyzing the cutoff values for coverability [BBM21] and adding local registers and identifiers [DST13; GMW24].

Many other means of communications have been considered by the community. We have already mentioned rendez-vous communication; it was the means of communication of the first positive results in parameterized verification [GS92]. Some subsequent works with rendez-vous communication have studied more sophisticated questions [AKRSV14]; for the target problem (where all processes must end up in the final state), deciding whether there is a cutoff (a limit value in the number of processes above which the answer does not change) is proved decidable in [HS20]; the variant of non-blocking rendez-vous is studied in [GSS23]. Another studied means of communication is the one of token-passing [EN03; CTTV04; AJKR14; BGS14] where the communication consists of a token that circulates among processes, typically in a ring. This token is often valued, meaning that it carries a value from a fixed set.

A paradigm that is of particular importance to us is the one where processes communicate by reading from and writing to a shared memory. More specifically, we are interested in communication using non-atomic registers, which was first considered in a context of parameterized verification in [Hag11]. Non-atomic registers do not allow a process to perform sequences of actions while excluding other processes: if a process performs a read and a write action, other processes may act in between the two actions. In other words, there is no locking mechanism. In [Hag11], the system is composed of a leader and of arbitrarily many followers, all described by pushdown machines, and coverability is proved to be decidable. This model was then further studied in [EGM13] (and its journal version [EGM16]) where the authors give it the name *asynchronous shared-memory systems* (ASMS for short). In [EGM13; EGM16], the complexity of coverability in the setting of [Hag11] is proved to be PSPACE-complete, and the complexity of coverability is settled in several variations on the computation powers of the leader and of the followers. In particular, if all processes are finite-state machines, then coverability is NP-complete in presence of a leader and in PTIME without a leader. An abstracted version of ASMS is studied in [TMW15] where more general decidability results are obtained. In [DEGM15], the liveness question (where the execution is infinite and must satisfy a Büchi condition) is proved to be NP-complete when all processes, leader and follower, are finite-state. In [BMRSS16], the question of almost-sure coverability is studied in ASMS under the assumption that all processes are identical, finite-state machines. In [FMW17], verification of stuttering-invariant LTL is proved to be NEXPTIME-complete in presence of a leader when all processes are pushdown machines. Another work with shared-memory is the one from [BRS21], which is however not on the ASMS model. In [BRS21], the system is composed of identical finite-state processes and

each process has a local register and an assigned shared register, so that the number of shared registers is unbounded. Processes may test the number of occurrences of their value in the shared registers up to some threshold; in this model, coverability is proved to PSPACE-complete.

Some parameterized models abstract communication using guards. This is the case of threshold automata [KVW14; KKW18] (see [KLSW23] for a survey of parameterized verification using threshold automata). Threshold automata are meant to model threshold-based fault-tolerant distributed algorithms; interestingly, the threshold automata community often combines theoretical results and practical model-checking experiments. In threshold automata, processes are abstracted into counters that can only be incremented (meant to represent the number of times that a given message was sent) and the behavior of the system is represented by a single finite automaton who transitions may be guarded by thresholds that require a counter value to be greater than a value depending on some parameters. A related model is the one of guarded protocols [EK03; EN03; AJK15; JS18], where processes are each described by instances of a finite-state description whose transitions are guarded by constraints related to the states of the other processes; guards can be either conjunctive (all other process must satisfy the guard) or disjunctive (some other process must satisfy the guard).

We end this overview of the literature on parameterized verification by presenting population protocols. Population protocols were first introduced as a model of distributed computing [AADFP04]; the communication mechanism consists in pairwise interactions between processes and is similar to rendez-vous communication. This model is also very similar to the standard model of Petri nets. We refer to [AR09] for a survey on the model of population protocols from the point of view of distributed computing. Population protocols were analyzed with a parameterized approach in [EGLM16], where the authors show that verification of LTL properties over population protocols is decidable when the LTL property refers to actions of the population protocol, but that the problem is as hard as Petri net reachability and therefore Ackermann-complete [CO21]. Many subsequent works have been performed on parameterized verification of population protocols, we refer to [Esp17; BEJK18] for surveys.

Contributions and Organization

Chapter 1 In the first chapter of the thesis, we start with some useful preliminaries. In Section 1.1, we introduce standard mathematical notions and their notations. We then introduce concepts of probability theory that will be useful in Chapter 5. In Section 1.2, we introduce random variables, which we use as basic bricks for our probabilistic definitions. In Section 1.3, we introduce some classic tools of probability theory. In Section 1.4, we give a quick introduction

to random walks, in particular in dimension 1. Finally, in Section 1.5, we present well-quasi-orders and some known results on the topic which will be useful in Chapter 3.

Chapter 2 The subject of this second chapter is to introduce and study asynchronous shared-memory systems (ASMS). In this model, first introduced in [EGM13], processes communicate via reading from and writing to a shared memory. In this thesis, all processes are assumed to be identical finite-state machines. While our model of ASMS is very similar to the one of [EGM13; EGM16], it constitutes a generalization in two ways: in our model, the shared-memory may consist of several registers, and the registers initially hold a special initial value. In parameterized models of distributed systems like ASMS, the questions studied are typically *reachability problems* of the form $\exists n,\ \exists \rho : \gamma_0(n) \xrightarrow{*} \gamma,\ \mathsf{prop}(\gamma)$, *i.e.*, they ask whether, for some number of processes, there is an execution that reaches a configuration satisfying some property prop. These problems are called *parameterized* because of the universal quantification over the number n of processes. The simplest such question is the *coverability problem* where the property is that γ has at least one process in a particular state q_f; here, coverability is denoted Cover. Cover in ASMS was studied in [EGM13; EGM16]; more precisely, the problem studied in [EGM13; EGM16] is the negation of Cover, called *safety*. In the safety problem, one asks that *no* execution puts a process on state q_f, which is seen as an error state. A more difficult question is the *target problem* that asks whether one can reach a configuration where *all* processes are in q_f; it is here denoted Target. We generalize Cover and Target into a more expressive parameterized reachability problem called *presence reachability problem*. The properties considered are presence constraints, which are formulas expressing that some states must be populated (*e.g.*, at least one process is in the state) and some must be empty (no process is in the state). The formal definition of the ASMS model and of our problems of interest can be found in Section 2.2. In Section 2.3, we present the copycat property: when a process goes from q_1 to q_2, any other process in q_1 may go in q_2 without affecting the rest of the system. This property yields some monotonicity property of the set of reachable configurations. Thanks to this property, we define a non-counting abstraction that is sound and complete for the presence reachability problem in Section 2.4. We establish that this problem is NP-complete in Section 2.5, and that NP-hardness already holds for Cover. This leads us to consider some restrictions of the model in the hope to obtain polynomial-time algorithms. In particular, the NP-hardness of Cover directly uses the initial values of the registers. Therefore, in Section 2.6, we study the *uninitialized* case where the registers initially contain no value. This restriction indeed makes Cover solvable in polynomial time, but harder problems such as Target remain

NP-hard. In Section 2.7, we consider another restriction where the system has one shared register only (as in [EGM13]). Inspired by a similar algorithm for RBN [Fou15], we prove that, when the system has a single register, TARGET can be solved in PTIME and that the same is true of the presence reachability problem assuming that the constraint is given in disjunctive normal form. In Section 2.8, we make a small incursion into the world of parameterized complexity by studying the dependency of our complexity results with respect to the number of registers. We prove that, with the number of registers as complexity parameter, COVER is FPT but TARGET is W[2]-hard.

Chapter 3 In the third chapter of this thesis, we introduce a more abstract model, called *copycat systems*, which is meant to capture models that enjoy the so-called copycat property. Such systems are composed of an arbitrary number of finite-state, identical processes. A configuration of the system is therefore a multiset of states along with a value from a finite set. This value, called *control location*, represents the global state of the system and can correspond, *e.g.*, to the values of shared variables or to the state of a leader. A transition takes the form of a so-called *transfer flow*, which consists in a source control location, a destination control location and a function that expresses how many processes may go from a given state to another. We define a mathematical framework for transfer flows and in particular a compositional product that describes the possibilities enabled by several transitions performed in a row. In transfer flows, if m processes are required to go from q_1 to q_2, then any integer in $[\![m, +\infty[\![$ is allowed. This hypothesis is crucial and models the copycat property described in Chapter 2, hence the name *copycat system*. Examples of models from the literature encodable into copycat systems include asynchronous shared-memory systems [EGM13], reconfigurable broadcast networks [DSTZ12] and immediate-observation population protocols [ERW19]. Section 3.2 is devoted to the definition of copycat systems, of transfer flows and to the proofs of basic properties of transfer flows. In Section 3.3, we provide a general-purpose bound on copycat systems called *structural bound*. In particular, given two configurations γ_1, γ_2 such that γ_2 can be reached from γ_1, we bound the number of steps of the semantics needed to go from γ_1 to γ_2. Also, our bound expresses that, if γ_1 and γ_2 have many processes, one can find γ_1' and γ_2' of bounded size that are similar to γ_1 and γ_2 up to some threshold in the number of processes and such that γ_2' can be reached from γ_1'. This structural bound is doubly-exponential in the number of states in the description of the processes, but polynomial in the rest of the description of the system. It is based on the bound on the length of descending chains on $\mathbb{N}^d$ [LS21; SS24] introduced in Section 1.5. In Section 3.4, we elaborate on what this structural bound implies

for copycat systems. In particular, it gives decidability results of several problems by bounding the size of the configurations to consider; this is for example the case of LTL verification. In Section 3.5, we present applications to other models, and in particular to ASMS from Chapter 2. When given an ASMS protocol, one can build a copycat system whose semantics corresponds to the accelerated semantics of the ASMS, *i.e.*, a step in the copycat system corresponds to a single transition of the ASMS performed arbitrarily many times. This means that our bounds on copycat systems carry over to ASMS.

Chapter 4 In this chapter, we introduce *round-based ASMS*, a model designed to capture round-based shared-memory algorithms for consensus, and in particular Aspnes' noisy consensus algorithm [Asp02]. To motivate our model, we introduce this algorithm in detail in Section 4.2. Round-based consensus algorithms typically are structured using a `for` loop in which a round value k goes from 0 to $+\infty$. In round-based ASMS, each round has its own set of registers, so that the total number of registers is unbounded. All processes start at round 0 and they progress asynchronously in rounds, so that there is no *a priori* bound in the difference of rounds between pairs of processes. Overall, the information about a given process at a given point in time takes the form of a pair (q, k) where q is its state and k is its round value. Processes may only interact with registers of nearby rounds. The first natural problem is again coverability, called here round-based Cover, where one asks to reach a configuration where at least one process is in state q_f (independently of its round). Again, the motivation for this problem is that q_f models an error state that must be avoided, so that the system is safe (for every number of processes) if the answer to Cover is negative. As in Chapter 2, we want to consider more general problems, and we define a generalization of the presence constraints from Chapter 2, which we call *round-based presence constraints*. In round-based presence constraints, we allow for quantifiers over round values, but quantifiers are not allowed to be nested. The associated reachability problem is called round-based presence reachability problem, or round-based PRP for short. The presence reachability problem is expressive enough to express validity and agreement of consensus algorithms and in particular of Aspnes' noisy consensus algorithm [Asp02]. The round-based ASMS model and the associated problems are introduced in Section 4.3. In Section 4.4, we start our analysis of the model by highlighting exponential lower bounds in the number of relevant rounds to consider for round-based Cover. In fact, even the number of rounds that must be considered *at a given point in time* may have to be as large as exponential when looking for a witness execution for round-based Cover. Therefore, a polynomial-space algorithm for round-based Cover cannot naively guess the execution con-

figuration by configuration. We also provide a PSPACE lower bound for round-based COVER and therefore for round-based PRP. In Section 4.5, we present a polynomial-space algorithm for COVER. This algorithm is non-deterministic; it guesses the execution with a sliding window mechanism. This sliding window relies on the notion of *footprint*, which corresponds to the projection of an execution onto a set of consecutive rounds. Using the same non-counting abstraction as in Chapter 2, we prove that the footprint of a well-chosen execution onto a small window of rounds can be stored in reasonable space, so that one can guess the execution footprint by footprint. In Section 4.6, we extend this PSPACE result to round-based PRP. The algorithm again guesses the execution footprint by footprint, but it must perform additional computations and non-deterministic choices to check that the presence constraint is satisfied. To guarantee that this algorithm works in polynomial space, we need to assume that the integer constants of the input are encoded in unary; in Section 4.7, we prove that, if the integers are encoded in binary, then round-based PRP and even round-based COVER become EXPSPACE-complete.

Chapter 5 Many round-based consensus algorithms require an unbounded number of rounds because they are randomized and terminate with probability 1 only. Randomization may take place at several different level: at the level of the process (*i.e.*, the code executed by the process performs coin tosses) or at the level of the system. Randomization at the level of the system may involve global coin tosses (*i.e.*, coin toss whose result is visible by everyone) or come from the scheduler (*i.e.*, the order in which processes act is selected randomly). We are interested in the last case, where the scheduler is stochastic. This is indeed the type of randomization used in our motivating example, Aspnes' noisy consensus algorithm [Asp02]. In Section 5.2, we formally introduce stochastic schedulers for round-based ASMS. Until Section 5.6 (included), the considered stochastic scheduler simply selects the next acting process uniformly at random among all processes, independently from the past. If the process selected has several possible actions, the action performed is selected uniformly at random among all possible actions. In Section 5.3, we introduce the two probabilistic properties that we are interested in. They are related to a special state q_f that processes cannot leave; this state models that the process has terminated. *Almost-sure coverability* is when, with probability 1, some process ends up in q_f whereas *almost-sure termination* is when, with probability 1, all processes end up in q_f. This last property is of particular relevance: when considering, *e.g.*, the protocol encoding Aspnes' noisy consensus algorithm, almost-sure termination of the algorithm is expressed as almost-sure termination of the round-based ASMS for every n. In Section 5.4, we present an overview of results from [BMRSS16] related to almost-sure coverability in ASMS without

rounds. In [BMRSS16], the authors prove an equivalence between almost-sure coverability and a non-probabilistic property. This equivalence states that, for a given n, q_f is covered almost surely if and only if it can be covered from every reachable configuration. This equivalence is very convenient, therefore we try to follow the same approach for round-based ASMS. In Section 5.5, we prove that, unfortunately, this equivalence does not hold in our round-based model. Indeed, we build a protocol that highlights random-walk behaviors and where there is a non-zero probability that processes drift away from each other, so that q_f is not covered almost surely even though it can be covered from any reachable configuration. With this example, the analysis of almost-sure coverability appears very challenging, because random walks are highly non-trivial from a mathematical point of view. We thus attempt to restrict the model to prevent random-walk behaviors. In Section 5.6, we present a restriction where processes are required to increment their rounds at the same average pace. However, we illustrate that this does not solve our issue and that random walks behaviors may still occur. In Section 5.7, we consider another family of schedulers, called *waiting-time schedulers*. A waiting-time scheduler relies on a continuous-time model; whenever a process performs an action, the time until the next action of this process is drawn according to a continuous probability distribution. Yet again, this choice of scheduler does not prevent random-walk behaviors. This forces us to lower our ambitions and to come back to our motivations. In Section 5.8, we define a restriction, called *almost-sure obstruction-freedom*, which heavily limits the expressive power of the systems but which is typically satisfied by protocols for round-based consensus algorithms. This is strongly related to the notion of obstruction-freedom in distributed algorithms [Asp24, Chapter 27] but extends it to non-deterministic processes. Almost-sure obstruction-freedom expresses that, in any reachable configuration, if all processes crash except one, then the remaining process ends up in q_f with probability 1. We prove that almost-sure obstruction-freedom implies almost-sure termination for every number of processes. Moreover, given a protocol, one can decide in polynomial space whether it satisfies the almost-sure obstruction-freedom property or not.

Personal Publications

A total of five publications correspond to work performed during the time of the PhD. A sixth publication [GSWW24] is in preparation; we mention it here because it is connected with some of the content of this thesis. We list them here by chronological order.

[BMSW22] Nathalie Bertrand, Nicolas Markey, Ocan Sankur and Nicolas Waldburger. *Parameterized Safety Verification of Round-Based Shared-Memory Systems*. ICALP'22.

[Wal23] Nicolas Waldburger. *Checking Presence Reachability Properties on Parameterized Shared-Memory Systems*. MFCS'23.

[GMW24] Lucie Guillou, Corto Mascle and Nicolas Waldburger. *Parameterized Broadcast Networks with Registers: from NP to the Frontiers of Decidability*. FoSSaCS'24.

[BGKMWW24] Steffen van Bergerem, Roland Guttenberg, Sandra Kiefer, Corto Mascle, Nicolas Waldburger and Chana Weil-Kennedy. *Verification of Population Protocols with Unordered Data*. ICALP'24.

[CLSW24] Dmitry Chistikov, Jérôme Leroux, Henry Sinclair-Banks and Nicolas Waldburger. *Invariants for One-Counter Automata with Disequality Tests*. CONCUR'24.

[GSWW24] Pierre Ganty, Cesar Sanchez, Nicolas Waldburger and Chana Weil-Kennedy. *Temporal Hyperproperties on Population Protocols*. In preparation. 2024.

Three works presented in the list above are related to the content of this thesis. [BMSW22] is a study of safety problems in round-based asynchronous shared-memory systems: it is related to Chapter 4, but it is mostly subsumed by [Wal23]. This second publication [Wal23] studies a more generic class of problems in asynchronous shared-memory systems, with and without rounds. The last work to mention is the study of hyperLTL verification in population protocols [GSWW24]; while this topic is relatively far from the content of this thesis, some of the proof techniques have inspired Chapter 3.

The other three publications do not appear in this thesis. The work from [GMW24] is about parameterized verification of broadcast networks with private registers. [BGKMWW24] is related to verification of population protocols with unordered data. Finally, [CLSW24] is about reachability in 1-VASS with tests.

Chapter 1

Preliminaries

In this chapter, we introduce some useful definitions, notations and results. We start with standard mathematical definitions in Section 1.1. We then introduce some notions of probability theory that will be useful in Chapter 5. In Section 1.2, we introduce the fundamental notion of random variables. In Section 1.3, we present classic probabilistic tools. In Section 1.4, we provide a short introduction to random walks. In Section 1.5, we introduce well-quasi-orders and some related results from the literature.

1.1 Standard Mathematical Definitions

In this section, we introduce, for the sake of completeness, standard terminology and notations. The set of all non-negative integers is written $\mathbb{N}$, the set of all integers is written $\mathbb{Z}$. Given $m, n \in \mathbb{N}$ such that $n \leqslant m$, we denote by $[\![n, m]\!] \coloneqq \{i \in \mathbb{N} \mid n \leqslant i \leqslant m\}$ the set of integers between n and m. Moreover, given $n \in \mathbb{N}$, we denote by $[\![n, +\infty[\![\coloneqq \{m \in \mathbb{N} \mid n \leqslant m\}$ the set of integers greater than or equal to n. The set of all real numbers is written $\mathbb{R}$. Given $a, b \in \mathbb{R}$, we let $[a, b] \coloneqq \{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$.

Given a finite set Σ called *alphabet*, we write Σ^* for the set of *finite words* over Σ and Σ^ω for the set of *infinite words* over Σ. Given a word $w \in \Sigma^* \cup \Sigma^\omega$ and $i \in \mathbb{N}$, we denote by $w(i) \in \Sigma$ the $(i+1)$-th symbol of w, if it exists; therefore, $w = w(0)w(1)\ldots$. If $w \in \Sigma^*$ is a finite word, its *length* ℓ is its number of letters, so that $w = w(0)w(1)\ldots w(\ell-1)$. The concatenation of two words $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^* \cup \Sigma^\omega$, with w_1 finite, is written $w_1 \cdot w_2$. A *language (of infinite words)* is a set of words $\mathcal{L} \subseteq \Sigma^\omega$.

Let S be a countable set. A *finite multiset* over S is a function $\mu : S \to \mathbb{N}$ such that $\sum_{s \in S} \mu(s) < \infty$. We denote by $\mathcal{M}(S) \subseteq S$ the set of all finite multisets over S. The *support* of a multiset $\mu \in \mathcal{M}(S)$ is defined by $\overline{\mu} \coloneqq \{s \in S \mid \mu(s) > 0\}$. Given two multisets $\mu_1, \mu_2 \in \mathcal{M}(S)$, we let $\mu_1 \subseteq \mu_2$ when, for all $s \in S$, $\mu_1(s) \leqslant \mu_2(s)$. We let $\mu_1 \oplus \mu_2 \in \mathcal{M}(S)$ be the multiset such that, for all $s \in S$, $(\mu_1 \oplus \mu_2)(s) = \mu_1(s) + \mu_2(s)$. Also, if $\mu_1 \subseteq \mu_2$, then we let $\mu_2 \ominus \mu_1$ be the multiset defined by $(\mu_2 \ominus \mu_1)(s) = \mu_2(s) \ominus \mu_1(s)$. Given $s \in S$, the *singleton multiset* obtained

from s is the multiset μ such that $\mu(s) = 1$ and $\mu(s') = 0$ for all $s' \neq s$; we abuse notations and denote this singleton multiset s. Also, given $k \in \mathbb{N}$ and $s \in S$, we denote by $k \cdot s$ the multiset μ such that $\mu(s) = k$ and $\mu(s') = 0$ for all $s' \neq s$. Finally, given $\mu \in \mathcal{M}(S)$, we denote by $|\mu| \coloneqq \sum_{s \in S} \mu(s)$ the *size* of μ.

A *directed graph* is a pair $G = (V, E)$ where $E \subseteq V^2$. V is the set of *vertices* while E is the set of *edges*. An *undirected graph* follows the same definition except that, for all $(u, v) \in E$, $(v, u) \in E$. A (finite) *path* from $u \in V$ to $v \in V$ is a sequence $u = v_0, v_1, \ldots, v_k = v$ such that, for all $i \in [\![0, k-1]\!]$, $(v_i, v_{i+1}) \in E$. The *length* of the path is defined to be k. The path *visits* $v' \in V$ when there is i such that $v_i = v'$. A set of vertices $S \subseteq V$ is *strongly connected* if, for every $u, v \in S$, there is a path from u to v. An *infinite path* of G is an infinite sequence $v_0, v_1, \ldots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. An infinite path $v_0, v_1, \ldots$ *visits* v *infinitely many times* when there are infinitely many indices i such that $v_i = v$. A *strongly connected component* of G is a subset of vertices $S \subseteq V$ that is strongly connected and maximal in that sense, *i.e.*, for all $v \in V \setminus S$, $S \cup \{v\}$ is not strongly connected. A strongly connected component S is *bottom* if, for every $u \in S$ and $v \in V$, if there is a path from u to v then $v \in S$.

1.2 Random Variables

In Chapter 5, we will consider a probabilistic model where the next process to act is selected randomly by a stochastic scheduler. Therefore, in the following sections (Section 1.2, Section 1.3 and Section 1.4), we introduce notions of probability theory and some classic results that will be useful in Chapter 5.

A formal definition of our probabilistic framework would require to introduce, among others, the notion of *probabilistic space* and to present some elements of *measure theory*. Such definitions would however add little value to the rest of this work. We therefore refer to, *e.g.*, [BK08; Pan01], for such definitions; instead, we will hide technical details behind the notion of *random variable*. Informally, a random variable X with values in a set S is a variable whose value is in S and is chosen at random. See, *e.g.*, [BH14], for a formal definition of random variables.

We first introduce *discrete random variables*, *i.e.*, random variables with values in countable sets. When a random variable X has values in a countable (non-empty) set S, the basic events are of the form "$X \in T$" with $T \subseteq S$, and the *probability distribution* of X takes the form of a function $P : S \to [0, 1]$ such that $\sum_{s \in S} P(s) = 1$. The probability associated to a subset of S is simply the sum of the probabilities of its elements. We denote by $\mathbb{P}(E)$ the probability of

a given event E, and denote by $\mathbb{P}(X = s)$ the probability of the event $X \in \{s\}$. The simplest random variables are *Bernoulli variables*. X is a *Bernoulli variable* of parameter $p \in [0, 1]$, denoted $X \sim \mathcal{B}(p)$, when X take values in $\{0, 1\}$ and $\mathbb{P}(X = 1) = p$. Another classic type of discrete random variables are uniformly distributed random variables. When S is a finite set, X is *uniformly distributed over* S, denoted $X \sim \mathcal{U}(S)$, when $\mathbb{P}(X = s) = \frac{1}{|S|}$ for all $s \in S$.

We will sometimes have to consider *continuous random variables*, *i.e.*, random variables with values in $\mathbb{R}$. In this case, the basic events are of the form "$X \in S$" with S a measurable subset of $\mathbb{R}$. The definition of measurability for subsets of $\mathbb{R}$ is not important for us, all that we need to know is that all intervals of $\mathbb{R}$ are measurable. The *probability distribution* of a continuous random variable is a measurable function $f_X : \mathbb{R} \to [0, 1]$ such that $\int_{-\infty}^{+\infty} f(x)dx = 1$. Given an interval $[a, b]$ with $a \leqslant b$, we have $\mathbb{P}(X \in [a, b]) = \int_a^b f(x)dx$. Note that, given $x \in \mathbb{R}$, $\mathbb{P}(X = x) = 0$. A classic continuous probability distribution is the normal distribution. Given $\mu \in \mathbb{R}$, $\sigma > 0$, a random variable X with values in $\mathbb{R}$ is *normally distributed* with mean μ and variance σ^2, denoted $X \sim \mathcal{N}(\mu, \sigma)$, when its probability distribution is $F_X : x \mapsto \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$.

A random variable is not fully characterized by its probability distribution: indeed, the values of different random variables may depend on one another. A countable family $(X_i)_{i \in I}$ of random variables with values in a finite set S is *independent* when, for all $J \subseteq I$, for all families $(T_j)_{j \in J}$ of subsets of S,

$$\mathbb{P}(\bigcap_{j \in J} X_j \in T_j) = \prod_{j \in J} \mathbb{P}(X_j \in T_j).$$

The random variables (X_i) are *independent identically distributed*, or i.i.d., when they are independent and have the same probability distribution.

Given two events A, B such that $\mathbb{P}(B) > 0$, let $\mathbb{P}(A \mid B) \coloneqq \frac{\mathbb{P}(A)}{\mathbb{P}(B)}$ be the conditional probability of A given B. More generally, given events $A, B_1, \ldots, B_m$, we use $\mathbb{P}(A \mid B_1, \ldots, B_m)$ as a shorthand for $\mathbb{P}(A \mid (B_1 \cap B_2 \cdots \cap B_m))$. Similarly, we use $\mathbb{P}(B_1, \ldots, B_m)$ as a shorthand for $\mathbb{P}(B_1 \cap \cdots \cap B_m)$.

Given a discrete random variable X with values in $\mathbb{Z}$, its *expected value* is $\mathbb{E}(X) \coloneqq \sum_{z \in \mathbb{Z}} z\, \mathbb{P}(X = z)$, which is properly defined when the sum converges to a value in $\mathbb{Z}$. Also, its *variance* is $V(X) \coloneqq \sum_{z \in \mathbb{Z}} (z - \mathbb{E}(X))^2 \mathbb{P}(X = z)$, again assuming that the sum converges. For a continuous random variable X with a probability distribution f_X, we let $\mathbb{E}(X) \coloneqq \int_{-\infty}^{+\infty} z\, f_X(z)dz$ and $V(X) \coloneqq \int_{-\infty}^{+\infty} (z - \mathbb{E}(X))^2 f_X(z)dz$.

1.3 Classic Tools for Probabilistic Analysis

We here introduce two classic tools of probability theory. The first one is the central limit theorem:

Theorem 1.1 (Central limit theorem)**.** *Let $\mu, \sigma \in \mathbb{R}$, let $(X_i)_{i \geqslant 1}$ be a sequence of i.i.d. random variable of expectation $\mathbb{E}(X_i) = \mu$ and of variance $V(X_i) = \sigma^2$ with $\sigma > 0$. For every $n \geqslant 1$, let $S_n \coloneqq \sum_{i \in [\![1,n]\!]} X_i$ and let $Y_n \coloneqq \sqrt{n}\frac{(S_n - \mu)}{\sigma}$. The probability distribution of Y_n converges, when n tends to infinity, to the normal distribution $\mathcal{N}(0,1)$. In particular, for all $a, b \in \mathbb{R}$ such that $a \leqslant b$, $\mathbb{P}(Y_n \in [a,b]) \underset{n \to +\infty}{\longrightarrow} \int_a^b e^{-\frac{x^2}{2}} dx$.*

The other useful tool is the so-called *Borel-Cantelli lemma*:

Lemma 1.2 (Borel-Cantelli lemma [Bor09])**.** *Let $(E_i)_{i \in \mathbb{N}}$ a countable family of events such that the sum $\sum_{i \in \mathbb{N}} \mathbb{P}(E_i)$ is finite. The probability that infinitely many events E_i occur is 0. Formally,*

$$\mathbb{P}(\bigcap_{i \in \mathbb{N}} \bigcup_{j \geqslant i} E_j) = 0.$$

1.4 Random Walks

We here introduce some notions of discrete random walks. Informally speaking, a discrete random walk in dimension d is a particle that moves randomly in an infinite d-dimensional grid, corresponding to the set $\mathbb{N}^d$. At each step, the particle takes a step in one of the $2d$ possible directions, chosen uniformly at random independently from the past.

The most important case for our purposes is the one-dimensional case, *i.e.*, $d = 1$. A *one-dimensional random walk* of parameter p is a sequence of random variables $(X_i)_{i \in \mathbb{N}}$ with values in $\mathbb{Z}$ such that $X_0 = 0$ and, for all $n \in \mathbb{N}$, for all $k_0, \ldots, k_n, k_{n+1} \in \mathbb{Z}$,

$$\mathbb{P}(X_{n+1} = k_{n+1} \mid X_0 = k_0, \ldots, X_n = k_n) = \begin{cases} p & \text{if } k_{n+1} = k_n + 1 \\ 1 - p & \text{if } k_{n+1} = k_n - 1 \\ 0 & \text{otherwise.} \end{cases}$$

Said otherwise, at every step, the value has probability p of increasing by 1 and probability $1 - p$ of decreasing by 1. An equivalent definition is that the value of the random walk at step n is $2\sum_{i=1}^{n} Y_i - n$ where the Y_i are i.i.d. Bernoulli variables of parameter p. The random walk is called *balanced* when $p = \frac{1}{2}$, *positively biased* when $p > \frac{1}{2}$ and *negatively biased* when $p < \frac{1}{2}$.

A natural question is whether a random walk *visits* a given integer z, *i.e.*, whether there is i such that $X_i = z$. This question has been extensively studied in the literature, see, *e.g.*, [Joh11]. We provide all results relevant for us in the following proposition.

Proposition 1.3. *Consider a one-dimensional random walk of parameter p.*

If $p = \frac{1}{2}$ (the random walk is balanced) then the random walk almost surely visits all integers.

If $p > \frac{1}{2}$ (the random walk is positively biased), then it almost surely visits all positive integers. Moreover, there is a non-zero probability that $X_i \geqslant 0$ for all $i \in \mathbb{N}$.

If $p < \frac{1}{2}$ (the random walk is negatively biased) then it almost surely visits all negative integers. Moreover, there is a non-zero probability that $X_i \leqslant 0$ for all $i \in \mathbb{N}$.

We will also informally refer to balanced random walks of higher dimension. For example, for dimension 2, a balanced random walk has, at every step, probability $\frac{1}{4}$ to go to the left, probability $\frac{1}{4}$ to go to the right, probability $\frac{1}{4}$ to go up and probability $\frac{1}{4}$ to go down. A classic result is that balanced random walks in dimensions 1 and 2 almost surely visit every point of the grid whereas, in dimension at least 3, for every point of the grid (except the origin), a balanced random walk has a non-zero probability of never visiting this point.

1.5 Well-Quasi-Orders and Descending Chains

In this section, we introduce some preliminary notions and results related to ordered sets and well-quasi-orders. The results introduced here, and more specifically the bound from Theorem 1.9, will be used in Chapter 3. We refer to [DFGSS17] for a complete introduction on the topic.

A *quasi-ordering* is a relation that is transitive and reflexive, but does not have to be antisymmetric. A *quasi-order* is a set E equipped with a quasi-ordering $\preceq$. Given a quasi-order $(E, \preceq)$, a subset $S \subseteq E$ is *upward-closed* (for $\preceq$) when, for all $s \in S$, for all $t \in E$, if $s \preceq t$ then $t \in S$. Similarly, a set $S \subset E$ is *downward-closed* when, for all $s \in S$, for all $t \in E$, if $t \preceq s$ then $t \in S$. Given a set $S \subseteq E$, we let $\uparrow S \coloneqq \{t \in E \mid \exists s \in S, s \preceq t\}$ its *upward-closure* and $\downarrow S \coloneqq \{t \in E \mid \exists s \in S, t \preceq s\}$ its *downward-closure*.

Definition 1.4. A *well-quasi-order* is a quasi-order $(E, \preceq)$ such that, for every infinite sequence $(x_i)_{i \in \mathbb{N}}$ of elements of E, there is $i < j$ such that $x_i \preceq x_j$.

In a well-quasi-order $(E, \preceq)$, any upward-closed set S has a finite set of minimal elements, denoted $\mathsf{basis}(S)$. This set $\mathsf{basis}(S)$ is called the *basis* of S and we have $S = \uparrow \mathsf{basis}(S)$.

A common well-quasi-ordered set is the set $\mathbb{N}^d$ equipped with the component-wise ordering. Let $d \geqslant 1$. Given $\vec{v}$ of $\mathbb{N}^d$ and $i \in [\![1, d]\!]$, we denote by $\vec{v}(i)$ its i-th component. Let $\leqslant_\times$ be the partial order over $\mathbb{N}^d$ such that $\vec{u} \leqslant_\times \vec{v}$ when, for all $i \in [\![1, d]\!]$, $\vec{u}(i) \leqslant \vec{v}(i)$.

Lemma 1.5 (Dickson's lemma)**.** $(\mathbb{N}^d, \leqslant_\times)$ *is a well-quasi-order.*

An alternative characterization of well-quasi-orders relies on the notion of descending chains. Given a quasi-order $(E, \preceq)$, a *descending chain* is a sequence $D_0 \supseteq D_1 \supseteq D_2 \dots$ of sets $D_k \subseteq E$ that are downward-closed for $\preceq$. A descending chain can either be *finite*, *i.e.*, of the form $(D_k)_{k \in [\![0,N]\!]}$ or *infinite*, *i.e.*, of the form $(D_k)_{k \in \mathbb{N}}$. To have a single notation for all descending chains, we denote them by (D_k), omitting the indexing set. The *length* of a finite descending chain $(D_k)_{k \in [\![0,N]\!]}$ is defined to be $N + 1$.

Proposition 1.6. *A quasi-order $(E, \preceq)$ is a well-quasi-order if and only if all its descending chains are finite.*

Although this implies that all descending chains of $(\mathbb{N}^d, \preceq)$ are finite, there is no general bound on their length. Even in the case of $\mathbb{N}^d$ with $d = 1$, which is $(\mathbb{N}, \leqslant)$, there is a descending chain of length L for every L, for example $(D_k)_{k \leqslant L-1}$ where $D_k \coloneqq \downarrow(L - k)$. However, it is possible to bound the length of a descending chain under some additional constraints. In this thesis, we will use a bound first introduced in [LS21], but in its improved form from [SS24]. This bound will apply under two conditions. The first condition is that the sequence is controlled, meaning that the size of the description of the set D_k (expressed below by its norm $\|D_k\|$) evolves in a reasonable way. The second condition, more technical, is that the sequence is *strongly monotone*; we present here a stronger condition, namely *ω-monotonicity*, that is sufficient for our needs. We now introduce some definitions needed to formally state the result from [SS24].

We extend $\mathbb{N}$ to $\mathbb{N}_\omega \coloneqq \mathbb{N} \cup \{\omega\}$ with $n < \omega$ for all $n \in \mathbb{N}$. Given a vector $\vec{v} \in \mathbb{N}_\omega^d$, its *norm* is $\|\vec{v}\| \coloneqq \max\{\vec{v}(i) \mid i \in [\![1, d]\!], \vec{v}(i) \neq \omega\}$, *i.e.*, the largest $\vec{v}(i)$ that is not ω, with the convention that $\max \emptyset = 0$. An *ideal* is the downward closure in $\mathbb{N}^d$ of a vector $\vec{v} \in \mathbb{N}_\omega^d$, *i.e.*, the set $I = \{\vec{u} \in \mathbb{N}^d \mid \forall i, \vec{u}(i) \leqslant \vec{v}(i)\}$; its *norm* $\|I\|$ is the norm $\|\vec{v}\|$. Note that the vector $\vec{v}$ is unique; it is called the *vector representing* I.

A downward-closed set $D \subseteq \mathbb{N}^d$ can be canonically represented as a finite union of ideals:

Proposition 1.7 (see, *e.g.*, [LS21])**.** *Let D be a downward-closed subset of $(\mathbb{N}^d, \leqslant_\times)$. The set D can be expressed as a union of ideals,* i.e.*, there is $n \in \mathbb{N}$ and $I_1, \dots, I_n$ ideals such that $D = I_1 \cup \dots \cup I_n$. Moreover, if we impose that, for all $i, j \in [\![1, n]\!]$, if $i \neq j$ then $I_i \not\subseteq I_j$, then $\{I_1, \dots, I_n\}$ is unique and called* decomposition *of D.*

The *norm* $\|D\|$ of a downward-closed set D is defined as the maximum of the norms of the ideals in its decomposition. Given $N > 0$, a descending chain (D_k) is *N-controlled* when, for all k, $\|D_k\| \leqslant (k+1)N$. An ideal I is *proper at step k* if I is in the decomposition of D_k but $I \notin D_{k+1}$. The sequence (D_k) is *ω-monotone* if, when an ideal I_{k+1} represented by vector $\vec{v}_{k+1}$ is proper at step $k+1$, there is an ideal I_k proper at step k whose representing vector $\vec{v}_k \in \mathbb{N}_\omega^d$ is such that, for all $i \in [\![1, d]\!]$, if $\vec{v}_{k+1}(i) = \omega$ then $\vec{v}_k(i) = \omega$.

Example 1.8. Let $d = 2$, $D_0 \coloneqq \downarrow(5, \omega), D_1 \coloneqq \downarrow(1, \omega), D_2 \coloneqq \downarrow(1, 13)$ and $D_3 \coloneqq \downarrow(1, 1)$. All D_k are downward-closed sets and they are also ideals (they each have a single ideal in their decomposition). The sequence $(D_k)_{k \leqslant 3}$ is a descending chain of length 4. This descending chain is 5-controlled, because $\|D_0\| = 5 \leqslant 5$, $\|D_1\| = 1 \leqslant 10$, $\|D_2\| = 13 \leqslant 15$ and $\|D_3\| = 1 \leqslant 20$. Also, (D_k) is ω-monotone; this is particularly easy to see in this example, because, for each k, the only proper ideal at step k is D_k itself. The ω-monotonicity entails from the fact that, for all $k \geqslant 1$, if the representing vector of D_k has ω at position i then the same is true for the representing vector of D_{k-1}. Intuitively, the descending chain D_k is ω-monotone because it only loses ω values but never gains new ones.

The length of descending chains that are controlled and ω-monotone can be bounded using the following result:

Theorem 1.9 ([SS24])**.** *Let $d, n > 0$. Every descending chain (D_k) of $(\mathbb{N}^d, \leqslant_\times)$ that is n-controlled and ω-monotone has length at most $n^{3^d(\log(d)+1)}$.*

Chapter 2

Asynchronous Shared-Memory Systems

2.1 Introduction

The first model considered in this thesis is the one of asynchronous shared-memory systems, or ASMS for short. In this model, the system is composed of arbitrarily many identical processes that run asynchronously and communicate via reading from and writing to a shared memory. This shared memory is composed of registers; each register contains, at a given point in time, a symbol from a finite alphabet. While a given read or a given write action is performed atomically, it is crucial that sequences of actions, and in particular read-write combinations, are performed non-atomically. By this, we mean that a process cannot performed several actions in a row while preventing other processes from acting: there is no locking mechanism. This hypothesis allows for the copycat property: whenever a process goes from q_1 to q_2, other processes in q_1 may also go to q_2 without affecting the rest of the system. This copycat property will play a central role in the analysis of the model. In fact, if we assume that sequences of actions are performed atomically, then the model gains an expressivity similar to the one of Petri nets [GS92; Esp14]. ASMS were first introduced in [EGM13; EGM16]; however, we highlight two differences between their model and ours. First, in our model, the shared-memory is composed of several independent registers, while the model of [EGM13; EGM16] has a single shared register. Combining several registers into one would not only imply an exponential blowup, but also alter the semantics of the model, as highlighted in Remark 2.28. Second, our registers initially hold a special symbol, denoted $\perp$. This choice is relatively common in parameterized verification of shared-memory systems [BMRSS16; AAR20]. As we will see, the presence of initial values increases the power of the model. Also, some shared-memory algorithms indeed require this hypothesis; for example, in [Asp02], registers initially hold value $\perp$, so that a process, upon reading a register, may notice whether someone has already written to this register or not.

In [EGM13; EGM16], the authors consider the case where the system has a distinguished

leader, and they consider different combinations of computing powers for the leader of follower, *e.g.*, depending on the leader is a finite-state machine or a pushdown machine, and similarly for the followers. In this chapter, we will consider only the case where there is no leader and where all followers are finite-state machines. With this choice, all processes are identical and a configuration of the system corresponds to a multiset of states, expressing how many processes are in each state, along with the value of the shared variables.

In this model, we study *reachability problems* where the question is whether there is a value of n for which one can reach, from the initial configuration with n processes, a configuration in an objective set. To describe a general family of objective sets, we define *presence constraints* where one may express, for each state q, whether there is at least one process in state q or none. More formally, a presence constraint is a Boolean combination of statements such as "state q is populated" and "register r contains symbol d". The associated reachability problem is called *presence reachability problem*, or PRP for short. Two special cases are Cover, where the objective set is the set of configurations with at least one process in some state q_f, and Target, where the objective set is the set of configurations where all processes are in q_f. Using a non-counting abstraction, we establish that PRP is in NP. Our aim is then to find cases decidable in polynomial time. In the general case, even Cover is NP-hard. If we make the assumption that the registers are *uninitialized*, *i.e.*, that the processes are not allowed to read $\perp$, then Cover is decidable in PTIME, but Target remains NP-hard. If the system has a single register, then Target is solvable in PTIME and the same is true for PRP if the formula is given is disjunctive normal form. The fact that the problem is easier with one register encourages us to consider the dependency of the complexity of our problems with respect to the number of registers, with the approach of parameterized complexity [Cyg+15]. We show that Cover in indeed FPT with respect to the number of registers; by contrast, Target is W[2]-hard.

This chapter is organized as follows. In Section 2.2, we introduce the model and the problems of interest. In Section 2.3, we present the copycat property. In Section 2.4, we introduce our non-counting abstraction for PRP. In Section 2.5, we study the complexity of PRP in the general case. In Section 2.6, we study the uninitialized case. In Section 2.7, we study the particular case with one register only. In Section 2.8, we study the dependency of the complexities of our problems with respect to the number of registers. We conclude this chapter with a closing discussion in Section 2.9. Most results presented in this section have been published in [Wal23].

Related works

The first study of a parameterized model with a shared memory and without atomic read-write combinations can be found in [Hag11]. In [Hag11], the system is composed of a leader and of arbitrarily many followers, all represented with pushdown machines. In this case, (control-state) coverability (*i.e.*, reachability of a configuration with at least one process in q_f) is decidable, which is interestingly not the case for the corresponding non-parameterized problem.

The model of ASMS is introduced in [EGM13] (and its journal version [EGM16]). In [EGM13; EGM16], the problem considered is safety, which is the dual of coverability: a safety instance is positive if, for every value of n, there is no execution that puts a process on q_f. The systems considered are with and without leader, and the descriptions of processes considered include finite-state machines and pushdown machines. Their approach is language-theoretical. It relies on copycat arguments and on the so-called *simulation lemma*, that expresses that a limited number of followers suffices to simulate the behavior of all followers. This allows the authors to prove that the safety problems is coNP-complete if the leader is described by a finite-state machine and the followers are described by pushdown machines, and also if the leader is described by a pushdown machine and the followers are described by finite-state machines. If both are described by pushdown machines, the problem becomes PSPACE-complete.

This work was extended to liveness verification in [DEGM15]. Liveness verification asks whether there is an *infinite* execution that satisfies an ω-regular property, expressed with a Büchi automaton. The liveness problems is NP-complete when all machines, leader and follower alike, are finite-state, and decidable in NEXPTIME when pushdown machines are considered instead. Notably, when leader and followers are finite-state machines, liveness lies in the same complexity class as coverability, which is rather unusual as liveness is typically much harder than coverability. A fine-grained analysis of the complexity of the two problems (with leader and follower, all finite-state machines) is performed in [CMS19], where the authors establish that the deterministic complexity coverability and liveness only differ by a polynomial factor.

One last significant work on ASMS is the study of almost-sure coverability under a stochastic scheduler [BMRSS16]. In [BMRSS16], the authors establish that there is a cutoff for almost-sure coverability, *i.e.*, a value N such that the answer to almost-sure coverability is the same for every $n \geqslant N$. They also provide a doubly-exponential bound on the smallest cutoff value, and prove that the answer to almost-sure coverability for large values of n can be decided in PSPACE. This result is extended to more general objectives such as almost-sure termination (all processes in q_f) in [Sta17]. We will present the work of [BMRSS16] in more detail in Section 5.4, as we will attempt to extend their results to our round-based setting from Chapter 4.

2.2 Asynchronous Shared-Memory Systems

2.2.1 Protocols

The *protocol* is the description of the common finite automaton, which intuitively corresponds to the code that the processes are running.

Definition 2.1. A (roundless) *Asynchronous Shared-Memory System*, or ASMS for short, is described by a tuple $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \perp, \Delta \rangle$ where

— Q is a finite set of *states* with a distinguished *initial state* $q_0 \in Q$;

— $\mathsf{dim} \in [\![1, +\infty[\![$ is the number of shared *registers*;

— $\mathbb{D}$ is a finite set of *symbols* containing the *initial symbol* $\perp$;

— $\Delta \subseteq Q \times \mathcal{A} \times Q$ is the set of *transitions*, where $\mathcal{A} := \{\mathsf{read}_r(\mathsf{d}) \mid r \in [\![1, \mathsf{dim}]\!], \mathsf{d} \in \mathbb{D}\} \cup \{\mathsf{write}_r(\mathsf{d}) \mid r \in [\![1, \mathsf{dim}]\!], \mathsf{d} \in \mathbb{D} \setminus \{\perp\}\} \cup \{\circledast\}$ is the set of *actions*. Given $\delta = (q, \mathsf{act}, q') \in \Delta$, q is the *source state* of δ, act is its *action* and q' is its *destination state*.

In this chapter and the next one, we sometimes refer to the above-defined ASMS as *roundless* to distinguish them from round-based ASMS introduced in Chapter 4. We refer to the tuple $\mathcal{P}$ as the *protocol* of the roundless ASMS: the system, in the traditional sense, is defined by the protocol along with a number of participating *processes*. The *size* of the protocol $\mathcal{P}$ is defined as $|\mathcal{P}| := |Q| + |\mathbb{D}| + |\Delta| + \mathsf{dim}$.

There are three types of actions. Actions of the form $\mathsf{write}_r(\mathsf{d})$ are *write actions* correspond to writing a symbol to a register, while actions of the form $\mathsf{read}_r(\mathsf{d})$ are *read actions* that correspond to reading a symbol from a register. Finally, action $\circledast$ is an *internal action* that does not interact with the shared registers, so that it simply corresponds to a process changing its state [1]. A transition with a read action is called *read transition*, a transition with a write action is called *write transition* and a transition with an internal action is called *internal transition*. The variable $r \in [\![1, \mathsf{dim}]\!]$ denotes a shared register and $\mathsf{d} \in \mathbb{D}$ denotes the symbol. Note that a given transition may only be labeled by one action. While a given read action or a given write action is performed atomically (they are modeled as instantaneous actions), read-write combinations are performed non-atomically: in order to read the content of a register and then write to that register, one needs two separate transitions. This has the important consequence that no process can

1. The internal action $\circledast$ does not change the expressive power of our model and is here for the sake of simplicity only. We could indeed encode a $\circledast$ transition with, *e.g.*, parallel read transitions that read all possible values from a register.

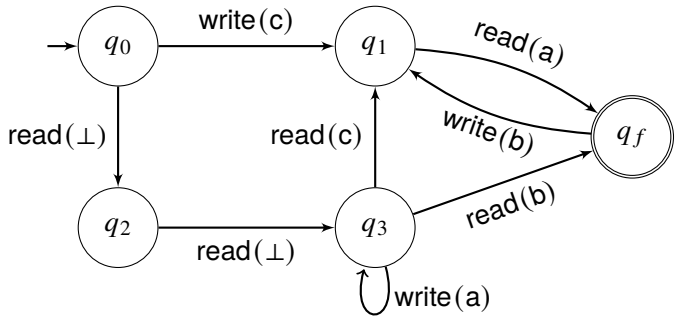


Figure 2.1 – An example of an ASMS protocol with only one register (dim = 1). The index 1 of the register is omitted from the labels of the transitions.

perform a read-write combination while excluding all other processes from acting. This choice will be fundamental in Section 2.3 because it gives the system strong monotonicity properties. The alternative would have been to allow transitions to be labeled by words of actions, so that a process may perform several actions in a row with the guarantee that other processes will remain idle in the meantime. The latter choice would in fact yield a model that is, arguably, too powerful; we will expand on this subject in Section 2.9.

In all the following, we visually represent protocols as in Fig. 2.1: states are represented as circles and transitions as arrows labeled by their actions.

2.2.2 Semantics

The shape of the system at a given point in time is called a *configuration*. Recall that the system is composed of a finite number of processes which all execute the same protocol. Because we assume that all processes are anonymous, the only relevant information about a given process is its state. Therefore, a configuration is characterized by how many processes are in each state of the protocol, which we represent by a multiset of states along with the content of the registers. Each register stores one *symbol* from the finite set $\mathbb{D}$ at a time. Hence, a *configuration* is a pair $\gamma = \langle \mu, \vec{d} \rangle$ in the set $\Gamma := \mathcal{M}(Q) \times \mathbb{D}^{\mathsf{dim}}$ such that $|\mu| > 0$. Given $\vec{d} \in \mathbb{D}^{[\![1,\mathsf{dim}]\!]}$ and $r \in [\![1, \mathsf{dim}]\!]$, we denote by $\vec{d}(r)$ the r-th component of $\vec{d}$. Let $\mathsf{st}(\gamma) := \mu$ be the multiset which indicates the number of processes in each state, and $\mathsf{data}(\gamma) := \vec{d}$ mapping to each register its symbol: for all $r \in [\![1, \mathsf{dim}]\!]$, $\mathsf{data}(\gamma)(\mathbf{reg})$ is the symbol contained in register **reg** in γ. We denote by $\mathsf{supp}(\gamma) := \overline{\mathsf{st}(\gamma)} = \{q \in Q \mid \mathsf{st}(\gamma)(q) > 0\}$ the support of the multiset $\mathsf{st}(\gamma)$. The *size* $|\gamma|$ of a configuration is the number of processes involved, *i.e.*, $|\gamma| := |\mathsf{st}(\gamma)|$. A configuration is *initial* if all processes are in q_0 while all registers have value $\perp$. Note that, for all $n \geqslant 1$, there is only

one initial configuration of size n, namely $\gamma_0(n) \coloneqq \langle n \cdot q_0, \bot^{\mathsf{dim}} \rangle$ where $\bot^{\mathsf{dim}}$ is the element of $\mathbb{D}^{\mathsf{dim}}$ whose components are all $\bot$. We denote by $\Gamma_0 \coloneqq \{\gamma_0(n) \mid n \geqslant 1\}$ the set of initial configurations.

Given $\gamma, \gamma' \in \Gamma$, γ' is a *successor* of γ when there exists $\delta = (q, \mathsf{act}, q') \in \Delta$ such that $\mathsf{st}(\gamma)(q) > 0$, $\mathsf{st}(\gamma') = (\mathsf{st}(\gamma) \ominus \{q\}) \oplus \{q'\}$ and:

— if $\mathsf{act} = \mathsf{read}_r(\mathsf{d})$ then $\mathsf{data}(\gamma)(r) = \mathsf{d}$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$,

— if $\mathsf{act} = \mathsf{write}_r(\mathsf{d})$ then $\mathsf{data}(\gamma')(r) = \mathsf{d}$ and $\forall r' \neq r$, $\mathsf{data}(\gamma')(r') = \mathsf{data}(\gamma)(r')$,

— if $\mathsf{act} = \circledast$ then $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$.

In that case, we write $\gamma \xrightarrow{\delta} \gamma'$ or simply $\gamma \to \gamma'$, which is called a *step*; this step is obtained by *applying* δ from γ. If there is a such a configuration γ', then δ *can be applied* from γ; in this case, γ' is unique. If act is a read action then the step $\gamma \xrightarrow{\delta} \gamma'$ is called a *read step*; it is called a *write step* when act is a write action and an *internal step* when $\mathsf{act} = \circledast$. An *execution* is a sequence $\rho = \gamma_0, \delta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \delta_\ell, \gamma_\ell$ such that, for all i, $\gamma_i \xrightarrow{\delta_{i+1}} \gamma_{i+1}$. Its *length* $\mathsf{len}(\rho)$ is simply defined as its number ℓ of steps. We write $\gamma_0 \xrightarrow{\delta_1 \ldots \delta_\ell} \gamma_\ell$ for the existence of an execution from γ_0 to γ_ℓ of sequence of transitions $\delta_1, \ldots, \delta_\ell$, and simply $\gamma_0 \xrightarrow{*} \gamma_\ell$ for the existence of an execution from γ_0 to γ_ℓ. Given an execution $\rho = \gamma_0, \delta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \delta_\ell, \gamma_\ell$, the configurations $\gamma_0, \ldots, \gamma_\ell$ are the configurations *visited* in ρ.

A configuration γ' is *reachable from* a configuration γ when $\gamma \xrightarrow{*} \gamma'$. Given a set $C \subseteq \Gamma$, we write $\mathsf{Post}^*(C) \coloneqq \{\gamma' \mid \exists \gamma \in C,\ \gamma \xrightarrow{*} \gamma'\}$. Dually, we write $\mathsf{Pre}^*(C) \coloneqq \{\gamma \mid \exists \gamma' \in C, \gamma \xrightarrow{*} \gamma'\}$. A configuration is *reachable* when it is in $\mathsf{Post}^*(\Gamma_0)$. A state $q \in Q$ is *coverable from* γ if there is an execution $\rho : \gamma \xrightarrow{*} \gamma'$ where $\mathsf{st}(\gamma')(q) > 0$; the execution ρ *covers* state q. A state q is *coverable* when it is coverable from Γ_0, *i.e.*, when there is $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\mathsf{st}(\gamma)(q) > 0$.

Example 2.2. Figure 2.1 provides an example of an ASMS protocol $\mathcal{P}$ with $\mathbb{D} = \{\bot, \mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathsf{dim} = 1$, hence read and write actions are implicitly on register 1. The following execution with two processes witnesses that $\langle q_f \oplus q_3, \mathsf{a} \rangle \in \mathsf{Post}^*(\Gamma_0)$:

$$\langle 2 \cdot q_0, \bot \rangle \xrightarrow{(q_0, \mathsf{read}(\bot), q_2)} \langle q_0 \oplus q_2, \bot \rangle \xrightarrow{(q_2, \mathsf{read}(\bot), q_3)} \langle q_0 \oplus q_3, \bot \rangle \xrightarrow{(q_0, \mathsf{write}(\mathsf{c}), q_1)}$$
$$\langle q_1 \oplus q_3, \mathsf{c} \rangle \xrightarrow{(q_3, \mathsf{write}(\mathsf{a}), q_3)} \langle q_1 \oplus q_3, \mathsf{a} \rangle \xrightarrow{(q_1, \mathsf{read}(\mathsf{a}), q_f)} \langle q_f \oplus q_3, \mathsf{a} \rangle.$$

We make the simple observation that additional processes cannot disable a transition:

Fact 2.3. If $\langle \mu_1, \vec{d}_1 \rangle \xrightarrow{\delta} \langle \mu_2, \vec{d}_2 \rangle$ then, for every $\mu_+ \in \mathcal{M}(Q)$, $\langle \mu_1 \oplus \mu_+, \vec{d}_1 \rangle \xrightarrow{\delta} \langle \mu_2 \oplus \mu_+, \vec{d}_2 \rangle$.

Given a configuration γ, we call a register r *blank* when $\mathsf{data}(\gamma)(r) = \bot$. By definition of the set of actions $\mathcal{A}$, processes are not allowed to write the initial symbol $\bot$. Registers are all initially blank, and once a register is written, it may never be blank again.

Remark 2.4. The choice to forbid transitions writing $\perp$ can be made without loss of generality. Indeed, consider an ASMS protocol where transitions are allowed to be labeled by write actions writing symbol $\perp$; we build an equivalent protocol with no such action. To do so, we replace actions writing $\perp$ by actions that write a fresh symbol $\perp' \neq \perp$ that, upon reading, is not distinguished from $\perp$ by the processes. Formally, this can be achieved by replacing each $\mathsf{write}_r(\perp)$ action in Δ by $\mathsf{write}_r(\perp')$ and by adding, for every transition of the form $(q, \mathsf{read}_r(\perp), q')$ in Δ, an alternative transition $(q, \mathsf{read}_r(\perp'), q')$ to Δ.

2.2.3 Reachability Problems

We are interesting in reachability problems, *i.e.*, problems asking whether the set of reachable configurations $\mathsf{Post}^*(\Gamma_0)$ intersects some particular set $R \subseteq \Gamma$, called *reachability objective*. Formally, one asks whether $\mathsf{Post}^*(\Gamma_0) \cap R \neq \emptyset$. Of course, this does not describe a problem but rather a family of problems, since we need to specify the shape and description of the set R. The first problem of interest is the *coverability problem* (Cover) where the reachability objective consists in all configurations covering a state q_f:

Cover for roundless ASMS
Input: A roundless ASMS protocol $\mathcal{P}$, $q_f \in Q$
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\mathsf{st}(\gamma)(q_f) > 0$?

Note that, because the model is parameterized, a witness execution of Cover may have an arbitrarily large number of processes. The dual of the coverability problem is the *safety problem*, the answer to which is yes when state q_f cannot be covered regardless of the number of processes. Here, q_f can be thought of as an error state that must be avoided: if q_f cannot be covered then the system is safe no matter the number of participants.

A similar problem is the *target problem* (Target) where processes must synchronize at q_f :

Target for roundless ASMS
Input: A roundless ASMS protocol $\mathcal{P}$, $q_f \in Q$
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ s.t. for all $q \neq q_f$, $\mathsf{st}(\gamma)(q) = 0$?

Remark 2.5. Target is a harder problem than Cover, because Cover reduces to Target in linear time. Indeed, it suffices to add a loop on q that writes a fresh symbol ok to register 1, and add, for every state $q \neq q_f$, a transition $(q, \mathsf{read}_1(\mathsf{ok}), q_f)$.

Presence constraints are Boolean combinations (with $\wedge$, $\vee$ and $\neg$) of atomic propositions of the form $\mathsf{popu}(q)$ with $q \in Q$, or of the form $\mathsf{cont}(r, \mathsf{d})$ with $r \in [\![1, \mathsf{dim}]\!]$ and $\mathsf{d} \in \mathbb{D}$. A

presence constraint is interpreted over a configuration γ by interpreting $\mathsf{popu}(q)$ as true if and only if $\mathsf{st}(\gamma)(q) > 0$ (q is *populated* in γ) and $\mathsf{cont}(r, \mathsf{d})$ as true if and only if $\mathsf{data}(\gamma)(\mathbf{reg}) = \mathsf{d}$ (register **reg** contains d). Note that presence constraints cannot refer to how many processes are on a given state. We write $\gamma \models \phi$ when configuration γ satisfies presence constraint ϕ.

Example 2.6. If $Q = \{q_1, q_2, q_3\}$, $\mathsf{dim} = 2$, $\mathbb{D} = \{\bot, a, b\}$ and $\phi := \mathsf{popu}(q_1) \vee (\mathsf{popu}(q_2) \wedge \mathsf{cont}(1, a))$ then $\langle q_1 \oplus q_3, \bot^2 \rangle \models \phi$, $\langle 2 \cdot q_2, (a, b) \rangle \models \phi$ but $\langle 2 \cdot q_2, b^2 \rangle \not\models \phi$.

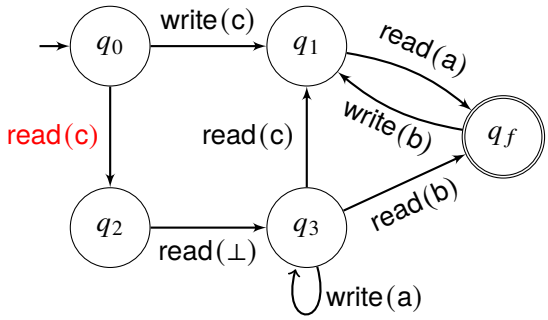
The *Presence Reachability Problem* (PRP) generalizes both Cover and Target. A similar problem has been studied in [DSTZ12] for reconfigurable broadcast networks, under the name of "cardinal reachability problem restricted to CC[$\geqslant 1, = 0$]".

PRP for roundless ASMS
Input: A roundless shared-memory protocol $\mathcal{P}$, a presence constraint ϕ
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \models \phi$?

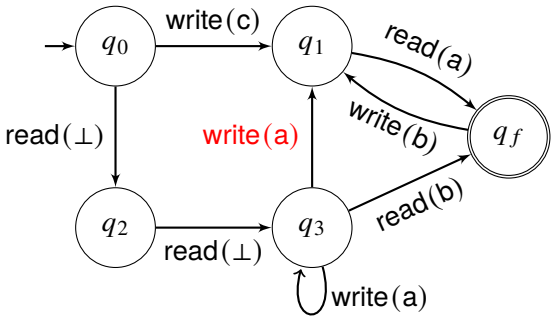
The formula ϕ automatically makes PRP NP-hard: one can easily encode the SAT problem. For this reason, we also consider the *DNF Presence Reachability Problem* (dnfPRP), in which ϕ is in disjunctive normal form. A formula ϕ is in *disjunctive normal form* when ϕ is of the form $\bigvee_{1 \leqslant i \leqslant m} \ell_{i,1} \wedge \cdots \wedge \ell_{i,k_i}$ where $\{\ell_{i,j} \mid i \in [\![1, m]\!], j \in [\![1, k_i]\!]\}$ is the set of *literals* which are atomic propositions or negation of atomic propositions. Formally, for all $i \in [\![1, m]\!]$ and $j \in [\![1, k_i]\!]$, $\ell_{i,j} \in \{\mathsf{popu}(q), \neg\mathsf{popu}(q), \mathsf{cont}(r, \mathsf{d}), \neg\mathsf{cont}(r, \mathsf{d}) \mid q \in Q, r \in [\![1, \mathsf{dim}]\!], \mathsf{d} \in \mathbb{D}\}$. Cover and Target are special cases of dnfPRP, with $\phi = \mathsf{popu}(q_f)$ for Cover and $\phi = \bigwedge_{q \neq q_f} \neg\mathsf{popu}(q)$ for Target.

Example 2.7. Consider again the protocol $\mathcal{P}$ defined in Figure 2.1. $(\mathcal{P}, q_f)$ is a positive instance of Cover, as proved in Example 2.2. However, consider the variant $\mathcal{P}_1$ from Fig. 2.2(a), obtained from $\mathcal{P}$ by changing to $\mathsf{read}(\mathsf{c})$ the label of the transition from q_0 to q_2. $(\mathcal{P}_1, q_f)$ is a negative instance of Cover. In fact, a process can only get in state q_2 once c has been written to the register, but then $\bot$ can no longer be read so no process may go in state q_3. This means that a cannot be written and no process may go from q_1 to q_f.

Again with $\mathcal{P}$ the protocol depicted in Fig. 2.1, $(\mathcal{P}, q_f)$ is a negative instance of Target. To prove this, assume by contradiction that we have a witness execution $\rho : \gamma_0 \xrightarrow{*} \gamma_f$ such that $\gamma_0 \in \Gamma_0$ and $q \notin \mathsf{st}(\gamma_f)$ for all $q \neq q_f$. Let γ be the penultimate configuration in ρ, so that $\gamma \xrightarrow{\delta} \gamma_f$. Transition δ is either the transition from q_1 to q_f or the transition from q_3 to q_f. To send a process from q_1 to q_f, a needs to be in the register. However, a can only be written from q_3 and a process cannot leave q_3 while a is in the register. This means that, whenever a is in the register, at least one process must be in q_3, hence the last transition δ performed in ρ cannot be



(a) The protocol $\mathcal{P}_1$ obtained from $\mathcal{P}$ by changing the label of the transition from q_0 to q_2.



(b) The protocol $\mathcal{P}_2$ obtained from $\mathcal{P}$ by changing the label of the transition from q_3 to q_1.

Figure 2.2 – Two variants of the protocol from Fig. 2.1.

the transition from q_1 to q_f. Similarly, b can only be in the register when there is a process in q_1, hence δ cannot be the transition from q_3 to q_f, a contradiction.

Consider now the protocol $\mathcal{P}_2$ from Fig. 2.2(b), obtained from $\mathcal{P}$ by changing to write(a) the label of the transition from q_3 to q_1. $(\mathcal{P}_2, q_f)$ is a positive instance of TARGET, as witnessed by the following execution:

$$\langle 2 \cdot q_0, \bot \rangle \xrightarrow{(q_0,\mathsf{read}(\bot),q_2)} \langle q_0 \oplus q_2, \bot \rangle \xrightarrow{(q_2,\mathsf{read}(\bot),q_3)} \langle q_0 \oplus q_3, \bot \rangle \xrightarrow{(q_0,\mathsf{write}(c),q_1)}$$
$$\langle q_1 \oplus q_3, \mathsf{c} \rangle \xrightarrow{(q_3,\mathsf{write}(a),q_1)} \langle 2 \cdot q_1, \mathsf{a} \rangle \xrightarrow{(q_1,\mathsf{read}(a),q_f)} \langle q_1 \oplus q_f, \mathsf{a} \rangle \xrightarrow{(q_1,\mathsf{read}(a),q_f)} \langle 2 \cdot q_f, \mathsf{a} \rangle.$$

Let $\phi := \neg\mathsf{popu}(q_3) \wedge (\mathsf{cont}(1,\mathsf{a}) \vee (\mathsf{cont}(1,\mathsf{b}) \wedge \neg\mathsf{popu}(q_1)))$ be a presence constraint, then $(\mathcal{P}, \phi)$ is a negative instance of PRP. Indeed, by the same reasoning as above, if a is in the register then q_3 must be populated and if b is in the register then q_1 must be populated.

Let $\phi := \mathsf{popu}(q_1) \wedge \neg\mathsf{popu}(q_3) \wedge \mathsf{popu}(q_f) \wedge \mathsf{cont}(1,\mathsf{c})$. ϕ is a presence constraint in DNF and $(\mathcal{P}, \phi)$ is a positive instance of PRP and DNFPRP. In fact, one can reach a configuration with four processes satisfying ϕ as follows:

$$\langle 4 \cdot q_0, \bot \rangle \xrightarrow{(q_0,\mathsf{read}(\bot),q_2)} \langle 3 \cdot q_0 \oplus q_2, \bot \rangle \xrightarrow{(q_2,\mathsf{read}(\bot),q_3)} \langle 3 \cdot q_0 \oplus q_3, \bot \rangle \xrightarrow{(q_0,\mathsf{write}(c),q_1)}$$
$$\langle 2 \cdot q_0 \oplus q_1 \oplus q_3, \mathsf{c} \rangle \xrightarrow{(q_0,\mathsf{write}(c),q_1)} \langle q_0 \oplus 2 \cdot q_1 \oplus q_3, \mathsf{c} \rangle \xrightarrow{(q_3,\mathsf{write}(a),q_3)}$$
$$\langle q_0 \oplus 2 \cdot q_1 \oplus q_3, \mathsf{a} \rangle \xrightarrow{(q_1,\mathsf{read}(a),q_f)} \langle q_0 \oplus q_1 \oplus q_3 \oplus q_f, \mathsf{a} \rangle \xrightarrow{(q_f,\mathsf{write}(b),q_1)} \langle q_0 \oplus 2 \cdot q_1 \oplus q_3, \mathsf{b} \rangle$$
$$\xrightarrow{(q_3,\mathsf{read}(b),q_f)} \langle q_0 \oplus 2 \cdot q_1 \oplus q_f, \mathsf{b} \rangle \xrightarrow{(q_0,\mathsf{write}(c),q_1)} \langle 3 \cdot q_1 \oplus q_f, \mathsf{c} \rangle.$$

2.3 Copycat and Accelerated Semantics

We start this section with the following observation. Assume that we have a configuration γ from which one process in q fires a transition δ. Another process in q may immediately perform δ without changing the content of the registers: indeed, if δ is a read transition then the symbol in the register is still there and may be read again, and if δ is a write transition then writing the symbol to the register again does not change its content. This means that, given an execution and a process in this execution, we are able to add a copycat [2] process that consistently follows the first process along the execution. This observation gives us a useful monotonicity property, formalized with the following lemma:

Lemma 2.8 (Copycat)**.** *Let* $\gamma_1, \gamma_2 \in \Gamma$, $\delta = (q_1, \mathsf{act}, q_2) \in \Delta$ *such that* $\gamma_1 \xrightarrow{\delta} \gamma_2$. *For every* $m \geqslant 0$, *we have* $\langle \mathsf{st}(\gamma_1) \oplus m \cdot q_1, \mathsf{data}(\gamma_1)\rangle \xrightarrow{\delta^{m+1}} \langle \mathsf{st}(\gamma_2) \oplus m \cdot q_2, \mathsf{data}(\gamma_2)\rangle$.

Proof. This proof is almost a direct consequence of Fact 2.3 combined with the observation made above. First, by Fact 2.3, we have that $\langle \mathsf{st}(\gamma_1) \oplus m \cdot q_1, \mathsf{data}(\gamma_1)\rangle \xrightarrow{\delta} \langle \mathsf{st}(\gamma_2) \oplus m \cdot q_1, \mathsf{data}(\gamma_2)\rangle$, hence we will prove that $\langle \mathsf{st}(\gamma_2) \oplus m \cdot q_1, \mathsf{data}(\gamma_2)\rangle \xrightarrow{\delta^m} \langle \mathsf{st}(\gamma_2) \oplus m \cdot q_2, \mathsf{data}(\gamma_2)\rangle$.

We prove by induction on k that, for all $k \geqslant 0$, $\langle \mathsf{st}(\gamma_2) \oplus k \cdot q_1, \mathsf{data}(\gamma_2)\rangle \xrightarrow{\delta^k} \langle \mathsf{st}(\gamma_2) \oplus k \cdot q_2, \mathsf{data}(\gamma_2)\rangle$. It it trivially true for $k = 0$. Assume that it is true for k and prove it for $k + 1$. By induction hypothesis and Fact 2.3, we have $\langle \mathsf{st}(\gamma_2) \oplus (k + 1) \cdot q_1, \mathsf{data}(\gamma_2)\rangle \xrightarrow{\delta^k} \langle \mathsf{st}(\gamma_2) \oplus q_1 \oplus k \cdot q_2, \mathsf{data}(\gamma_2)\rangle$. Let $\gamma \coloneqq \langle \mathsf{st}(\gamma_2) \oplus q_1 \oplus k \cdot q_2, \mathsf{data}(\gamma_2)\rangle$. We make a case disjunction on act:

— if $\mathsf{act} = \mathsf{write}_r(\mathsf{d})$ is a write action then, because $\gamma_1 \xrightarrow{\delta} \gamma_2$, we have $\mathsf{data}(\gamma)(r) = \mathsf{data}(\gamma_2)(r) = \mathsf{d}$ and applying δ from γ does not change the content of the registers;

— if $\mathsf{act} = \mathsf{read}_r(\mathsf{d})$ is a read action then, because $\gamma_1 \xrightarrow{\delta} \gamma_2$, we have $\mathsf{data}(\gamma_1)(r) = \mathsf{data}(\gamma_2)(r) = \mathsf{data}(\gamma)(r) = \mathsf{d}$ and δ can be applied from γ,

— if $\mathsf{act} = \circledast$ is an internal action then δ can be applied from γ.

This proves that $\langle \mathsf{st}(\gamma_2) \oplus q_1 \oplus k \cdot q_2, \mathsf{data}(\gamma_2)\rangle \xrightarrow{\delta} \langle \mathsf{st}(\gamma_2) \oplus (k+1) \cdot q_2, \mathsf{data}(\gamma_2)\rangle$, concluding the proof. □

The previous lemma tells us that, whenever a transition δ from q_1 to q_2 is fired, we can freely move processes from q_1 to q_2 by applying δ as many times as we want. This leads us

2. According to *Cambridge dictionary*, a copycat is "something that is intentionally done or made to be very similar to something else". The word is in fact a compound word of *copy* and *cat*, as in old English the word *cat* was used as a derogatory term for a person.

to defining the *accelerated semantics*: given $\gamma, \gamma' \in \Gamma$ and $\delta = (q_1, \mathsf{act}, q_2) \in \Delta$, we define $\gamma \xrightarrow[\mathsf{acc}]{\delta} \gamma'$ whenever there is $k \geqslant 1$ such that $\gamma \xrightarrow{\delta^k} \gamma'$.

Again, we extend the definition to define the reflexive and transitive closure $\xrightarrow[\mathsf{acc}]{*}$. Obviously, the accelerated semantics defines the same reachability relation as the normal semantics:

Fact 2.9. For every $\gamma, \gamma' \in \Gamma$, one has $\gamma \xrightarrow{*} \gamma'$ if and only if $\gamma \xrightarrow[\mathsf{acc}]{*} \gamma'$.

One can also make the following, similar observation:

Lemma 2.10. *Consider γ_1, γ_2, q_2 such that $\gamma_1 \xrightarrow{*} \gamma_2$, $q_2 \in \mathsf{supp}(\gamma_2)$. There exists $q_1 \in \mathsf{supp}(\gamma_1)$ s.t. $\langle \mathsf{st}(\gamma_1) \oplus q_1, \mathsf{data}(\gamma_1)\rangle \xrightarrow{*} \langle \mathsf{st}(\gamma_2) \oplus q_2, \mathsf{data}(\gamma_2)\rangle$.*

Proof. The intuition is that we take one process ending in q_2, let q_1 its state in γ_1 and add another process that mimics the behavior of this process so that it goes from q_1 to q_2 as well. We will prove that a process may mimic another one using iterated applications of Lemma 2.8. To prove this formally, we prove the result by induction on the length of the execution. If the execution is of length 0 then one simply considers $q_1 := q_2$. Let $\rho : \gamma_1 \xrightarrow{*} \gamma_2$ and suppose that the property is true for all executions of length $\mathsf{len}(\rho) - 1$. Decompose ρ into $\gamma_1 \xrightarrow{*} \gamma_3 \xrightarrow{\delta} \gamma_2$. If q_2 is not the destination of δ, then $q_2 \in \mathsf{supp}(\gamma_3)$: we directly apply the induction hypothesis on $\gamma_1 \xrightarrow{*} \gamma_3$ and q_2 and conclude by applying δ to get to $\langle \mathsf{st}(\gamma_2) \oplus q_2, \mathsf{data}(\gamma_2)\rangle$ by Fact 2.3. Assume that q_2 is the destination of δ; let q_3 be the source state of δ. We have $q_3 \in \mathsf{supp}(\gamma_3)$, so we apply the induction hypothesis on $\gamma_1 \xrightarrow{*} \gamma_3$ and q_3: we obtain that there exists $q_1 \in \mathsf{supp}(\gamma_1)$ such that $\langle \mathsf{st}(\gamma_1) \oplus q_1, \mathsf{data}(\gamma_1)\rangle \xrightarrow{*} \langle \mathsf{st}(\gamma_3) \oplus q_3, \mathsf{data}(\gamma_3)\rangle$. Moreover, by Lemma 2.8 we have $\langle \mathsf{st}(\gamma_3) \oplus q_3, \mathsf{data}(\gamma_3)\rangle \xrightarrow{\delta} \langle \mathsf{st}(\gamma_2) \oplus q_2, \mathsf{data}(\gamma_2)\rangle$, concluding the proof. □

Moreover, the set of initial configurations expresses that all processes are in q_0. Therefore, it is not sensitive to the number of processes. Combined with the result above, this gives a monotonicity property for the set $\mathsf{Post}^*(\Gamma_0)$:

Corollary 2.11. *For all $\gamma \in \mathsf{Post}^*(\Gamma_0)$ and $q \in \mathsf{supp}(\gamma)$, $\langle \mathsf{st}(\gamma) \oplus q, \mathsf{data}(\gamma)\rangle \in \mathsf{Post}^*(\Gamma_0)$.*

Proof. Because $\gamma \in \mathsf{Post}^*(\Gamma_0)$, by letting $n := |\gamma|$, we have $\gamma_0(n) \xrightarrow{*} \gamma$. Let $q \in \mathsf{supp}(\gamma)$; applying Lemma 2.10 proves that $\gamma_0(n+1) \xrightarrow{*} \langle \mathsf{st}(\gamma) \oplus q, \mathsf{data}(\gamma)\rangle$ and therefore that $\langle \mathsf{st}(\gamma) \oplus q, \mathsf{data}(\gamma)\rangle \in \mathsf{Post}^*(\Gamma_0)$. □

2.4 An Abstraction for Presence Reachability Problems

In this section, we define an abstract semantics that is sound and complete with respect to PRP. This abstraction is a *non-counting abstraction*, a classic tool of parameterized verification

of systems that enjoy strong monotonicity properties (see, *e.g.*, [DSTZ12; DEGM15]). The intuition is that there is no need to count processes, thanks to the following three observations:

(Obs1) initial configurations allow for arbitrarily many processes in q_0,

(Obs2) the copycat principle presented in Section 2.3 allows us, given an execution that sends at least one process from q_0 to q, to build an execution equal to the original one except with arbitrarily many processes going from q_0 to q,

(Obs3) presence constraints are not able to count processes, as they may only express whether a given state is populated or not.

We thus define an abstraction where we do not store how many processes are in each state.

Definition 2.12. An *abstract configuration* is a pair $\sigma = \langle \mathsf{st}(\sigma), \mathsf{data}(\sigma) \rangle \in \overline{\Gamma} \coloneqq 2^Q \times \mathbb{D}^{\mathsf{dim}}$. Given a configuration $\gamma \in \Gamma$, its *abstract projection* $\overline{\gamma}$ is the abstract configuration $\langle \mathsf{supp}(\gamma), \mathsf{data}(\gamma) \rangle$.

We will sometimes refer to actual configurations as *concrete* to distinguish them from abstract configurations. In an abstract configuration, the only information kept about each state is whether there is at least one process in the state, *i.e.*, whether the state is populated. There is only one initial abstract configuration $\sigma_0 \coloneqq \langle \{q_0\}, \bot^{\mathsf{dim}} \rangle$: for all n, $\overline{\gamma_0(n)} = \sigma_0$.

We define the abstract semantics as the abstract projection of the accelerated concrete semantics:

Definition 2.13. For every $\delta \in \Delta$, for every $\sigma_1, \sigma_2 \in \overline{\Gamma}$, we let $\sigma_1 \xrightarrow{\delta} \sigma_2$, called an *abstract step*, if and only if there is $\gamma_1, \gamma_2 \in \Gamma$ such that $\overline{\gamma_1} = \sigma_1$, $\overline{\gamma_2} = \sigma_2$ and $\gamma_1 \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$.

For the sake of clarity, we explicit under what conditions one has $\sigma_1 \xrightarrow{\delta} \sigma_2$ in the abstract semantics. Let $(q_1, \mathsf{act}, q_2) \coloneqq \delta$. The conditions on the content of the registers to have $\sigma_1 \xrightarrow{\delta} \sigma_2$ are identical to the ones we had for concrete configurations. However, for the set of populated states, there are now two options, making the semantics non-deterministic even for a fixed transition. The first option is $\mathsf{st}(\sigma_2) = \mathsf{st}(\sigma_1) \cup \{q_2\}$. This corresponds to sending some processes from q_1 to q_2, but not all of them, and is called a *non-deserting* step. The second option (possible when $q_1 \neq q_2$ only) is $\mathsf{st}(\sigma_2) = (\mathsf{st}(\sigma_1) \setminus \{q_1\}) \cup \{q_2\}$; it corresponds to sending all processes in q_1 to q_2, and is therefore called a *deserting* step.

In the (non-accelerated) concrete semantics, implementing a deserting step might require several steps, in fact as many steps as there are processes in q_1. In the accelerated semantics, emptying a state can always be done in one step. Hence, the abstract semantics is more similar

to the accelerated concrete semantic. This is the reason why we used the accelerated semantics in Definition 2.13. In fact, the definition of an abstract step would have been exactly the same if we replaced $\gamma_1 \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$ with $\gamma_1 \xrightarrow{\delta} \gamma_2$, using non-accelerated semantics instead. Indeed, if we have $\gamma_1 \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$ with $\delta = (q_1, \mathsf{act}, q_2)$ then we have $\gamma_1' \xrightarrow{\delta} \gamma_2'$ with $\overline{\gamma_1'} = \overline{\gamma_1}$ and $\overline{\gamma_2'} = \overline{\gamma_2}$. To build γ_1' and γ_2', if $\mathsf{st}(\gamma_2)(q_1) > 0$ then we let $\gamma_1' \coloneqq \gamma_1$ and if $\gamma_2(q_1) = 0$ then we let γ_1' equal to γ_1 except that $\mathsf{st}(\gamma_1')(q_1) = 1$, so that $\mathsf{st}(\gamma_2')(q_1) = 0$ and $\overline{\gamma_2'} = \overline{\gamma_2}$.

As in Section 2.2, we extend the abstract semantics $\xrightarrow{\delta}$ to define relations $\to$ and $\xrightarrow{*}$ on abstract configurations. We also define *abstract executions* similarly to concrete ones. The *length* of an abstract execution is given by its number of steps. We define the reachability set $\mathsf{Post}^*(A)$ and the notion of coverability as in the concrete case. This abstraction is sound and complete for PRP. Indeed, thanks to (Obs1) and (Obs2), the set $\mathsf{Post}^*(\Gamma_0)$ of reachable concrete configurations and the set $\mathsf{Post}^*(\sigma_0)$ of reachable abstract configurations are strongly related:

Proposition 2.14. $\overline{\mathsf{Post}^*(\Gamma_0)} = \mathsf{Post}^*(\sigma_0)$. *In other words, for all* $\sigma \in \overline{\Gamma}$*:*

$$(\exists \gamma \in \mathsf{Post}^*(\Gamma_0) : \overline{\gamma}{=}\sigma) \iff \sigma \in \mathsf{Post}^*(\sigma_0).$$

Proof. We start with the high-level idea of the proof. The converse implication is the interesting one; it relies on the monotonicity of $\mathsf{Post}^*(\Gamma_0)$ highlighted in Corollary 2.11, which allows us to obtain concrete configurations with enough processes. Indeed, if we have an abstract step $\sigma \xrightarrow{\delta} \sigma'$ and a concrete configuration γ such that $\overline{\gamma} = \sigma$, although there is indeed γ' such that $\gamma \xrightarrow[\mathsf{acc}]{\delta} \gamma'$, it could be that γ' has no process in the source state q of δ (if γ had only one process in q) while $q \in \mathsf{st}(\sigma')$ so that $\overline{\gamma'} \neq \sigma'$; we solve this issue by guaranteeing that $\mathsf{st}(\gamma)(q) \geqslant 2$.

First, by definition, if there is $\gamma_0 \in \Gamma_0$, $\gamma \in \Gamma$ such that $\overline{\gamma} = \sigma$ then, by definition of the abstract semantics, $\overline{\gamma_0} = \sigma_0 \xrightarrow{*} \overline{\gamma} = \sigma$.

We now prove the converse implication. Fix an abstract execution $\rho : \sigma_0 \xrightarrow{*} \sigma$; we prove the existence of $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\overline{\gamma} = \sigma$. The proof is by induction in the length of the abstract execution ρ. If ρ has 0 steps then $\sigma = \sigma_0$ and the statement is trivially true. Assume that the statement is true for abstract executions of length at most $\mathsf{len}(\rho) - 1$, and decompose ρ into $\sigma_0 \xrightarrow{*} \sigma_m \xrightarrow{\delta} \sigma$. By induction hypothesis, there is $\gamma_m \in \mathsf{Post}^*(\Gamma_0)$ such that $\overline{\gamma_m} = \sigma_m$. Let $(q_1, \mathsf{act}, q_2) \coloneqq \delta$. Because δ can be applied from σ_m, it can be applied from γ_m. If $q_1 \notin \mathsf{st}(\sigma)$, it suffices to perform δ from γ_m $\mathsf{st}(\gamma_m)(q_1)$ times, so that there is no process left on q_1 (in the accelerated semantics, this corresponds to an accelerated step that sends $\mathsf{st}(\gamma_m)(q_1)$ processes from q_1 to q_2). If $q_1 \in \mathsf{st}(\sigma)$, we could have an issue if $\mathsf{st}(\gamma_m)(q_1) = 1$, as applying δ from γ_m would empty q_1. We know that $\mathsf{st}(\gamma_m)(q_1) > 0$, thus by Corollary 2.11

$\langle \mathsf{st}(\gamma_m) \oplus q_1, \mathsf{data}(\gamma_m)\rangle \in \mathsf{Post}^*(\Gamma_0)$; applying δ once from this configuration brings us to a configuration γ such that $\overline{\gamma} = \sigma$. □

(Obs3) can be formalized as follows. Given $\gamma_1, \gamma_2 \in \Gamma$, if $\overline{\gamma_1} = \overline{\gamma_2}$ then γ_1 and γ_2 have the same truth value for presence constraints. We interpret presence constraints on abstract configurations in a straightforward manner. We can study directly presence reachability problems in the abstract world:

Proposition 2.15. *Let ϕ be a presence constraint. There exists $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\phi \models \gamma$ if and only if there exists $\sigma \in \mathsf{Post}^*(\sigma_0)$ such that $\phi \models \sigma$.*

This proves that presence reachability problems may be studied in the abstract semantics. We now bound the length of abstract executions needed to witness reachability between two configurations. To do that, we first need to define a notion of abstract execution in normal form.

Given an abstract step $\sigma_1 \xrightarrow{\delta} \sigma_2$ and a state q, the step *deserts* state q whenever $q \in \mathsf{st}(\sigma_1) \setminus \mathsf{st}(\sigma_2)$, and it *populates* state q whenever $q \in \mathsf{st}(\sigma_2) \setminus \mathsf{st}(\sigma_1)$.

Definition 2.16. An abstract execution ρ is in *normal form* if each of its steps satisfies one of the following three properties:

(2.16.1) it deserts a state, or

(2.16.2) it populates that was never populated before in the execution, or

(2.16.3) it is a write step on register r, its start and end configurations are distinct and the next step involving register r in ρ, if it exists, is a read step.

Lemma 2.17. *For every $\sigma_1, \sigma_2 \in \overline{\Gamma}$, if $\sigma_1 \xrightarrow{*} \sigma_2$ then there is an abstract execution from σ_1 to σ_2 that is in normal form.*

Proof. The high-level idea is that any step that violates all three conditions of Definition 2.16 is useless and can be removed. Let $\rho : \sigma_1 \xrightarrow{*} \sigma_2$. We build an execution ρ' starting with $\rho' = \rho$ and applying the following transformations:

1. for every state q, if q is deserted and later populated again, add q to all intermediate abstract configurations between the first step deserting q and the last step populating q;
2. remove from ρ' all steps whose start and end configurations are equal;
3. remove from ρ' all write steps that do not desert nor populate a state and whose written symbol is overwritten; a write step on register r in ρ' is *overwritten* if, in the suffix ρ'_s of ρ' that follows this step, there is a step that involves register r and the first step in ρ'_s involving register r is a write step;

4. remove again from ρ' all steps whose start and end configurations are equal.

It is important that we apply these four transformations in this order: 1 then 2 then 3 then 4. Indeed, transformation 1 may generate steps with same start and end configurations, so that those steps are deleted at transformation 2. In transformation 2, we delete useless read steps with same start and end configurations; such steps must not be taken into account during transformation 3 to justify the presence of write steps. In transformation 3, we may generate write steps with the same start and end configuration: it could be that, before this transformation, some write step $\mathsf{write}_r(\mathsf{d})$ was applied to some configuration with a symbol distinct from d in register r, but that, after removal of anterior write steps in transformation 3, the start configuration of this step now has symbol d.

We claim that, after these four transformations, the obtained execution ρ' is in normal form. We need to prove the conditions of Definition 2.16 for each step of ρ'. We therefore consider an arbitrary step in ρ'; to do so, we split ρ' into $\rho_p : \sigma_1 \xrightarrow{*} \sigma_2$, $\sigma_2 \xrightarrow{\delta} \sigma_3$ and $\rho_s : \sigma_3 \xrightarrow{*} \sigma_4$ and we are interested in the step $\sigma_2 \xrightarrow{\delta} \sigma_3$. If this step does not satisfy (2.16.1), then $\mathsf{st}(\sigma_2) \subseteq \mathsf{st}(\sigma_3)$. If it does not satisfy (2.16.2), then $\mathsf{st}(\sigma_3) \subseteq \mathsf{st}(\sigma_2)$. Indeed, if there is $q \in \mathsf{st}(\sigma_3) \setminus \mathsf{st}(\sigma_2)$ then, because q is not a witness for (2.16.2), q appears in ρ_s; but then, q would have been detected in transformation 1 so that it would be in $\mathsf{st}(\sigma_2)$, a contradiction. We have proved that if the step $\sigma_2 \xrightarrow{\delta} \sigma_3$ violates (2.16.1) and (2.16.2) then $\mathsf{st}(\sigma_2) = \mathsf{st}(\sigma_3)$. Because the step was not removed at transformations 2 and 4, we have $\sigma_2 \neq \sigma_3$ therefore δ is a write transition. Let r be the register that δ writes to. Because the step $\sigma_2 \xrightarrow{\delta} \sigma_3$ was not removed during transformation 3, if ρ_p has some step involving register r then the first such step in ρ_p is a read step. We have proved that a step in ρ' that violates (2.16.1) and (2.16.2) satisfies (2.16.3). This proves that the obtained execution ρ' is in normal form; it remains to prove that ρ' connects the two desired configurations. Let $\rho' : \sigma'_1 \xrightarrow{*} \sigma'_2$; we must argue that $\sigma'_1 = \sigma_1$ and $\sigma'_2 = \sigma_2$. Transformation 1 does not change the set of states of the first and last configurations because it only add states to intermediate configurations of the execution. Transformations 2, 3 and 4 may only remove steps between configurations of same set of states. This proves that $\mathsf{st}(\sigma'_1) = \mathsf{st}(\sigma_1)$ and $\mathsf{st}(\sigma'_2) = \mathsf{st}(\sigma_2)$. The content of the registers of the first configuration are never modified, therefore $\mathsf{data}(\sigma'_1) = \mathsf{data}(\sigma_1)$. Finally, for every register r, if there is a write transition to r in ρ, then the last write transition δ to r in ρ is also the last write transition to r in ρ', as this step will not be removed during transformation 3. If there is no step writing to r in ρ, then there is also none in ρ'. This proves that $\mathsf{data}(\sigma'_2) = \mathsf{data}(\sigma_2)$, which concludes the proof. □

Abstract executions in normal form have their length bounded by a polynomial, making them valid certificates for the abstract reachability relation.

Lemma 2.18. *Abstract executions in normal form are of length at most* $4|Q| + |\mathbb{D}|$.

Proof. At most $|Q|$ steps populate a state and at most $|Q|$ steps desert a state. All steps that do not desert nor populate a state are write steps. Write steps that do not desert not populate a state can be split into two categories: those whose register is not written again in the remainder of the execution (there are at most $|\mathbb{D}|$ of them) and those whose register is written again later in the execution. Write steps of the second category can be injectively mapped to the corresponding read step, which exists thanks to (2.16.3), and there are at most $2|Q|$ read steps hence at most $2|Q|$ write steps of this second category. This makes at most $4|Q| + |\mathbb{D}|$ steps in total. □

2.5 Complexity of the General Case

In this section, we establish that the presence reachability problem is an NP-complete problem, and that NP-hardness already holds for Cover.

Theorem 2.19. *The presence reachability problem is in NP.*

Proof. Consider an instance $(\mathcal{P}, \phi)$ of the presence reachability problem. Thanks to Proposition 2.15, the instance is positive if and only if there is $\sigma \in \mathsf{Post}^*(\sigma_0)$ such that $\phi \models \sigma$. Thanks to Lemma 2.17 and Lemma 2.18, $\sigma \in \mathsf{Post}^*(\sigma_0)$ can be witnessed by an abstract execution of length at most $4|Q| + |\mathbb{D}|$. An NP certificate therefore consists in providing an abstract execution in normal form. Such an execution can be stored in space $O((|Q| + |\mathbb{D}| + |\Delta|)^2)$. Moreover, given a certificate in the shape of an execution, one can easily check, in linear time in the number of steps, that the execution is valid. Lastly, checking whether a given abstract configuration satisfies ϕ can be done in polynomial time. □

This argument heavily relies on the unlimited supply of processes initially in q_0. In fact, when the set of initial configurations considered is allowed to express upper bounds on the number of processes present in some states, the presence reachability problem and even Cover are PSPACE-complete [BGW22; BW21; BGW23].

Since the presence reachability problem includes satisfiability of a SAT formula, it is automatically NP-hard. What is less obvious is that even Cover is NP-hard.

Proposition 2.20. *Cover is NP-hard.*

Proof. The proof is by a reduction from 3-SAT. Consider a 3-CNF formula $\phi = \bigwedge_{i=1}^{m} \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ over n variables $x_1, \ldots, x_n$ where, for all $i \in [\![1, m]\!]$, for all $k \in [\![1, 3]\!]$, $\ell_{i,k} \in \{x_j, \neg x_j \mid j \in$

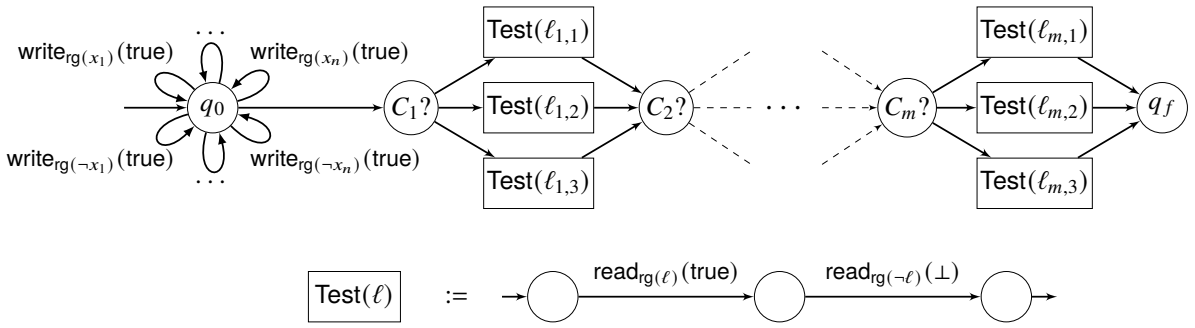


Figure 2.3 – The protocol $\mathcal{P}_\phi$ built in Proposition 2.20 to reduce 3-SAT to Cover. Transitions with no depicted label are labeled by the internal action $\circledast$.

$[\![1, n]\!]\}$. We define a roundless protocol $\mathcal{P}_\phi$ with a distinguished state q_f which is coverable if and only if ϕ is satisfiable. In $\mathcal{P}_\phi$, one has $\mathbb{D} = \{\bot, \mathsf{true}\}$, $\mathsf{dim} = 2n$ and there are two registers for each variable x_i, namely $\mathsf{rg}(x_i) \coloneqq 2i - 1$ and $\mathsf{rg}(\neg x_i) \coloneqq 2i$. The protocol is represented on Figure 2.3.

While any register may be set to true thanks to the loops on q_0, a register set to true can never be set back to the initial symbol $\bot$. The general idea is that literal ℓ being true corresponds to $\mathsf{rg}(\ell)$ being set to true while $\mathsf{rg}(\neg\ell)$ still has value $\bot$ (of course, if $\ell = \neg x_j$ then $\mathsf{rg}(\ell) = \mathsf{rg}(\neg x_j)$ and $\mathsf{rg}(\neg\ell) = \mathsf{rg}(x_j)$). This is indeed the condition that must be satisfied to go across the gadget $\mathsf{Test}(\ell)$ at the bottom of Fig. 2.5.

Suppose that the instance of 3-SAT is positive, *i.e.*, that ϕ is satisfied by some assignment ν. Consider an abstract execution that writes true exactly to all $\mathsf{rg}(\ell)$ with ℓ true in ν. For each clause C_i, there is $k_i \in \{1, 2, 3\}$ such that ℓ_{i,k_i} is true in ν. In this case, $\mathsf{rg}(\ell_{i,k_i})$ is written to true while $\mathsf{rg}(\neg\ell_{i,k_i})$ is left blank, so that processes can go through $\mathsf{Test}(\ell_{i,k_i})$. Therefore the execution may cover $C_i?$ for all i, so that it may cover q_f and the instance of Cover is positive. Conversely, if the instance of Cover is positive, there exists an execution $\rho : \sigma_0 \xrightarrow{*} \sigma$ with $q_f \in \mathsf{st}(\sigma)$. Consider ν that assigns to each variable x_j value true if, in ρ, $\mathsf{rg}(x_j)$ is written for the first time while $\mathsf{rg}(\neg x_j)$ is still blank; ν assigns to x_j value false otherwise. Given a literal ℓ, ρ may only send processes through $\mathsf{Test}(\ell)$ if $\nu(\ell)$ is true. Note that the value of ν is in fact only relevant for variables x_j such that ρ either crossing $\mathsf{Test}(x_j)$ or $\mathsf{Test}(\neg x_j)$; otherwise, the value of $\nu(x_j)$ is irrelevant. In particular, if ρ writes to neither $\mathsf{rg}(\ell)$ nor $\mathsf{rg}(\neg\ell)$ then the value $\nu(x_j)$ is arbitrarily set to false. Because ρ covers q_f, this proves that $\nu \models \phi$. □

Given the simplicity of the model, it is a bit disappointing that all interesting problems are NP-hard. In fact, in the first study of asynchronous shared-memory systems [EGM13; EGM16], the coverability problem was proved to be in PTIME in the case where there is no leader and

where all processes are finite-state machine, which is similar to our model. There are however an important differences between our model of the one of [EGM13; EGM16], which is that our registers have an initial value. This initial value breaks a key principle from [EGM13; EGM16], which can be phrased as: "in an execution witnessing Cover, whenever a symbol is read from a register, one may assume that this symbol can be written at will to this register". The underlying idea is to apply the copycat principle to the process writing the symbol. For a symbol that was initially in the register, this idea breaks down.

2.6 The Uninitialized Case

The reduction used in Proposition 2.20 directly relies on the initialization of registers. Initial values may force an execution to make global and irreversible choices, which is a common source of NP-hardness. We now consider the case where the registers are *uninitialized*. Formally, a protocol $\mathcal{P}$ is *uninitialized* when it has no transition reading $\perp$: for all $(q, \mathsf{act}, q') \in \Delta$, $\mathsf{act} \notin \{\mathsf{read}_r(\perp) \mid r \in [\![1, \dim]\!]\}$. In other words, in an uninitialized protocol, processes are not able to use the initial symbol in the registers, which is a way to model that this initial value does not exist. In [AAR20, Section 7], where the systems considered also work with shared memory, a similar uninitialized hypothesis yields lower complexity than the general case. The uninitialized restriction indeed lowers the complexity of Cover for roundless ASMS:

Proposition 2.21. *Cover for uninitialized roundless ASMS is PTIME-complete.*

Proof. We first prove membership in PTIME. In an abstract execution witnessing Cover, one may assume that there is no deserting step. In such an execution, whenever symbol d is written to register r using a transition $(q_1, \mathsf{write}_r(\mathsf{d}), q_2)$, then q_1 stays populated and it remains possible to write d to this register for the remainder of the execution. We abstract this observation by working with the assumption that a symbol d can be read from a register r if and only if there is a transition $(q_1, \mathsf{write}_r(\mathsf{d}), q_2)$ such that q_1 is populated (note that this does not hold for initialized ASMS where the initial symbol $\perp$ may be read although it cannot be written). This observation allows us to apply a standard saturation technique (see, *e.g.*, [DSTZ12]) to obtain a polynomial-time algorithm. This saturation technique consists in a fixpoint computation of $S \subseteq Q$. Initially, we set $S \leftarrow \{q_0\}$. Then, we iteratively add to S all states q such that one of the three following conditions is satisfied:

(i) there is $q' \in S$ such that $(q', \mathsf{write}_r(\mathsf{d}), q) \in \Delta$ for some $\mathsf{d} \in \mathbb{D}$ and $r \in [\![1, \dim]\!]$; or

(ii) there is $q' \in S$ such that $(q', \mathsf{read}_r(\mathsf{d}), q) \in \Delta$ for some $\mathsf{d} \in \mathbb{D}$ and $r \in [\![1, \mathsf{dim}]\!]$ for which there is $q_1 \in S$, $q_2 \in Q$ such that $(q_1, \mathsf{write}_r(\mathsf{d}), q_2) \in \Delta$ [3] ; or

(iii) there is $q' \in S$ such that $(q', \circledast, q) \in \Delta$.

We claim that, once a fixpoint is reached for S, S contains the set of coverable states so that the instance is positive if and only if $q_f \in S$. First, from a computation of the above algorithm, one can easily build an execution that covers all states added to S. The proof is by induction on the number of states added to S so far. Let q be the next state added, and let S_i be the content of S right before q is added. By induction hypothesis, we obtain an execution $\rho_i : \sigma_0 \xrightarrow{*} \sigma_i$ such that $S_i \subseteq \mathsf{st}(\sigma_i)$. We make a case disjunction on how q is added to S. In the case where q is added with (i), it suffices to apply the write transition detected by the algorithm from σ_i. In the case where q is added with (ii), we first apply the write transition then the read transition. If q is added with (iii), we simply apply the transition. Conversely, for every abstract execution ρ from σ_0, all states visited in ρ are eventually added to S in the computation of the algorithm. This is proved by induction on the length of the execution. If the last step of ρ is a write step or an internal step, then this step is captured by (i) and (iii) respectively. The interesting case is when the last step of the execution is a read transition $\mathsf{read}_r(\mathsf{d})$: in this case, we argue that a transition $(q_1, \mathsf{write}_r(\mathsf{d}), q_2)$ is applied before in ρ and, by induction hypothesis, q_1 is added to S so that this transition can be detected as a witness for (ii) in the algorithm. This is where we need that the ASMS is uninitialized, as if $\mathsf{d} = \perp$ then there is no such write transition.

We now prove that Cover is PTIME-hard. For the sake of reusability, we show that this holds even for uninitialized protocols with a single register ($\mathsf{dim} = 1$). The proof is similar to the one presented in [DSTZ12, Proposition 1] for broadcast protocols. It uses a LOGSPACE reduction from the Circuit Value Problem, which is PTIME-complete for LOGSPACE reductions [Lad75]. The latter problem consists in determining the output value of an acyclic Boolean circuit with given input values and Boolean gates that are negations $\neg$, disjunctions $\vee$ and conjunctions $\wedge$.

Consider an instance of the Circuit Value Problem, we write V for the set of input, intermediate and output values of the circuit. A gate is represented as a tuple of the form $(\neg, i, o)$, $(\vee, i_1, i_2, o)$ or $(\wedge, i_1, i_2, o)$ where we denote by $i, i_1, i_2 \in V$ input(s) and by $o \in V$ the output of the gate. We construct an instance $(\mathcal{P}_{\mathrm{CVP}}, q_f)$ of Cover with $\mathsf{dim} = 1$ and where $\mathcal{P}_{\mathrm{CVP}}$ is uninitialized. In $\mathbb{D}$, we have $\perp$ (which is never read) along with, for every $v \in V$, symbols $\mathsf{true}(v)$ and $\mathsf{false}(v)$, denoting that v is respectively true and false. First, $\mathcal{P}_{\mathrm{CVP}}$ has a part from which one may write the symbols corresponding to the assignment of the input values of the

3. One could alternatively choose to enforce that $q_2 \in S$; this choice is irrelevant, as case (i) guarantees that q_2 will eventually be added to S.

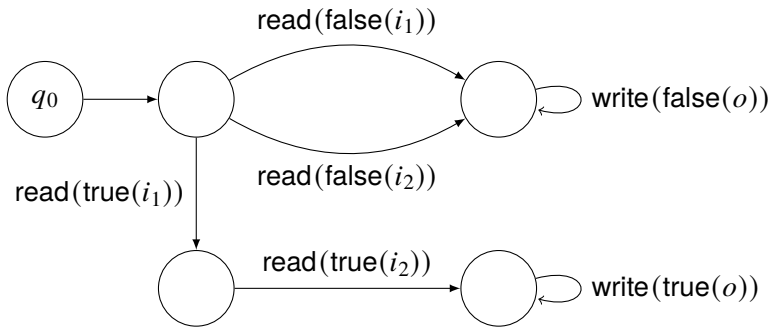


Figure 2.4 – The gadget in $\mathcal{P}_{\text{CVP}}$ corresponding to a gate $(\wedge, i_1, i_2, o)$

circuit. Moreover, for every gate of the circuit, there is a part of the protocol corresponding to this gate, in which a process may read the values of the inputs and write the corresponding value of the output. A depiction for gate $(\wedge, i_1, i_2, o)$ may be found in Fig. 2.4. Lastly, state q_f is the destination of the transition writing symbol $\mathsf{true}(\mathsf{out})$ where $\mathsf{out} \in V$ is the output variable. This reduction can be computed in logarithmic space because we can build the protocol by going through all gates one by one and building all corresponding gadgets. □

The uninitialized hypothesis therefore makes Cover a tractable problem; this is however not the case for our other problems of interest and in particular for Target.

Proposition 2.22. *Target for uninitialized protocols is NP-hard.*

Proof. Once again, we proceed by reduction from 3-SAT. Consider a 3-CNF formula $\phi = \bigwedge_{i=1}^{m} \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ over n variables $x_1, \ldots, x_n$ where, for all $i \in [\![1, m]\!]$, for all $k \in [\![1, 3]\!]$, $\ell_{i,k} \in \{x_j, \neg x_j \mid j \in [\![1, n]\!]\}$. We define an instance $(\mathcal{P}_\phi, q_f)$ of Target with $\mathcal{P}_\phi$ uninitialized, so that $(\mathcal{P}_\phi, q_f)$ is positive if and only if ϕ is satisfiable. We start by explaining how this reduction compares with the one used to prove Proposition 2.20. Here, we only need one register per variable, and not two. The encoding of the truth value is straightforward: x_j being true corresponds to true being written to register j, and x_j being false corresponds to false being written to register j. The definitive choice of the valuation is made by the last process to leave the initial state q_0, because q_0 is the only state from which processes can write values to the registers. Indeed, in every witness execution for Target, there is a last process to leave the initial state. This is a crucial difference with Cover, where it is always possible to make processes stay and write the same values over and over again.

Let $\mathsf{dim} := n$, *i.e.*, the protocol has a register for each variable of the formula. The symbols of $\mathbb{D}$ are true and false (along with $\perp$ which cannot be read nor written). A depiction of the

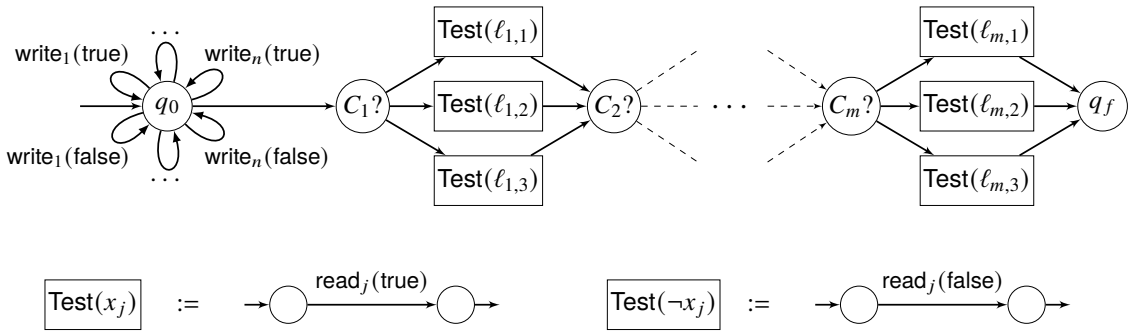


Figure 2.5 – The protocol $\mathcal{P}_\phi$ for NP-hardness of uninitialized TARGET.

protocol can be found in Figure 2.5.

Suppose first that ϕ is satisfiable by an assignment ν. For all $i \in [\![1, m]\!]$, there exists $k(i) \in [\![1, 3]\!]$ such that $\nu(\ell_{i,k(i)}) = \mathsf{true}$. Consider the following execution ρ with one process. First, the process writes true to all literals ℓ such that ℓ is true in ν. The process then leaves q_0; it goes through each state C_i? and through each gadget $\mathsf{Test}(\ell_{i,k(i)})$, which is possible because $\ell_{i,k(i)}$ is true in ν. This execution goes from $\langle q_0, \bot^{[\![1,\mathsf{dim}]\!]}\rangle$ to $\langle q_f, \vec{d}\rangle$ for some $\vec{d} \in \mathbb{D}^{\mathsf{dim}}$ hence the instance of TARGET is positive. Conversely, suppose that there exists such an execution $\rho : \sigma_0 \xrightarrow{*} \langle q_f, \vec{d}\rangle$. All processes end up in q_f, therefore there is a step in ρ that deserts q_0 for the last time, *i.e.*, where the last process to leave q_0 does so. Let γ be the configuration immediately after this step. Let ν be the valuation that assigns true to x_j if and only if $\mathsf{data}(\gamma)(j) = \mathsf{true}$; this means that ν assigns false to x_j if and only if $\mathsf{data}(\gamma)(j) \in \{\mathsf{false}, \bot\}$. We claim that $\nu \models \phi$. There is at least one process in ρ, therefore for every $i \in [\![1, m]\!]$, there is $k(i) \in \{1, 2, 3\}$ such that at least one process in ρ goes through $\mathsf{Test}(\ell_{i,k(i)})$. From this, we deduce that, for all $i \in [\![1, m]\!]$, ν assigns true to $\ell_{i,k(i)}$, which proves that $\nu \models \phi$. □

2.7 One-register Asynchronous Shared-Memory Systems

In the previous section, we have seen that, while the NP-hardness of COVER stems from the initialization of registers, this is not true for more complex problems such as TARGET. Another natural question is whether this hardness comes from the number of registers. Motivated by this consideration, we study presence reachability problems in ASMS with one register only.

In this section, we prove that DNFPRP (therefore in particular TARGET) can be solved in polynomial time when the protocol has only one register.

Theorem 2.23. *DNFPRP for roundless ASMS with* $\mathsf{dim} = 1$ *is PTIME-complete.*

The rest of this section is devoted to proving Theorem 2.23. Thanks to Proposition 2.15, we work with the abstract semantics in this proof. The technique used for the complexity upper bound is inspired by the one of [Fou15] in the setting of broadcast networks. We describe here this general idea for TARGET, the proof for DNFPRP relies on the same idea but with additional technical details.

Let $(\mathcal{P}, q_f)$ be an instance of TARGET. We are looking for a witness (abstract) execution for TARGET, *i.e.*, an execution $\sigma_0 \xrightarrow{*} \langle \{q_f\}, \vec{d} \rangle$ for some $\vec{d}$. We first prove that such a witness execution can be rearranged so that it starts by an increasing phase where it populates all needed states, followed by a decreasing phase where all states except q_f are deserted. With this observation, the algorithm works with two separate computations: one that computes the set of states that can be populated in the first phase, and one that computes the set of states that can be deserted in the second phase. Any state that is not in the intersection of these two sets has no hope of appearing in a witness execution, so that such states can be deleted from the protocol. After this removal, we apply the same procedure on the new, smaller protocol. Once a fixpoint set of states is reached, it suffices to check whether this fixpoint is empty or not.

In the ASMS setting, the proof is however more complex than the one of [Fou15] because of the initialization of the register, a concept that has no equivalent in broadcast networks, and because of persistency of symbols in the register. We start by getting rid of the first obstacle by reducing the general problem to the one for uninitialized ASMS.

Lemma 2.24. *There exists a polynomial-time reduction from initialized DNFPRP with* $\mathsf{dim} = 1$ *to uninitialized DNFPRP with* $\mathsf{dim} = 1$.

Proof. Let $\mathcal{P} = \langle Q, q_0, 1, \mathbb{D}, \bot, \Delta \rangle$ be a roundless protocol with a single register. Any execution of $\mathcal{P}$ is composed of two (potentially empty) phases: the phase where the register has symbol $\bot$ and no write transition is applied and the phase where the register has a symbol in $\mathbb{D} \setminus \{\bot\}$ and $\bot$ may no longer be read. The reduction relies on the observation that, in the first phase, only read transitions labeled by $\mathsf{read}(\bot)$ and internal transitions may be taken, and processes do not interact during this phase. Therefore, we can consider as initial any state that may be covered from q_0 with a path composed only of transitions labeled by $\mathsf{read}(\bot)$ and $\circledast$.

Consider the graph $G = (Q, E)$ whose vertices are the states of the system and whose edges are the transitions labeled by $\mathsf{read}(\bot)$ or $\circledast$: $(q_1, q_2) \in E$ if and only if $(q_1, \mathsf{read}(\bot), q_2) \in \Delta$ or $(q_1, \circledast, q_2) \in \Delta$. Let Q_0 be the set of vertices with a path from q_0 in G. This set can trivially be computed in polynomial time. We let $q_0' \notin Q$ and we build an uninitialized protocol $\mathcal{P}' = \langle Q \uplus \{q_0'\}, q_0', 1, \mathbb{D}, \bot, \Delta' \rangle$ where Δ' is obtained by removing from Δ all transitions labeled by $\mathsf{read}(\bot)$ and, for each $q \in Q_0$, adding an internal transition from q_0' to q. We now prove that

$(\mathcal{P}, \phi)$ is a positive instance of DNFPRP if and only if $(\mathcal{P}', \phi)$ is. Note that ϕ can be interpreted over $\mathcal{P}'$ because $Q \subseteq Q'$. For the same reason, for every abstract configurations $\langle S, \mathsf{d}\rangle$ of $\mathcal{P}$, $\langle S, \mathsf{d}\rangle$ and $\langle S \cup \{q'_0\}, \mathsf{d}\rangle$ are abstract configurations of $\mathcal{P}'$.

First, assume that $(\mathcal{P}, \phi)$ is positive and let $\rho : \sigma_0 \xrightarrow{*} \sigma$ be an abstract execution of $\mathcal{P}$ such that $\sigma \models \phi$. Let $\langle S, \bot\rangle$ be the last configuration visited in ρ with a blank register (we can have $\langle S, \bot\rangle = \sigma$ if ρ never writes to the register). We first build $\rho_m : \sigma_0 \xrightarrow{*} \langle S \cup \{q'_0\}, \bot\rangle$ in $\mathcal{P}'$ that uses the internal transitions to populate every state in S. If $\sigma \neq \langle S, \bot\rangle$, the next transition performed in ρ is a write transition, so that the remainder of the execution can be mimicked in $\mathcal{P}'$ to obtain $\rho' : \sigma_0 \xrightarrow{*} \sigma'$ where $\sigma' = \langle \mathsf{st}(\sigma) \cup \{q'_0\}, \mathsf{data}(\sigma)\rangle$. We have $\sigma' \models \phi$, which proves that $(\mathcal{P}', \phi)$ is positive.

Conversely, assume that $(\mathcal{P}', \phi)$ is positive and let $\rho' : \sigma_0 \xrightarrow{*} \sigma'$ be an abstract execution of $\mathcal{P}'$ such that $\sigma' \models \phi$. In ρ', each process remains idle in q'_0 until it takes an internal transition to some state in Q_0. Because this involves no read or write transition, we may rearrange ρ' so that all such transitions take place in some initial phase. Let $\langle S', \bot\rangle$ be the configuration obtained after this initial phase. Let $S \coloneqq S' \setminus \{q'_0\}$, so that $S \subseteq Q_0$. From $\langle S', \bot\rangle$, ρ' has a (possibly empty) second phase $\langle S', \bot\rangle \xrightarrow{*} \sigma'$ that uses no transition with source state q'_0. All transitions of the second phase can be mimicked in $\mathcal{P}$ to obtain an execution $\langle S, \bot\rangle \xrightarrow{*} \sigma$ where $\sigma \coloneqq \langle \mathsf{st}(\sigma') \setminus \{q'_0\}, \mathsf{data}(\sigma')\rangle$; we have $\sigma \models \phi$ therefore it suffices to build $\rho : \sigma_0 \xrightarrow{*} \langle S, \bot\rangle$ in $\mathcal{P}$. This execution ρ uses only transitions labeled by $\mathsf{read}(\bot)$ and $\circledast$; by definition of Q_0 and because $S \subseteq Q_0$, all states in S can be populated using only sequences of $\mathsf{read}(\bot)$ and $\circledast$ transitions from q_0. We populate each state in S one by one, while making sure to always desert states that are not in S. This way, we construct an execution from σ_0 to $\langle S, \bot\rangle$ in $\mathcal{P}$. We have proved that $(\mathcal{P}, \phi)$ is positive if and only if $(\mathcal{P}', \phi)$ is positive. Because $\mathcal{P}'$ can be constructed from $\mathcal{P}$ in polynomial time and $\mathcal{P}'$ has no transition labeled $\mathsf{read}(\bot)$, this concludes the proof of Lemma 2.24. □

Thanks to the previous lemma, it is sufficient to prove Theorem 2.23 in the uninitialized case. Consider an instance $(\mathcal{P}, \phi)$ of DNFPRP where $\mathcal{P}$ is uninitialized and only has one register. Because ϕ is assumed to be in disjunctive normal form, $(\mathcal{P}, \phi)$ is positive if and only if $(\mathcal{P}, C)$ is positive for some clause C of ϕ. The algorithm hence iterates over all clauses in ϕ. A clause in ϕ is a conjunction of literals, hence we see clauses as sets of atomic propositions and negations of atomic propositions that the final configuration has to satisfy. We first rule out the case where there is a witness execution composed of internal steps only:

Lemma 2.25. *One can decide in polynomial-time, giving an instance $(\mathcal{P}, \phi)$ of DNFPRP, whether there is an execution $\rho : \sigma_0 \xrightarrow{*} \sigma$ with $\sigma \models \phi$ where ρ has internal steps only.*

Proof. The intuition is that, in execution with internal steps only, there is no communication between processes so that processes independently follow paths of internal transitions. With the same technique as in the proof of Lemma 2.24, we can compute the set S of states that can be covered from σ_0 using executions with internal steps only. It then suffices, for each clause C in ϕ, to check whether there is $T \subseteq S$ such that $T \neq \emptyset$ and $\langle T, \bot \rangle \models C$. This can be done in polynomial time by performing the following computation. First, check that, for all $\mathsf{d} \in \mathbb{D}$, if $\mathsf{cont}(1, \mathsf{d}) \in C$ then $\mathsf{d} = \bot$ and check that $(\neg\mathsf{cont}(1, \bot)) \notin C$. Then, check that, for all $q \in Q$, if $\mathsf{popu}(q) \in C$ then $q \in S$ and $(\neg\mathsf{popu}(q)) \notin C$. Finally, check that, if $(\neg\mathsf{popu}(q_0)) \in C$ then there is $q \in S$ such that $(\neg\mathsf{popu}(q)) \notin C$. □

From now on, we assume that there is no witness execution of $(\mathcal{P}, \phi)$ using only internal steps. Let C be a clause in ϕ; C is a conjunction of literals, hence we see C as a set of atomic propositions and negations of atomic propositions that the final configuration has to satisfy. Let $Q_+(C)$ be the set of states that need to be populated in the final configuration, $Q_-(C)$ the states that need to not be populated, and $\mathbb{D}_{\mathsf{ok}}(C)$ the symbols that are allowed in the final configuration. Formally,

$$Q_+(C) := \{q \mid \mathsf{popu}(q) \in C\}$$
$$Q_-(C) := \{q \mid (\neg\mathsf{popu}(q)) \in C\}$$
$$\mathbb{D}_{\mathsf{ok}}(C) := \{\mathsf{d} \in \mathbb{D} \mid (\neg\mathsf{cont}(1, \mathsf{d})) \notin C \text{ and } \forall \mathsf{d}' \neq \mathsf{d}, \mathsf{cont}(1, \mathsf{d}') \notin C\}.$$

In words, a symbol d is in $\mathbb{D}_{\mathsf{ok}}(C)$ if, in C, it is not forbidden to be in the register (formally, $(\neg\mathsf{cont}(1, \mathsf{d})) \notin C$) and if no other symbol has to be in the register (formally, for all $\mathsf{d}' \neq \mathsf{d}$, $\mathsf{cont}(1, \mathsf{d}') \notin C$). In particular, if there is d such that $\mathsf{cont}(1, \mathsf{d}) \in C$ then $\mathbb{D}_{\mathsf{ok}}(C) = \{\mathsf{d}\}$ or $\mathbb{D}_{\mathsf{ok}}(C) = \emptyset$. This last case may happen if two different symbols must in the register, so that clause C cannot be satisfied.

For all $\langle S, \mathsf{d} \rangle \in \overline{\Gamma}$, $\langle S, \mathsf{d} \rangle \models C$ if and only if $Q_+(C) \subseteq S \subseteq Q \setminus Q_-(C)$ and $\mathsf{d} \in \mathbb{D}_{\mathsf{ok}}(C)$. Let

$$\mathcal{F}(C) := \{S \subseteq Q \mid Q_+(C) \subseteq S \subseteq Q \setminus Q_-(C)\}$$

be the collection containing sets of states that could form a valid final configuration. We now prove that, if there is a witness execution, then we can rearrange it into an increasing phase followed by a decreasing phase. We call an execution ρ *write-first* if its first step interacting with the register exists and is a write step. An execution ρ is write-first if it has some prefix $\gamma_1 \xrightarrow{*} \gamma_2 \xrightarrow{\delta} \gamma_3$ where the execution from γ_1 to γ_2 only has internal steps and δ is a write

transition. Because $\mathcal{P}$ is uninitialized, any execution starting from a configuration with symbol $\perp$ that has at some write transition is write-first.

Lemma 2.26. *Let $\rho : \sigma_0 \xrightarrow{*} \sigma_f = \langle S_f, \mathsf{d}_f \rangle$ be an execution with at least one write transition. There are write-first executions ρ_{inc}, ρ_{dec} such that $\rho_{\text{inc}} : \sigma_0 \xrightarrow{*} \langle S, \mathsf{d}_f \rangle$, $\rho_{\text{dec}} : \langle S, \mathsf{d}_f \rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f \rangle$ where S contains all states appearing in ρ, ρ_{inc} has no deserting step and ρ_{dec} does not populate any state.*

Proof. The execution ρ_{inc} is obtained by mimicking ρ (*i.e.*, applying the same sequence of transitions as in ρ) but turning into non-deserting all deserting steps, so that all states visited in ρ are populated in the configuration obtained. The execution ρ_{dec} is obtained by mimicking ρ but from a larger set of states, which is possible thanks to Fact 2.3. For every state $q \in S \setminus S_f$, the last step in ρ where q appears is a deserting step with a transition leaving q. By making these steps deserting in ρ_{dec} and all other steps non-deserting, we obtain an execution from $\langle S, \mathsf{d}_f \rangle$ to $\langle S_f, \mathsf{d}_f \rangle$. Because the protocol is uninitialized, we know that ρ is write-first, so that ρ_{dec} and ρ_{inc} also are. Also, ρ_{dec} populates no state because the first configuration of ρ_{dec} contains all states seen in ρ_{dec}, only states in $S \setminus S_f$ are deserted in ρ_{dec} and these states are deserted only once. □

Given a protocol $\mathcal{P}$ and a clause C, we define two sets of sets of states as follows:

— $\mathcal{R}(\mathcal{P}) := \{S \subseteq Q \mid \exists \sigma_0 \in \Gamma_0, \exists \mathsf{d} \in \mathbb{D},\ \sigma_0 \xrightarrow{*} \langle S, \mathsf{d} \rangle\}$,

— $\mathcal{BR}(\mathcal{P}, C) = \{S \subseteq Q \mid \exists \mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C), \exists S_f \in \mathcal{F}(C), \exists \rho : \langle S, \mathsf{d} \rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f \rangle \text{ write-first}\}$.

We put $\mathcal{P}$ as argument because Algorithm 1 will modify the protocol by removing states. To avoid overloading notations, we consider that the sets Q and $\mathbb{D}$ and the reachability relation are implicitly those of the protocol $\mathcal{P}$ taken as argument. In the definition of $\mathcal{BR}(\mathcal{P}, C)$, the symbol d is irrelevant because the execution is write-first, so that it in fact guarantee that $\langle S, \mathsf{d}' \rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f \rangle$ for all $\mathsf{d}' \neq \mathsf{d}$.

Lemma 2.27. *The sets $\mathcal{R}(\mathcal{P})$ and $\mathcal{BR}(\mathcal{P}, C)$ are stable by union.*

Proof. For both sets, the proof essentially consists in concatenating executions, which will be possible because all executions considered are write-first. We first prove it for $\mathcal{R}(\mathcal{P})$. Let $\sigma_0, \sigma_0' \in \Gamma_0$, $\rho : \sigma_0 \xrightarrow{*} \langle S, \mathsf{d} \rangle$, $\rho' : \sigma_0' \xrightarrow{*} \langle S', \mathsf{d}' \rangle$. We show that we can merge ρ and ρ' into a single execution $\langle \mathsf{st}(\sigma_0) \cup \mathsf{st}(\sigma_0'), \perp \rangle \xrightarrow{*} \langle S \cup S', \mathsf{d}' \rangle$. To do so, we first mimic ρ, but we make non-deserting any deserting step in ρ whose transition leaves a state in $\mathsf{st}(\sigma_0')$. This way, we obtain an execution $\langle \mathsf{st}(\sigma_0) \cup \mathsf{st}(\sigma_0'), \perp \rangle \xrightarrow{*} \langle S \cup \mathsf{st}(\sigma_0'), \mathsf{d} \rangle$. We then mimic ρ, but we make

non-deserting any deserting step in ρ' whose transition leaves a state in S. ρ' is write-first and can be mimicked regardless of the symbol in the register. This way, we obtain an execution $\langle S \cup \mathsf{st}(\sigma'_0), \mathsf{d}\rangle \xrightarrow{*} \langle S \cup S', \mathsf{d}'\rangle$. This proves that $S \cup S' \in \mathcal{R}(\mathcal{P})$.

Suppose now that we have $S, S' \in \mathcal{BR}(\mathcal{P}, C)$. By hypothesis on S, there exist $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$, $\mathsf{d} \in \mathbb{D}$, $S_f \in \mathcal{F}(C)$, $\rho : \langle S, \mathsf{d}\rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle$ where ρ is write-first, and therefore $\langle S \cup S', \mathsf{d}\rangle \xrightarrow{*} \langle S_f \cup S', \mathsf{d}_f\rangle$. By hypothesis on S', there exist $\mathsf{d}'_f \in \mathbb{D}_{\mathsf{ok}}(C)$, $S_f \in \mathcal{F}(C)$ and $\rho' : \langle S', \mathsf{d}_f\rangle \xrightarrow{*} \langle S'_f, \mathsf{d}'_f\rangle$ where ρ' is write-first. Therefore we also have $\langle S_f \cup S', \mathsf{d}_f\rangle \xrightarrow{*} \langle S_f \cup S'_f, \mathsf{d}'_f\rangle$, which combined with the previous execution provides a write-first execution $\langle S \cup S', \mathsf{d}\rangle \xrightarrow{*} \langle S_f \cup S'_f, \mathsf{d}'_f\rangle$, proving that $S \cup S' \in \mathcal{BR}(\mathcal{P}, C)$. □

Because $\mathcal{R}(\mathcal{P})$ and $\mathcal{BR}(\mathcal{P}, C)$ are stable by union and contain subsets of the finite set Q, we can define $\mathrm{R}(\mathcal{P}) \coloneqq \max \mathcal{R}(\mathcal{P})$ and $\mathrm{BR}(\mathcal{P}, C) \coloneqq \max \mathcal{BR}(\mathcal{P})$, where the maximum is for inclusion of sets, so that $\mathrm{R}(\mathcal{P}), \mathrm{BR}(\mathcal{P}, C) \subseteq Q$. By convention, we set $\max(\emptyset) = \emptyset$; this cannot happen for $\mathcal{R}(\mathcal{P})$ because $\{q_0\} \in \mathcal{R}(\mathcal{P})$ but this can happen for $\mathcal{BR}(\mathcal{P}, C)$, for example if $Q_-(\mathcal{P}, C) = Q$.

Algorithm 1 provides functions computing $\mathrm{R}(\mathcal{P}, C)$ and $\mathrm{BR}(C, \mathcal{P})$ along with the procedure solving DNFPRP when the protocol is uninitialized and $\mathsf{dim} = 1$.

The function `Compute_R`$(\mathcal{P})$ used the same technique as in Proposition 2.21, but with one register only. The proof that `Compute_R`$(\mathcal{P})$ returns $\mathrm{R}(\mathcal{P})$ follows from the proof of Proposition 2.21. We now claim that `Compute_BR`$(\mathcal{P}, C)$ returns $\mathrm{BR}(\mathcal{P}, C)$.

First, we prove by induction in the number of write transitions in ρ the following statement: if there is $\rho : \langle S_s, \mathsf{d}_s\rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle$ write-first such that $S_f \in \mathcal{F}(C)$ and $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$, then all states in S_s are added to S in the computation of `Compute_BR`$(\mathcal{P}, C)$. First, let ρ be such an execution with only one write step. The symbol written in ρ is d_f and by hypothesis $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$. We claim that all states of S_s are added to S during the call to `PreviousSymbol`$(Q \setminus Q_-(C), \mathbb{D}_{\mathsf{ok}}(C))$. Indeed, $S_f \subseteq Q \setminus Q_-(C)$ and $q_f \in \mathbb{D}_{\mathsf{ok}}(C)$: from each state in S_s, there is a path to a state in S_f using only transitions reading and writing d_f. This proves that all states in S_s are added to T during the iteration of the loop of line 23 where $\mathsf{d} = \mathsf{d}_f$; this iterative of the `for` loop passes the test at line 27 therefore all states in S_s are added to T in `PreviousSymbol`$(Q \setminus Q_-(C), \mathbb{D}_{\mathsf{ok}}(C))$, so that they are all in the result of `Compute_BR`$(\mathcal{P}, C)$. Let now $\rho : \langle S_s, \mathsf{d}_s\rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle$ starting with a write transition and such that $S_f \in \mathcal{F}(C)$ and $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$, and suppose that ρ has several write steps. We split ρ on its second write transition; let $\langle S_m, \mathsf{d}_m\rangle$ be the configuration in ρ right before this second write transition. By induction hypothesis, all states in S_m are in the result of `Compute_BR`$(\mathcal{P}, C)$. There is a call to `PreviousSymbol`$(S, \mathbb{D} \setminus \{\perp\})$ that is performed at line 19 with $S \supseteq S_m$. In this call, with the same reasoning as above, all states in $S_s \setminus S_m$ are

Function `DNFPRP_Oneregister_Uninit`($\mathcal{P}$)**:**
  for C *clause in* ϕ **do**
    $\mathcal{P}_C \leftarrow \mathcal{P}$; `// copy of` $\mathcal{P}$ `that will be modified`
    Until $Q(\mathcal{P}_C)$ **reaches a fixpoint do**
      $Q(\mathcal{P}_C) \leftarrow Q(\mathcal{P}_C) \cap \mathrm{R}(\mathcal{P}_C) \cap \mathrm{BR}(\mathcal{P}_C, C)$;
      `/* modifies` $\mathcal{P}_C$`, only keeps in` $\mathcal{P}_C$ `transitions with source state and destination state in` $Q(\mathcal{P}_C)$ `*/`
    if $Q_+(C) \cup \{q_0\} \subseteq Q(\mathcal{P}_C)$ **then** Accept;
  Reject ;
Function `Compute_R`($\mathcal{P}$)**:**
  $S \leftarrow q_0$;
  Until S **reaches a fixpoint do**
    $S \leftarrow S \cup \{q' \mid \exists q \in S, \exists \mathsf{d} \in \mathbb{D},\ (q, \mathsf{write}(\mathsf{d}), q') \in \Delta\}$;
    $S \leftarrow S \cup \{q' \mid \exists q, q_1, q_2 \in S, \exists \mathsf{d},\ (q, \mathsf{read}(\mathsf{d}), q') \in \Delta, (q_1, \mathsf{write}(\mathsf{d}), q_2) \in \Delta\}$;
  return S ;
Function `Compute_BR`($\mathcal{P}, C$)**:**
  `/* Compute_BR(`$\mathcal{P}, C$`) builds an execution backwards, from configurations` $\langle S, \mathsf{d}_f \rangle$ `with` $S \in \mathcal{F}(C)$ `and` $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$`. */`
  if `PreviousSymbol(`$Q \setminus Q_-(C), \mathbb{D}_{\mathsf{ok}}(C)$`)` $\neq$ "`Not found`" **then**
   $S \leftarrow$`PreviousSymbol(`$Q \setminus Q_-(C), \mathbb{D}_{\mathsf{ok}}(C)$`)`;
  else return $\emptyset$;
  Until S **reaches a fixpoint do**
    if `PreviousSymbol(`$S, \mathbb{D} \setminus \{\perp\}$`)`$\neq$"`Not found`" **then**
      $S \leftarrow$`PreviousSymbol(`$S, \mathbb{D} \setminus \{\perp\}$`)` ; `// This is not the last symbol, it does not have to be in` $\mathbb{D}_{\mathsf{ok}}(C)$`.`
  return S;
Function `PreviousSymbol(`S, Symbols`)`**:**
  `/* PreviousSymbol(`S, Symbols`) looks in set` Symbols `for the previous symbol written: for each` d`, it adds all states from which one can reach` S `with` read(d) `transitions, then checks that` d `can be written; if not, it backtracks and tries another symbol. If no symbol works, returns "Not found". */`
  Found $\leftarrow$ False ;
  for d $\in$ Symbols **do**
    $T \leftarrow S$;
    Until T **reaches a fixpoint do**
      $T \leftarrow T \cup \{q \in Q \mid \exists q' \in T, (q, \mathsf{read}(\mathsf{d}), q') \in \Delta\}$;
    if *there exist* $q \in Q$, $q' \in T$ *s.t.* $(q, \mathsf{write}(\mathsf{d}), q') \in \Delta$ **then**
      $S \leftarrow T \cup \{q\}$;
      Found $\leftarrow$ True ;
  if Found **then return** S **else return** "`Not found`";

Algorithm 1: A polynomial-time algorithm for DNFPRP for uninitialized ASMS with a single register.

detected during the iteration of the `for` loop at line 23 where $\mathsf{d} = \mathsf{d}_m$. Also, the test at line 27 passes because it detects the first transition performed in ρ. Overall, this proves that all states in S_i are in the result of $\texttt{Compute\_BR}(\mathcal{P}, C)$.

Conversely, we prove that all states in the result of $\texttt{Compute\_BR}(\mathcal{P}, C)$ are in $\mathrm{BR}(\mathcal{P}, C)$. To do so, it suffices to prove that, for all $S \subseteq \mathrm{BR}(\mathcal{P}, C)$, for all $D \subseteq \mathbb{D}$, the set computed by $\texttt{PreviousSymbol}(S, D)$ contains only states q for which there is $\mathsf{d}_f \in D$, $S_s \supseteq S$ such that $q \in S_s$ and such that there is a write-first execution $\langle S_s, \mathsf{d}_s\rangle \xrightarrow{*} \langle S, \mathsf{d}_f\rangle$. This would indeed guarantee, by direct induction, that the set S in $\texttt{Compute\_BR}(\mathcal{P}, C)$ only ever contains states in $\mathrm{BR}(\mathcal{P}, C)$. Let $S \subseteq \mathrm{BR}(\mathcal{P}, C)$ and $D \subseteq \mathbb{D}$. It suffices to prove the assertion for one iteration of the loop at line 23; let d be the symbol of the considered iteration and let $S' \supseteq S$ be the set of states obtained after the iteration. We suppose that the test at line 27 passed, as otherwise the iteration adds no state. It suffices to observe that, with the transition detected at line 27 and those detected at line 26, we can build a write-first execution $\langle S', \mathsf{d}_s\rangle \xrightarrow{*} \langle S, \mathsf{d}\rangle$ that starts with the write transition followed by the read transitions (in reversed order compared to the order of detection at 26). This proves the desired assertion about $\texttt{PreviousSymbol}(S, D)$. By iteratively applying this argument, first once with $D = \mathbb{D}_{\mathsf{ok}}(C)$ then with $D = \mathbb{D} \setminus \{\bot\}$, this proves that all states added to S in $\texttt{Compute\_BR}(\mathcal{P}, C)$ are in $\mathrm{BR}(\mathcal{P}, C)$.

We now prove that `DNFPRP_Oneregister_Uninit` of Algorithm 1 solves DNFPRP for uninitialized protocols with one register. First, suppose that the algorithm accepts during the iteration corresponding to clause C. It ends with a protocol $\mathcal{P}_C$ such that $Q_+(C) \subseteq Q(\mathcal{P}_C) = \mathrm{R}(\mathcal{P}_C) \cap \mathrm{BR}(\mathcal{P}_C, C)$. In this protocol, by definition of $\mathrm{R}(\mathcal{P}_C)$ there exist $\sigma_0 \in \Gamma_0$ and $\mathsf{d} \in \mathbb{D}$ such that $\sigma_0 \xrightarrow{*} \langle \mathrm{R}(\mathcal{P}_C), \mathsf{d}\rangle$; since $\mathrm{R}(\mathcal{P}_C) = \mathrm{BR}(\mathcal{P}_C, C)$ we also have $\langle \mathrm{R}(\mathcal{P}_C), \mathsf{d}\rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle \models C$ and the instance is positive. Conversely, suppose that the instance is positive. There exists a clause C in ϕ and a witness execution $\rho : \sigma_0 \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle$ with $\mathsf{d}_f \in \mathbb{D}_{\mathsf{ok}}(C)$ and $S_f \in \mathcal{F}(C)$. Thanks to Lemma 2.25 and because the protocol is uninitialized, we may assume that ρ is write-first. Let S be the set of states appearing in ρ; let $\rho_{\mathrm{inc}} : \sigma_0 \xrightarrow{*} \langle S, \mathsf{d}_f\rangle$ and $\rho_{\mathrm{dec}} : \langle S, \mathsf{d}_f\rangle \xrightarrow{*} \langle S_f, \mathsf{d}_f\rangle$ obtained with Lemma 2.26. We have $S \subseteq \mathrm{R}(\mathcal{P}_C)$ thanks to ρ_{inc} and $S \subseteq \mathrm{BR}(\mathcal{P}_C, C)$ thanks to ρ_{dec}, so that no states of S are removed during the first iteration of the fixpoint computation at line 4 of `DNFPRP_Oneregister_Uninit`. By direct induction, ρ_{inc} and ρ_{dec} remain witnesses that $S \subseteq \mathrm{R}(\mathcal{P}) \cap \mathrm{BR}(\mathcal{P}, C)$ at every iteration, so that it remains true that $S \subseteq Q(\mathcal{P}_C)$. Moreover, $Q_+(C) \cup \{q_0\} \subseteq S$ therefore the algorithm accepts. This proves that DNFPRP is in PTIME.

The PTIME-hardness proof can be obtained using the reduction in the proof of Proposition 2.21. This concludes the proof of Theorem 2.23.

Remark 2.28. The result above proves that many problems can be solved more efficiently on

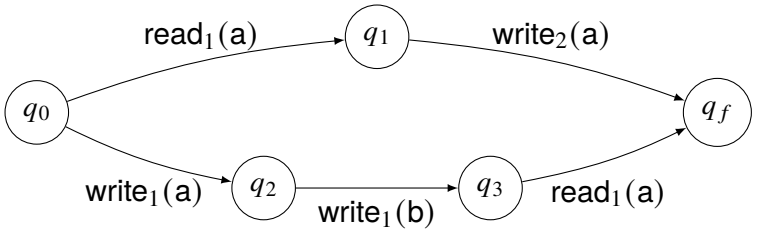


Figure 2.6 – An example of negative instance of TARGET that becomes positive when merging all registers into one with the procedure described in Remark 2.28. In this uninitialized protocol, $\mathsf{dim} = 2$ and $\mathbb{D} = \{\perp, \mathsf{a}, \mathsf{b}\}$.

systems with only one register. A natural idea is to generalize the result above by simply encoding several registers into one. The hope is to find algorithms that pay a high cost in the number of registers but a reasonable cost in the number of states and transitions. We explain here why this is a wrong track. The natural way to encode several registers into one works as follows. Let $\mathcal{P}$ be a protocol with dim registers and alphabet $\mathbb{D}$, we build a protocol $\mathcal{P}'$ with one register and whose alphabet is $\mathbb{D}^{\mathsf{dim}}$, *i.e.*, the merged register of $\mathcal{P}'$ contains a dim-tuple whose components correspond to register values in $\mathcal{P}$. A write transition $(q_1, \mathsf{write}_i(\mathsf{d}), q_2)$ in $\mathcal{P}$ is translated in $\mathcal{P}'$ by adding $|\mathbb{D}|^{\mathsf{dim}}$ intermediate states between q_1 and q_2 so that one can read and remember the value of the register, and then write the new dim-tuple with component i set to d. A read transition $(q_1, \mathsf{read}_i(\mathsf{d}), q_2)$ of $\mathcal{P}$ is transformed to $|\mathbb{D}|^{\mathsf{dim}-1}$ parallel transitions from q_1 to q_2, *i.e.*, read transitions that read all dim-tuple in which the i-th component has symbol d. With this construction, we would pay an exponential blowup in dim but not in the rest of the system. However, such a construction is actually not correct. While every execution of $\mathcal{P}$ can be simulated in $\mathcal{P}'$, some executions of $\mathcal{P}'$ do not correspond to executions of $\mathcal{P}$. Intuitively, a merged transition in $\mathcal{P}'$ allows a process to write symbols to registers that it should not be able to write to. We illustrate this on an example in Fig. 2.6. The depicted $(\mathcal{P}, q_f)$ is a negative instance of TARGET: there must be at least one process going in q_2, but the last process to leave q_2 can never go from q_3 to q_f. Nonetheless, if we let $\mathcal{P}'$ be the protocol where the two registers are merged into one using the procedure described above, then $(\mathcal{P}', q_f)$ is a positive instance of TARGET. Indeed, a process going from q_1 to q_f may, for every $\mathsf{d} \in \mathbb{D}$, read tuple (a, d) for some d, go to an intermediate state then write (a, a). Such a process may stop in the intermediate state while all processes in q_2 go in q_3, and then this process can write (a, a) which allows processes in q_3 to go in q_f.

2.8 Dependency in the Number of Registers

In the previous section, we established that TARGET and DNFPRP are decidable in polynomial time when the systems considered only have one register, although both problems are NP-complete with an arbitrary number of registers. This highlights that the number of registers is a source of difficulty. In this section, we study more finely the dependency of the complexities of our problems with respect to the number of registers. We start by introducing a few so-called parameterized complexity classes. Note that the term *parameterized* here has a different meaning as in the rest of this thesis. Here, we study *parameterized complexity*, which asks how the complexity of our problems depends on some relevant metrics different from the size of the input, this metric being here the number of registers in our system. We elsewhere refer to *parameterized verification* to express the fact that we do not fix the number of processes so that we consider an infinite family of systems.

We introduce here a few notions of parameterized complexity that will be useful for our purposes; we refer to, *e.g.*, [Cyg+15] for a complete overview. A *parameterized problem* is a decision problem that takes as input a tuple (x, k) with x a binary input and $k \in \mathbb{N}$. The component k is called *parameter* of the problem. The complexity class FPT is the class of parameterized problems that can be decided in deterministic time $O(f(k)|x|^{O(1)})$ where f is a computable function. In words, this time complexity is a polynomial in $|x|$ multiplied by a function in k. To define other classes, we rely on the concept of FPT reductions. An FPT reduction from a parameterized problem P_1 to a parameterized problem P_2 transforms an instance (x, k) of P_1 into an equivalent instance (x', k') of P_2 such that there is a computable function g for which $k' \leqslant g(k)$ for some computable function g and such that the transformation can be computed in time $O(f(k)p(|x|))$ where f is a computable function and p is a polynomial. In particular, the class FPT is stable under FPT reductions. Another class is W[1], which we define here as problems that are FPT-interreducible with the k-clique problem. The *k-clique problem* takes as input (G, k) with G an undirected graph and asks whether G contains a clique of size at least k, *i.e.*, a set of k vertices that are pairwise connected in G. It is known that $\mathsf{FPT} \subseteq W[1]$, and this inclusion is commonly believed to be strict. Yet another class is W[2], which we define here as problems that are FPT-interreducible with the k-dominating set problem. The *k-dominating set problem* takes as input (G, k) with G an undirected graph and asks whether G contains a dominating set of size at most k, *i.e.*, a set of k vertices such that every vertex of G is at distance at most 1 of the set. It is known that $W[1] \subseteq W[2]$, and this inclusion is commonly believed to be strict.

We start with the Cover problem. Recall that this problem is in PTIME for ASMS with one register and for uninitialized ASMS but NP-complete in general.

Proposition 2.29. *The Cover problem is FPT with respect to the number of registers.*

Proof. Let $(\mathcal{P}, q_f)$ be an instance of Cover. The high-level idea of the algorithm is to enumerate all *first-write orders*, *i.e.*, all possible orders in which registers lose the initial value. For a given such first-write order, using a simple saturation technique, one can build an execution that covers all states that can be covered by executions with the same first-write order. It then suffices to prove that this maximal set of states coverable with this first-write order can be computed efficiently.

First-write orders are non-repeating sequences of elements of $[\![1, \mathsf{dim}]\!]$, there are at most $\sum_{k \leqslant \mathsf{dim}} k! \leqslant (\mathsf{dim}+1)!$ such sequences. For $f_1, \ldots, f_k$ such a sequence, let $\mathsf{Execs}(f_1, \ldots, f_k)$ be the set of abstract executions in which the registers that are written for the first time are $f_1, \ldots, f_k$, in this order. Formally, $\rho : \sigma_0 \xrightarrow{*} \sigma$ is in $\mathsf{Execs}(f_1, \ldots, f_k)$ if all its write transitions are to registers in $\{f_1, \ldots, f_k\}$, if ρ has a write transition to register i for all $i \in \{f_1, \ldots, f_k\}$ and if, for all $i < j$, there is a configuration $\langle S, \vec{d} \rangle$ in ρ such that $\vec{d}(f_i) \neq \bot$ but $\vec{d}(f_j) = \bot$.

We need to provide a polynomial-time algorithm that decides the following problem: given $(\mathcal{P}, q_f)$ and a non-repeating sequence $f_1 \ldots f_k$ of $[\![1, \mathsf{dim}]\!]$, is there an execution of $\mathsf{Execs}(f_1, \ldots, f_k)$ covering q_f? We prove the following property by induction on k. For every non-repeating sequence $f_1, \ldots, f_k$ of elements of $[\![1, \mathsf{dim}]\!]$, there is a set S such that, for all $\langle S', \vec{d} \rangle$ reachable from σ_0 with an execution in $\mathsf{Execs}(f_1, \ldots, f_k)$, $S' \subseteq S$; moreover, S can be computed in polynomial time.

For $k = 0$, the sequence is empty and the only allowed executions are those that only use transitions reading $\bot$, hence the result. Suppose that the result is true for k, and let $f_1, \ldots, f_{k+1}$ be a non-repeating sequence of $[\![1, \mathsf{dim}]\!]$. Let S_k be the set obtained by induction hypothesis on $f_1, \ldots, f_k$; we know that S_k can be computed in polynomial time. We set $S \leftarrow S_k$, and we iteratively add to S all states q that satisfy one of the following conditions:

— there is $(q', \mathsf{write}_i(\mathsf{d}), q) \in \Delta$ with $q' \in S$ and $i \in \{f_1, \ldots, f_{k+1}\}$; or

— there is $(q', \mathsf{read}_i(\bot), q) \in \Delta$ with $q' \in S$ and $i \notin \{f_1, \ldots, f_{k+1}\}$; or

— there is $(q', \mathsf{read}_i(\mathsf{d}), q), (q_1, \mathsf{write}_i(\mathsf{d}), q_2) \in \Delta$ with $q', q_1 \in S$ and $i \in \{f_1, \ldots, f_{k+1}\}$,

— there is $(q', \circledast, q) \in \Delta$ with $q' \in S$.

This computation reaches a fixpoint in polynomial time. We then check that at least one write transition on register f_{k+1} has source state in S; if no, then we return $\emptyset$ because

$\mathsf{Execs}(f_1, \ldots, f_{k+1})$ is empty. Otherwise, we return the set S, which satisfies the desired property. Indeed, by direct induction, all states added to S can be covered by executions of $\mathsf{Execs}(f_1, \ldots, f_{k+1})$. We prove that, for all $\rho : \sigma_0 \xrightarrow{*} \langle T, \mathsf{d}\rangle$ such that $\rho \in \mathsf{Execs}(f_1, \ldots, f_{k+1})$, we have $T \subseteq S$ with S the set computed above. Let $\rho : \sigma_0 \xrightarrow{*} \langle T, \vec{d}_f\rangle$ such that $\rho \in \mathsf{Execs}(f_1, \ldots, f_{k+1})$; we split ρ into $\rho_p : \sigma_0 \xrightarrow{*} \langle S_m, \vec{d}_m\rangle$ and $\rho_s : \langle S_m, \vec{d}_m\rangle \xrightarrow{*} \langle T, \vec{d}_f\rangle$ where $\vec{d}_m(f_{k+1}) = \bot$ and ρ_s starts with a write transition on f_{k+1}. By induction hypothesis on ρ_p, all states in S_m are in S at the beginning of the computation. We then proceed inductively along ρ_s; for a given transition, it suffices to make a case disjunction on the type of transition, with the same four cases highlighted in the fixpoint computation above. This directly proves that the algorithm detects all transitions along ρ_s, so that $T \subseteq S$ with S obtained at the end of the fixpoint computation. We have proven that we can decide, given $(\mathcal{P}, q_f)$ and a non-repeating sequence $f_1 \ldots f_k$ of $[\![1, \mathsf{dim}]\!]$, if there is an execution of $\mathsf{Execs}(f_1, \ldots, f_k)$ covering q_f. Iterating over all non-repeating sequences gives an algorithm for Cover that works in time $O((\mathsf{dim}+1)!\,\mathsf{poly}(|\mathcal{P}|))$ with poly a polynomial. □

We now consider the more complex problems of Target and dnfPRP. Recall that both problems are in PTIME for protocols with one register, but NP-complete for protocols with several registers, even for uninitialized protocols. We now provide a hardness result that highlights that these problems are not easily tractable with respect to the number of registers.

Proposition 2.30. *Target and dnfPRP are $W[2]$-hard with respect to the number of registers.*

Proof. Because Target is FPT-reducible to dnfPRP, we provide an FPT reduction from the k-dominating set problem to Target. Let (G, k) be an instance of the k-dominating set problem; for simplicity, we assume that every vertex is its own neighbor. We build a protocol $\mathcal{P}$ with k registers and a special state q_f such that $(\mathcal{P}, q_f)$ is a positive instance of Target if and only if (G, k) is a positive instance of the k-dominating set problem. Let $\{v_1, \ldots, v_n\}$ be the set of vertices of G; for every $i \in [\![1, n]\!]$, let d_i be the number of neighbors of v_i in G, and let $\{w_i^1, \ldots, w_i^{d_i}\}$ be its set of neighbors. The built protocol $\mathcal{P}$ is depicted in Fig. 2.7. $\mathcal{P}$ is uninitialized and has k registers, one for each of the vertices of the dominating set. It works with set of symbols $\mathbb{D} = \{\bot\} \cup \{v_1, \ldots, v_n\}$. This construction is polynomial in k and n, so that this reduction is a polynomial-time reduction hence an FPT reduction. If there is a dominating set $\{v_{i_1}, \ldots, v_{i_\ell}\}$ with $\ell \leqslant k$, then it is easy to obtain a witness execution that starts by writing v_{i_j} to register j for all $j \leqslant \ell$, then sends all processes to q_f: from every v_i?, it takes the transition reading a register in which the vertex written is a neighbor of v_i. Conversely, if there is an execution ρ witnessing that the instance of Target is positive, then let $(u_1, \ldots, u_k)$ be the

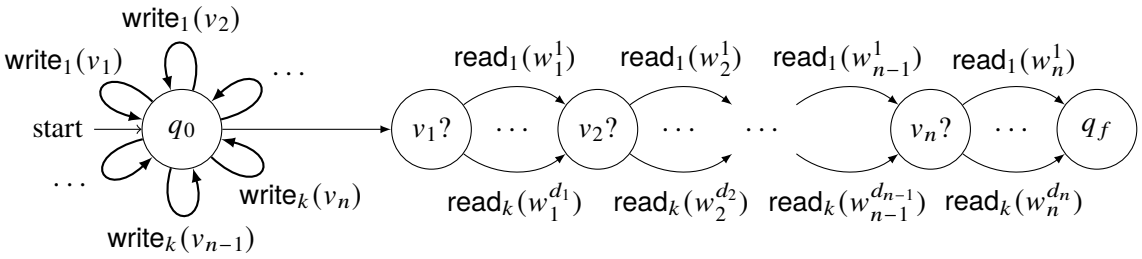


Figure 2.7 – The protocol built in the proof of Proposition 2.30. From q_0, one can write anything to any register, so that there is a loop $(q_0, \mathsf{write}_i(v_j), q_0)$ for all $i \in [\![1, k]\!]$ and $j \in [\![1, n]\!]$. From $v_i?$, the leaving edges correspond to an exhaustive check for a neighbor of v_i in the registers. Therefore, for all $i \in [\![1, n]\!]$, there is an edge $(v_i?, \mathsf{read}_j(w_i^\ell), v_{i+1}?)$ for every $r \in [\![1, \mathsf{dim}]\!]$ and for every $\ell \in [\![1, d_i]\!]$ (with convention $v_{n+1}? = q_f$).

content of the registers when q_0 is deserted in ρ. Because all states $v_1?$ to $v_n?$ are eventually deserted in ρ, the set $\{u_1, \ldots, u_k\}$ is a k-dominating set. □

This hardness result puts an end to our hope to design efficient algorithms for protocols with small numbers of registers. It is also instructive with respect to the discussion of Remark 2.28. In Remark 2.28, we tried to build a transformation of protocols with dim registers into protocols with one register. The hardness result from Proposition 2.30 proves that (unless $W[2] = \mathsf{FPT}$) there exists no such transformation computable in $O(f(\mathsf{dim})p(|\mathcal{P}|))$ with f computable and p a polynomial. Indeed, by contradiction, combining such a transformation with Theorem 2.23 would prove that Target is FPT with respect to dim. However, it remains an open question whether Target and dnfPRP are slice-wise polynomial, *i.e.*, whether they can be solved in polynomial time for any fixed value of dim. The class of slice-wise polynomial parameterized problems is the parameterized complexity class XP, which subsumes FPT and W[2] (see [Cyg+15] for a formal definition of XP).

2.9 Perspectives

We recall the complexity results obtained in this chapter in Fig. 2.8.

All of the problems studied in this chapter lie in NP, a complexity class that, in formal verification, can sometimes be considered reasonable. Several restrictions of our problems are even solvable in polynomial time. It is interesting to compare the results from Fig. 2.8 to the ones known in the related but simpler model of RBN (discussed in the introduction of this chapter). Indeed, presence reachability problems have their counterparts in RBN, which have studied in

	COVER	TARGET	DNFPRP	PRP
General case	NP-complete (2.19 & 2.20)	NP-complete (2.19 & 2.20)	NP-complete (2.19 & 2.20)	NP-complete (2.19)
Uninitialized	PTIME-complete (2.21)	NP-complete (2.19 & 2.22)	NP-complete (2.19 & 2.22)	NP-complete (2.19)
One register	PTIME-complete (2.23)	PTIME-complete (2.23)	PTIME-complete (2.23)	NP-complete (2.19)

Figure 2.8 – Summary of complexity results for roundless ASMS. All problems in the rightmost column are automatically NP-hard because the presence constraint is a general SAT formula.

[DSTZ12] under the name of "cardinal reachability problem restricted to CC[$\geqslant 1, = 0$]". All such problems are also in NP in RBN [DSTZ12], and the general PRP problem is NP-complete because of the general SAT formula. However, COVER and TARGET in RBN are solvable in PTIME without any restriction [Fou15], a result that could be extended to DNFPRP. The study of the parameterized complexity with respect to the number of registers is, to the best of our knowledge, new; it has no equivalent in related models such as RBN. It remains, however, quite superficial. A possible future work would be to complete this study; in particular, it is not known whether TARGET lies in XP, *i.e.*, whether TARGET can be solved in polynomial time for a fixed number of registers. Another possibly interesting question would be to perform a fine-grained complexity analysis and to extend the work from [CMS19] to ASMS with multiple registers and to the problem of TARGET.

The main issue with the roundless ASMS model and presence reachability problems is their lack of expressive power. We will explore in the next chapters some extensions of this model and some more general problems. We discuss here one important choice that we have made in Section 2.2, which is to forbid atomic read-write combinations, *i.e.*, transitions labeled by a read and a write action that the process performs instantly. Atomic read-write combinations allow a process to perform a read and a write action while no other process may act between the two actions. This choice in particular breaks the copycat property: a process may ensure that it is the only one taking a transition. Therefore, a given process has the ability to pick a role while ensuring that no other process is playing the same role. The low complexities we obtained in the non-atomic case are due to monotonicity property, so that it seems likely that atomic read-write combinations make the complexity jump. In fact, it was first observed in [GS92] that, when allowing read-write combination, roundless ASMS gain an expressive power similar to the one of Petri nets and population protocols. In particular, COVER becomes an EXPSPACE-complete problem. This high complexity, along with the fact that we would essentially end up with

the known model of Petri nets, discourages a formal study of ASMS with atomic read-write combinations. This is why, in this thesis, we only consider non-atomic read-write combinations.

In the next chapter, we will embed roundless ASMS into a more general family of systems, named *copycat systems*. This will allow us to provide general-purpose bounds that we leverage to prove decidability of problems beyond presence reachability problems. In Chapter 4, we will extend the ASMS model to a round-based version meant to capture round-based distributed algorithms.

Chapter 3

Copycat Systems

3.1 Introduction

Copycat Systems In this chapter, we introduce *copycat systems*, a new model meant to capture parameterized distributed systems enjoying a monotonicity property called *copycat property*. In parameterized verification of distributed systems, it is common to consider models composed of arbitrarily many identical processes described by a common finite-state machine, so that the overall configuration of the system is described by a multiset of states along with some global state. This global state can be, *e.g.*, the values of shared variables or the state of a leader. Sometimes, such systems enjoy a monotonicity property called *copycat property*. Whenever a step is performed where some process goes from q_1 to q_2, any other process in q_1 may go to q_2 without impacting the rest of the system [1]. While the copycat property hints that the computational power of the model is limited, it also gives the system convenient monotonicity properties. These properties often allow for decidability of verification and for somewhat low complexity when compared with more powerful models such as Petri nets. The aim of this chapter is to design a generic model for systems enjoying the copycat property and to provide a general-purpose bound that allows to prove decidability of most problems on this model. This model is meant to implement the *accelerated* semantics of traditional models such as ASMS: in one step of the copycat system, arbitrarily many processes are allowed to perform the same transition. The main result of this chapter is called *structural theorem*. This theorem provides us with two different bounds for copycat systems: one on the number of steps needed to reach a configuration from another, and one on the number of processes needed in an execution, *i.e.*, of processes that cannot be removed without affecting the rest of the execution. The adjective *structural* expresses that the bound is related to *any* execution of the system, so that it structurally bounds the system. To the best of our knowledge, the literature, for example in ASMS, has no

1. Sometimes, less powerful properties are also referred to as "copycat properties", for example the ability to combine several copies of an execution. The property describes here corresponds to the strongest notion that bears the name "copycat property"

equivalent bound with the same level of generality. In our view, this theorem is an important result by itself. To illustrate the power of this theorem, we apply it to some decision problems such as emptiness of so-called generalized reachability expressions (inspired by [Wei23]) and LTL verification. In ASMS, this theorem allows to easily reprove the results from [BMRSS16] with almost no additional work. The structural theorem is the core result of this chapter; the subsequent results are here to demonstrate the power of the structural theorem and are not extraordinarily new on their own.

This chapter is organized as follows. In Section 3.2, we define copycat systems and associated notions. In Section 3.3, we introduce and prove the structural bound. In Section 3.4, we elaborate on what this bound implies for copycat systems. In Section 3.5, we connect copycat systems with other models and in particular ASMS. We conclude the chapter with some perspectives in Section 3.6. Some proofs from this chapter are technical and less important; to ease the read of the chapter, these proofs are relegated to Section 3.7. The work presented in this chapter is partly based on a joint work with Pierre Ganty, Cesar Sanchez and Chana Weil-Kennedy [GSWW24] that remains to be published. This joint work was performed during a stay at IMDEA Madrid from January to March 2024. Many thanks to Rennes Métropole for funding this stay.

Related Works We start with a presentation of models that satisfy the conditions above and are captured by copycat systems. The first such model is the one of asynchronous shared-memory systems (ASMS) from Chapter 2. Numerous works on ASMS and similar models rely on copycat arguments, be it explicitly [EGM13; EGM16; DEGM15] or implicitly [BMRSS16; CMS19]. See Section 2.1 for a detailed overview of the literature on ASMS.

Another related model is the one of *reconfigurable broadcast networks* (RBN). Reconfigurable broadcast networks are similar to ASMS but, instead of using a shared memory, processes communicate via broadcasting and receiving messages. A message consists of a symbol from a finite alphabet; when a message is broadcast, it can be received by any subset of the other processes. Reception of messages is instantaneous. This model was first defined in [DSZ10; DSTZ12], and has been widely studied since [BFS14; Fou15; BBM18; BW21; BGW22; BGW23]. In broadcast networks, the copycat property requires the *reconfigurable* hypothesis [DSZ10]: that every process can receive a message sent, but that no process has to do so. This hypothesis is intended to model unreliable communication where messages may get lost. Reconfigurable Broadcast Networks are in fact equivalent to ASMS for problems that allow to enforce that some processes play the role of the shared memory [BW21]. In [BW21; BGW22], the authors define a generic class of expression, named *nice expressions*, and consider

the problem of deciding whether the set defined by such an expression is empty or not. This problem is meant to encompass known problems from the literature of ASMS and RBN; in particular, they claimed to have solved a complexity gap left open in [BMRSS16], related to almost-sure coverability. However, the proof turned out to be incorrect [BGW23]. The associated question remains an open problem. In this chapter, we formulate a more general version of this open problem, but unfortunately we do not solve it.

A third model that lies under the scope of copycat systems is the one of Immediate Observation (IO) population protocols (also known as immediate observation Petri nets). IO population protocols are a restriction of population protocols; they were first defined in [AAER07], where their expressive power was established, and the complexity of standard parameterized verification problems was identified in [ERW19]. The immediate observation hypothesis enforces that, when two processes interact, only one of them is allowed to change state: it *observes* the other process, while this other process remains passive. This hypothesis allows for the copycat principle, and makes IO population protocols easier to verify (and less expressive) than usual population protocols. Overall, IO population protocols are simpler than RBN and ASMS.

This chapter constitutes an attempt to provide a general framework that encompasses parameterized models of distributed systems enjoying the copycat property. We present two unification attempts from the literature that are arguably related to ours. The first one corresponds to a general study of models of population protocols where the communication is unreliable [Ras21; Ras23], among which IO population protocols; [Ras21; Ras23] is however specific to population protocols and only interested in questions related to expressive power, and not in decidability and complexity of verification problems. Another unification attempt can be found in [EJW24]; the latter work, however, only considers simple problems such as the coverability problem, and its main contribution is to give a unified version of proofs relying on a non-counting abstraction similar to the one we used in Chapter 2.

3.2 Copycat Systems

In this section, we define copycat system and transfer flows and we prove some basic properties. We will often manipulate functions with several arguments and express the growth rate of such a function with respect to a given parameter. Given a function $f(x_1, \ldots, x_n)$, we call f polynomial (resp. exponential, doubly-exponential) in x_i if, for fixed values of x_j for each $j \neq i$, the function $x_i \mapsto f(x_1, \ldots, x_n)$ is polynomial (resp. exponential, doubly-exponential). For example, $f : (n, m) \mapsto m^n$ is exponential in n but polynomial in m, so that the value would

overall remain exponential if we pay the cost of an exponential blowup in the value of m.

3.2.1 Definition and Semantics

Copycat systems are meant to capture distributed systems where the copycat principle holds, *i.e.*, whenever a process goes from a state q_1 to a state q_2, another process in q_1 may do the same with no difference in the rest of the execution. In particular, we consider systems where a configuration is composed of a multiset of states (representing how many processes are in each state), where the set of states Q is finite, along with an element from a finite set ℓ (representing a global state of the system).

It will be convenient to extend the natural numbers with a special element, written #. Let $\mathbb{N}_\# \coloneqq \mathbb{N} \cup \{\#\}$; we extend the order on $\mathbb{N}$ to a partial order on $\mathbb{N}_\#$ by making # incomparable to all integers: for all $a, b \in \mathbb{N}_\#$, $a \leqslant b$ when either $a, b \in \mathbb{N}$ and $a \leqslant b$ or $a = b = \#$. We extend addition by $\# + n = n$ for all $n \in \mathbb{N}$ and $\# + \# = \#$. Note that $\mathbb{N}_\#$ differs from $\mathbb{N}_\omega$ in that # is not comparable with the integers, whereas $\omega \geqslant n$ for all $n \in \mathbb{N}$. A subtlety to keep in mind is that, for $x \in \mathbb{N}_\#$ and $m \in \mathbb{N}$, we do not always have $x \leqslant x + m$, because if $x = \#$ then $x + m = m$ and # and m are incomparable.

Definition 3.1. A *copycat system* is a tuple $C = (Q, \mathcal{L}, \mathcal{T}_{\min})$ where:

— Q is a finite set of *states*,

— $\mathcal{L}$ is a finite set of *control locations*,

— $\mathcal{T}_{\min} \subseteq (Q^2 \to \mathbb{N}_\#) \times \mathcal{L}^2$ is a finite set of transfer flows.

We additionally require that, for all $(f, \ell, \ell') \in \mathcal{T}_{\min}$, for all $q \in Q$, $f(q, q) \neq \#$. We call this property *idle-compliance*.

We start by explaining the sets Q and $\mathcal{L}$; the role of the set $\mathcal{T}_{\min}$ will be explained later in this subsection. The set Q represents the states of the processes composing the system, while the set $\mathcal{L}$ represents a global variable. Therefore, a *configuration* γ of the copycat system is an element of the set $\Gamma \coloneqq \mathcal{M}(Q) \times \mathcal{L}$. The *size* of a configuration $\gamma = \langle \mu, \ell \rangle$ is $|\gamma| \coloneqq |\mu|$. From now on, we fix a copycat system $(Q, \mathcal{L}, \mathcal{T}_{\min})$.

We now explain the set $\mathcal{T}_{\min}$. To do so, we start by defining the concept of *transfer flows*.

Definition 3.2. A *transfer flow* is a triplet $\mathsf{tf} = (f, \ell, \ell')$ where $f : Q^2 \to \mathbb{N}_\#$ and $\ell, \ell' \in \mathcal{L}$. We denote by $\mathcal{F}$ the set of all transfer flows.

With this definition, $\mathcal{T}_{\min}$ is a subset of $\mathcal{F}$. Transfer flows are meant to represent the possibility offered by transitions of the system. The function f represents a transfer of processes. ℓ and ℓ' represent the starting and ending control locations. We call ℓ the *source control location* and ℓ' the *destination control location*. Having $f(q_1, q_2) = \#$ represents that one is unable to send processes from q_1 to q_2, while having $f(q_1, q_2) = n$ means that one has to send at least n processes from q_1 to q_2, but that one may send any number in $[\![n, +\infty[\![$.

Given a transfer flow $\mathsf{tf} = (f, \ell, \ell') \in \mathcal{F}$, we define the *weight* of tf as $\mathsf{weight}(\mathsf{tf}) \coloneqq \sum_{q,q'} f(q, q')$; by convention, if $f(q, q') = \#$ for all q, q', we set $\mathsf{weight}(\mathsf{tf}) \coloneqq 0$.

We define a partial order $\preceq$ on $\mathcal{F}$ as follows. For $\mathsf{tf}_1 = (f_1, \ell_1, \ell_2)$ and $\mathsf{tf}_2 = (f_2, \ell_2, \ell_2')$, we let $\mathsf{tf}_1 \preceq \mathsf{tf}_2$ when $\ell_1 = \ell_1'$, $\ell_2 = \ell_2'$ and $f_1 \leqslant f_2$ (*i.e.*, for all q, q', $f_1(q, q') \leqslant f_2(q, q')$). In particular, this requires that $f_1(q, q') = \#$ if and only if $f_2(q, q') = \#$, because $\#$ is incomparable with all integers. Intuitively, $\mathsf{tf}_1 \preceq \mathsf{tf}_2$ means that tf_1 gives more possibilities than tf_2. We therefore highlight the following rule of thumb, which provides the reader with an intuition that shall be useful for the rest of this chapter:

Rule of thumb. *Smaller transfer flows are more powerful.*

The role played by transfer flows is made more concrete with the following definition:

Definition 3.3. Given $\gamma_1 = \langle \mu_1, \ell_1 \rangle, \gamma_2 = \langle \mu_2, \ell_2 \rangle \in \Gamma$ and $\mathsf{tf} = (f, \ell, \ell') \in \mathcal{F}$, we let $\gamma_1 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_2$ when the following conditions are satisfied:

- $\ell_1 = \ell$,
- $\ell_2 = \ell'$, and
- there exists $h : Q^2 \to \mathbb{N}_{\#}$ such that $f \leqslant h$ and:
  - for all $q \in Q$, $\mu_1(q) = \sum_{q'} h(q, q')$, and
  - for all $q' \in Q$, $\mu_2(q') = \sum_{q} h(q, q')$.

The function h is called a *witness function* that $\gamma_1 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_2$.

Notice that $\langle \mu_1, \ell_1 \rangle \overset{\mathsf{tf}}{\Longrightarrow} \langle \mu_2, \ell_2 \rangle$ is possible only when the source control location of tf is equal to ℓ_1 and the destination control location of tf is equal to ℓ_2. Also, it requires that $|\mu_1| = |\mu_2|$. The function h represents how many processes indeed go from one state to another; $f \leqslant h$ expresses that f and h have the same $\#$ values and that, when $f(q, q') \neq \#$, we have that $h(q, q') \geqslant f(q, q')$. This last inequality corresponds to the fact that the number of processes sent from q to q' must be in $[\![f(q, q'), +\infty[\![$. Observe that $\mu_1(q) = \sum_{q'} f(q, q')$ for all q is only

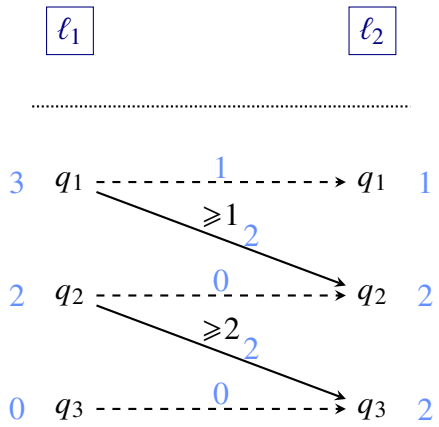


Figure 3.1 – A depiction of Example 3.4. In black, the transfer flow $\mathsf{tf} = (f, \ell_1, \ell_2)$. For each state q, $f(q,q) = 0$; every such pair is represented by an unlabeled, dashed edge indicating that any number of processes may cross using this edge. A pair (q,q') with $f(q,q') > 0$ is represented by an edge with label $\geqslant n$, to indicate that the number of processes to cross using this edge must be at least n, but can be higher. The pairs (q,q') for which $f(q,q') = \#$ are not represented. The blue labels correspond to the multisets μ_1 (left) and μ_2 (right), as well as the non-# values of the function g.

possible when there is q' such that $f(q,q') \neq \#$; symmetrically, $\mu_2(q') = \sum_q f(q,q')$ is only possible when there is q such that $f(q,q') \neq \#$. In fact, this will not be restrictive because we will work with transfer flows for which $f(q,q) \neq \#$ for all q, thanks to idle-compliance. Intuitively, it is always possible to make an extra process remain idle in its state.

Example 3.4. Let $Q \coloneqq \{q_1, q_2, q_3\}$ and $\mathcal{L} \coloneqq \{\ell_1, \ell_2\}$. Consider the transfer flow $\mathsf{tf} = (f, \ell_1, \ell_2)$ where $f(q,q) = 0$ for all $q \in Q$, $f(q_1,q_2) = 1$, $f(q_2,q_3) = 2$ and $f(q,q') = \#$ for all $q \neq q'$ such that $(q,q') \neq (q_1,q_2)$ and $(q,q') \neq (q_2,q_3)$. This transfer flow is represented in Fig. 3.1 (in black). Let $\mu_1 \in \mathcal{M}(Q)$ such that $\mu_1(q_1) = 3$, $\mu_1(q_2) = 2$ and $\mu_1(q_3) = 0$; also, let $\mu_2 \in \mathcal{M}(Q)$ such that $\mu_2(q_1) = 1, \mu_2(q_2) = 2$ and $\mu_2(q_3) = 2$. Let $\gamma_1 \coloneqq \langle \mu_1, \ell_1 \rangle$ and $\gamma_2 \coloneqq \langle \mu_2, \ell_2 \rangle$; we have $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_2$. First, the control locations matches: tf goes from the control location ℓ_1 of γ_1 to the control location ℓ_2 of γ_2. Moreover, a witness function of $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_2$ is the function $g : Q^2 \to \mathbb{N}_\#$ such that $g(q_1,q_1) = 1$, $g(q_1,q_2) = 2$, $g(q_2,q_2) = 0$, $g(q_2,q_3) = 2$, $g(q_3,q_3) = 0$ and $g(q,q') = \#$ for all other (q,q'). The multisets μ_1 and μ_2 and the function g are represented with the blue labels in Fig. 3.1. Function g is a valid witness function for $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_2$ because $f \leqslant g$, $\mu_1(q) = \sum_{q'} g(q,q')$ for all q and $\mu_2(q') = \sum_q g(q,q')$ for all q'.

The following observation is a direct consequence of Definition 3.3:

Fact 3.5. If $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_2$ then $\gamma_1 \overset{\mathsf{tf}'}{\Longrightarrow} \gamma_2$ for every $\mathsf{tf}' \in \mathcal{F}$ such that $\mathsf{tf}' \preceq \mathsf{tf}$.

This matches the intuition from the rule of thumb that a smaller transfer flow is more powerful. Indeed, let $\mathsf{tf} = (f, \ell, \ell')$ and $q, q' \in Q$, and suppose that $f(q, q') = k \in \mathbb{N}$. This means that, with tf, at least k processes may be sent from q to q', but possibly more, so that the set of values allowed for the number of processes sent from q to q' is $[\![k, +\infty[\![$. Therefore, it we change f by setting $f(q, q') \coloneqq \ell \leqslant k$, then the new set of values allowed for the number of processes sent from q to q' is $[\![\ell, +\infty[\![$ and $[\![k, +\infty[\![\subseteq [\![\ell, +\infty[\![$.

It is now time to discuss again the role of the set $\mathcal{T}_{\min} \subseteq \mathcal{F}$. This set is meant to encode the transfer flows allowed by the system. Of course, with the intuition above, for all $t \in \mathcal{T}_{\min}$, for every $\mathsf{tf}' \in \mathcal{F}$ such that $t \preceq \mathsf{tf}'$, tf' should also be allowed by the system. Recall the rule of thumb: smaller transfer flows are more powerful, hence if one can achieve a transfer flow, then one is also able to achieve all larger transfer flows. For this reason, all transfer flows in the upward-closure of $\mathcal{T}_{\min}$ are allowed by the system. We let $\mathcal{T}$ denote this upward-closure. More formally:

$$\mathcal{T} \coloneqq \uparrow\mathcal{T}_{\min} = \{\mathsf{tf} \in \mathcal{F} \mid \exists t \in \mathcal{T}_{\min},\ t \preceq \mathsf{tf}\}.$$

We now define a notion of execution of the copycat system. An *execution* is an alternating sequence $\rho = \gamma_0, \mathsf{tf}_1, \gamma_1, \mathsf{tf}_2, \gamma_2, \ldots, \mathsf{tf}_k, \gamma_k$ with $k \in \mathbb{N}$ and, for all $i \in [\![1, k]\!]$, $\mathsf{tf}_i \in \mathcal{T}_{\min}$ and $\gamma_{i-1} \overset{\mathsf{tf}_i}{\Longrightarrow} \gamma_i$. In this context, $\gamma_{i-1} \overset{\mathsf{tf}_i}{\Longrightarrow} \gamma_i$ is called a *step*, and configurations γ_1 to γ_k are *visited* by ρ. The *length* of execution ρ is defined to be k. We denote by $\gamma_0 \overset{*}{\Longrightarrow} \gamma_k$ the existence of such an execution. In this case, γ_k is said to *reachable from* γ_0. Note that the reachability relation preserves the size of the configurations: if $\gamma \overset{*}{\Longrightarrow} \gamma'$ then $|\gamma| = |\gamma'|$. While we require an execution to only contain transfer flows in $\mathcal{T}_{\min}$, transfer flows in $\mathcal{T}$ would define the same reachability relation, as a direct consequence of Fact 3.5:

Fact 3.6. For every $\gamma, \gamma' \in \Gamma$, $\gamma \overset{*}{\Longrightarrow} \gamma'$ if and only if there is $k \in \mathbb{N}$, $\mathsf{tf}_1, \ldots, \mathsf{tf}_k \in \mathcal{T}$ and $\gamma = \gamma_0, \gamma_1, \ldots, \gamma_k = \gamma'$ such that $\gamma_{i-1} \overset{\mathsf{tf}_i}{\Longrightarrow} \gamma_i$ for every i.

Example 3.7. In order to give some intuition about the definition of copycat systems, we sketch here how ASMS can be encoded into copycat systems. This transformation will be presented in detail in Section 3.5. Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \delta\rangle$ be an ASMS protocol. The corresponding copycat system is $C \coloneqq \langle Q, \mathbb{D}^{\mathsf{dim}}, \mathcal{T}_{\min}\rangle$, so that the states of C are those of $\mathcal{P}$ and the control locations are used to encode the values of the registers. The set $\mathcal{T}_{\min}$ must encode, for each transition $\delta \in \Delta$, the possibilities offered by δ. More precisely, for every $\delta = (q_1, \mathsf{act}, q_2) \in \Delta$, $\mathcal{T}_{\min}$ contains a family of transfer flows $(f, \vec{d}_1, \vec{d}_2)$ where:

— f expresses that at least one process must go from q_1 to q_2 and that any number of

processes may go from q to q for all $q \in Q$;

— $\vec{d}_1$ and $\vec{d}_2$ are so that there are μ_1, μ_2 such that $\langle \mu_1, \vec{d}_1 \rangle \xrightarrow{\delta} \langle \mu_2, \vec{d}_2 \rangle$ (this condition can be made explicit with a case disjunction on act).

Note that there are exponentially many transfer flows in $\mathcal{T}_{\min}$ for each transition in Δ, because one must list all the possibilities $\vec{d}_1$ and $\vec{d}_2$ for the content of the registers. While this definition, one step in C corresponds to an accelerated step in $\mathcal{P}$; indeed, one step in $\mathcal{P}$ allows arbitrarily many processes to go from q_1 to q_2.

3.2.2 Flow Composition

Our aim is to better understand the concept of reachability defined above. To do that, we are now interested in the possibilities offered by using several transfer flows in a row. Given transfers flows $\mathsf{tf}_1, \ldots, \mathsf{tf}_k$, can we give a convenient characterization of the configurations γ_0, γ_k for which there are $\gamma_1, \ldots, \gamma_{k-1}$ such that $\gamma_i \xRightarrow{\mathsf{tf}_{i+1}} \gamma_{i+1}$ for all i? As we will see, the possibilities offered by a sequence of transfer flows can be conveniently described with a set of transfer flows.

We start with two transfer flows used consecutively. To do so, we define the *compositional product* $\mathsf{tf}_1 \otimes \mathsf{tf}_2$ as the set of transfer flows defined as follows.

Definition 3.8. Let $\mathsf{tf}_1 = (f_1, \ell_1, \ell_1'), \mathsf{tf}_2 = (f_2, \ell_2, \ell_2') \in \mathcal{F}$. If $\ell_1' \neq \ell_2$, then we set $\mathsf{tf}_1 \otimes \mathsf{tf}_2 = \emptyset$. Assume now that $\ell_1' = \ell_2$. The set $\mathsf{tf}_1 \otimes \mathsf{tf}_2 \subseteq \mathcal{F}$ is the set that contains all transfer flows of the form (h, ℓ_1, ℓ_2') for which there is $H : Q^3 \to \mathbb{N}_{\#}$ such that:

(3.8.i) for all (q_1, q_3), $\sum_{q_2} H(q_1, q_2, q_3) = h(q_1, q_3)$;

(3.8.ii) for all (q_1, q_2), $\sum_{q_3} H(q_1, q_2, q_3) \geqslant f_1(q_1, q_2)$;

(3.8.iii) for all (q_2, q_3), $\sum_{q_1} H(q_1, q_2, q_3) \geqslant f_2(q_2, q_3)$.

The function H is called a *witness function* that $(h, \ell_1, \ell_2') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$.

In particular, for all q_1, q_2, $f(q_1, q_2) = \#$ if and only if, for all q_3, $H(q_1, q_2, q_3) = \#$. Similarly, $g(q_2, q_3) = \#$ if and only if, for all q_1, $H(q_1, q_2, q_3) = \#$.

We extend the operator $\otimes$ to sets of transfer flows in a natural manner: for $F, F' \subseteq \mathcal{F}$, $F \otimes F' \coloneqq \bigcup_{\mathsf{tf} \in F, \mathsf{tf}' \in F'} \mathsf{tf} \otimes \mathsf{tf}'$. We start with some basic properties of the compositional product:

Lemma 3.9. *For all* $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$*:*

(3.9.i) the set $\mathsf{tf}_1 \otimes \mathsf{tf}_2$ *is upward-closed with respect to* $\preceq$*: for all* $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$*, for all* $\mathsf{tf}' \in \mathcal{F}$*, if* $\mathsf{tf} \preceq \mathsf{tf}'$ *then* $\mathsf{tf}' \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$*;*

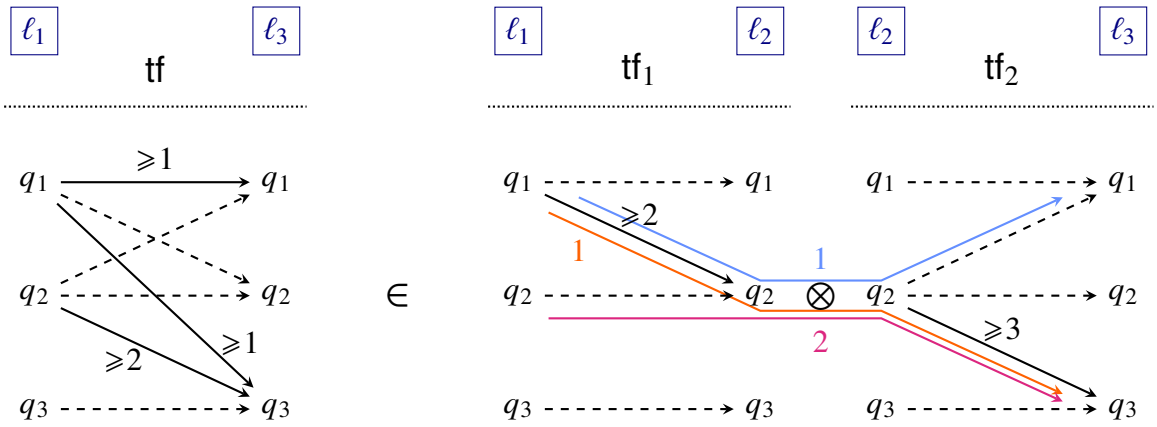


Figure 3.2 – An example of element in a compositional product: here, $\mathsf{tf}_1 \in \mathsf{tf}_2 \otimes \mathsf{tf}_3$. Unlabeled arrows correspond to value 0. The witness function H is represented with dashed arrows. Only values of H greater than 0 are depicted.

(3.9.ii) the compositional product is decreasing with respect to $\preceq$ and $\subseteq$: for all $\mathsf{tf}'_1 \preceq \mathsf{tf}_1$ and $\mathsf{tf}'_2 \preceq \mathsf{tf}_2$, we have $\mathsf{tf}_1 \otimes \mathsf{tf}_2 \subseteq \mathsf{tf}'_1 \otimes \mathsf{tf}'_2$.

Proof. Relegated to Section 3.7.1. □

Example 3.10. An example to illustrate the compositional product can be found in Fig. 3.2. Here, we have $\mathsf{tf}_1 = (f_1, \ell_1, \ell_2)$ and $\mathsf{tf}_2 = (f_2, \ell_2, \ell_3)$ where $f_1(q_1, q_2) = 2$, $f_2(q_2, q_3) = 3$, $f_1(q, q) = f_2(q, q) = 0$ for all q, $f(q_2, q_1) = 0$ and all other values equal to #. Also, let $\mathsf{tf} = (f, \ell_1, \ell_3)$ where $f(q_1, q_1) = 1$, $f(q_1, q_3) = 1$, $f(q_2, q_3) = 2$, $f(q_1, q_1) = f(q_2, q_2) = f(q_3, q_3) = f(q_1, q_2) = f(q_2, q_1) = 0$ and $f(q, q') = \#$ for all other (q, q'). We have $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. Indeed, the control locations match and we have the following witness function $H : Q^3 \to \mathbb{N}_\#$:

— $H(q_1, q_2, q_1) = 1$,

— $H(q_1, q_2, q_3) = 1$,

— $H(q_2, q_2, q_3) = 2$,

— $H(q_1, q_2, q_2) = H(q_2, q_2, q_1) = H(q_2, q_2, q_2) = H(q_3, q_3, q_3) = H(q_1, q_1, q_1) = 0$,

— all other values equal to #.

In fact, tf_1 is minimal for $\preceq$ in $\mathsf{tf}_2 \otimes \mathsf{tf}_3$. Because $\mathsf{tf}_2 \otimes \mathsf{tf}_3$ is upward-closed, increasing the value of non-# components of tf_1 would yield transfer flows that are also in $\mathsf{tf}_2 \otimes \mathsf{tf}_3$.

A necessary but tedious task is to prove that the compositional product $\otimes$ is associative.

Lemma 3.11. *The compositional product $\otimes$ is associative,* i.e.*, for all $\mathsf{tf}_1, \mathsf{tf}_2, \mathsf{tf}_3 \in \mathcal{F}$, $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 = \mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3)$.*

Proof. Relegated to Section 3.7.1. □

Another convenient property of the compositional product is related to the weight of the transfer flows:

Lemma 3.12. *Let* $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$. *For every* $\mathsf{tf} \in \mathsf{basis}(\mathsf{tf}_1 \otimes \mathsf{tf}_2)$, $\mathsf{weight}(\mathsf{tf}) \leqslant \mathsf{weight}(\mathsf{tf}_1) + \mathsf{weight}(\mathsf{tf}_2)$.

Proof. Relegated to Section 3.7.1. □

Given $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$, the set $\mathsf{tf}_1 \otimes \mathsf{tf}_2$ corresponds to the set of transfer flows that can be obtained by using tf_1 then tf_2, as formalized in the following lemma.

Lemma 3.13. *Let* $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$, $\gamma_1, \gamma_3 \in \Gamma$. *We have the following equivalence:*

$$(\exists\, \mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2, \gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_3) \iff (\exists\, \gamma_2 \in \Gamma, \gamma_1 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_2 \overset{\mathsf{tf}_2}{\Longrightarrow} \gamma_3).$$

Proof. Let $\mathsf{tf}_1 =: (f_1, \ell_1, \ell_2)$, $\mathsf{tf}_2 =: (f_2, \ell_2', \ell_3)$. If we have $\ell_2 \neq \ell_2'$ then $\mathsf{tf}_1 \otimes \mathsf{tf}_2 = \emptyset$, both assertions are false and the equivalence holds. Similarly, if the control location of γ_1 is not equal to ℓ_1, then both assertions are false and the equivalence holds, and same for γ_3 and ℓ_3. We now suppose that $\gamma_1 =: \langle \mu_1, \ell_1 \rangle$ and $\gamma_3 =: \langle \mu_3, \ell_3 \rangle$.

Assume first that there is $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ such that $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_3$, let $\mathsf{tf} =: (f, \ell_1, \ell_3)$. Let $g \geqslant f$ witnessing that $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_3$. By hypothesis, $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. By Lemma 3.9, $\mathsf{tf}_1 \otimes \mathsf{tf}_2$ is upward-closed, therefore $\mathsf{tf}' \coloneqq (g, \ell_1, \ell_3) \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. Let $H : Q^3 \to \mathbb{N}_\#$ be a witness function of that. Let $\mu_2 : q_2 \in Q \mapsto \sum_{q_1, q_3} H(q_1, q_2, q_3)$, and let $\gamma_2 \coloneqq \langle \mu_2, \ell_2 \rangle$. Let $h : (q_1, q_2) \in Q^2 \mapsto \sum_{q_3} H(q_1, q_2, q_3)$, we prove that h is a witness function that $\gamma_1 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_2$. By definition of H, for all q_1, q_2, $\sum_{q_3} H(q_1, q_2, q_3) \geqslant f_1(q_1, q_2)$ hence $h(q_1, q_2) \geqslant f_1(q_1, q_2)$, so that $h \geqslant f_1$. By definition of H, for all q_1, $\sum_{q_2} H(q_1, q_2, q_3) = g(q_1, q_3)$ and by definition of g, $\sum_{q_3} g(q_1, q_3) = \mu_1(q_1)$. This gives, for all q_1, $\sum_{q_2} h(q_1, q_2) = \sum_{q_2, q_3} H(q_1, q_2, q_3) = \sum_{q_3} \sum_{q_2} H(q_1, q_2, q_3) = \sum_{q_3} g(q_1, g_3) = \mu_1(q_1)$. Moreover, $\sum_{q_1} h(q_1, q_2) = \sum_{q_1, q_3} H(q_1, q_2, q_3) = \mu_2(q_2)$ by definition of μ_2. This proves that $\gamma_1 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_2$; the proof that $\gamma_2 \overset{\mathsf{tf}_2}{\Longrightarrow} \gamma_3$ is similar.

Conversely, assume that there is γ_2 such that $\gamma_1 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_2 \overset{\mathsf{tf}_2}{\Longrightarrow} \gamma_3$. Let $g_1 \geqslant f_1$ be a witness function that $\gamma_1 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_2$ and $g_2 \geqslant f_2$ a witness function that $\gamma_2 \overset{\mathsf{tf}_2}{\Longrightarrow} \gamma_3$. We build $H : Q^3 \to \mathbb{N}_\#$ that satisfies the following conditions:

(i) for all q_1, q_2, $\sum_{q_3} H(q_1, q_2, q_3) = g_1(q_1, q_2)$,

(ii) for all q_2, q_3, $\sum_{q_1} H(q_1, q_2, q_3) = g_2(q_2, q_3)$.

Indeed, the existence of H would imply that, by letting $h : (q_1, q_3) \mapsto \sum_{q_2} H(q_1, q_2, q_3)$ and $\mathsf{tf} \coloneqq (h, \ell_1, \ell_3)$, we have $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ (with H as witness function, because $g_1 \geqslant f_1$ and $g_2 \geqslant f_2$) and $\gamma_1 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_3$ because $\sum_{q_2} g_1(q_1, q_2) = \mu_1(q_1)$ and $\sum_{q_2} g_2(q_2, q_3) = \mu_3(q_3)$.

We now prove the following statement:

> For every $g_1 : Q^2 \to \mathbb{N}_\#$ and $g_2 : Q^2 \to \mathbb{N}_\#$, if $\sum_{q_1} g_1(q_1, q_2) = \sum_{q_3} g_2(q_2, q_3)$ for every q_2, then there is $H : Q^3 \to \mathbb{N}_\#$ such that $\sum_{q_3} H(q_1, q_2, q_3) = g_1(q_1, q_2)$ and $\sum_{q_1} H(q_1, q_2, q_3) = G(q_2, q_3)$.

First, if F and G are constant equal to # then we set H constant equal to #. Suppose that it is not the case; let $n \coloneqq \sum_{q_1,q_2} g_1(q_1, q_2) = \sum_{q_2,q_3} g_2(q_2, q_3) \in \mathbb{N}$. We proceed by induction on n.

If $n = 0$ then all values in g_1 and g_2 are in $\{0, \#\}$. For each q_1, q_2, q_3, we let $H(q_1, q_2, q_3) \coloneqq 0$ whenever both $g_1(q_1, q_2) = 0$ and $g_2(q_2, q_3) = 0$, and $H(q_1, q_2, q_3) \coloneqq \#$ otherwise. We first prove that, for all q_1, q_2, $\sum_{q_3} H(q_1, q_2, q_3) = g_1(q_1, q_2)$. Let $q_1, q_2 \in Q$; if $g_1(q_1, q_2, q_3) = \#$ then $H(q_1, q_2, q_3) = \#$ for all q_3 hence $\sum_{q_3} H(q_1, q_2, q_3) = \#$. Suppose now that $g_1(q_1, q_2) = 0$. This implies that $\sum_{q_3} g_2(q_2, q_3) = 0$ therefore there is $\tilde{q_3}$ such that $g_2(q_2, \tilde{q_3}) = 0$, so that $H(q_1, q_2, \tilde{q_3}) = 0$ and $\sum_{q_3} H(q_1, q_2, q_3) = 0$. Similarly, for every q_2, q_3, if $g_2(q_2, q_3) = \#$ then $\sum_{q_1} H(q_1, q_2, q_3) = \#$ and if $g_2(q_2, q_3) = 0$ then there is $\tilde{q_1}$ such that $g_1(\tilde{q_1}, q_2) = 0$ hence $H(\tilde{q_1}, q_2, q_3) = 0$ and $\sum_{q_1} H(q_1, q_2, q_3) = 0$.

Suppose now that $n > 0$. There exists $\tilde{q_2}$ such that $\sum_{q_1} g_1(q_1, \tilde{q_2}) = \sum_{q_3} g_2(\tilde{q_2}, q_3) > 0$. Let $\tilde{q_1}$ such that $g_1(\tilde{q_1}, \tilde{q_2}) > 0$ and $\tilde{q_3}$ such that $g_2(\tilde{q_2}, \tilde{q_3}) > 0$. Let g'_1 equal to g_1 except that $g'_1(\tilde{q_1}, \tilde{q_2}) \coloneqq g_1(\tilde{q_1}, \tilde{q_2}) - 1$ and let g'_2 equal to g_2 except that $g'_2(\tilde{q_2}, \tilde{q_3}) \coloneqq g_2(\tilde{q_2}, \tilde{q_3}) - 1$. We have $\sum_{q_1} g'_1(q_1, q_2) = \sum_{q_3} g'_2(q_2, q_3)$ for all q_2, and $\sum_{q_1,q_2} g'_1(q_1, q_2) = \sum_{q_2,q_3} g_2(q_2, q_3) - 1 = n - 1$. We apply the induction hypothesis on g'_1 and g'_2 to obtain H' such that $\sum_{q_3} H'(q_1, q_2, q_3) = g'_1(q_1, q_2)$ for all q_1, q_2 and $\sum_{q_1} H'(q_1, q_2, q_3) = g'_2(q_2, q_3)$ for all q_2, q_3. It suffices to let H equal to H' except that $H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) \coloneqq H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) + 1$. Note that it could be that $H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) = \#$, in which case $H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) = 1$. We know that $g'_1(\tilde{q_1}, \tilde{q_2}) \neq \#$ therefore $\sum_{q_3} H'(\tilde{q_1}, \tilde{q_2}, q_3) \neq \#$ so that we indeed have $\sum_{q_3} H(\tilde{q_1}, \tilde{q_2}, q_3) = g'_1(\tilde{q_1}, \tilde{q_2}) + 1 = g_1(\tilde{q_1}, \tilde{q_2})$. With the same argument, $\sum_{q_1} H(q_1, \tilde{q_2}, \tilde{q_3}) = g_2(\tilde{q_2}, \tilde{q_3})$. This concludes the induction.

By letting $h : (q_1, q_3) \mapsto \sum_{q_2} H(q_1, q_2, q_3)$ and $\mathsf{tf} = (h, \ell_1, \ell_3)$, we have $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ and $\gamma_1 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_3$, concluding the proof. □

By iterating Lemma 3.13, we obtain a similar property for a larger number of transfer flows:

Lemma 3.14. *Let $k \geqslant 1$, $\mathsf{tf}_1, \dots, \mathsf{tf}_k \in \mathcal{F}$, $\gamma_0, \dots, \gamma_k \in \Gamma$. We have the following equivalence:*

$$\exists \gamma_1, \dots, \gamma_{k-1},\ \gamma_0 \overset{\mathsf{tf}_1}{\Longrightarrow} \gamma_1 \overset{\mathsf{tf}_2}{\Longrightarrow} \dots \overset{\mathsf{tf}_k}{\Longrightarrow} \gamma_k \quad \Longleftrightarrow \quad \exists \mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2 \otimes \dots \otimes \mathsf{tf}_k,\ \gamma_0 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_k.$$

Proof. We proceed by induction on k. The case $k = 1$ is immediate. Let $k \geqslant 1$. By Lemma 3.13, there is $\mathsf{tf} \in \mathsf{tf}_1 \otimes \dots \otimes \mathsf{tf}_{k+1}$ such that $\gamma_0 \overset{\mathsf{tf}}{\Longrightarrow} \gamma_{k+1}$ if and only if there is $\mathsf{tf}' \in \mathsf{tf}_1 \otimes \dots \otimes \mathsf{tf}_k$ and $\gamma_k \in \Gamma$ such that $\gamma_0 \overset{\mathsf{tf}'}{\Longrightarrow} \gamma_k \overset{\mathsf{tf}_{k+1}}{\Longrightarrow} \gamma_{k+1}$. By induction hypothesis, for a given γ_k, the existence of $\mathsf{tf}' \in \mathsf{tf}_1 \otimes \dots \otimes \mathsf{tf}_k$ such that $\gamma_0 \overset{\mathsf{tf}'}{\Longrightarrow} \gamma_k$ is equivalent to the existence of $\gamma_1, \dots, \gamma_{k-1}$ such that $\gamma_0 \overset{\mathsf{tf}_1}{\Longrightarrow} \dots \overset{\mathsf{tf}_k}{\Longrightarrow} \gamma_k$, which concludes the proof. □

Let $\mathcal{F}_0 \subseteq \mathcal{F}$ be the set of transfer flows (f, ℓ, ℓ) with $\ell \in \mathcal{L}$ and f such that $f(q, q') = \#$ for all $q \neq q'$ and $f(q, q) \in \mathbb{N}$ for all $q \in Q$. Intuitively, $\mathcal{F}_0$ contains trivial transfer flows where all processes remain idle. Note that $\mathcal{F}_0$ is upward-closed and that $\mathsf{basis}(\mathcal{F}_0) = \{(f_0, \ell, \ell) \mid \ell \in \mathcal{L}\}$ where $f_0(q, q) = 0$ for all q and $f_0(q, q') = \#$ for all $q \neq q'$.

Lemma 3.15. *For all $T \subseteq \mathcal{F}$ upward-closed for $\preceq$, $\mathcal{F}_0 \otimes T = T \otimes \mathcal{F}_0 = T$.*

Proof. Relegated to Section 3.7.1. □

Given $T \subseteq \mathcal{F}$ and $k \in \mathbb{N}$, we define $T^k \subseteq \mathcal{F}$ by $T^0 \coloneqq \mathcal{F}_0$ and $T^{k+1} \coloneqq T^k \otimes T$. In particular, we have $T^1 = T$. The set T^k contains all possibilities obtained by using exactly k transfer flows in T. Similarly, we let $T^{\leqslant k} \coloneqq \bigcup_{\ell \leqslant k} T^\ell$. Finally, we let $T^* \coloneqq \bigcup_{k \in \mathbb{N}} T^k$. The set T^* encodes all transfer flows possible by combining transfer flows from $\mathcal{T}$:

Lemma 3.16. *For all $\gamma, \gamma' \in \Gamma$, $\gamma \overset{*}{\Rightarrow} \gamma'$ if and only if there is $\mathsf{tf} \in \mathcal{T}^*$ such that $\gamma \overset{\mathsf{tf}}{\Rightarrow} \gamma'$.*

Proof. By definition, $\gamma \overset{*}{\Rightarrow} \gamma'$ if and only if there exist $k \in \mathbb{N}$ and $\gamma = \gamma_0, \mathsf{tf}_1, \gamma_1, \mathsf{tf}_2, \gamma_2, \dots, \mathsf{tf}_k, \gamma_k = \gamma'$ such that, for all $i \in [\![1, k]\!]$, $\mathsf{tf}_i \in \mathcal{T}_{\min}$ and $\gamma_{i-1} \overset{\mathsf{tf}_i}{\Rightarrow} \gamma_i$. If $k = 0$, then we must have $\gamma = \gamma'$, which is equivalent to the existence of $\mathsf{tf}_0 \in \mathcal{F}_0$ such that $\gamma \overset{\mathsf{tf}_0}{\Longrightarrow} \gamma'$. By Lemma 3.14 and Fact 3.6, this proves that $\gamma \overset{*}{\Rightarrow} \gamma'$ if and only if there is $k \geqslant 0$ and $\mathsf{tf} \in \mathcal{T}^k$ such that $\gamma \overset{*}{\Rightarrow} \gamma'$, so that $\gamma \overset{*}{\Rightarrow} \gamma'$ if and only if there is $\mathsf{tf} \in \mathcal{T}^*$ such that $\gamma \overset{*}{\Rightarrow} \gamma'$. □

The set $\mathcal{T}^*$ is upward-closed with respect to $\preceq$, therefore it has a finite basis $\mathsf{basis}(\mathcal{T}^*)$. This basis fully characterizes the reachability relation: for all γ, γ', we have $\gamma \overset{*}{\Rightarrow} \gamma'$ if and only if there is $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^*)$ such that $\gamma \overset{\mathsf{tf}}{\Rightarrow} \gamma'$. For this reason, we are interested in the set $\mathsf{basis}(\mathcal{T}^*)$.

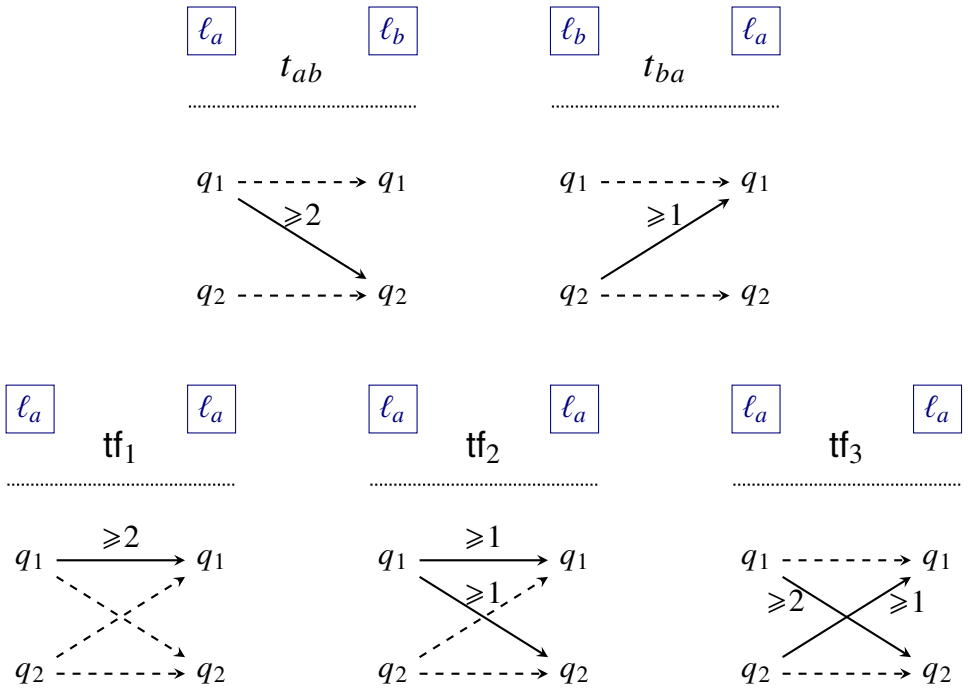


Figure 3.3 – The example of copycat system from Example 3.17. As before, dashed edge correspond to value 0, and # edges are not represented.

Example 3.17. Consider the copycat system $C = (\{q_1, q_2\}, \{\ell_a, \ell_b\}, \{t_{ab}, t_{ba}\})$ where:

— $t_{ab} = (f_{ab}, \ell_a, \ell_b)$ with $f_{ab}(q_1, q_1) = f_{ab}(q_2, q_2) = 0$, $f_{ab}(q_1, q_2) = 2$ and $f_{ab}(q_2, q_1) = \#$,

— $t_{ba} = (f_{ba}, \ell_b, \ell_a)$ with $f_{ba}(q_1, q_1) = f_{ba}(q_2, q_2) = 0$, $f_{ba}(q_2, q_1) = 1$ and $f_{ba}(q_1, q_2) = \#$.

The two transfer flows are depicted in Fig. 3.3. Note that they both satisfy the idle-compliance property, because $f_{ab}(q, q) \neq \#$ and $f_{ba}(q, q) \neq \#$ for all $q \in Q$. Let $\mathcal{F}_{aa} \coloneqq \{(f, \ell_a, \ell_a) \mid f : Q^2 \to \mathbb{N}_\#\}$ denote transfer flows from ℓ_a to ℓ_a. In Fig. 3.3, we illustrate three transfer flows, denoted $\mathsf{tf}_1, \mathsf{tf}_2$ and tf_3, that are in $\mathcal{T}^* \cap \mathcal{F}_{aa}$. It is easy to see that they are incomparable and that they are exactly the minimal elements of $t_{ab} \otimes t_{ba}$, so that $\{\mathsf{tf}_1, \mathsf{tf}_2, \mathsf{tf}_3\} = \mathsf{basis}(t_{ab} \otimes t_{ba})$. This implies that $\mathsf{basis}((t_{ab} \otimes t_{ba})^2) \subseteq \{\mathsf{tf}_i \otimes \mathsf{tf}_j \mid i, j \in \{1, 2, 3\}\}$. Moreover, with a (tedious) case disjunction, one cna show that, for all $i, j \in \{1, 2, 3\}$, $\mathsf{tf}_i \otimes \mathsf{tf}_j \subseteq t_{ab} \otimes t_{ba}$. This implies that $(t_{ab} \otimes t_{ba})^2 \subseteq t_{ab} \otimes t_{ba}$. With a symmetric argument, one can prove that $(t_{ba} \otimes t_{ab})^2 \subseteq t_{ba} \otimes t_{ab}$. This proves that $\mathcal{T}^4 \subseteq \mathcal{T}^2$, so that $\mathcal{T}^* = \mathcal{T}^{\leqslant 3}$.

Remark 3.18. The model of transfer flows arguably constitutes an interesting formalism on its own. To the best of our knowledge, this notion is new. In [BDGG17], the authors use so-called *transfer graphs*, which are subsets of Q^2 describing from which state to which state processes may go. In our formalism, this would correspond to transfer flows where $\mathcal{L} = \emptyset$ and whose

values are in $\{\#, 0\}$. No notion of composition of transfer graphs is defined in [BDGG17]. In the probabilistic extension of the previous work [CFO20], the authors consider flows, which are elements of $Q^2 \to \mathbb{N}$; they also rely on an implicit flow composition. In their case, however, an edge from q to q' labeled n expresses that *at most* n processes may go from q to q', so that the set of achievable flows is downward-closed and not upward-closed.

Another family of models that is related to transfer flows is the one of threshold automata [KVW14]. The threshold constraints are somewhat reminiscent of our constraints on the number of processes in transfer flows. An important result in threshold automata is that the diameter is bounded, so that any two configurations can be connected with an accelerated execution of bounded length, a result analogue to our structural theorem. However, the analogy between copycat systems and threshold automata is limited: the two models have several crucial differences, and the bound on the diameter for threshold automata is polynomial while our bound from the structural theorem is doubly-exponential.

The main contribution of this chapter is to provide two bounds related to $\mathsf{basis}(\mathcal{T}^*)$. The first bound is related to the smallest value of K such that $\mathsf{basis}(\mathcal{T}^*) \subseteq \mathcal{T}^{\leqslant K}$. In other words, it bounds the number of steps needed to connect two configurations. The second bound is related to the norm of the elements of $\mathsf{basis}(\mathcal{T}^*)$; it bounds, in a given execution, the number of processes that cannot be removed without affecting the rest of the execution.

We are yet to define a notion of size of a copycat system. In fact, we will not rely on a single such notion but on two different ones. The first one is $n(C) \coloneqq |Q|$, and the other one is $M(C) \coloneqq |\mathcal{L}| + |\mathcal{T}_{\min}| + \max_{t \in \mathcal{T}_{\min}} \mathsf{weight}(t)$. Note that a copycat system can be stored in space polynomial in $n(C) + M(C)$. The reason for this distinction is that we will, in the next section, obtain a bound of the form $O(M(C))^{2^{O(n(C))}}$, whose dependency in $n(C)$ and in $M(C)$ are therefore very different. Tracking this difference will allow us, in Section 3.4 and Section 3.5, to allow for exponential blowups in $M(C)$ without affecting the overall order of magnitude of our bounds.

The following result constitutes the main contribution of this chapter. Its proof relies on the bound on descending chains from Theorem 1.9 and is the topic of the next section.

Theorem 3.19 (Structural Theorem)**.**
Let $L : (n, M) \mapsto (M^2 2^{n^2})^{3^{n^2+2}(\log(n^2+2)+1)}$ *and* $B : (n, M) \mapsto M \cdot L(n, M)$. *We have* $\mathcal{T}^* = \mathcal{T}^{\leqslant L(n(C), M(C))}$ *and, for all* $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^*)$, $\mathsf{weight}(\mathsf{tf}) \leqslant B(n(C), M(C))$.

3.3 The Structural Bound

In this section, we use the result from [LS21; SS24] introduced in Theorem 1.9 to prove Theorem 3.19. Again, we fix a copycat system $C = (Q, \mathcal{L}, \mathcal{T}_{\min})$.

3.3.1 Mapping Transfer Flows to Vectors

Let $n := n(C) = |Q|$ and $M := M(C) = |Q| + |\mathcal{L}| + |\mathcal{T}_{\min}| + \max_{t \in \mathcal{T}_{\min}} \mathsf{weight}(t)$. Also, let $d := n^2 + 2$ and $N := |\mathcal{L}^2 \times 2^{Q^2}| \leqslant M^2 2^{n^2}$. We fix two arbitrary bijective mappings $\theta : \mathcal{L}^2 \times 2^{Q^2} \to [\![1, N]\!]$ and $\mathsf{index} : Q^2 \to [\![1, n^2]\!]$.

We map transfer flows to sets of vectors with a mapping $\chi : \mathcal{F} \to 2^{\mathbb{N}^d}$ defined as follows. Let $\mathsf{tf} = (f, \ell, \ell') \in \mathcal{F}$. Let $S := \{(q, q') \mid f(q, q') = \#\}$. A vector $\vec{v} \in \mathbb{N}^d$ is in $\chi(\mathsf{tf})$ if and only if:

— for all $(q, q') \notin S$ (*i.e.*, such that $f(q, q') \neq \#$), $\vec{v}(\mathsf{index}(q, q')) = f(q, q')$;

— $\vec{v}(n^2 + 1) = \theta(\ell, \ell', S)$,

— $\vec{v}(n^2 + 2) = N + 1 - \theta(\ell, \ell', S)$.

Note that there is no restriction to the value $\vec{v}(i)$ when the corresponding pair $(q, q') = \mathsf{index}^{-1}(i)$ is in S, *i.e.*, when $f(q, q') = \#$. Also, this definition guarantees that, if $\vec{v} \in \chi(\mathsf{tf})$ and $\vec{u} \in \chi(\mathsf{tf}')$ are such that $\vec{v} \leqslant_\times \vec{u}$, then $\vec{u}(n^2 + 1) = \vec{v}(n^2 + 1)$ and $\vec{u}(n^2 + 2) = \vec{v}(n^2 + 2)$, so that tf and tf' have the same control locations and the same # values. This mapping satisfies a convenient property which we dub *strong injectivity*:

Lemma 3.20 (Strong injectivity)**.** *For all* $\mathsf{tf} \in \mathcal{F}$, $\chi(\mathsf{tf}) \neq \emptyset$ *and for all* $\mathsf{tf}' \neq \mathsf{tf}$, $\chi(\mathsf{tf}) \cap \chi(\mathsf{tf}') = \emptyset$.

Proof. Let $\mathsf{tf} = (f, \ell_1, \ell_2), \mathsf{tf}' = (f', \ell'_1, \ell'_2) \in \mathcal{F}$. Let $S := \{(q, q') \mid f(q, q') = \#\}$ and $S' := \{(q, q') \mid f(q, q') = \#\}$. Let $\vec{v}$ be the vector such that $\vec{v}(i) = f(\mathsf{index}^{-1}(i))$ when $\mathsf{index}^{-1}(i) \in S$, $\vec{v}(i) = 0$ when $\mathsf{index}^{-1}(i) \notin S$, $\vec{v}(n^2 + 1) = \theta(\ell_1, \ell_2, S)$ and $\vec{v}(n^2 + 2) = N + 1 - \theta(\ell_1, \ell_2, S)$. Trivially, $\vec{v} \in \chi(\mathsf{tf})$ so that $\chi(\mathsf{tf}) \neq \emptyset$. We now prove that $\chi(\mathsf{tf}) \cap \chi(\mathsf{tf}') \neq \emptyset$ implies that $\mathsf{tf} = \mathsf{tf}'$; suppose that $\chi(\mathsf{tf}) \cap \chi(\mathsf{tf}') \neq \emptyset$ and let $\vec{v} \in \chi(\mathsf{tf}) \cap \chi(\mathsf{tf}')$. We have $\vec{v}(n^2 + 1) = \theta(\ell_1, \ell_2, S) = \theta(\ell'_1, \ell'_2, S')$ and, by injectivity of θ, $\ell_1 = \ell'_1$, $\ell_2 = \ell'_2$ and $S = S'$. This implies that $f(q, q') = g(q, q') = \#$ for all $(q, q') \in S$. Moreover, for every $(q, q') \notin S$, $f(q, q') = \vec{v}(\mathsf{index}(q, q')) = f'(q, q')$. We have proved that $\mathsf{tf} = \mathsf{tf}'$, concluding the proof. □

Given a set F of transfer flows, we write $\chi(F) := \bigcup_{\mathsf{tf} \in F} \chi(\mathsf{tf})$. The vectors of $\mathbb{N}^d$ that are in $\chi(\mathcal{F})$ are exactly those whose last two components are (strictly) positive and sum to $N + 1$. Note that, even for $T \subseteq \mathcal{F}$ is upward-closed in$(\mathcal{F}, \preceq)$, then $\chi(T)$ is *not* upward-closed in $(\mathbb{N}^d, \leqslant_\times)$.

Indeed, the upward-closure of $\chi(T)$ would include vectors whose last two components sum to values strictly greater than $N+1$, and such vectors are not in $\chi(\mathcal{F})$.

Our aim is to build a descending chain (D_k) of $\mathbb{N}^d$ such that D_k contains the images by χ of transfer flows in $\mathcal{F} \setminus \mathcal{T}^{\leqslant k}$. Let V_0 be the set of vectors $\vec{v}$ such that either $\vec{v}(n^2+1) = N+1$ and $\vec{v}(n^2+2) = 0$ or $\vec{v}(n^2+1) = 0$ and $\vec{v}(n^2+2) = N+1$. For technical reasons, we will enforce that $D_k \cap V_0 = \emptyset$ for every k. Note that $V_0 \cap \chi(\mathcal{F}) = \emptyset$: vectors in V_0 have no relevance in terms of transfer flows.

3.3.2 Construction of the Descending Chain

We now build a descending chain in $\mathbb{N}^d$ related to the sets of transfer flows $\mathcal{T}^{\leqslant k}$. For all $k \geqslant 0$, let U_k denote the upward-closure, in $(\mathbb{N}^d, \leqslant_\times)$, of $\chi(\mathcal{T}^{\leqslant k}) \cup V_0$:

$$U_k \coloneqq \uparrow(\chi(\mathcal{T}^{\leqslant k}) \cup V_0).$$

Intuitively, U_k corresponds to the vector counterpart of $\mathcal{T}^{\leqslant k}$. For technical reasons, we want the set V_0 to be included in U_k for each k. Because $\mathcal{T}^{\leqslant k}$ is upward-closed with respect to $\preceq$, taking the upward-closure for $\leqslant_\times$ of $\chi(\mathcal{T}^{\leqslant k})$ only adds in U_k vectors with higher values in the last two components:

Lemma 3.21. *For all $k \geqslant 0$, $U_k \cap \chi(\mathcal{F}) = \chi(\mathcal{T}^{\leqslant k})$.*

Proof. Trivially, $\chi(\mathcal{T}^{\leqslant k}) \subseteq U_k \cap \chi(\mathcal{F})$. Conversely, let $\vec{v} \in U_k \cap \chi(\mathcal{F})$. There exists $\vec{u} \in \chi(\mathcal{T}^{\leqslant k}) \cup V_0$ such that $\vec{u} \leqslant_\times \vec{v}$. Since $\vec{v} \in \chi(\mathcal{F})$, the last two components of $\vec{v}$ sum to $N+1$ and same for $\vec{u}$, so that $\vec{u}(n^2+1) = \vec{v}(n^2+1)$ and $\vec{u}(n^2+2) = \vec{v}(n^2+2)$. This proves that $\vec{u} \notin V_0$ because $\vec{v} \in \chi(\mathcal{F})$, therefore $\vec{u} \in \chi(\mathcal{T}^{\leqslant k})$. Let $\mathsf{tf}_u = (f_u, \ell_u, \ell'_u) \in \mathcal{T}^{\leqslant k}$ such that $\vec{u} \in \chi(\mathsf{tf}_u)$; let $\mathsf{tf}_v = (f_v, \ell_v, \ell'_v) \in \mathcal{F}$ such that $\vec{v} \in \chi(\mathsf{tf}_v)$. Because $\vec{u}$ and $\vec{v}$ coincide on the last two component, we have $\ell_u = \ell_v$ and $\ell'_u = \ell'_v$; also, $f_u(q,q') = \#$ whenever $f_v(q,q') = \#$. When $f_u(q,q'), f_v(q,q') \neq \#$, we have $f_v(q,q') \leqslant f_u(q,q')$. This proves that $\mathsf{tf}_u \preceq \mathsf{tf}_v$, hence $\mathsf{tf}_v \in \mathcal{T}^{\leqslant k}$ because $\mathcal{T}^{\leqslant k}$ is upward-closed. □

For every k, we let $D_k \coloneqq \mathbb{N}^d \setminus U_k$. Trivially, the sets D_k are downward-closed. Also for all k, $\mathcal{T}^{\leqslant k} \subseteq \mathcal{T}^{\leqslant k+1}$ so that $U_k \subseteq U_{k+1}$ thus $D_{k+1} \subseteq D_k$. Therefore, because all descending chains are finite by Proposition 1.6, there exists k such that $D_k = D_{k+1}$. Let L be the smallest k such that $D_k = D_{k+1}$: $(D_k)_{k \leqslant L}$ is a descending chain. This length L has a direct implication on $\mathcal{T}^{\leqslant k}$:

Lemma 3.22. $\mathcal{T}^* = \mathcal{T}^{\leqslant L}$.

Proof. We have $D_{L+1} = D_L$ therefore $U_{L+1} = U_L$ so that, thanks to Lemma 3.21, $\chi(\mathcal{T}^{\leqslant L+1}) = \chi(\mathcal{T}^{\leqslant L})$. Lemma 3.20 tells us that χ is strongly injective hence in particular injective, so that $\mathcal{T}^{\leqslant L+1} = \mathcal{T}^{\leqslant L}$. Said otherwise, $\mathcal{T}^{\leqslant L} \otimes \mathcal{T} = \mathcal{T}^{\leqslant L}$; by direct induction, for all $k \geqslant L$, $\mathcal{T}^{\leqslant k} = \mathcal{T}^{\leqslant L}$, so that $\mathcal{T}^* = \mathcal{T}^{\leqslant L}$. □

Towards proving Theorem 3.19, we want to argue that $L \leqslant L(n, M)$. To apply Theorem 1.9, we will prove that $(D_k)_{k \leqslant L}$ is C-controlled for some C and is ω-monotone.

3.3.3 The Descending Chain is Controlled and ω-monotone

Let $\lambda \coloneqq \max_{t \in \mathcal{T}_{\min}} \mathsf{weight}(t)$. TWe first prove that $(D_k)_{k \leqslant L}$ is C-controlled where $C \coloneqq N + \lambda$.

Lemma 3.23. *The descending chain $(D_k)_{k \leqslant L}$ is C-controlled.*

Towards proving Lemma 3.23, we start by bounding the norm of minimal elements of U_k. For all k, U_k is upward-closed for $\leqslant_\times$ hence it has a finite basis $\mathsf{basis}(U_k)$. Because U_k is the upward-closure of $\chi(\mathcal{T}^{\leqslant k}) \cup V_0$, we have $\mathsf{basis}(U_k) \subseteq \chi(\mathcal{T}^{\leqslant k}) \cup V_0$. A vector $\vec{v} \in \mathsf{basis}(V_0)$ is such that $\vec{v}(i) = 0$ for all $i \in [\![1, n^2]\!]$, and $\max(\vec{v}(n^2+1), \vec{v}(n^2+2)) = N+1$, so that $\|\vec{v}\| = N+1$. We now consider vectors in $\mathsf{basis}(U_k) \cap \chi(\mathcal{T}^{\leqslant k})$; such vectors must be minimal in $\chi(\mathcal{T}^{\leqslant k})$.

Lemma 3.24. *Minimal vectors of $\chi(\mathcal{T}^{\leqslant k})$ are in $\chi(\mathsf{basis}(\mathcal{T}^{\leqslant k}))$.*

Proof. Let $\vec{v}$ minimal in $\chi(\mathcal{T}^{\leqslant k})$. In particular, $\vec{v} \in \chi(\mathcal{T}^{\leqslant k})$; let $\mathsf{tf} = (f, \ell_1, \ell_2) \in \mathcal{T}^{\leqslant k}$ such that $\vec{v} \in \chi(\mathsf{tf})$. Our aim is to prove that $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$. Let $S \coloneqq \{(q, q') \mid f(q, q') = \#\}$. Let $\mathsf{tf}' = (f', \ell_1', \ell_2) \preceq \mathsf{tf}$; we prove that $\mathsf{tf}' = \mathsf{tf}$. Because $\mathsf{tf}' \preceq \mathsf{tf}$, by letting $S' \coloneqq \{(q, q') \mid f(q, q') = \#\}$, we have $S' = S$. Therefore, there exists $\vec{u} \in \chi(\mathsf{tf}')$ such that $\vec{u}(i) = 0$ for all $i \in \mathsf{index}^{-1}(S)$. We claim that $\vec{u} \leqslant_\times \vec{v}$. We have $\vec{u}(n^2+1) = \vec{v}(n^2+1)$ and $\vec{u}(n^2+2) = \vec{v}(n^2+2)$; for all $i \in \mathsf{index}^{-1}(S)$, $\vec{u}(i) = 0 \leqslant \vec{v}(i)$; for all $i \notin \mathsf{index}^{-1}(S)$, by letting $(q, q') \coloneqq \mathsf{index}^{-1}(i)$, we have $\vec{u}(i) = f'(q, q') \leqslant f(q, q') = \vec{v}(i)$. We have therefore $\vec{u} \in \chi(\mathcal{T}^{\leqslant k})$ and $\vec{u} \leqslant_\times \vec{v}$, but $\vec{v}$ is minimal in $\mathsf{basis}(\chi(\mathcal{T}^{\leqslant k}))$ therefore $\vec{u} = \vec{v}$. This implies that $f = f'$ hence that $\mathsf{tf} = \mathsf{tf}'$. □

Note that, because $\chi(\mathcal{T}^{\leqslant k})$ is not upward-closed, we cannot write that minimal vectors of $\chi(\mathcal{T}^{\leqslant k})$ are in the basis of the set. The remaining task is to bound the values of transfer flows in $\mathsf{basis}(\mathcal{T}^{\leqslant k})$, which is achieved with the following lemma:

Lemma 3.25. *For all $k \geqslant 0$, for all $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$, $\mathsf{weight}(\mathsf{tf}) \leqslant \lambda k$.*

Proof. The proof is by induction on k and relies on Lemma 3.12. For $k = 0$, $\mathcal{T}^{\leqslant 0} = \mathcal{F}_0$ and transfer flows in $\mathsf{basis}(\mathcal{F}_0)$ only have values 0 and #, so that they have weight 0. Suppose that the statement is true for k, and prove it for $k+1$. Let $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^{\leqslant k+1})$. If $\mathsf{tf} \in \mathcal{T}^{\leqslant k}$ then, because $\mathcal{T}^{\leqslant k} \subseteq \mathcal{T}^{\leqslant k+1}$, $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$ and it suffices to apply the induction hypothesis on tf. Otherwise, there is $\mathsf{tf}_k \in \mathcal{T}^{\leqslant k}$, $t \in \mathcal{T}_{\min}$ such that $\mathsf{tf} \in \mathsf{tf}_k \otimes t$. By Lemma 3.9, we may assume that $\mathsf{tf}_k \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$. By applying the induction hypothesis, we have $\mathsf{weight}(\mathsf{tf}_k) \leqslant \lambda k$; by definition of λ, we have $\mathsf{weight}(t) \leqslant \lambda$. By Lemma 3.12, we obtain that $\mathsf{weight}(\mathsf{tf}) \leqslant \lambda k + \lambda = (k+1)\lambda$, concluding the induction. □

We now conclude the proof of Lemma 3.23. Let $k \in \mathbb{N}$, and let $\vec{v} \in \mathbb{N}_\omega^d$ be the representing vector of some ideal of D_k. First, we argue that $\vec{v}(i) \leqslant N$ for $i \in \{n^2+1, n^2+2\}$. Indeed, we would otherwise have a vector $\vec{u} \leqslant_\times \vec{v}$ such that $\vec{u} \in V_0$, which contradicts the fact that $V_0 \subseteq U_k$. Let $i \in [\![1, n^2]\!]$ such that $\vec{v}(i) \neq \omega$. Let $\vec{u}$ denote the vector equal to $\vec{v}$ except that $\vec{u}(i) \coloneqq \vec{v}(i) + 1$. Because $\vec{v}$ is maximal in D_k, $\downarrow\{\vec{u}\} \not\subseteq D_k$. Therefore, there is a vector $\vec{u}_m \in \mathsf{basis}(U_k)$ such that $\vec{u}_m \leqslant_\times \vec{u}$. We must have $\vec{u}_m(i) = \vec{v}(i) + 1$ because we would otherwise have $\vec{u}_m \leqslant_\times \vec{v}$, which would imply that $\vec{u}_m \in D_k$ and would contradict $\vec{u}_m \in U_k$. By definition of U_k, we have $\vec{u}_m \in V_0 \cup \chi(\mathcal{T}^{\leqslant k})$; but $\vec{u}_m(i) > 0$, hence $\vec{u}_m \notin V_0$ therefore $\vec{u}_m \in \chi(\mathcal{T}^{\leqslant k})$. Moreover, because $\vec{u}_m \in \mathsf{basis}(U_k)$, by Lemma 3.24, there is $\mathsf{tf}_m = (f_m, \ell, \ell') \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$ such that $\vec{u}_m \in \chi(\mathsf{tf}_m)$. By Lemma 3.25, we have $\mathsf{weight}(\mathsf{tf}_m) \leqslant \lambda k$ so that $f_m(q, q') \in [\![0, \lambda k]\!] \cup \{\#\}$ for all q, q'. This proves in particular that $\vec{v}(i) \leqslant \vec{u}_m(i) \leqslant \lambda k$. Overall, we have proved that, for all $i \in [\![1, n^2]\!]$ such that $\vec{v}(i) \neq \omega$, $\vec{v}(i) \leqslant \lambda k$, and that $\vec{v}(n^2+1) \leqslant N$ and $\vec{v}(n^2+2) \leqslant N$, so that $\|\vec{v}\| \leqslant \max(\lambda k, N)$. This proves that the norm of D_k is bounded by $\max(N, \lambda k) \leqslant (N+\lambda)(k+1)$, concluding the proof of Lemma 3.23.

The last ingredient needed to apply Theorem 1.9 is the ω-monotonicity. While this may seem like a technical detail (and it is), the proof is actually quite involved. The reason for that is that our encoding into $\mathbb{N}^d$ is quite artificial and makes it tedious to reason about the ideals of D_k. The proof of the following lemma is therefore fully relegated to Section 3.7.2.

Lemma 3.26. *$(D_k)_{k \leqslant L}$ is ω-monotone.*

3.3.4 The Structural Theorem

We are now ready to conclude the proof of the main result of this chapter:

Theorem 3.19 (Structural Theorem)**.**
Let $L : (n, M) \mapsto (M^2 2^{n^2})^{3^{n^2+2}(\log(n^2+2)+1)}$ and $B : (n, M) \mapsto M \cdot L(n, M)$. We have $\mathcal{T}^* = \mathcal{T}^{\leqslant L(n(C), M(C))}$ and, for all $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^*)$, $\mathsf{weight}(\mathsf{tf}) \leqslant B(n(C), M(C))$.

Proof. Let $n \coloneqq n(C)$ and $M \coloneqq M(C)$. We construct the descending chain $(D_k)_{k \leqslant L}$ from Section 3.3.2. As before, $N \coloneqq |\mathcal{L}^2 \times 2^{Q^2}|$, $d \coloneqq n^2+2$, $\lambda \coloneqq \max_{t \in \mathcal{T}_{\min}} \mathsf{weight}(t)$ and $C \coloneqq N+\lambda$. We have $C \leqslant |\mathcal{L}|^2 2^{n^2} + \lambda \leqslant (|\mathcal{L}|+\lambda)^2 2^{n^2} \leqslant M^2 2^{n^2}$. Thanks to Lemma 3.23, $(D_k)_{k \leqslant L}$ is C-controlled. Thanks to Lemma 3.26, $(D_k)_{k \leqslant L}$ is ω-monotone. We apply Theorem 1.9: $L \leqslant C^{3^d(\log(d)+1)} = (M^2 2^{n^2})^{3^{n^2+2}(\log(n^2+2)+1)} = L(n, M)$. By Lemma 3.22, $\mathcal{T}^* = \mathcal{T}^{\leqslant L}$, thus $\mathcal{T}^* = \mathcal{T}^{\leqslant L(n,M)}$. Moreover, by Lemma 3.25, for every k, for every $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^{\leqslant k})$, $\mathsf{weight}(\mathsf{tf}) \leqslant \lambda k$. Applying this with $k = L(n, M)$ gives that, for all $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}^*)$, $\mathsf{weight}(\mathsf{tf}) \leqslant \lambda L(n, M) \leqslant B(n, M)$. □

Theorem 3.19 allows us to bound the length of relevant executions of the copycat system:

Corollary 3.27. *For all $\gamma_1 = \langle \mu_1, \ell_1 \rangle, \gamma_2 = \langle \mu_2, \ell_2 \rangle \in \Gamma$ such that $\gamma_1 \overset{*}{\Rightarrow} \gamma_2$, there is an execution of C from γ_1 to γ_2 of length at most $L(n(C), M(C))$.*

Proof. Because $\gamma_1 \overset{*}{\Rightarrow} \gamma_2$, there is $\mathsf{tf} \in \mathcal{T}^*$ such that $\gamma_1 \overset{\mathsf{tf}}{\Rightarrow} \gamma_2$. By Theorem 3.19, we have $\mathsf{tf} \in \mathcal{T}^{\leqslant L(n(C),M(C))}$, so that we obtain the result by applying Lemma 3.14. □

While the exact bounds provided by Theorem 3.19 are not important to us, their dependency with respect to the two parameters will play a role later. Both $L(n(C), M(C))$ and $L(n(C), M(C))$ depend doubly-exponential in $n(C)$, but polynomially in $M(C)$. In other words, the bounds obtained are very large with respect to the number of states $|Q|$, but reasonable with respect to all other metrics of the input.

A natural question is whether the bound above can be improved. It is easy to show that the dependency in the number of states may have to be as large as exponential: for example, one could encode a binary counter over n bits using $2n$ states, using control locations of $\mathcal{L}$ to propagate information from bit to bit. We do not present this construction here. In Section 3.5, we will show an encoding of ASMS into copycat systems, which allows to derive such an exponential lower bound from the PSPACE-hardness proof in [BMRSS16].

However, no example is known where the bound of Theorem 3.19 has to be as large as doubly-exponential in the number of states. It constitutes a major open question whether this bound can be improved to exponential in $|Q|$ or not. As we will see in Section 3.5, this open question is in fact an abstracted version of a long-standing open problem first stated in [BMRSS16], that was wrongly claimed to be solved in [BGW22; BGW23].

3.4 Implications for Copycat Systems

The aim of this section is to illustrate the power of the structural theorem (Theorem 3.19) with a few case studies. Most the results presented in this section consist in generalized versions

of previously known results; we present them here as a way to highlight how the structural theorem allows us to easily obtain decidability of many problems on copycat systems.

3.4.1 K-Blind Sets

We start this section by introducing K-blind sets. A K-blind set is a set that is sensitive to the number of processes only up to threshold K. This notion shares similarities with the one of cubes from [BGW22]. As we will see, many natural sets of configurations of copycat systems can be proved to be K-blind for some threshold K of same order of magnitude as the bounds of Theorem 3.19. In this subsection, we fix a copycat system $\mathcal{C} = (Q, \mathcal{L}, \mathcal{T}_{\min})$.

Definition 3.28. Let $K \in \mathbb{N}$ and let $S \subseteq \Gamma$ be a set of configurations. The set S is *K-blind* when, for every $\langle \mu, \ell \rangle \in \Gamma$, for every $q \in Q$ such that $\mu(q) \geqslant K$, $\langle \mu, \ell \rangle \in S$ if and only if $\langle \mu \oplus q, \ell \rangle \in S$.

Example 3.29. Let $q_0 \in Q$. The set I of configurations $\langle \mu, \ell \rangle$ such that $\mu(q) = 0$ for every $q \neq q_0$ is a 1-blind set. Indeed, it only distinguishes value 0 from values greater than 1. More generally, any set expressed by conditions related to which states are empty and which are not is 1-blind. This would be the case of the sets of configurations expressed by presence constraints from Chapter 2, should we extend this notion to copycat systems.

This definition is robust with respect to Boolean combinations:

Fact 3.30. Let $K \in \mathbb{N}$. If S_1 and S_2 are two K-blind sets of configurations, then so are $\overline{S_1} \coloneqq \Gamma \setminus S_1$, $S_1 \cup S_2$ and $S_1 \cap S_2$.

From now on, we use notation $\overline{S} \coloneqq \Gamma \setminus S$ to write the set complement of S. As in Chapter 2, given a configuration γ, we let $\mathsf{Post}^*(\gamma) \coloneqq \{\gamma' \in \Gamma \mid \gamma \overset{*}{\Rightarrow} \gamma'\}$ and $\mathsf{Pre}^*(\gamma) \coloneqq \{\gamma' \in \Gamma \mid \gamma' \overset{*}{\Rightarrow} \gamma\}$. We extend this notion to sets of configurations in a straightforward manner. We now exploit the structural theorem (Theorem 3.19) to obtain the following result related to K-blind sets.

Proposition 3.31. *Let $K \in \mathbb{N}$, let $S \subseteq \Gamma$ be a K-blind set. The sets $\mathsf{Post}^*(S)$ and $\mathsf{Pre}^*(S)$ are K'-blind with $K' \coloneqq n(\mathcal{C}) \max(K, B(n(\mathcal{C}), M(\mathcal{C})))$.*

Proof. Let $n \coloneqq n(\mathcal{C})$, $M \coloneqq M(\mathcal{C})$. We prove the result for $\mathsf{Post}^*(S)$, the proof for $\mathsf{Pre}^*(S)$ is similar. Let $\gamma = \langle \mu, \ell \rangle \in \Gamma$ and $q \in Q$ such that $\mu(q) \geqslant K'$, let $\gamma' \coloneqq \langle \mu \oplus q, \ell \rangle$. We show that $\gamma \in \mathsf{Post}^*(S)$ if and only if $\gamma' \in \mathsf{Post}^*(S)$. The high-level idea is as follows. The easy direction is to prove that $\gamma \in \mathsf{Post}^*(S)$ implies that $\gamma' \in \mathsf{Post}^*(S)$, because if $\gamma_S \overset{*}{\to} \gamma$ with $\gamma_S \in S$, one can find r such that there are many processes sent from r to q, and simply increase this value by

one. The opposite direction is a bit more complex, as it requires Theorem 3.19 to argue that a process can be removed.

First, suppose that $\gamma = \langle \mu, \ell \rangle \in \mathsf{Post}^*(S)$. Let $\gamma_S = \langle \mu_S, \ell_S \rangle$, $\mathsf{tf} \in \mathcal{T}^*$ such that $\gamma_S \overset{\mathsf{tf}}{\Rightarrow} \gamma$. Let $g : Q^2 \to \mathbb{N}_\#$ be a witness function that $\gamma_S \overset{\mathsf{tf}}{\Rightarrow} \gamma$. We have $\sum_{r \in Q} g(r, q) = \mu(q) \geqslant K'$. By the pigeonhole principle, there is $r \in Q$ such that $g(r, q) \geqslant \frac{K'}{n} \geqslant K$. In particular, we have $\mu_S(r) \geqslant K$. Let $\gamma'_S \coloneqq \langle \mu_S \oplus r, \ell_S \rangle$. Because S is K-blind, $\gamma'_S \in S$; also, $\gamma'_S \overset{\mathsf{tf}}{\Rightarrow} \gamma' = \langle \mu \oplus q, \ell \rangle$ as it suffices to increase $g(r, q)$ by one to obtain a witness function of that. This proves that $\gamma' \in \mathsf{Post}^*(S)$.

Conversely, suppose that $\gamma' = \langle \mu \oplus q, \ell_S \rangle \in \mathsf{Post}^*(S)$. Let $\mathsf{tf} \in \mathcal{T}^*$, $\gamma'_S \in S$ such that $\gamma'_S \overset{\mathsf{tf}}{\Rightarrow} \gamma'$. Thanks to Fact 3.5, without loss of generality, we may consider that $\mathsf{tf} = (f, \ell_S, \ell) \in \mathsf{basis}(\mathcal{T}^*)$. By Theorem 3.19, we have $\mathsf{weight}(\mathsf{tf}) \leqslant B(n, M)$. Let $g : Q^2 \to \mathbb{N}_\#$ be a witness function of $\gamma'_S \overset{\mathsf{tf}}{\Rightarrow} \gamma'$. By hypothesis, $\mu(q) \geqslant K' + 1$, so that $\sum_r g(r, q) \geqslant K' + 1 = n \max(K, B(n, M)) + 1$. By the pigeonhole principle, there exists $r \in Q$ for which $g(r, q) \geqslant \max(K, B(n, M)) + 1$. Let $h : Q^2 \to \mathbb{N}_\#$ equal to g except that $h(r, q) \coloneqq g(r, q) - 1$. We have $f \leqslant g$ so that $f(q_1, q_2) \leqslant h(q_1, q_2)$ for all $(q_1, q_2) \neq (r, q)$. Also, $g(r, q) \geqslant B(n, M) + 1$ so that $h(r, q) \geqslant B(n, M)$ which proves that $f(r, q) \leqslant \mathsf{weight}(\mathsf{tf}) \leqslant B(n, M) \leqslant h(r, q)$. Therefore, $f \leqslant h$, so that h is a witness function that $\gamma_S \overset{\mathsf{tf}}{\Rightarrow} \gamma$ where $\gamma_S = \langle \mu_S \ominus r, \ell_S \rangle$. Moreover, $\mu_S(r) = \sum_{q'} g(r, q') > K$ and, because S is K-blind, $\gamma_S \in S$. Because $\gamma_S \overset{*}{\Rightarrow} \gamma$, $\gamma \in \mathsf{Post}^*(S)$. □

The bound from Proposition 3.31 yields naive decision procedures that consist in a non-deterministic exploration of the space of configurations of a given, bounded size. For example, a generalized version of the presence reachability problem from Chapter 2 asks, given two 1-blind sets I and F (for example described as presence constraints), whether $\mathsf{Post}^*(I) \cap F \neq \emptyset$. Thanks to Proposition 3.31, we know that $\mathsf{Post}^*(I) \cap F$ is a K-blind set with $K \coloneqq n(C)B(n(C), M(C))$. Therefore, to decide emptiness of this set, we can look for a witness execution whose configurations have size at most K; by storing configurations in binary and guessing the execution configuration by configuration, this can be decided in non-deterministic space exponential in $|Q|$ and polynomial in the rest of the input.

This can be extended to expressions obtained by using K-blind sets as basic blocks and applying Boolean operations and Post^* and Pre^* operators. This leads us to define *generalized reachability expressions* [Wei23] as the expressions produced by the grammar:

$$E ::= \phi \mid E \cup E \mid E \cap E \mid \overline{E} \mid \mathsf{Post}^*(E) \mid \mathsf{Pre}^*(E)$$

where $\overline{E}$ denotes the set complement of E and ϕ denotes *basic predicates*. These basic predicates range over a family of constraints; a basic predicate defines a set of configuration. To define generalized reachability expressions, one must choose what family of basic predicates is allowed. Possible such families could be, *e.g.*, presence constraints from Chapter 2 or so-called cubes from [BGW22]. For us, the exact choice of the family of basic predicates is not really important, as long as they define, in an effective way, K-blind sets for a reasonable K.

In order to illustrate some techniques related to generalized reachability expressions, we make an arbitrary choice on the family of basic predicates allowed. To save us some definitions, we choose a very simple family: we consider that ϕ is allowed to take the form of a pair (S, ℓ) with $S \subseteq Q$, and that the set described is $\{\langle \mu, \ell' \rangle \mid \forall q \notin S, \mu(q) = 0\}$. Given a GRE E, we denote by $[\![E]\!] \subseteq \Gamma$ the set of configurations that it defines.

Example 3.32. Consider again the copycat system $C = (\{q_1, q_2\}, \{\ell_a, \ell_b\}, \{t_{ab}, t_{ba}\})$ from Example 3.17. Let ϕ_a be the predicate that describes the set $\Gamma \times \{\ell_a\}$; let $E \coloneqq \mathsf{Post}^*(\phi_a) \setminus \mathsf{Pre}^*(\phi_a)$. We have that $[\![E]\!] = \emptyset$. In words, for every γ with control location ℓ_a, for every γ' reachable from γ, there is an execution from γ' to a configuration in $[\![\phi_a]\!]$, *i.e.*, a configuration with control location ℓ_a. Indeed, if $\gamma' = \langle \mu', \ell_a \rangle$ then it is trivially in $[\![\phi_a]\!]$; if $\gamma' = \langle \mu', \ell_b \rangle$ then the execution from γ to γ' ends with t_{ab}, so that $\mu'(q_2) \geqslant 2$ and one may apply t_{ba} from γ', proving that $\gamma' \in \mathsf{Pre}^*([\![\phi_a]\!])$.

We can now define the *GRE emptiness problem*:

GRE EMPTINESS PROBLEM FOR COPYCAT SYSTEMS
Input: A copycat system C, a GRE E
Question: Is $[\![E]\!]$ empty?

The size of an instance (C, E) is defined by $n(C) + M(C) + |E|$ where $|E|$ is the number of operators of E.

Lemma 3.33. *The following problem is in PSPACE: given a copycat system C, a GRE E over $\mathcal{P}$ and a configuration γ encoded in binary, is γ in $[\![E]\!]$?*

Proof. This proof follows the same procedure used to check GRE membership in [Wei23] and [BGKMWW24]. Let $n \coloneqq n(C)$, $M \coloneqq M(C)$ and $N \coloneqq |\gamma|$. Observe that a configuration of size N can be stored in space polynomial in M and n and logarithmic in N using binary encoding. Thanks to Savitch's theorem, we may allow for non-deterministic choices. Using a non-deterministic exploration of the configuration space, we can decide, given γ_1 and γ_2 of size N, whether $\gamma_1 \overset{*}{\Rightarrow} \gamma_2$ in space polynomial in $M(C)$ and $n(C)$ and logarithmic in N.

We prove Lemma 3.33 by structural induction on E. Suppose that, for every E' appearing in E, we have a polynomial-space procedure to decide membership of configurations of size N. To check that $\gamma \in [\![E]\!]$, we operate a case disjunction on E:

— if $E = \phi$ then it is easy to check whether $\gamma \in [\![\phi]\!]$;

— if $E = \mathsf{Post}^*(E')$ then we guess $\gamma' \in E'$ of same size as γ, and we check that $\gamma \overset{*}{\Rightarrow} \gamma'$ and that $\gamma' \in [\![E']\!]$;

— if $E = \mathsf{Pre}^*(E')$ then we guess $\gamma' \in E'$ of same size as γ, and we check that $\gamma' \overset{*}{\Rightarrow} \gamma$ and that $\gamma' \in [\![E']\!]$;

— if $E = E_1 \cup E_2$ then we check whether $\gamma \in [\![E_1]\!]$ and whether $\gamma \in [\![E_2]\!]$, and accept if one of these holds;

— if $E = E_1 \cap E_2$ then we check whether $\gamma \in [\![E_1]\!]$ and whether $\gamma \in [\![E_2]\!]$, and accept if both hold;

— if $E = \overline{E'}$ then we check whether $\gamma \in [\![E']\!]$ and negate the answer.

This concludes the proof. □

We now give decidability and a complexity upper bound for GRE emptiness:

Proposition 3.34. *The GRE emptiness problem is in EXPSPACE.*

Proof. We want to apply Lemma 3.33; to do that, we need a doubly-exponential bound in the configuration γ to consider. Let (C, E) be an instance of GRE emptiness, let $n \coloneqq n(C)$ and $M \coloneqq M(C)$. Let k denote the highest number of nested Post^* and Pre^* operators in E. By iterative applications of Proposition 3.31, the set $[\![E]\!]$ is K-blind for $K \coloneqq n^k B(n, M)$. Therefore, $[\![E]\!] \neq \emptyset$ if and only if it contains some configuration γ of size at most K. It therefore suffices to guess γ of size at most K and to apply the membership procedure from Lemma 3.33. This procedure works in space polynomial in $\log(K)$ because of binary encoding. Because K is doubly-exponential in the size of the input, this procedure uses exponential space. □

We will now provide another case study to illustrate the power of the structural theorem: verification of linear-time properties in copycat systems.

3.4.2 LTL for Copycat Systems

We start by defining *linear-time logic*, or LTL for short [Pnu77]. An LTL formula expresses properties over infinite words, *i.e.*, words in Σ^ω.

Definition 3.35. LTL formulas are defined by the following grammar:

$$\psi ::= x \mid \psi \vee \psi \mid \psi \wedge \psi \mid \neg\psi \mid \mathsf{X}\,\psi \mid \mathsf{G}\,\psi \mid \mathsf{F}\,\psi \mid \psi\,\mathsf{U}\,\psi$$

where $x \in \Sigma$ is a symbol. LTL formula are interpreted over $\Sigma^\omega \times \mathbb{N}$ as follows:

(w,i)	$\models x$	iff	$w(i) = x$
(w,i)	$\models \psi_1 \vee \psi_2$	iff	$(w,i) \models \psi_1$ or $(w,i) \models \psi_2$
(w,i)	$\models \psi_1 \wedge \psi_2$	iff	$(w,i) \models \psi_1$ and $(w,i) \models \psi_2$
(w,i)	$\models \neg\psi$	iff	$(w,i) \not\models \psi$
(w,i)	$\models \mathsf{X}\,\psi$	iff	$(w,i+1) \models \psi$
(w,i)	$\models \mathsf{G}\,\psi$	iff	for all $j \geqslant 0$ $(w,i+j) \models \psi$
(w,i)	$\models \mathsf{F}\,\psi$	iff	for some $j \geqslant 0$ $(w,i+j) \models \psi$
(w,i)	$\models \psi_1\,\mathsf{U}\,\psi_2$	iff	for some $j \geqslant 0$ $(w,i+j) \models \psi_2$ and for all $k < j$, $(w,i+k) \models \psi_1$.

The language $\mathcal{L}(\psi)$ of an LTL formula ψ is defined as the set of words $w \in \psi^\omega$ such that $(w,0) \models \psi$.

In this thesis, we will not work directly with LTL formula, but rather with automata. Given a finite set Σ, a *deterministic Rabin automaton* over Σ is a tuple $\mathcal{A} = (\mathcal{R}, \Delta, r_0, \Omega)$, where $\mathcal{R}$ is the finite set of states, r_0 is the initial state, $\Delta : \mathcal{R} \times \Sigma \to \mathcal{R}$ is the transition function and $\Omega \subseteq 2^{\mathcal{R}} \times 2^{\mathcal{R}}$ is a finite set of *Rabin pairs* of the form (F,G) with $F, G \subseteq \mathcal{R}$. Given an infinite word $w \in \Sigma^\omega$, the *run* of $\mathcal{A}$ reading w is the sequence of states $r_0, r_1, r_2, \ldots$ such that, for all $i \in \mathbb{N}$, $\Delta(r_i, w(i)) = r_{i+1}$. A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists a pair $(F,G) \in \Omega$ such that the run of $\mathcal{A}$ reading w visits F finitely often and G infinitely often. The language $\mathcal{L}(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. The *size* of $\mathcal{A}$ is $|\mathcal{A}| := |\mathcal{R}|$.

Theorem 3.36 (see, *e.g.*, [EKS18])**.** *Given a finite set Σ and an LTL formula ψ over Σ, there is a deterministic Rabin automaton $\mathcal{A}_\psi$ over Σ that accepts the same language as ψ. Moreover $\mathcal{A}_\psi$ has number of states at most doubly-exponential in $|\psi|$, and can be computed in space doubly-exponential in $|\psi|$.*

Let $C = (Q, \mathcal{L}, \mathcal{T}_{\min})$ be a copycat system. Our aim is to define LTL formulas over executions of C. For this purpose, we consider infinite executions; an *infinite execution* is a sequence $\gamma_0, t_1, \gamma_1, \ldots$ such that, for all $i \in \mathbb{N}$, $\gamma_i \xRightarrow{t_{i+1}} \gamma_{i+1}$ and $t_{i+1} \in \mathcal{T}$. We also need a notion of initial configuration of a copycat system. To do that, we define a notion of initial copycat system. An *initial copycat system* is a tuple $C = (Q, \mathcal{L}, \mathcal{T}_{\min}, q_0, \ell_0)$ where $(Q, \mathcal{L}, \mathcal{T}_{\min})$ is a copycat system,

$q_0 \in Q$ and $\ell_0 \in \mathcal{L}$. The set of initial configurations of an initial copycat system is defined by

$$\mathcal{I}(C) \coloneqq \{\langle \mu_0, \ell_0 \rangle \mid \forall q \neq q_0,\ \mu_0(q) = 0\}.$$

An infinite execution of C is called *initial* when its first configuration is in $\mathcal{I}(C)$. We assume that there is no deadlock, *i.e.*, that from every γ, there is $t \in \mathcal{T}_{\min}$ and γ' such that $\gamma \overset{t}{\Rightarrow} \gamma'$; this can be enforced by adding to $\mathcal{T}_{\min}$ transfer flows of $\mathsf{basis}(\mathcal{F}_0)$, so that there is always $t \in \mathcal{T}_{\min}$ such that $\gamma \overset{t}{\Rightarrow} \gamma$.

We interpret LTL formulas over infinite executions as follows. Let ψ be an LTL formula over $\mathcal{T}_{\min}$ and $\rho = \gamma_0, t_1, \dots$ be an infinite execution. Let $w \in \mathcal{T}_{\min}^{\omega}$ such that $w(i) \coloneqq t_i$ for all $i \in \mathbb{N}$; ψ *accepts* ρ when $(w, 0) \models \psi$. We abuse notation and denote this by $\rho \models \psi$.

When considering infinite executions, it is common to enforce an additional constraint to ensure that the execution is reasonable. This takes the form of a *fairness condition*. A typical fairness condition, called here *weak fairness*, is that any step available infinitely often appears infinitely often. Formally, an infinite execution $\gamma_0, t_1, \gamma_1, \dots$ is *weakly fair* when it is initial and when, for every $\gamma \in \Gamma$, for every $\gamma' \in \mathsf{Post}^*(\gamma)$, if $\gamma = \gamma_i$ for infinitely many $i \in \mathbb{N}$, for every $t \in \mathcal{T}_{\min}$ then for every $\gamma' \in \Gamma$ such that $\gamma \overset{t}{\Rightarrow} \gamma'$, there are infinitely many i such that $\gamma_i = \gamma$, $t_{i+1} = t$ and $\gamma_{i+1} = \gamma'$. This notion of fairness is however too weak for fairness verification, as argued in Section 3.4.3.

For this reason, we define a more restrictive notion of fairness, called *strong fairness*. Given an infinite execution $\rho = \gamma_0, t_1, \gamma_1, \dots$ and a finite execution $\rho' = \gamma'_0, t'_1, \dots, t'_k, \gamma'_k$, we say that ρ' *appears infinitely often* in ρ when there are infinitely many indices i such that $\gamma_{i+j} = \gamma'_j$ for all $j \in [\![0, k]\!]$ and $t_{i+j} = t'_j$ for all $j \in [\![1, k]\!]$. An infinite execution is *strongly fair* when it is initial and, for every finite execution $\rho' = \gamma'_0, \delta_1, \dots, \gamma'_k$, if γ_0 is visited infinitely often in ρ then ρ' appears infinitely often in ρ. This definition is harder to grasp, because it involves a universal quantification over an infinite (but countable) set, namely the set of finite executions. An argument to justify that this definition is sound is that, under a reasonable stochastic scheduler, the execution selected will be strongly fair with probability 1. By *reasonable*, we mean that the stochastic scheduler is memoryless (the choice only depends on the current configuration) and gives non-zero probability to all possible candidates for the next step.Under such a scheduler, given a finite execution $\rho' = \gamma'_0, \delta_1, \dots, \gamma'_k$, whenever γ'_0 is the current configuration, there is a fixed non-zero probability that the next k steps correspond to ρ', regardless of the past. If γ'_0 is visited infinitely often, we have infinitely many independent repetitions of this experiment and ρ' appears infinitely often. Because the set of finite executions is countable, by summing over

this set, the probability of existence of a finite execution disproving strong fairness is equal to 0, so that the drawn execution is strongly fair with probability 1.

This allows us to define the LTL verification problem for copycat systems. There are two versions of this problem, depending on whether (strong) fairness is enforced or not.

LTL verification without Fairness
Input: An initial copycat system $C = (Q, \mathcal{L}, \mathcal{T}_{\min}, q_0, \ell_0)$, an LTL formula ψ over $\mathcal{T}_{\min}$
Question: Does there exist an initial execution ρ of C such that $\rho \models \psi$?

LTL verification with (strong) Fairness
Input: An initial copycat system $C = (Q, \mathcal{L}, \mathcal{T}_{\min}, q_0, \ell_0)$, an LTL formula ψ over $\mathcal{T}_{\min}$
Question: Does there exist a strongly fair execution ρ of C such that $\rho \models \psi$?

3.4.3 Strong Fairness and Stochastic Schedulers

Our decision to rely on strong fairness instead of on the traditional, weak fairness may seem surprising. It is even more so given that a prior work on LTL verification for population protocols (a parameterized model) [EGLM16] considers weak fairness for LTL verification. To do so, the authors relate weak fairness to stochastic schedulers. More precisely, they claim that there is a weakly fair execution satisfying an LTL formula if and only if, under a reasonable stochastic scheduler, there is a non-zero probability that the LTL formula is satisfied. However, this claim is incorrect, both for population protocols and in our model.

The high-level intuition is that weak fairness allows for regular patters, whereas a stochastic scheduler does not. We highlight this on an example [2]. Suppose that $\mathcal{T}_{\min} = \{t_1, t_2, t_3, t_4\}$ and that we have three distinct configurations γ_1, γ_2, γ_3 (with γ_1 initial) such that $\gamma_1 \overset{t_1}{\Longrightarrow} \gamma_2$, $\gamma_2 \overset{t_2}{\Longrightarrow} \gamma_1$, $\gamma_1 \overset{t_3}{\Longrightarrow} \gamma_3$ and $\gamma_3 \overset{t_4}{\Longrightarrow} \gamma_1$. Also, suppose that these are the only possible steps from each of these configurations. Consider $\psi \coloneqq \neg \mathsf{F}\,(t_1 \wedge (\mathsf{X}\, t_2) \wedge (\mathsf{X}^2 t_1) \wedge (\mathsf{X}^3 t_2))$, which expresses that sequence of transitions $t_1\, t_2\, t_1\, t_2$ does not appear in the execution. Under a reasonable stochastic scheduler, from γ_1, ψ is satisfied with probability 0. However, there is a weakly fair execution satisfying ψ: it suffices to repeat infinitely $\gamma_1 \overset{t_1}{\Longrightarrow} \gamma_2 \overset{t_2}{\Longrightarrow} \gamma_1 \overset{t_3}{\Longrightarrow} \gamma_3 \overset{t_4}{\Longrightarrow} \gamma_1$.

Let $C = (Q, \mathcal{L}, \mathcal{T}_{\min}, q_0, \ell_0)$ be an initial copycat system and let ψ be an LTL formula over $\mathcal{T}_{\min}$. Thanks to Theorem 3.36, we build a Rabin automaton $\mathcal{A} = (\mathcal{R}, \Delta, r_0, \Omega)$ over $\mathcal{T}_{\min}$ that accepts $\mathcal{L}(\psi)$. Given $t = (f, \ell_1, \ell_2) \in \mathcal{T}_{\min}$ and $r \in \mathcal{R}$, we define $\mathsf{tr}(t, r) \coloneqq (f, (\ell_1, r), (\ell_2, \Delta(r, t)))$.

2. For simplicity, we keep this example abstract. It is not hard to build an actual copycat system or population protocol that implements this example.

We build a new initial copycat system

$$C_\psi \coloneqq (Q, \mathcal{L} \times \mathcal{R}, \mathcal{T}_{\min,\psi}, q_0, (\ell_0, r_0)) \text{ with } \mathcal{T}_{\min,\psi} \coloneqq \{\mathsf{tr}(t, r) \mid t \in \mathcal{T}_{\min}, r \in \mathcal{R}\}.$$

Intuitively, C_ψ consists in C where the Rabin automaton $\mathcal{A}$ has been implemented using control locations in $\mathcal{L}$; this Rabin automaton takes as input the transitions applied in C. We denote by $\Gamma_\psi \coloneqq \mathcal{M}(Q) \times (\mathcal{L} \times \mathcal{R})$ the set of configurations of C_ψ. For simplicity, we denote by (γ, r) the configuration $\langle \mu, (\ell, r)\rangle \in \Gamma_\psi$ where $\gamma = \langle \mu, \ell\rangle \in \Gamma$. In order to distinguish configurations of C_ψ and of C, we use symbol c for configurations of C_ψ. Also, we abuse notations and write $(\gamma, r) \overset{t}{\Rightarrow} (\gamma', r')$ instead of $(\gamma, r) \overset{\mathsf{tr}(t,r)}{\Longrightarrow} (\gamma', r')$. For every $R \subseteq \mathcal{R}$, we write $\Gamma_\psi(R) \coloneqq \{(\gamma, r) \in \Gamma_\psi \mid \gamma \in \Gamma, r \in R\}$ for the set of configurations of C_ψ whose Rabin state is in R. Also, we let $\mathcal{G}(C_\psi)$ denote the graph whose vertices are configurations of C_ψ and whose edges correspond to reachability in one step in C_ψ, *i.e.*, given $c, c' \in C_\psi$, there is an edge from c to c' whenever there is $t \in \mathcal{T}_{\min}$ such that $c \overset{t}{\Rightarrow} c'$. A set of vertices S of $\mathcal{G}(C_\psi)$ is called *reachable from* c when there is a path from c to some vertex in S. Also, S is called *winning* when there is $(F, G) \in \Omega$ such that $S \cap \Gamma_F = \emptyset$ but $S \cap \Gamma_G \neq \emptyset$. For every (finite or infinite) execution $\rho_C = \gamma_0, t_1, \gamma_1, \dots$ of C and every $r \in \mathcal{R}$, we let $\mathsf{Ex}_r(\rho_C) \coloneqq (\gamma_0, r), t_1, (\gamma_1, r_1), \dots$ be the (finite or infinite) execution of C_ψ where $r_1 = \Delta(r, t_1)$ and $r_{i+1} = \Delta(r_i, t_{i+1})$ for all i. In words, $\mathsf{Ex}_r(\rho_C)$ is the execution of C_ψ corresponding to performing ρ starting with Rabin state r. From an execution ρ of C_ψ starting from (γ_0, r), we let $\mathsf{Ex}^{-1}(\rho)$ be the unique execution ρ_C of C such that $\rho = \mathsf{Ex}_r(\rho_C)$.

We call an execution ρ of C_ψ *system-fair* when the corresponding execution $\mathsf{Ex}^{-1}(\rho)$ of C is strongly fair. Note that the system-fairness condition is only related to the first component of configurations of C_ψ. For example, if a system-fair execution ρ of C_ψ visits infinitely many times some configuration (γ, r_1) from which there exists a step $(\gamma, r_1) \overset{t}{\Rightarrow} (\gamma', r_2)$, fairness guarantees the existence of r_3, r_4 such that $(\gamma, r_3) \overset{t}{\Rightarrow} (\gamma', r_4)$ appears infinitely often in ρ, but *a priori* does not guarantee that the step $(\gamma, r_1) \overset{t}{\Rightarrow} (\gamma', r_2)$ ever appears. This is in fact the mistake made in [EGLM16, Proposition 7]. However, because we rely on strong fairness, system-fair executions satisfy a property that is sufficient for our needs.

Lemma 3.37. *Any system-fair execution of C_ψ ends in a bottom SCC S of $\mathcal{G}(C_\psi)$ and visits infinitely many times every configuration in S.*

Proof. Note that, because steps in C_ψ may only connect configurations of same size (*i.e.*, same number of processes), all SCC of $\mathcal{G}(C_\psi)$ are finite. Let ρ be a system-fair execution of C_ψ,

there is a unique SCC S visited infinitely many times by ρ. Let $\rho_C \coloneqq \mathsf{Ex}^{-1}(\rho)$ denote the corresponding execution of C; ρ_C is strongly fair. We first prove that S is bottom.

By contradiction, suppose that S is not bottom. There is $t \in \mathcal{T}_{\min}$, $(\gamma_a, r_a) \in S, (\gamma_b, r_b) \notin S$ such that $(\gamma_a, r_a) \overset{t}{\Rightarrow} (\gamma_b, r_b)$. Let $C_a \coloneqq S \cap (\{\gamma_a\} \times \mathcal{R})$. Trivially, ρ is infinite hence visits infinitely often some configuration $(\gamma, r) \in S$. There is a finite execution of C from γ to γ_a hence, by strong fairness of ρ_C, γ_a is visited infinitely often in ρ_C, so that C_a is visited infinitely often in ρ.

Let $C'_a \coloneqq \{(\gamma_a, r) \in C_a \mid (\gamma_b, \Delta(r, t)) \in S\}$; C'_a contains configurations $(\gamma_a, r) \in C_a$ from which the step $(\gamma_a, r) \overset{t}{\Rightarrow} (\gamma_b, r')$ makes us stay in S. Let $C'_a =: \{(\gamma_a, r_1), \dots, (\gamma_a, r_m)\}$ with $m = |C'_a|$. Our aim is to build a finite execution σ of C from γ_a such that, for all $i \in [\![1, m]\!]$, the execution $\mathsf{Ex}_{r_i}(\sigma)$ leaves S; said otherwise, performing σ from any configuration in C'_a takes us out of S. For all $i \in [\![1, m]\!]$, we build a finite execution σ_i of C from γ_a to γ_a as follows. First, we let σ_0 be the execution from γ_a of length 0. Let $i < m$ and suppose that σ_i is constructed; let (γ_a, s_{i+1}) be the last configuration of $\mathsf{Ex}_{r_{i+1}}(\sigma_i)$, *i.e.*, the configuration obtained when translating σ_i to an execution of C_ψ from (γ_a, r_{i+1}). If $(\gamma_a, s_{i+1}) \notin C'_a$ then we let $\sigma_{i+1} = \sigma_i$. If $(\gamma_a, s_{i+1}) \in C'_a$, by letting $s'_{i+1} \coloneqq \Delta(s_{i+1}, t)$, we have $(\gamma_a, s_{i+1}) \overset{t}{\Rightarrow} (\gamma_b, s'_{i+1})$ and $(\gamma_b, s'_{i+1}) \in S$. Because S is strongly connected, there is an execution π_{i+1} of C_ψ from (γ_b, s'_{i+1}) to (γ_a, r_a). Let $\tau_{i+1} \coloneqq \mathsf{Ex}^{-1}(\pi_{i+1})$ be the corresponding execution of C from γ_b to γ_a. We let σ_{i+1} denote the finite execution of C from γ_a to γ_a that consists in following σ_i, then the step $\gamma_a \overset{t}{\Rightarrow} \gamma_b$ and finally τ_{i+1}. This concludes the induction step. Finally, we let σ be the finite execution of C obtained by following σ_m then the step $\gamma_a \overset{t}{\Rightarrow} \gamma_b$.

For all $j \in [\![1, m]\!]$, let $\rho_j \coloneqq \mathsf{Ex}_{r_j}(\sigma)$; ρ_j corresponds to performing σ from (γ_a, r_j). By strong fairness of ρ_C, σ appears infinitely often in ρ_C. Because σ starts with step $\gamma_a \overset{t}{\Rightarrow} \gamma_b$, the corresponding steps in ρ must be from C'_a as ρ would otherwise leave S. Therefore, there is $j \in [\![1, m]\!]$ such that ρ_j appears infinitely often in ρ. Because ρ does not leave S, it must be that ρ_j remains in S. Consider step j of the construction above. First, ρ_j follows $\mathsf{Ex}_{r_j}(\sigma_{j-1})$ and gets to (γ_a, s_j). If $(\gamma_a, s_j) \notin C'_a$ then $j > 0$ and we have set $\sigma_j = \sigma_{j-1}$; the next step in ρ_j is then $(\gamma_a, s_j) \overset{t}{\Rightarrow} (\gamma_b, r)$ where r is such that $(\gamma_b, r) \notin S$ since $(\gamma_a, s_j) \notin C'_a$: ρ_j leaves S, a contradiction. Suppose now that $(\gamma_a, s_j) \in C'_a$. In this case, by construction of σ, ρ_j follows $\mathsf{Ex}_{r_j}(\sigma_{j-1})$ to (γ_a, s_j), performs step $(\gamma_a, s_{i+1}) \overset{t}{\Rightarrow} (\gamma_b, s'_{i+1})$, follows π_{j+1} while leads to (γ_a, r_a) from where ρ_j performs step $(\gamma_a, r_a) \overset{t}{\Rightarrow} (\gamma_b, r_b)$. This last step makes ρ_j leave S, a contradiction.

We have proven that ρ eventually leaves any non-bottom SCC, so that the SCC S visited

infinitely often in ρ is bottom. We now must prove that ρ visits all configurations in S infinitely often. Suppose by contradiction that there are $(\gamma_a, r_a), (\gamma_b, r_b) \in S$ and $t \in \mathcal{T}_{\min}$ such that $(\gamma_a, r_a) \overset{t}{\Rightarrow} (\gamma_b, r_b)$ and such that ρ visits (γ_a, r_a) infinitely often but (γ_b, r_b) finitely often. We proceed similarly to above, except that now $C'_a = C_a$, which makes this case easier. Let $C_a = S \cap (\{\gamma_a\} \times \mathcal{R}) = \{(\gamma_a, r_1), \dots, (\gamma_a, r_m)\}$ with $m = |C_a| > 0$. We construct σ as above, except that the case $(\gamma_a, s_j) \notin C'_a$ may no longer happen since $C'_a = C_a$. Again, for all $j \in [\![1, m]\!]$, we let ρ_j be the finite execution of C_ψ that corresponds to performing σ from (γ_a, r_j). By strong fairness of ρ_C, there is j such that ρ_j appears infinitely often in ρ. We then argue that ρ_j visits (γ_b, r_b). Again, the reasoning is the same as above: once at (γ_a, s_j), by construction, we visit (γ_a, r_a) from where the next step makes ρ_j visit (γ_b, r_b). We have proven that, if $(\gamma_a, r_a) \overset{t}{\Rightarrow} (\gamma_b, r_b)$ and (γ_a, r_a) is visited infinitely many times in ρ then the same is true for (γ_b, r_b). By direct induction, ρ visits infinitely often each configuration in S. □

We argue that strong fairness, unlike weak fairness, is equivalent to stochastic schedulers for LTL verification. This justifies our choice to consider strong fairness. We fix a reasonable stochastic scheduler for C. Given a configuration γ, we denote by $\mathbb{P}_\gamma(\rho \models \psi)$ the probability that an execution from γ satisfies ψ. We may now express that strong fairness is equivalent for LTL to a stochastic scheduler:

Proposition 3.38. *Let $\gamma_s \in \Gamma$. The following are equivalent:*

(i) there is a strongly fair execution ρ from γ_s such that $\rho \models \psi$;

(ii) $\mathbb{P}_{\gamma_s}(\rho \models \psi) > 0$;

(iii) there is a winning bottom SCC S of $\mathcal{G}(C_\psi)$ that is reachable from (γ_s, r_0).

Proof. We follow the same proof scheme as in [EGLM16, Proposition 7]. The equivalence between (ii) and (iii) is fairly easy to prove using the same proof as [EGLM16, Proposition 6]. It is a direct consequence of the fact that, under a reasonable stochastic scheduler, there is probability one that the execution of C_ψ ends in a bottom SCC of $\mathcal{G}(C_\psi)$ and visits all configurations in this bottom SCC infinitely many times.

We now prove the equivalence between (i) and (iii). We first prove that (iii) implies (i). Suppose that there is such an SCC S. Under a reasonable stochastic scheduler, there is probability one that the obtained execution of C_ψ from (γ_s, r_0) is system-fair, ends in a bottom SCC of $\mathcal{G}(C_\psi)$ and visits all configurations in the SCC infinitely often; there is a non-zero probability that this SCC is S (because S is reachable from (γ_s, r_0)) hence that the corresponding execution of C satisfies ψ. The fact that the stochastic scheduler, with non-zero probability, selects a strongly

fair execution from γ satisfying ψ implies the existence of such an execution, which proves (i). For the converse implication, suppose that there is a strongly fair execution ρ of C such that $\rho \models \psi$. Let $\rho' \coloneqq \mathsf{Ex}_{r_0}(\rho)$ denote the corresponding execution of C_ψ; trivially, $\rho \models \psi$ implies the existence of $(F, G) \in \Omega$ such that ρ' visits $\Gamma_\psi(F)$ finitely often and $\Gamma_\psi(G)$ infinitely often. By Lemma 3.37, we have that the SCC S visited infinitely often in ρ' is bottom and that ρ' visits infinitely often each configuration in S; this implies that $S \cap \Gamma_\psi(F) = \emptyset$ and that $S \cap \Gamma_\psi(G) \neq \emptyset$, which concludes the proof. □

Note that the proof above does not use many features copycat system, and in particular does not use the copycat property. Therefore, it could easily apply to other parameterized models where processes are finite-state and their number is preserved in an execution, by considering the larger system obtained by putting side by side the original system and the Rabin automaton. In particular, it applies to population protocols, so that strong fairness fixes the mistake from [EGLM16].

3.4.4 Verification of LTL for Copycat Systems

This subsection is devoted to proving the following theorem:

Theorem 3.39. *The LTL verification problems without fairness and with fairness are in* 2-*EXPSPACE*.

We start by considering the problem without fairness, which is equivalent to asking for the existence of an initial execution ρ of C_ψ such that, for some $(F, G) \in \Omega$, ρ visits $\Gamma_\psi(F)$ finitely often but $\Gamma_\psi(G)$ infinitely often. Our aim is to prove that, when looking for such an execution, one may consider configurations of bounded size. Let $C'_\psi = (Q, \mathcal{L}', \mathcal{T}'_{\min,\psi})$ be the copycat system obtained from C_ψ by removing all elements of $\mathcal{L}$ and $\mathcal{T}_{\min,\psi}$ involving states in F. Formally, $\mathcal{L}' \coloneqq \mathcal{L} \times (\mathcal{R} \setminus F)$ and $\mathcal{T}'_{\min,\psi} \coloneqq \mathcal{T}_{\min,\psi} \setminus \{(f, (\ell, r), (\ell', r')) \mid r \in F \ \vee \ r' \in F\}$. The upward-closure of $\mathcal{T}'_{\min,\psi}$ is denoted $\mathcal{T}'_\psi$. Observe that the set of configurations of C'_ψ is $\Gamma'_\psi = \Gamma_\psi \setminus \Gamma_\psi(F)$.

Lemma 3.40. *Let $n \coloneqq n(C_\psi)$, $M \coloneqq M(C_\psi)$ and $\lambda \coloneqq \max_{t \in \mathcal{T}_{\min,\psi}} \mathsf{weight}(t)$. Let $N \coloneqq \lambda + n^n B(n, M)$. The instance (C, ψ) of LTL verification without fairness is positive if and only if there is $c \in \mathsf{Post}^*(\mathcal{I}(C_\psi)) \cap (\Gamma_\psi(G) \setminus \Gamma_\psi(F))$ of size at most N and $\mathsf{tf} \in \mathcal{T}'^*_{\min,\psi} \otimes (\mathcal{T}'_\psi)^*$ such that $c \overset{\mathsf{tf}}{\Longrightarrow} c$.*

Proof. We know that (C, ψ) is positive if and only if there exist $(F, G) \in \Omega$ and $\rho = c_0, t_1, c_1, t_2 \dots$ an initial execution of C_ψ that visits $\Gamma_\psi(F)$ finitely often but $\Gamma_\psi(G)$ infinitely often. If suffices to prove that the existence of ρ above is equivalent to the existence of c that satisfies the conditions of the lemma. We start by omitting the condition on the size of c, which we will treat afterwards. First, assume that we have such a configuration c, and let $t_1, \dots, t_k \in \mathcal{T}'_{\min,\psi}$, $k > 0$, such that there is an execution ρ_f from c to c whose sequence of transitions is $t_1, \dots, t_k$. We build an initial execution ρ that starts with a prefix execution from $c_0 \in \mathcal{I}(C_\psi)$ to c then repeats infinitely many times the finite execution ρ_f. By hypothesis, $c \in \Gamma_\psi(G)$ and ρ_f does not visit $\Gamma_\psi(F)$, so that ρ visits $\Gamma_\psi(F)$ finitely many times but $\Gamma_\psi(G)$ infinitely many times.

Conversely, suppose that we have $\rho = c_0, t_1, c_1, t_2 \dots$ an initial execution of C_ψ that visits $\Gamma_\psi(F)$ finitely often but $\Gamma_\psi(G)$ infinitely many times. Recall that all configurations in ρ have the same size, so that the set of configurations from $\Gamma_\psi(G)$ visited in ρ is finite. By the pigeonhole principle, there is $c \in \Gamma_\psi(G)$ that is visited infinitely many times. There is k such that $c_k = c$ and, for all $i \geqslant k$, $c_i \notin \Gamma_\psi(F)$. Moreover, there is $j > i$ such that $c_j = c$, which proves that there is a non-trivial execution from c to c in C'_ψ. This implies that there is $\mathsf{tf} \in \mathcal{T}'_\psi \otimes (\mathcal{T}'_\psi)^*$ such that $c \overset{\mathsf{tf}}{\Longrightarrow} c$. By Fact 3.5, we may assume that $\mathsf{tf} \in \mathsf{basis}(\mathcal{T}'_\psi \otimes (\mathcal{T}'_\psi)^*) \subseteq \mathcal{T}'_{\min,\psi} \otimes \mathsf{basis}((\mathcal{T}'_\psi)^*)$.

We now prove that we may assume that c has size at most N. Suppose that we have c of size strictly greater than N; we show that we can find c' that satisfies the conditions of the lemma and with $|c'| < |c|$. Let $\mathsf{tf} \in \mathcal{T}'_{\min,\psi} \otimes \mathsf{basis}((\mathcal{T}'_\psi)^*)$ such that $c \overset{\mathsf{tf}}{\Longrightarrow} c$. By definition, all transfer flows in $\mathcal{T}'_{\min,\psi}$ have weight bounded by λ, and by Theorem 3.19 applied in C'_ψ, all transfer flows in $\mathsf{basis}((\mathcal{T}'_\psi)^*)$ have weight bounded by $B(n(C_{\psi'}), M(C'_\psi)) \leqslant B(n, M)$. By considering tf minimal, we obtain by Lemma 3.12 that $\mathsf{weight}(\mathsf{tf}) \leqslant \lambda + B(n, M)$. Let $\langle \mu, \ell \rangle \coloneqq c$ and $(f, \ell, \ell) \coloneqq \mathsf{tf}$; let $g : Q^2 \to \mathbb{N}_\#$ be a witness function that $c \overset{\mathsf{tf}}{\Longrightarrow} c$. We want to build a set of states $S \subseteq Q$ so that we can remove, in c, one process from every state in S while maintaining that $c \overset{\mathsf{tf}}{\Longrightarrow} c$. We have supposed that $|c| > n^n(\lambda + B(n, M))$ so that there is $q_1 \in Q$ such that $\mu(q_1) > n^{n-1}(\lambda + B(n, M))$. There is therefore q_2 such that $g(q_1, q_2) > n^{n-2}(\lambda + B(n, M))$, thus also $\mu(q_2) > n^{n-2}(\lambda + B(n, M))$. If $q_2 = q_1$ then we let $S \coloneqq \{q_1, q_2\}$. If not, there is q_3 such that $g(q_2, q_3) > n^{n-3}(\lambda + B(n, M))$, and so on. By iterating this construction, we obtain $q_1, \dots, q_k$ such that $k \leqslant n$ and $q_k = q_j$ for some $j \leqslant k$. For all $i \in [\![j, k-1]\!]$, $g(q_i, q_{i+1}) > \lambda + B(n, M)$ and, for all $q \in S$, $\mu(q) > n(\lambda + B(n, M))$. We let $S \coloneqq \{q_i \mid j \leqslant i < k\}$. We let $c' = \langle \mu', \ell \rangle$ where $\mu'(q) = \mu(q) - 1$ for all $q \in S$ and $\mu'(q) = \mu(q)$ for all $q \notin S$. We also let g' such that $g'(q_i, q_{i+1}) = g(q_i, q_{i+1}) - 1$ for all $i \in [\![j, k-1]\!]$ and $g'(q, q') = g(q, q')$ otherwise. For all $i \in [\![j, k-1]\!]$, $g'(q_i, q_{i+1}) \geqslant n(\lambda + B(n, M)) \geqslant f(q_i, q_{i+1})$, so that g' is a witness function that

$c' \overset{\mathsf{tf}}{\Longrightarrow} c'$. Moreover, by Proposition 3.31, we have that $\mathsf{Post}^*(\mathcal{I}(C_\psi))$ is $(nB(n,M))$-blind so that $c' \in \mathsf{Post}^*(\mathcal{I}(C_\psi))$. This proves that c' satisfies the conditions of the lemma. By iterating this shortening procedure, we prove that we can assume that $|c| \leqslant N$. □

We now provide a similar result for the problem with fairness. To do that, we actually reduce the problem to a GRE emptiness question:

Lemma 3.41. *(C, ψ) is a positive instance of the LTL verification problem with fairness if and only if, in C_ψ,*

$$\mathcal{S}_\psi \coloneqq \mathcal{I}(C_\psi) \cap \mathsf{Pre}^*\Big(\bigcup_{(F,G)\in\Omega} \overline{\mathsf{Pre}^*(\Gamma_\psi(F))} \cap \overline{\mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\Gamma_\psi(G))})} \Big) \neq \emptyset$$

where, for all $S \subseteq \Gamma_\psi$, $\overline{S} \coloneqq \Gamma_\psi \setminus S$ denotes the set complement operation.

Proof. Thanks to Proposition 3.38, we know that the instance is positive if and only if there is, in the graph $\mathcal{G}(C_\psi)$, a bottom SCC that is winning and reachable from (γ_0, r_0) for some $\gamma_0 \in \mathcal{I}(C)$. Note that $\mathcal{I}(C_\psi) = \{(\gamma_0, r_0) \mid \gamma_0 \in \mathcal{I}(C)\}$. We need to prove that such an SCC exists if and only if $\mathcal{S}_\psi \neq \emptyset$.

Suppose first that $\mathcal{S}_\psi \neq \emptyset$, let $c_0 \in \mathcal{S}_\psi$. By definition of $\mathcal{S}_\psi$, there is a Rabin pair $(F, G) \in \Omega$ and a configuration $\gamma \in \overline{\mathsf{Pre}^*(\Gamma_\psi(F))} \cap \mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\Gamma_\psi(G))})$ such that $c_0 \overset{*}{\Longrightarrow} c$. Consider a bottom SCC S reachable from c. Because $c \notin \mathsf{Pre}^*(\Gamma_\psi(F))$ and $c \in \mathsf{Pre}^*(S)$, $S \cap \Gamma_\psi(F) = \emptyset$. For similar reasons, $S \subseteq \mathsf{Pre}^*(\Gamma_\psi(G))$ so that $S \cap \Gamma_\psi(G) \neq \emptyset$.

Conversely, suppose that we have $c_0 \in \mathcal{I}(C)$, a bottom SCC S reachable from c_0 and $(F, G) \in \Omega$ such that $S \cap \Gamma_\psi(F) = \emptyset$ and $G \cap \Gamma_\psi(G) \neq \emptyset$. Let $c \in S$; we have that $c_0 \in \mathsf{Pre}^*(c) \cap \mathcal{I}(C_\psi)$. We have $c \notin \mathsf{Pre}^*(\Gamma_\psi(F))$ because $\mathsf{Post}^*(c) \subseteq S$. Also, because S is an SCC and $S \cap \Gamma_\psi(G) \neq \emptyset$, we have $S \subseteq \mathsf{Pre}^*(\Gamma_\psi(G))$ and $\mathsf{Post}^*(c) \subseteq S$ so that $c \notin \mathsf{Pre}^*(\overline{\mathsf{Pre}^*(\Gamma_\psi(G))})$. We have proven that $c \in \mathcal{S}_\psi$, so that $\mathcal{S}_\psi \neq \emptyset$. □

We are now able to conclude the proof of Theorem 3.39. Let $n \coloneqq n(C_\psi) = n(C)$ and $M \coloneqq M(C_\psi) = O(M(C) \cdot |\psi|)$. We first transform the LTL formula automaton ψ to the Rabin automaton $\mathcal{A}_\psi$, which can be done in doubly-exponential space in ψ according to Theorem 3.36. Configurations of C_ψ of size N can be stored in space doubly-exponential in $|\psi|$, polynomial in the parameters of C and logarithmic in N, using binary encoding for the multiset; with the same space constraints, we can enumerate the set of successors, which in particular allows to decide reachability between two configurations using non-determinism.

For the LTL verification problem without fairness, by Lemma 3.40 it suffices to guess $(F, G) \in \Omega$, a configuration $c \in \Gamma_\psi(G)$ of size at most $n(C_\psi)^{n(C_\psi)}(\lambda + B(n(C_\psi), M(C_\psi)))$, and to check that $c \in \mathsf{Post}^*(\mathcal{I}(C_\psi))$ and that there is a non-trivial execution of C_ψ from c to c that does not visit $\Gamma_\psi(F)$. This can all be done in non-deterministic doubly-exponential space. Moreover, note that the double-exponential cost in only in $|\psi|$, because of the construction of the automaton using Theorem 3.36. In fact, we can drop determinism of the Rabin automaton, which is not needed in the case without fairness, to obtain $\mathcal{A}_\psi$ simply-exponential in $|\psi|$ and thus membership in EXPSPACE.

For the LTL verification problem with fairness, by Lemma 3.41, it suffices to decide emptiness of the set $\mathcal{S}_\psi$ defined in Lemma 3.41. To do so, we apply Proposition 3.34 on C_ψ. Although C_ψ has a doubly-exponential blowup in $|\psi|$, the space used in the proof of Proposition 3.34 is only polynomial in $|\psi|$, so that we remain doubly-exponential in ψ and not triply-exponential. This proves that LTL verification with fairness is in 2-EXPSPACE. In this case, we need determinism of the Rabin automaton, in particular in the proof of Lemma 3.37. Whether we can drop determinism to improve this complexity result to EXPSPACE is an open question.

3.5 Applications to ASMS and Other Models

In this section, we make connections between copycat systems and other models. The models considered are the ones mentioned in the introduction of this chapter: asynchronous shared-memory systems (ASMS), reconfigurable broadcast networks and immediate observation population protocols. We present here in detail the connection with ASMS; the links with the two other models can be obtained similarly.

ASMS can be encoded into copycat systems as follows. Given an ASMS protocol $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \Delta\rangle$, we let $C_\mathcal{P} = (Q, \mathbb{D}^{\mathsf{dim}}, \mathcal{T}_{\min})$ with $\mathcal{T}_{\min} \coloneqq \bigcup_{\delta\in\Delta} T_\delta$ where, given $\delta \in \Delta$, the set T_δ is defined as follows. Given two states q_1, q_2, we let $f_{q_1,q_2} : Q^2 \to \mathbb{N}_\#$ be the function such that:

— $f_{q_1,q_2}(q, q) = 0$ for all $q \in Q$ such that $(q, q) \neq (q_1, q_2)$;

— $f_{q_1,q_2}(q_1, q_2) = 1$;

— $f_{q_1,q_2}(q, q') = \#$ for all $q \neq q'$ such that $(q, q') \neq (q_1, q_2)$.

Let $\delta = (q_1, \mathsf{act}, q_2) \in \Delta$. We define the set T_δ as the subset of $\mathcal{F}$ that contains exactly the transfer flows $(f_{q_1,q_2}, \vec{d}_1, \vec{d}_2)$ where $\vec{d}_1$ and $\vec{d}_2$ satisfy the following condition:

— if $\mathsf{act} = \mathsf{read}_r(\mathsf{d})$ is a read transition, then $\vec{d}_1(r) = \mathsf{d}$ and $\vec{d}_1 = \vec{d}_2$;

— if $\mathsf{act} = \mathsf{write}_r(\mathsf{d})$ is a write transition, $\vec{d}_2(r) = \mathsf{d}$ and $\vec{d}_1(r') = \vec{d}_2(r')$ for all $r' \neq r$;

— if $\mathsf{act} = \circledast$ is an internal action then $\vec{d}_1 = \vec{d}_2$.

The obtained set $\mathcal{L}$ has size $|\mathbb{D}|^{\mathsf{dim}}$ and $\mathcal{T}_{\min}$ has size bounded by $|\Delta||\mathbb{D}|^{\mathsf{dim}}$, so that $|\mathcal{L}|$ and $|\mathcal{T}_{\min}|$ are exponential in the size of the ASMS protocol. Therefore, $n(C_{\mathcal{P}})$ is bounded by $|\mathcal{P}|$ and $M(C_{\mathcal{P}})$ is bounded by an exponential in $|\mathcal{P}|$. The semantics of $C_{\mathcal{P}}$ are equivalent to the accelerated semantics of $\mathcal{P}$ defined in Section 2.3.

Proposition 3.42. *Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \Delta\rangle$ be an ASMS protocol. For every $\mu_1, \mu_2 \in \mathcal{M}(Q)$, $\vec{d}_1, \vec{d}_2 \in \mathbb{D}^{\mathsf{dim}}$, $\delta \in \Delta$:*

$$\langle \mu_1, \vec{d}_1\rangle \xrightarrow[\mathsf{acc}]{\delta} \langle \mu_2, \vec{d}_2\rangle \text{ in } \mathcal{P} \quad \Longleftrightarrow \quad \exists t \in T_\delta, \langle \mu_1, \vec{d}_1\rangle \overset{t}{\Rightarrow} \langle \mu_2, \vec{d}_2\rangle \text{ in } C_{\mathcal{P}}.$$

Proof. Let $\gamma_1 \coloneqq \langle \mu_1, \vec{d}_1\rangle$, $\gamma_2 \coloneqq \langle \mu_2, \vec{d}_2\rangle$, and let $\delta = (q_1, \mathsf{act}, q_2)$; γ_1 and γ_2 can be seen both as configurations of $\mathcal{P}$ and as configurations of $C_{\mathcal{P}}$. Let $t = (f_{q_1,q_2}, \vec{d}_1, \vec{d}_2)$. By a direct case disjunction on act, we have $t \in T_\delta$ whenever $\vec{d}_1$ and $\vec{d}_2$ satisfy the conditions for $\gamma_1 \xrightarrow{\delta} \gamma_2$. Therefore, we focus on the conditions on μ_1 and μ_2. We distinguish two cases based on whether $q_1 = q_2$ or $q_1 \neq q_2$.

Case $q_1 = q_2$. If $\gamma \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$ in $\mathcal{P}$ then $\mu_1(q_1) > 0$ and $\mu_1 = \mu_2$. We let $g : Q^2 \to \mathbb{N}_\#$ such that $g(q,q) = \mu_1(q)$ for all $q \in Q$ and $g(q,q') = \#$ for all $q \neq q'$. We obtain that $f_{q_1,q_1} \leqslant g$, because $g(q_1,q_1) = \mu_1(q_1) > 0$. Therefore, g is a witness function that $\gamma_1 \overset{t}{\Rightarrow} \gamma_2$. Conversely, if $\gamma_1 \overset{t}{\Rightarrow} \gamma_2$ then let $g : Q^2 \to \mathbb{N}_\#$ be a witness function that $\gamma_1 \overset{t}{\Rightarrow} \gamma_2$. We have $f_{q_1,q_1} \leqslant g$. In particular, $\mu_1(q_1) = \sum_q g(q_1,q) = g(q_1,q_1) \geqslant f_{q_1,q_1}(q_1,q_1) = 1$ so that $\mu_1(q_1) \geqslant 1$. Moreover, because $f_{q_1,q_1} \leqslant g$, we have $g(q,q') = \#$ if and only if $q \neq q'$, so that $\mu_1 = \mu_2$. This proves that $\gamma_1 \xrightarrow{\delta} \gamma_2$.

Case $q_1 \neq q_2$. If $\gamma_1 \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$ in $\mathcal{P}$ then let $k \geqslant 1$ such that $\gamma_1 \xrightarrow{\delta^k} \gamma_2$. We have $\mu_1(q_1) \geqslant k$ and $\mu_2 = (\mu_1 \ominus k \cdot q_1) \oplus k \cdot q_2$. Let $g : Q^2 \to \mathbb{N}_\#$ such that $g(q,q) = \mu_1(q)$ for all $q \neq q_1$, $g(q_1,q_2) = k$, $g(q_1,q_1) = \mu_1(q_1) - k$ and $g(q,q') = \#$ for all $q \neq q'$ such that $(q,q') \neq (q_1,q_2)$. We have $f_{q_1,q_2} \leqslant g$; also, for all q, $\sum_{q'} g(q,q') = \mu_1(q)$ and for all q', $\sum_q g(q,q') = \mu_2(q')$. Therefore, g is a witness function that $\gamma_1 \overset{t}{\Rightarrow} \gamma_2$. Conversely, if $\gamma_1 \overset{t}{\Rightarrow} \gamma_2$ then let $g : Q^2 \to \mathbb{N}_\#$ be a witness function of that. We have $f_{q_1,q_2} \leqslant g$, so that $g(q_1,q_2) \geqslant f_{q_1,q_2}(q_1,q_2) = 1$; let $k \coloneqq g(q_1,q_2) > 0$. We have $\mu_1(q_1) = \sum_q g(q_1,q) \geqslant g(q_1,q_2) \geqslant k$. Because $g(q,q') = \#$ for all $q \neq q'$ except for $q = q_1$ and $q' = q_2$, we have $\mu_2 = (\mu_1 \ominus k \cdot q_1) \oplus k \cdot q_2$. This proves that $\gamma_1 \xrightarrow{\delta^k} \gamma_2$ so that $\gamma_1 \xrightarrow[\mathsf{acc}]{\delta} \gamma_2$. □

Therefore, ASMS with accelerated semantics can be seen as copycat systems. Given an ASMS, a *finite accelerated execution* is a sequence $\alpha = \gamma_0, \delta_1, \gamma_1, \ldots, \delta_k, \gamma_k$ such that $\gamma_i \xrightarrow[\mathsf{acc}]{\delta_{i+1}} \gamma_{i+1}$. The *length* of this accelerated execution is defined to be k. An *infinite accelerated execution* is defined similarly. With this notion, we translate Theorem 3.19 to ASMS:

Proposition 3.43. *Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \Delta \rangle$ be an ASMS protocol. For every $\gamma, \gamma' \in \Gamma$ such that $\gamma_1 \xrightarrow{*} \gamma_2$, there is an accelerated execution from γ_1 to γ_2 whose length is bounded by a function doubly-exponential in $|\mathcal{P}|$.*

Proof. It suffices to use Proposition 3.42 and to apply Corollary 3.27 on the corresponding execution of $C_{\mathcal{P}}$. Note that the fact that $M(C_{\mathcal{P}})$ is exponential in $\mathcal{P}$ has no impact on the bound, because the bound of Theorem 3.19 is polynomial in $M(C)$. □

We may also apply this connection to retrieve results related to generalized reachability expressions and to GRE emptiness in ASMS. Let us consider, for ASMS, GRE whose basic predicates ϕ range over presence constraints from Chapter 2. Note that, when ϕ is a presence constraint, the set $[\![\phi]\!]$ is 1-blind. With this definition, we have a result on the shape of the set $[\![E]\!]$ for E a GRE:

Proposition 3.44. *Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \Delta \rangle$ be an ASMS protocol, let E be a GRE over $\mathcal{P}$. The set $[\![E]\!]$ is K-blind set with K a bound doubly-exponential in $|\mathcal{P}|$ and simply exponential in the number of operators in E.*

Proof. We apply the exact same reasoning as in the proof of Proposition 3.34: by iterative applications of Proposition 3.31, $[\![E]\!]$ is K-blind for $K \coloneqq |Q|^k B(n(C_{\mathcal{P}}), M(C_{\mathcal{P}}))$ where k denotes the maximal number of nested Pre^* and Post^* operators in E, which is less than the total number of operators in E. Moreover, $C_{\mathcal{P}}$ is polynomial in $|\mathcal{P}|$ and $M(C_{\mathcal{P}})$ is exponential in $\mathcal{P}$. Because $B(n, M)$ is doubly-exponential in n and polynomial in M, the obtained bound is doubly-exponential in $|\mathcal{P}|$. □

This allows us to decide of GRE emptiness:

Proposition 3.45. *The following problem is in EXPSPACE: given an ASMS protocol $\mathcal{P}$ and a GRE E over $\mathcal{P}$, is it the case that $[\![E]\!] = \emptyset$?*

Proof. The proof is very similar to the one of Proposition 3.34 applied to $C_{\mathcal{P}}$. One must again notice that the fact that $M(C_{\mathcal{P}})$ is exponential in $|\mathcal{P}|$ does not affect the complexity class, because the space used in the proof of Proposition 3.34 was polynomial in $M(C)$. □

We present here an application of the two propositions above. In [BMRSS16], the problem of almost-sure coverability in ASMS (with one uninitialized register) is studied. Assuming that there is no transition escaping q_f, the authors show that, for a given n, q_f is covered almost surely from $\gamma_0(n)$ if and only if $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ where $\uparrow q_f$ denotes the set of configurations with at least one process in q_f. There are two main results in [BMRSS16]: that there is a bound N doubly-exponential in $|\mathcal{P}|$ such that either $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ for all $n \geqslant N$ or $\mathsf{Post}^*(\gamma_0(n)) \not\subseteq \mathsf{Pre}^*(\uparrow q_f)$ for all $n \geqslant N$, and that the problem of deciding which of the two cases holds is in EXPSPACE. The techniques from [BMRSS16] are very different from ours, and are very specific to this particular question. Nonetheless, our framework allows us to prove this same fact using K-blind sets and the structural theorem. We know by Proposition 3.44 that the set $\mathsf{Post}^*(\Gamma_0) \setminus \mathsf{Pre}^*(\uparrow q_f)$ is K-blind with K doubly-exponential in $|\mathcal{P}|$, which proves the existence of N above. Moreover, thanks to the proof of Proposition 3.34, we know that membership of a doubly-exponential configuration in a GRE can be decided in EXPSPACE, proving the second part of the result. We are thus able to reprove the main results from [BMRSS16], which highlights the level of generality and the power of the structural theorem.

There is another theoretical consequence of our results on copycat systems: verification of stuttering-invariant LTL on ASMS. A language is called *stuttering-invariant* when, for every $w_p \in \Sigma^*, x \in \Sigma$ and $w_s \in \Sigma^\omega$, for every $k \geqslant 1$, $w_p \cdot x \cdot w_s \in \mathcal{L}$ if and only if $w_p \cdot x^k \cdot w_s \in \mathcal{L}$. The stuttering-invariant fragment of LTL contains all LTL formula whose language is stuttering-invariant. This fragment is a natural restriction of LTL for computer science application and in particular for concurrent systems, as first argued by Lamport [Lam83]. In fact, any LTL formula without the X operator is stuttering-invariant, and stuttering-invariant LTL is as expressive as LTL without X [PW97].

Remark 3.46. In ASMS, it makes little sense to consider LTL formulas that are not stuttering-invariant. Indeed, if we do not have stuttering-invariance, then the copycat property (as expressed in Lemma 2.8) breaks down. In fact, if we allow for general LTL formulas, then we may encode atomic read-write combinations (recall that atomic read-write combinations were discussed in Section 2.2.1 and Section 2.9). Indeed, consider a transition $(q_1, \mathsf{read}_r(\mathsf{d}_1) - \mathsf{write}_r(\mathsf{d}_2), q_2)$ with an atomic read-write combination: a process that takes this transition must first read symbol d_1 from r and then, *right away*, write symbol d_2 to r. We argue that such a transition can be implemented in an ASMS under standard LTL constraints as follows. We add an intermediate state q_{int} and two transitions $\delta = (q_1, \mathsf{read}_r(\mathsf{d}_1), q_{\mathsf{int}})$ and $\delta' = (q_{\mathsf{int}}, \mathsf{write}_r(\mathsf{d}_2), q_2)$. Because processes may stay in q_{int}, this does not suffice to guarantee that δ' is always taken immediately after δ. However, this last condition can be encoded in LTL by $\mathsf{G}\,(\delta \implies \mathsf{X}\,\delta')$, a formula that

is not stuttering-invariant. Therefore, under standard LTL, ASMS are essentially equivalent to Petri nets and population protocols, so that verification of general LTL in ASMS lies under the scope of [EGLM16] and is therefore an Ackermann-complete problem [CO21].

It is easy to define the counterpart in ASMS of the LTL verification problems from Section 3.4. Using Theorem 3.39 applied to $C_{\mathcal{P}}$, it directly yields decidability of the LTL verification problem without fairness for ASMS. For the version with fairness, we need a an additional result. We extend the notion of strong fairness to accelerated infinite executions in a natural manner: an infinite accelerated execution is *strongly fair* when, if γ is visited infinitely often, every finite accelerated execution from γ appears infinitely often. An infinite accelerated execution $\alpha = \gamma_0, \delta_1, \gamma_1, \ldots, \delta_k, \gamma_k$ is an *acceleration* of an (unaccelerated) infinite execution ρ when ρ can be put under the form $\gamma_0 \xrightarrow{\delta_1^{k_1}} \gamma_1 \xrightarrow{\delta^{k_2}} \gamma_2$.

Lemma 3.47. *Given a strongly fair accelerated execution α, there is a strongly fair execution ρ such that α is an acceleration of ρ. Conversely, given a strongly fair execution ρ, there is a strongly fair acceleration α of ρ.*

Proof. We start with the first statement. Let $\alpha = \gamma_0, \delta_1, \gamma_1, \delta_2, \ldots$ be a strongly fair accelerated execution. Let ρ be the infinite non-accelerated execution corresponding to $\gamma_0 \xrightarrow{\delta_1^{k_1}} \gamma_1 \xrightarrow{\delta_2^{k_2}} \gamma_2 \ldots$ where, for all $i \geqslant 1$, k_i is the minimal integer $k \geqslant 1$ such that $\gamma_{i-1} \xrightarrow{\delta_i^{k}} \gamma_i$. Note that all k_i exist because α is an accelerated execution. Clearly, α is an acceleration of ρ. We now claim that ρ is strongly fair. Let $\rho' = \gamma'_0, \delta'_1, \gamma'_1, \ldots, \delta'_m, \gamma'_m$ where γ'_0 is visited infinitely often in ρ. We claim that γ'_0 appears infinitely often in α. Trivially, there is a configuration $\gamma \in \Gamma$ that is visited infinitely often in α. Both γ and γ'_0 are visited infinitely often in ρ, therefore there is a finite execution from γ to γ'_0, and hence there is an accelerated finite execution from γ to γ'_0. Because α is strongly fair, this finite accelerated execution appears infinitely often in α so that γ'_0 is visited infinitely often in α. Let α' be the accelerated execution equal to ρ', but seen as an accelerated execution. By strong fairness, α' appears infinitely often in α. For each $i \in [\![1, m]\!]$, we have $\gamma'_{i-1} \xrightarrow{\delta'_i} \gamma'_i$. Therefore, whenever α' appears in α, all the corresponding k_i are equal to 1 by minimality. This proves that, for each occurrence of α' in α, there is an occurrence of ρ' in ρ. We conclude that ρ' appears infinitely often in ρ and that ρ is strongly fair.

We now prove the second statement. Let us fix a probability distribution $f : \mathbb{N} \to [0, 1]$ such that $f(n) > 0$ for all n (*e.g.*, a geometric distribution). We first define a random variable R that takes value over the set of infinite accelerated executions. We build R as follows. We proceed (accelerated) step by (accelerated) step by grouping consecutive steps of ρ with the same

transition. Suppose that the acceleration has been built until the i-th configuration of ρ; let γ denote this configuration, and let t denote the next transition in ρ (the i-th transition of ρ, which is fired from γ). We pick an integer $m \in \mathbb{N}$ according to f, independently from the past. If steps i to $i + m - 1$ of ρ use transition t then we accelerated all those steps into one accelerated step from the i-th configuration of ρ to the $i + m$-th configuration of ρ, and we repeat the procedure from the $(i + m)$-th configuration of ρ. Otherwise, we define the next accelerated step as equal to the step from the i-th configuration to the $(i + 1)$-th configuration of ρ (the next step is not grouped with other steps), and we repeat the procedure from the $(i + 1)$-th configuration of ρ.

By repeating this construction, we obtained an infinite accelerated execution R. Trivially, R is an acceleration of ρ. We claim that R is strongly fair with probability 1. Let $\gamma_0 \in \Gamma$ be a configuration that appears infinitely often in R and let $\alpha = \gamma_0, \delta_1, \gamma_1, \delta_2, \ldots, \delta_m\gamma_m$ be an accelerated finite execution from γ_0. For each $i \in [\![1, m]\!]$, let k_i be an integer in $[\![1, +\infty[\![$ such that $\gamma_{i-1} \xrightarrow{\delta_i^{k_i}} \gamma_i$. Let σ denote the (unaccelerated) finite execution corresponding to $\gamma_0 \xrightarrow{\delta_1^{k_1}} \gamma_1 \xrightarrow{\delta_2^{k_2}} \ldots \xrightarrow{\delta_m^{k_m}} \gamma_m$. Because γ_0 is visited infinitely often in R, it is also visited infinitely often in ρ so that σ appears infinitely often in ρ. Whenever σ is visited in ρ, at the corresponding point in R, there is probability at least $\prod_{i=1}^{m} f(k_i) > 0$ that the next m accelerated steps are the same as in α. This proves that there is probability 0 that γ_0 is visited infinitely often in R but that α appears finitely often in R. By summing over the set of finite accelerated executions (which is countable), there is probability zero that there is a finite accelerated execution α disproving strong fairness. We have proven that R is strongly fair with probability one; in particular, this implies the existence of a strongly fair acceleration of ρ. □

Therefore, we can apply the result from Theorem 3.39 to obtain decidability of verification of LTL in ASMS, both without fairness and with fairness. The following result is not very new or powerful on its own, because it has been shown that this verification problem is decidable in EXPTIME for the more powerful model of shared-memory pushdown systems, although without the fairness condition [FMW17]. For this reason and for the sake of simplicity, we keep the following statement and proof informal.

Proposition 3.48. *Given an ASMS protocol $\mathcal{P}$ and a stuttering-invariant LTL formula ψ, the problem of the existence of an execution of $\mathcal{P}$ that satisfies ψ is in 2-EXPSPACE, and the same is true if the execution is required to be strongly fair.*

Proof. Because the LTL formula ψ is stuttering-invariant, one may equivalently study whether it is satisfied in the accelerated semantics. In the case with strong fairness, Lemma 3.47 guarantees

that, because ψ is stuttering-invariant, it is satisfied by an (unaccelerated) strongly fair execution if and only if it is satisfied by an accelerated strongly fair execution. We then apply the procedure from the proof of Theorem 3.39 to C_ψ. One must simply observe that the exponential blowup of $M(C_\mathcal{P})$ with respect to $|\mathcal{P}|$ does not affect the complexity, because the decidability procedures from the proof of Theorem 3.39 work in space polynomial in $M(C)$. □

Other known model mentioned in the introduction of this chapter can be encoded in copycat systems. This is the case of reconfigurable broadcast networks, which is not surprising given the strong connection between this model and ASMS [BW21]. Another one is immediate observation population protocols (or, equivalently, immediate observation petri nets). The encoding of such systems in copycat systems works in a straightforward manner similar to the one introduced above for ASMS. Therefore, the results from Proposition 3.43, Proposition 3.44, Proposition 3.45 and Proposition 3.48 can be easily extended to these two models.

3.6 Perspectives

In this section, we defined copycat systems, a general model meant to capture systems with arbitrarily many identical processes, each described by a finite-state system, and that satisfy the copycat property: whenever a process goes from state q_1 to state q_2, other processes in q_1 may do the same without impacting the rest of the system. We have proved a general-purpose doubly-exponential bound on these systems, which limits the number of (accelerated) steps needed to connect two configurations and the number of processes needed in an execution.

The motivation for defining copycat systems is to provide a unified model so that results can be stated in the most general manner. The proof techniques rely on so-called transfer flows which express the possibilities provided by a sequence of transitions. In our eyes, the mathematical framework of transfer flows is interesting on its own and may have other applications. So far in the literature, known results on parameterized distributed systems with the copycat property are proved in separate models, requiring additional work whenever trying to convert results from one model to another. Moreover, the level of generality of our general-purpose doubly-exponential bound does not appear in the literature of the closely-related models of ASMS and RBN. The closest results from the literature would be the techniques relying on so-called symbolic graphs (see [Sta17, Chapter 9] for a detailed use of symbolic graphs), which also allow to prove doubly-exponential bounds on the description on some sets; these techniques are however more *ad hoc* to specific problems and models and therefore, in our opinion, less powerful and convenient than Theorem 3.19.

Copycat systems are meant to be as general as possible, so that the results can be translated to other models without having to duplicate the proofs. In particular, the set $\mathcal{L}$ allows to encode information about the global state of the system, and it may be doubly-exponentially large without affecting the overall scale of the bound from Theorem 3.19. For example, in Section 3.5, we have used the set $\mathcal{L}$ to encode the content of shared registers, but one can also use it to, *e.g.*, encode the presence of a leader in the system.

There is one crucial open question related to copycat systems, which also appears very challenging. Indeed, it remains unknown whether the bound from Theorem 3.19 can be improved to simple-exponential in the number of states $n(C)$. This open question is related to the open complexity gap on almost-sure coverability in ASMS [BMRSS16]. This question was wrongly claimed to be solved in the model of reconfigurable broadcast networks [BGW22; BGW23]. Despite several independent efforts, it remains open at the time of writing this thesis.

3.7 Technical Proofs

3.7.1 Proofs of Basic Properties of the Compositional Product

Lemma 3.9. *For all* $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$*:*

(3.9.i) the set $\mathsf{tf}_1 \otimes \mathsf{tf}_2$ *is upward-closed with respect to* $\preceq$*: for all* $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$*, for all* $\mathsf{tf}' \in \mathcal{F}$*, if* $\mathsf{tf} \preceq \mathsf{tf}'$ *then* $\mathsf{tf}' \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$*;*

(3.9.ii) the compositional product is decreasing with respect to $\preceq$ *and* $\subseteq$*: for all* $\mathsf{tf}'_1 \preceq \mathsf{tf}_1$ *and* $\mathsf{tf}'_2 \preceq \mathsf{tf}_2$*, we have* $\mathsf{tf}_1 \otimes \mathsf{tf}_2 \subseteq \mathsf{tf}'_1 \otimes \mathsf{tf}'_2$.

Proof. We first prove (3.9.i). Let $\mathsf{tf} = (h, \ell, \ell') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. We apply the definition to obtain a witness function $H : Q^3 \to \mathbb{N}_\#$. Let $\mathsf{tf}' = (h', \ell, \ell') \in \mathcal{F}$ such that $\mathsf{tf} \preceq \mathsf{tf}'$. This implies that $h \leqslant h'$. We define $H' : Q^3 \to \mathbb{N}_\#$ as follows. Let $q_1, q_3 \in Q$. If we have $H(q_1, q_2, q_3) = \#$ for all q_2 then we set $H'(q_1, q_2, q_3) \coloneqq \#$ for all q_2. Suppose now that there is $\tilde{q_2}$ such that $H(q_1, \tilde{q_2}, q_3) \neq \#$. This in particular implies, by (3.8.i), that $h(q_1, q_3) \neq \#$ therefore $h'(q_1, q_3) \neq \#$. We set $H'(q_1, \tilde{q_2}, q_3) \coloneqq H(q_1, \tilde{q_2}, q_3) + h'(q_1, q_3) - h(q_1, q_3)$, and we set $H'(q_1, q_2, q_3) = H(q_1, q_2, q_3)$ for every $q_2 \neq \tilde{q_2}$. We claim that H' is a witness function that $(h', \ell, \ell') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. First, H' has the same # values as H, so that we have $H' \geqslant H$ by construction. Note that $H' \geqslant H$ requires that they have the same # values (# is incomparable with all integers), which would not hold if we had set $H'(q_1, q_2, q_3) = H(q_1, q_2, q_3) + h'(q_1, q_3) - h(q_1, q_3)$ for some q_1, q_2 and q_3 such that $h'(q_1, q_3) \neq \#$ but $H(q_1, q_2, q_3) = \#$. We therefore have that H' satisfies (3.8.ii) and (3.8.iii). Also, for all q_1, q_3, if $h'(q_1, q_3) = \#$ then $\sum_{q_2} H'(q_1, q_2, q_3) = \sum_{q_2} H(q_1, q_2, q_3) = h(q_1, q_3) =$

#. If $h(q_1, q_3) \neq \#$ then $\sum_{q_2} H'(q_1, q_2, q_3) = \sum_{q_2} H(q_1, q_2, q_3) + h'(q_1, q_3) - h(q_1, q_3) = h(q_1, q_3) + h'(q_1, q_3) - h(q_1, q_3) = h'(q_1, q_3)$. We have proved that H' satisfies Item (3.8.i) for h', so that $(h', \ell, \ell') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$.

To prove (3.9.ii), it suffices to observe that, when we decrease the values of f_1 and f_2 in Definition 3.8, conditions (3.8.ii) and (3.8.iii) become less restrictive. □

Lemma 3.11. *The compositional product* $\otimes$ *is associative,* i.e.*, for all* $\mathsf{tf}_1, \mathsf{tf}_2, \mathsf{tf}_3 \in \mathcal{F}$, $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 = \mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3)$.

Proof. Let $\mathsf{tf}_i =: (f_i, \ell_i, \ell_i')$ for all $i \in \{1, 2, 3\}$. If we have $\ell_1' \neq \ell_2$ or $\ell_2' \neq \ell_3$ then $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 = \mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3) = \emptyset$. Suppose now that $\ell_1' = \ell_2$ and $\ell_2' = \ell_3$.

Let $T_{1,2,3} \subseteq \mathcal{F}$ denote the set of transfer flows $\mathsf{tf} = (f, \ell_1, \ell_3')$ for which there exists a function $H : Q^4 \to \mathbb{N}_\#$ that satisfies the following properties:

1. for all q_1, q_4, $\sum_{q_2, q_3} H(q_1, q_2, q_3, q_4) = f(q_1, q_4)$;
2. for all q_1, q_2, $\sum_{q_3, q_4} H(q_1, q_2, q_3, q_4) \geqslant f_1(q_1, q_2)$;
3. for all q_2, q_3, $\sum_{q_1, q_4} H(q_1, q_2, q_3, q_4) \geqslant f_2(q_2, q_3)$;
4. for all q_3, q_4, $\sum_{q_1, q_2} H(q_1, q_2, q_3, q_4) \geqslant f_3(q_3, q_4)$.

We claim that $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 = T_{1,2,3} = \mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3)$.

We first prove that $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 \subseteq T_{1,2,3}$. Let $\mathsf{tf} = (f, \ell_1, \ell_3') \in (\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3$. Let $\mathsf{tf}_{1,2} = (f_{1,2}, \ell_1, \ell_2') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ such that $\mathsf{tf} \in \mathsf{tf}_{1,2} \otimes \mathsf{tf}_3$; let $G : Q^3 \to \mathbb{N}_\#$ be a witness function of that. We have $\sum_{q_4} G(q_1, q_3, q_4) \geqslant f_{1,2}(q_1, q_3)$ for all q_1, q_3, $\sum_{q_1} G(q_1, q_3, q_4) \geqslant f_3(q_3, q_4)$ for all q_3, q_4 and $\sum_{q_3} G(q_1, q_3, q_4) = f(q_1, q_4)$ for all q_1, q_4. Let $F_{1,2}$ be a witness function that $\mathsf{tf}_{1,2} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$, *i.e.*, $\sum_{q_2} F_{1,2}(q_1, q_2, q_3) = f_{1,2}(q_1, q_3)$ for all q_1, q_3, $\sum_{q_3} F_{1,2}(q_1, q_2, q_3) \geqslant f_1(q_1, q_2)$ for all q_1, q_2 and $\sum_{q_1} F_{1,2}(q_1, q_2, q_3) \geqslant f_2(q_2, q_3)$ for all q_2, q_3. For every q_1, q_3, $\sum_{q_4} G(q_1, q_3, q_4) \geqslant f_{1,2}(q_1, q_3) = \sum_{q_2} F_{1,2}(q_1, q_2, q_3)$. For all q_1, q_3, if $f_{1,2}(q_1, q_3) \neq \#$ then there is $\tilde{q_2}$ such that $F_{1,2}(q_1, \tilde{q_2}, q_3) \neq \#$. Let F equal to $F_{1,2}$ except that, for all q_1, q_3 such that $f_{1,2}(q_1, q_3) \neq \#$, we choose $\tilde{q_2}$ such that $F_{1,2}(q_1, \tilde{q_2}, q_3) \neq \#$ and set $F(q_1, \tilde{q_2}, q_3) \coloneqq F_{1,2}(q_1, \tilde{q_2}, q_3) + \sum_{q_4} G(q_1, q_3, q_4) - f_{1,2}(q_1, q_3)$. This way, F satisfies the same conditions as $F_{1,2}$ related to f_1 and f_2 but also, for all q_1, q_3, $\sum_{q_2} F(q_1, q_2, q_3) = \sum_{q_4} G(q_1, q_3, q_4)$. To provide H that satisfies the conditions above, it suffices to build $H : Q^4 \to \mathbb{N}_\#$ so that $\sum_{q_4} H(q_1, q_2, q_3, q_4) = F(q_1, q_2, q_3)$ and $\sum_{q_2} H(q_1, q_2, q_3, q_4) = G(q_1, q_3, q_4)$. Indeed, this would imply conditions 1 and 4 thanks to F and conditions 2 and 3 thanks to G.

We now prove the following statement:

> For every $F : Q^3 \to \mathbb{N}_\#$ and $G : Q^3 \to \mathbb{N}_\#$, if $\sum_{q_2} F(q_1, q_2, q_3) = \sum_{q_4} G(q_1, q_3, q_4)$ for every q_1, q_3, then there is $H : Q^4 \to \mathbb{N}_\#$ such that $\sum_{q_4} H(q_1, q_2, q_3, q_4) = F(q_1, q_2, q_3)$ and $\sum_{q_2} H(q_1, q_2, q_3, q_4) = G(q_1, q_3, q_4)$.

First, if F and G are constant equal to # then we set H constant equal to #. Suppose now that it is not the case; let $n \coloneqq \sum_{q_1,q_2,q_3} F(q_1, q_2, q_3) = \sum_{q_1,q_3,q_4} G(q_1, q_3, q_4) \in \mathbb{N}$. We proceed by induction on n.

If $n = 0$ then all values in F and G are in $\{0, \#\}$. We let $H(q_1, q_2, q_3, q_4) \coloneqq 0$ whenever both $F(q_1, q_2, q_3) = 0$ and $G(q_1, q_3, q_4) = 0$, and $H(q_1, q_2, q_3, q_4) \coloneqq \#$ otherwise. We claim that, for all q_1, q_2, q_3, $\sum_{q_4} H(q_1, q_2, q_3, q_4) = F(q_1, q_2, q_3)$. Let $q_1, q_2, q_3 \in Q$; if $F(q_1, q_2, q_3) = \#$ then $H(q_1, q_2, q_3, q_4) = \#$ for all q_4 hence $\sum_{q_4} H(q_1, q_2, q_3, q_4) = \#$. Suppose now that $F(q_1, q_2, q_3) = 0$. This implies $\sum_{q_4} G(q_1, q_3, q_4) = 0$ therefore there is $\tilde{q_4}$ such that $G(q_1, q_3, \tilde{q_4}) = 0$, so that $H(q_1, q_2, q_3, \tilde{q_4}) = 0$ and $\sum_{q_4} H(q_1, q_2, q_3, q_4) = 0$. Similarly, for every q_1, q_3, q_4, if $G(q_1, q_3, q_4) = \#$ then $\sum_{q_2} H(q_1, q_2, q_3, q_4) = \#$ and if $G(q_1, q_3, q_4) = 0$ then there is $\tilde{q_2}$ such that $F(q_1, \tilde{q_2}, q_3) = 0$ hence $H(q_1, \tilde{q_2}, q_3, q_4) = 0$ and $\sum_{q_2} H(q_1, q_2, q_3, q_4) = 0$.

Suppose now that $n > 0$. Let $\tilde{q_1}, \tilde{q_3}$ such that $\sum_{q_2} F(\tilde{q_1}, q_2, \tilde{q_3}) = \sum_{q_4} G(\tilde{q_1}, \tilde{q_3}, q_4) > 0$. Let $\tilde{q_2}$ such that $F(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) > 0$ and $\tilde{q_4}$ such that $G(\tilde{q_1}, \tilde{q_3}, \tilde{q_4}) > 0$. Let F' equal to F except that $F'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) \coloneqq F(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) - 1$ and let G' equal to G except that $G'(\tilde{q_1}, \tilde{q_3}, \tilde{q_4}) \coloneqq G(\tilde{q_1}, \tilde{q_3}, \tilde{q_4}) - 1$. We have $\sum_{q_2} F'(q_1, q_2, q_3) = \sum_{q_4} G'(q_1, q_3, q_4)$ for all q_1 and q_3, and $\sum_{q_1,q_2,q_3} F'(q_1, q_2, q_3) = \sum_{q_1,q_2,q_3} F(q_1, q_2, q_3) - 1 = n - 1$. We apply the induction hypothesis on F' and G' to obtain H' such that $\sum_{q_4} H'(q_1, q_2, q_3, q_4) = F'(q_1, q_2, q_3)$ for all q_1, q_2, q_3 and $\sum_{q_2} H'(q_1, q_2, q_3, q_4) = G'(q_1, q_3, q_4)$ for all q_1, q_3, q_4. It suffices to let H equal to H' except that $H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, \tilde{q_4}) = H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, \tilde{q_4}) + 1$. Note that it could be that $H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, \tilde{q_4}) = \#$, in which case $H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, \tilde{q_4}) = 1$. We know that $F'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) \neq \#$ therefore $\sum_{q_4} H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, q_4) \neq \#$ so that we indeed have $\sum_{q_4} H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}, q_4) = F'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) + 1 = F(\tilde{q_1}, \tilde{q_2}, \tilde{q_3})$. With the same argument, $\sum_{q_2} H'(\tilde{q_1}, q_2, \tilde{q_3}, \tilde{q_4}) = G(\tilde{q_1}, \tilde{q_3}, \tilde{q_4})$. This concludes the induction.

We have proved that $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 \subseteq T_{1,2,3}$. The fact that $\mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3) \subseteq T_{1,2,3}$ follows by a symmetric argument. We claim that $T_{1,2,3} \subseteq (\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3$. Indeed, let $\mathsf{tf} \in T_{1,2,3}$ and let $H : Q^4 \to \mathbb{N}_\#$ that satisfies conditions 1 to 4 for tf. Let $f : (q_1, q_3) \mapsto \sum_{q_3,q_4} H(q_1, q_2, q_3, q_4)$, we have $(f, \ell_1, \ell_2') \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ with $F : (q_1, q_2, q_3) \mapsto \sum_{q_4} H(q_1, q_2, q_3, q_4)$ as witness function. Moreover, let $g : (q_3, q_4) \mapsto \sum_{q_1,q_2} H(q_1, q_2, q_3, q_4)$; we have $\mathsf{tf}_3 \preceq (g, \ell_3, \ell_3')$. Finally, we have $\mathsf{tf} \in (f, \ell_1, \ell_2') \otimes (g, \ell_3, \ell_3')$ with $(q_1, q_3, q_4) \mapsto \sum_{q_2} H(q_1, q_2, q_3, q_4)$ as witness function, hence by (3.9.ii) we conclude that $\mathsf{tf} \in (\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3$. This proves that $T_{1,2,3} \subseteq (\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3$; a symmetric argument proves that $T_{1,2,3} \subseteq \mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3)$. In the end, we obtain $(\mathsf{tf}_1 \otimes \mathsf{tf}_2) \otimes \mathsf{tf}_3 =$

$\mathsf{tf}_1 \otimes (\mathsf{tf}_2 \otimes \mathsf{tf}_3) = T_{1,2,3}$. □

Lemma 3.12. *Let* $\mathsf{tf}_1, \mathsf{tf}_2 \in \mathcal{F}$. *For every* $\mathsf{tf} \in \mathsf{basis}(\mathsf{tf}_1 \otimes \mathsf{tf}_2)$, $\mathsf{weight}(\mathsf{tf}) \leqslant \mathsf{weight}(\mathsf{tf}_1) + \mathsf{weight}(\mathsf{tf}_2)$.

Proof. Let $\mathsf{tf}_1 = (f_1, \ell_1, \ell_2)$, $\mathsf{tf}_2 = (f_2, \ell_2, \ell_3)$ and $\mathsf{tf} = (f, \ell_1, \ell_3) \in \mathsf{basis}(\mathsf{tf}_1 \otimes \mathsf{tf}_2)$; let $H : Q^3 \to \mathbb{N}_\#$ be a witness function that $\mathsf{tf} \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$. We know that $\mathsf{weight}(\mathsf{tf}) = \sum_{q_1,q_3} f(q_1, q_3) = \sum_{q_1,q_2,q_3} H(q_1, q_2, q_3)$. We thus prove that $\sum_{q_1,q_2,q_3} H(q_1, q_2, q_3) \leqslant \mathsf{weight}(\mathsf{tf}_1) + \mathsf{weight}(\mathsf{tf}_2)$. Suppose by contradiction that $\sum_{q_1,q_2,q_3} H(q_1, q_2, q_3) > \mathsf{weight}(\mathsf{tf}_1) + \mathsf{weight}(\mathsf{tf}_2)$. We claim that there is $H' : Q^3 \to \mathbb{N}_\#$ such that:

— $H' \leqslant H$,

— $\sum_{q_1,q_2,q_3} H'(q_1, q_2, q_3) < \sum_{q_1,q_2,q_3} H(q_1, q_2, q_3)$,

— $\sum_{q_3} H'(q_1, q_2) \geqslant f_1(q_1, q_2)$ for all q_1, q_2,

— $\sum_{q_1} H'(q_1, q_2, q_3) \geqslant f_2(q_2, q_3)$ for all q_2, q_3.

Indeed, if we have such a function H', then letting $f' : (q_1, q_2) \mapsto H'(q_1, q_2, q_3)$, we would have $(f', \ell_1, \ell_3) \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$ and $(f', \ell_1, \ell_3) \preceq \mathsf{tf}$, contradicting minimality of tf in $\mathsf{tf}_1 \otimes \mathsf{tf}_2$.

To build H', it suffices to prove the existence of $\tilde{q_1}, \tilde{q_2}, \tilde{q_3}$ such that $\sum_{q_3} H(\tilde{q_1}, \tilde{q_2}, q_3) > f_1(\tilde{q_1}, \tilde{q_2})$ and $\sum_{q_1} H(q_1, \tilde{q_2}, \tilde{q_3}) > f_2(\tilde{q_2}, \tilde{q_3})$, so that we can set H' equal to H except that $H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) = H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) - 1$.

To find $\tilde{q_1}, \tilde{q_2}$ and $\tilde{q_3}$, we prove the following statement:

> For all $h : Q^3 \to \mathbb{N}_\#$, $g_1 : Q^2 \to \mathbb{N}_\#$ and $g_2 : Q^2 \to \mathbb{N}_\#$ such that $\sum_{q_3} h(q_1, q_2, q_3) \geqslant g_1(q_1, q_2)$ for all q_1, q_2, $\sum_{q_1} h(q_1, q_2, q_3) \geqslant g_2(q_2, q_3)$ for all q_2, q_3 and $\sum_{q_1,q_2,q_3} h(q_1, q_2, q_3) > \sum_{q_1,q_2} g_1(q_1, q_2) + \sum_{q_2,q_3} g_2(q_2, q_3)$, there are $\tilde{q_1}, \tilde{q_2}$ and $\tilde{q_3}$ such that $\sum_{q_3} h(\tilde{q_1}, \tilde{q_2}, q_3) > g_1(\tilde{q_1}, \tilde{q_2})$ and $\sum_{q_1} h(q_1, \tilde{q_2}, \tilde{q_3}) > g_2(\tilde{q_2}, \tilde{q_3})$.

The proof is by induction on $\sum_{q_1,q_2,q_3} h(q_1, q_2, q_3)$. The base case is when $\sum_{q_1,q_2,q_3} h(q_1, q_2, q_3) = 1$ and g_1 and g_2 only have value 0 and #, in which case it suffices to take $\tilde{q_1}, \tilde{q_2}, \tilde{q_3}$ such that $h(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) = 1$. For the induction step, let r_1, r_2, r_3 such that $h(r_1, r_2, r_3) > 0$. This implies that $g_1(r_1, r_2), g_2(r_2, r_3) \in \mathbb{N}$. If $g_1(r_1, r_2) = 0$ and $g_2(r_2, r_3) = 0$ then we let $(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) \coloneqq (r_1, r_2, r_3)$ and we are done. Assume now that $g_1(r_1, r_2) > 0$ or $g_2(r_2, r_3) > 0$. Let h' equal to h except that $h'(r_1, r_2, r_3) = h(r_1, r_2, r_3) - 1$; let g'_1 equal to g_1 except if $g_1(r_1, r_2) > 0$ in which case $g'_1(r_1, r_2) = g_1(r_1, r_2) - 1$; let g'_2 equal to g_2 except if $g_2(r_2, r_3) > 0$ in which case $g'_2(r_2, r_3) = g_2(r_2, r_3) - 1$. For every $(q_1, q_2) \neq (r_1, r_2)$, we have $\sum_{q_3} h'(q_1, q_2, q_3) = \sum_{q_3} h(q_1, q_2, q_3) \geqslant g_1(q_1, q_2) = g'_1(q_1, q_2)$. Moreover, if $g_1(r_1, r_2) = 0$ then $g'_1(r_1, r_2) = 0$ and $\sum_{q_3} h'(r_1, r_2, q_3) \in \mathbb{N}$ so that $\sum_{q_3} h'(r_1, r_2, q_3) \geqslant$

$0 = g'_1(r_1, r_2)$. If $g_1(r_1, r_2) > 0$ then $g'_1(r_1, r_2) = g_1(r_1, r_2) - 1$ and $\sum_{q_3} h'(r_1, r_2, q_3) = \sum_{q_3} h(r_1, r_2, q_3) - 1 \geqslant g_1(r_1, r_2) - 1 = g'_1(r_1, r_2)$. Overall, we have proved that, for all q_1, q_2, $\sum_{q_3} h'(q_1, q_2) \geqslant g'_1(q_1, q_2)$. A similar argument proves that, for all q_2, q_3, $\sum_{q_1} h'(q_1, q_2, q_3) \geqslant g_2(q_2, q_3)$. Finally, by hypothesis, we have either $g_1(r_1, r_2) > 0$ or $g_2(r_2, r_3) > 0$ so that $\sum_{q_1,q_2} g'_1(q_1, q_2) + \sum_{q_2,q_3} g'_2(q_2, q_3) \leqslant \sum_{q_1,q_2} g_1(q_1, q_2) + \sum_{q_2,q_3} g_2(q_2, q_3) - 1$. Therefore, $\sum_{q_1,q_2,q_3} h'(q_1, q_2, q_3) = \sum_{q_1,q_2,q_3} h(q_1, q_2, q_3) - 1 \geqslant \sum_{q_1,q_2} g_1(q_1, q_2) + \sum_{q_2,q_3} g_2(q_2, q_3) - 1 \geqslant \sum_{q_1,q_2} g'_1(q_1, q_2) + \sum_{q_2,q_3} g'_2(q_2, q_3)$. We have proved that we may apply the induction hypothesis on h', g'_1 and g'_2. By doing so, we obtain $\tilde{q_1}, \tilde{q_2}, \tilde{q_3}$ such that $\sum_{q_3} h'(\tilde{q_1}, \tilde{q_2}, q_3) > g'_1(\tilde{q_1}, \tilde{q_2})$ and $\sum_{q_1} h'(q_1, \tilde{q_2}, \tilde{q_3}) > g'_2(\tilde{q_2}, \tilde{q_3})$. We prove that the same holds for h, g_1 and g_2. If $(\tilde{q_1}, \tilde{q_2}) \neq (r_1, r_2)$ then $\sum_{q_3} h(\tilde{q_1}, \tilde{q_2}, q_3) = \sum_{q_3} h'(\tilde{q_1}, \tilde{q_2}, q_3) > g'_1(\tilde{q_1}, \tilde{q_2}) = g_1(\tilde{q_1}, \tilde{q_2})$. Moreover, if $(\tilde{q_1}, \tilde{q_2}) = (r_1, r_2)$, we have $\sum_{q_3} h(\tilde{q_1}, \tilde{q_2}, q_3) = \sum_{q_3} h'(\tilde{q_1}, \tilde{q_2}, q_3) + 1 > g'_1(\tilde{q_1}, \tilde{q_2}) + 1 \geqslant g_1(\tilde{q_1}, \tilde{q_2})$. Overall, this proves that $\sum_{q_3} h(\tilde{q_1}, \tilde{q_2}, q_3) > g_1(\tilde{q_1}, \tilde{q_2})$; a similar argument proves that $\sum_{q_1} h(q_1, \tilde{q_2}, \tilde{q_3}) > g_2(\tilde{q_2}, \tilde{q_3})$. This concludes the induction.

Applying the property to H, f_1 and f_2 allow to obtain $\tilde{q_1}, \tilde{q_2}, \tilde{q_3}$ such that $\sum_{q_3} H(\tilde{q_1}, \tilde{q_2}, q_3) > f_1(\tilde{q_1}, \tilde{q_2})$ and $\sum_{q_1} H(q_1, \tilde{q_2}, \tilde{q_3}) > f_2(\tilde{q_2}, \tilde{q_3})$, so that we can set H' equal to H except that $H'(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) = H(\tilde{q_1}, \tilde{q_2}, \tilde{q_3}) - 1$. We then let $f' : (q_1, q_2) \mapsto H'(q_1, q_2, q_3)$; H' is a witness function that $(f', \ell_1, \ell_3) \in \mathsf{tf}_1 \otimes \mathsf{tf}_2$, but $(f', \ell_1, \ell_3) \preceq \mathsf{tf}$, contradicting minimality of tf in $\mathsf{tf}_1 \otimes \mathsf{tf}_2$. □

Lemma 3.15. *For all $T \subseteq \mathcal{F}$ upward-closed for $\preceq$, $\mathcal{F}_0 \otimes T = T \otimes \mathcal{F}_0 = T$.*

Proof. Let $T \subseteq \mathcal{F}_0$ be an upward-closed set of transfer flows. For every $\ell \in \mathcal{L}$, let $\mathsf{tf}_\ell \coloneqq (f_0, \ell, \ell)$ with f_0 as defined above. We trivially have $\mathsf{tf}_\ell \in \mathcal{F}_0$ for all $\ell \in \mathcal{L}$. Let $\mathsf{tf} = (f, \ell, \ell') \in T$. We claim that $\mathsf{tf} \in \mathsf{tf}_\ell \otimes \mathsf{tf}$. Indeed, the control locations match and, for the witness function, it suffices to let $H : Q^3 \to \mathbb{N}_\#$ such that $H(q_1, q_2, q_3) = \#$ when $q_1 \neq q_2$ and $H(q_1, q_1, q_3) = f(q_1, q_3)$ for all $q_1, q_3 \in Q$. With a similar argument, $\mathsf{tf} \in \mathsf{tf} \otimes f_{\ell'}$. We have proved that $T \subseteq \mathcal{F}_0 \otimes T$ and $T \subseteq T \otimes \mathcal{F}_0$.

Let now $\mathsf{tf}_0 \in \mathcal{F}_0$ and $\mathsf{tf}' \in \mathsf{tf}_0 \otimes \mathsf{tf}$. We must have $\mathsf{tf}' = (f', \ell, \ell')$ for some $f' : Q^2 \to \mathbb{N}_\#$. Let $H : Q^3 \to \mathbb{N}_\#$ be a witness function that $\mathsf{tf}' \in \mathsf{tf}_0 \otimes \mathsf{tf}$. We must have $H(q_1, q_2, q_3) = \#$ whenever $q_1 \neq q_2$, so that $f'(q_1, q_3) = H(q_1, q_1, q_3)$ for all $q_1, q_3 \in Q$. Moreover, $\sum_{q_1} H(q_1, q_2, q_3) \geqslant f(q_2, q_3)$ for all $q_1, q_2, q_3 \in Q$ therefore $H(q_1, q_1, q_3) \geqslant f(q_1, q_3)$ for all $q_1, q_3 \in Q$. This proves that $\mathsf{tf} \preceq \mathsf{tf}'$ so that, because T is upward-closed, $\mathsf{tf}' \in T$. We have proved that $\mathcal{F}_0 \otimes T \subseteq T$; a similar argument proves that $T \otimes \mathcal{F}_0 \subseteq T$. □

3.7.2 Proof of ω-monotonicity

Lemma 3.26. *$(D_k)_{k\leqslant L}$ is ω-monotone.*

This subsection is devoted to proving Lemma 3.26. Let $k \geqslant 0$, let $I_{k+1} \subseteq D_{k+1}$ be a proper ideal at step $k+1$. Let $\vec{v}_{k+1} \in \mathbb{N}_\omega^d$ be the vector representing I_{k+1}. Because I_{k+1} is proper, $I_{k+1} \not\subseteq D_{k+2}$.

Lemma 3.49. $I_{k+1} \cap (\chi(\mathcal{T}^{k+2}) \setminus \chi(\mathcal{T}^{\leqslant k+1})) \neq \emptyset$.

Proof. We know that $I_{k+1} \cap (U_{k+2} \setminus U_{k+1}) \neq \emptyset$ because I_{k+1} is a proper ideal at step $k+1$. Let $\vec{v} \in I_{k+1} \cap (U_{k+2} \setminus U_{k+1})$. Because $\vec{v} \in U_{k+2}$, there is $\vec{u} \in \chi(\mathcal{T}^{\leqslant k+2}) \cup V_0$ such that $\vec{u} \leqslant_\times \vec{v}$. We have $\vec{u} \notin V_0$ as it would otherwise imply that $\vec{v} \in U_{k+1}$, therefore $\vec{u} \in \chi(\mathcal{T}^{\leqslant k+2})$. Also, $\vec{u} \notin \chi(\mathcal{T}^{\leqslant k+1})$ as it would otherwise imply that $\vec{v} \in U_{k+1}$. Because I_{k+1} is downward-closed, $\vec{u} \in I_{k+1}$, so that $\vec{u} \in I_{k+1} \cap (\chi(\mathcal{T}^{k+2}) \setminus \chi(\mathcal{T}^{\leqslant k+1}))$. □

Given a set $I \subseteq \mathbb{N}^d$ of vectors and $t \in \mathcal{T}_{\min}$, let $\mathsf{Pre}_t(I)$ be the set of vectors $\vec{v}$ such that there are transfer flows $\mathsf{tf}_I \in \mathcal{F}$, $\mathsf{tf}_{\vec{v}} \in \mathsf{tf}_I \otimes t$ with $\vec{v} \in \chi(\mathsf{tf}_{\vec{v}})$ and $\chi(\mathsf{tf}_I) \subseteq I$.

Lemma 3.50. *For all $t \in \mathcal{T}_{\min}$, $\mathsf{Pre}_t(I_{k+1}) \subseteq D_k$.*

Proof. Suppose by contradiction that we have $\vec{v} \in \mathsf{Pre}_t(I_{k+1}) \cap U_k$. There are $\mathsf{tf}_k, \mathsf{tf}_{k+1}$ such that $\vec{v} \in \chi(\mathsf{tf}_k)$ and $\chi(\mathsf{tf}_{k+1}) \subseteq I_{k+1}$. In particular $\vec{v} \in \chi(\mathcal{F}) \cap U_k$ hence $\vec{v} \in \chi(\mathcal{T}^{\leqslant k})$ by Lemma 3.21. By strong injectivity (Lemma 3.20) this implies that $\mathsf{tf}_k \in \mathcal{T}^{\leqslant k}$, so that $\mathsf{tf}_{k+1} \in \mathcal{T}^{\leqslant k+1}$. Therefore, $\chi(\mathsf{tf}_{k+1}) \cap I_{k+1} \neq \emptyset$ but $\chi(\mathsf{tf}_{k+1}) \subseteq \chi(\mathcal{T}^{\leqslant k+1}) \subseteq U_{k+1}$, which contradicts $I_{k+1} \subseteq D_{k+1}$. □

Lemma 3.51. *There is $t \in \mathcal{T}_{\min}$ such that $\mathsf{Pre}_t(I_{k+1}) \cap U_{k+1} \neq \emptyset$.*

Proof. By Lemma 3.49, there is $\vec{v} \in I_{k+1} \cap (\chi(\mathcal{T}^{k+2}) \setminus \chi(\mathcal{T}^{\leqslant k+1}))$. Let $\mathsf{tf} = (f, \ell, \ell') \in \mathcal{T}^{k+2}$ such that $\vec{v} \in \chi(\mathsf{tf})$. Because $\vec{v} \notin \chi(\mathcal{T}^{\leqslant k+1})$, $\mathsf{tf} \notin \mathcal{T}^{\leqslant k+1}$. Also, $\chi(\mathsf{tf}) \subseteq I_{k+1}$. Indeed, by Lemma 3.21, $\chi(\mathsf{tf}) \cap U_{k+1} \subseteq \chi(\mathcal{T}^{\leqslant k+1})$ but $\chi(\mathsf{tf}) \cap \chi(\mathcal{T}^{\leqslant k+1}) = \emptyset$ by strong injectivity, so that $\chi(\mathsf{tf}) \subseteq D_{k+1}$. This means that the representing vector of I_{k+1} must have value ω on every i such that $f(\mathsf{index}^{-1}(i)) = \#$ by maximality of I_{k+1} in D_{k+1}, so that $\vec{v} \in I_{k+1}$ implies that $\vec{u} \in I_{k+1}$ for every $\vec{u} \in \chi(\mathsf{tf})$. Overall, we have proved that $\chi(\mathsf{tf}) \subseteq I_{k+1}$.

Because $\mathsf{tf} \in \mathcal{T}^{k+2}$, there is $t \in \mathcal{T}_{\min}$, $\mathsf{tf}' \in \mathcal{T}^{k+1}$ such that $\mathsf{tf} \in \mathsf{tf}' \otimes t$. By definition of $\mathsf{Pre}_t(I_{k+1})$, $\chi(\mathsf{tf}') \subseteq \mathsf{Pre}_t(I_{k+1})$. Also, $\chi(\mathsf{tf}') \subseteq \chi(\mathcal{T}^{\leqslant k+1}) \subseteq U_{k+1}$, so that $\chi(\mathsf{tf}') \subseteq \mathsf{Pre}_t(I_{k+1}) \cap U_{k+1}$. By Lemma 3.20, $\chi(\mathsf{tf}') \neq \emptyset$ therefore $\mathsf{Pre}_t(I_{k+1}) \cap U_{k+1} \neq \emptyset$. □

In all the following, we fix $t \in \mathcal{T}_{\min}$ such that $\mathsf{Pre}_t(I_{k+1}) \cap U_{k+1} \neq \emptyset$. By applying the definition, there are $\mathsf{tf}_k, \mathsf{tf}_{k+1}$ such that $\mathsf{tf}_{k+1} \in \mathsf{tf}_k \otimes t$, $\chi(\mathsf{tf}_k) \subseteq \mathsf{Pre}_t(I_{k+1}) \cap U_{k+1}$ and $\chi(\mathsf{tf}_{k+1}) \subseteq I_{k+1}$. We write $\mathsf{tf}_{k+1} = (f_{k+1}, \ell_{k+1}, \ell'_{k+1})$. Also, let $E \subseteq [\![1, d]\!]$ be the set of components at which the representing vector of I_{k+1} is equal to ω. We know that $n^2+1, n^2+2 \notin E$ as it would otherwise imply that $V_0 \cap D_k \neq \emptyset$, which contradicts definition of U_k [3]. This means that $E \subseteq [\![1, n^2]\!]$. Let $S \coloneqq \mathsf{index}^{-1}(E)$; S is the set of pairs of states (q, q') such that $\mathsf{index}(q, q') \in E$. For every $j \in \mathbb{N}$, for every transfer flow $\mathsf{tf} = (f, \ell, \ell')$, we denote by $\mathsf{tf}^{(j)}$ the transfer flow $(f^{(j)}, \ell, \ell')$ where $f^{(j)}$ is such that, for all $q, q' \in Q$:

— if $(q, q') \notin S$ then $f^{(j)}(q, q') = f(q, q')$;

— if $(q, q') \in S$ and $f(q, q') \neq \#$ then $f^{(j)}(q, q') = \max(f(q, q'), j)$;

— if $(q, q') \in S$ and $f(q, q') = \#$ then $f^{(j)}(q, q') = \#$.

Intuitively, $\mathsf{tf}^{(j)}$ is equal to tf except that, in all components in E where tf does not have value #, the values will tend to infinity as j grows. We define a similar notion for vectors. For every $j \in \mathbb{N}$, for every vector $\vec{v}$, we let $\vec{v}^{(j)}$ be the vector defined by, for all $i \in [\![1, d]\!]$:

— if $i \in E$ and $\vec{v}^{(j)}(i) = \max(\vec{v}(i), j)$

— if $i \notin E$ then $\vec{v}^{(j)}(i) = \vec{v}(i)$.

We connect the definition of $\vec{v}^{(j)}$ and of $\mathsf{tf}^{(j)}$ with the following lemma:

Lemma 3.52. *Let* $\mathsf{tf} \in \mathcal{F}$ *and* $\vec{v} \in \chi(\mathsf{tf})$. *For all* $j \in \mathbb{N}$, $\vec{v}^{(j)} \in \chi(\mathsf{tf}^{(j)})$.

Proof. Let $j \in \mathbb{N}$, $\mathsf{tf} = (f, \ell, \ell')$ and $\vec{v} \in \chi(\mathsf{tf})$. Let $i \in [\![1, n^2]\!]$ and $(q, q') \coloneqq \mathsf{index}^{-1}(i)$. First, if $i \notin E$ then $(q, q') \notin S$, so that $\vec{v}^{(j)}(i) = \vec{v}(i) = f(q, q') = f^{(j)}(q, q')$. Suppose now that $i \in E$. If $f(q, q') = \#$ then $f^{(j)}(q, q') = \#$ and the value at component i in $\vec{v}^{(j)}$ plays no role in whether $\vec{v}^{(j)} \in \chi(\mathsf{tf}^{(j)})$. If $f(q, q') \neq \#$ then $f(q, q') = \vec{v}(i)$ so that $\vec{v}^{(j)}(i) = \max(\vec{v}(i), j) = \max(f(q, q'), j) = f^{(j)}(q, q')$, concluding the proof. □

By applying this construction to tf_{k+1}, we obtain a sequence that remains in $\chi^{-1}(I_{k+1})$:

Lemma 3.53. *For all* j, *we have* $\chi(\mathsf{tf}_{k+1}^{(j)}) \subseteq I_{k+1}$.

Proof. By definition of tf_{k+1}, $\chi(\mathsf{tf}_{k+1}) \subseteq I_{k+1}$. Let $\vec{v}_j \in \chi(\mathsf{tf}_{k+1}^{(j)})$. Let $\vec{v}$ equal to $\vec{v}_j$ except that $\vec{v}(\mathsf{index}(q, q')) = f(q, q')$ for all $q, q' \in S$ such that $f(q, q') \neq \#$. We obtain that $\vec{v} \in \chi(\mathsf{tf}_{k+1})$ so that $\vec{v} \in I_{k+1}$. Observe that, for such values of q and q', $\vec{v}_j(q, q') = \max(j, f(q, q'))$, so that $\vec{v} \leqslant_\times \vec{v}_j$. Moreover, $\vec{v}_j$ is equal to $\vec{v}$ on components that are not in E, because by definition

3. In fact, this argument is the reason why we enforced that $V_0 \subseteq U_k$.

$E = \mathsf{index}(S)$. By definition of E, the representing vector of I_{k+1} is equal to ω on all components in E, so that membership in E is not sensitive to the values at these components; this proves that $\vec{v}_j \in I_{k+1}$. □

The following lemma is where the magic really happens:

Lemma 3.54. *For all $j \in \mathbb{N}$, there is $m \in \mathbb{N}$ such that $\mathsf{tf}_{k+1}^{(m)} \in \mathsf{tf}_k^{(j)} \otimes t$.*

Proof. We proceed by induction on j. For $j = 0$, $\mathsf{tf}_k^{(0)} = \mathsf{tf}_k$, $\mathsf{tf}_{k+1}^{(0)} = \mathsf{tf}_{k+1}$ and indeed $\mathsf{tf}_{k+1} \in \mathsf{tf}_k \otimes t$ so that the property holds by letting $m = 0$.

We suppose that the property is true for j, and we prove it for $j+1$. By induction hypothesis, there is m such that $\mathsf{tf}_{k+1}^{(m)} \in \mathsf{tf}_k^{(j)} \otimes t$; let $H_j : Q^3 \to \mathbb{N}_\#$ be a witness function of that. We build a function $H_{j+1} : Q^3 \to \mathbb{N}_\#$ as follows. For every $(q_1, q_2) \in Q^2$:

— if $f_k^{(j)}(q_1, q_2) \neq j$ or $f_k^{(j+1)}(q_1, q_2) \neq j+1$, we set $H_{j+1}(q_1, q_2, q_3) \coloneqq H_j(q_1, q_2, q_3)$ for all q_3;

— if $f_k^{(j)}(q_1, q_2) = j$ and $f_k^{(j+1)}(q_1, q_2) = j+1$, we set $H_{j+1}(q_1, q_2, q_2) \coloneqq H_j(q_1, q_2, q_2)+1$ and, for all $q_3 \neq q_2$, $H_{j+1}(q_1, q_2, q_3) \coloneqq H_j(q_1, q_2, q_3)$.

There is a subtlety in the second case: it could be that $H_j(q_1, q_2, q_2) = \#$, in which case we set $H_{j+1}(q_1, q_2, q_2) = 1$ (recall that $\# + 1 = 1$). We therefore do not always have $H_{j+1} \geqslant H_j$. Let $f : (q_1, q_3) \mapsto \sum_{q_2} H_{j+1}(q_1, q_2, q_3)$. We claim that $(f, \ell_{k+1}, \ell'_{k+1}) \in \mathsf{tf}_k^{(j+1)} \otimes t$, with H_{j+1} as witness function.

First, we prove that, for all q_1, q_2, $\sum_{q_3} H_{j+1}(q_1, q_2, q_3) \geqslant f_k^{(j+1)}(q_1, q_2)$. Let $q_1, q_2 \in Q$. Because H_j is a witness function that $\mathsf{tf}_{k+1}^{(m)} \in \mathsf{tf}_k^{(j)} \otimes t$, we have $\sum_{q_3} H_j(q_1, q_2, q_3) \geqslant f_k^{(j)}(q_1, q_2)$. If $f_k^{(j)}(q_1, q_2) \neq j$ or $f_k^{(j+1)}(q_1, q_2) \neq j+1$, we have $f_k^{(j+1)}(q_1, q_2) = f_k^{(j)}(q_1, q_2)$ and $H_{j+1}(q_1, q_2, q_3) = H_j(q_1, q_2, q_3)$ for all q_3, so that $\sum_{q_3} H_{j+1}(q_1, q_2, q_3) = \sum_{q_3} H_j(q_1, q_2, q_3) \geqslant f_k^{(j)}(q_1, q_2) = f_k^{(j+1)}(q_1, q_2)$. If $f_k^{(j)}(q_1, q_2) = j$ and $f_k^{(j+1)}(q_1, q_2) = j+1$, then $\sum_{q_3} H_j(q_1, q_2, q_3) \geqslant j$ and $\sum_{q_3} H_{j+1}(q_1, q_2, q_3) = \sum_{q_3} H_j(q_1, q_2, q_3) + 1 \geqslant j+1$.

Let $t =: (f_t, \ell_t, \ell'_t)$. We now prove that, for all q_2, q_3, $\sum_{q_1} H_{j+1}(q_1, q_2, q_3) \geqslant f_t(q_2, q_3)$. We know that this is true for H_j. For every $q_2 \neq q_3$, we have $\sum_{q_1} H_{j+1}(q_2, q_3) = \sum_{q_1} H_j(q_2, q_3) \geqslant f_t(q_2, q_3)$. For every q_2, we have either $\sum_{q_1} H_{j+1}(q_1, q_2, q_2) = \sum_{q_1} H_j(q_1, q_2, q_2)$ or $\sum_{q_1} H_{j+1}(q_1, q_2, q_2) = \sum_{q_1} H_j(q_1, q_2, q_2) + 1$. By idle-compliance, we have $f_t(q_2, q_2) \neq \#$ so that $\sum_{q_1} H_j(q_1, q_2, q_2) \neq \#$, therefore $\sum_{q_1} H_{j+1}(q_1, q_2, q_2) \geqslant \sum_{q_1} H_j(q_1, q_2, q_2) \geqslant f_t(q_2, q_2)$. Note that we need idle-compliance here: if we had $f_t(q_2, q_2) = \#$ then we would have $\sum_{q_1} H_j(q_1, q_2, q_2) = \#$ but $\sum_{q_1} H_{j+1}(q_1, q_2, q_2) = 1$ so that $\sum_{q_1} H_{j+1}(q_1, q_2, q_2)$ would be incomparable with $f_t(q_2, q_2)$.

It remains to prove that H_{j+1} is a witness function that $\mathsf{tf}_{k+1}(m') \in \mathsf{tf}_k^{(j+1)} \otimes t$ for some m'. To do that, let $f : (q_1, q_3) \mapsto \sum_{q_2} H_{j+1}(q_1, q_2, q_3)$ and $\mathsf{tf}'_{k+1} \coloneqq (f, \ell_{k+1}, \ell'_{k+1})$. By the above arguments, we know that $\mathsf{tf}'_{k+1} \in \mathsf{tf}_k^{(j+1)} \otimes t$. It therefore suffices to prove that there exists m' such that $(f, \ell_{k+1}, \ell'_{k+1}) \preceq (f_{k+1}^{(m')}, \ell_{k+1}, \ell'_{k+1}) = \mathsf{tf}_{k+1}^{(m')}$. Indeed, this would imply that $\mathsf{tf}_{k+1}^{(m')} \in \mathsf{tf}_k^{(j+1)} \otimes t$ by Lemma 3.9.

We now prove that there is $m' \in \mathbb{N}$ such that $\mathsf{tf}'_{k+1} \preceq \mathsf{tf}_{k+1}^{(m')}$. Let $q_1, q_3 \in Q$. If $(q_1, q_3) \notin S$ then we must have $f_k^{(j)}(q_1, q_2) = f_k^{(j+1)}(q_1, q_2)$ so that, by definition of H_{j+1}, $\sum_{q_2} H_{j+1}(q_1, q_2, q_3) = \sum_{q_2} H_j(q_1, q_2, q_3) = f_{k+1}^{(m)}(q_1, q_3) = f_{k+1}^{(m')}(q_1, q_3)$ for all m'. Suppose now that $(q_1, q_3) \in S$. There are two cases: $f_{k+1}(q_1, q_3) = \#$ and $f_{k+1}(q_1, q_3) \neq \#$.

Let $(q_1, q_3) \in S$ such that $f_{k+1}(q_1, q_3) = \#$. We must prove that $f(q_1, q_3) = \#$. Recall that we have $\mathsf{tf}_{k+1} \in \mathsf{tf}_k \otimes t$. Upon defining the compositional product $\otimes$ in Definition 3.8, we have made sure that $\mathsf{tf}_{k+1} \in \mathsf{tf}_k \otimes t$ implies that, for all q, q', if $f_{k+1}(q, q') = \#$ then $f_k(q, q') = \#$. This proves that $f_k(q_1, q_3) = \#$. By definition of $f_k^{(j+1)}$, this implies that $f_k^{(j+1)}(q_1, q_3) = \#$. In particular, $f_k^{(j+1)}(q_1, q_3) \neq j + 1$ so that, by definition of H_{j+1}, for all q_2, we have $H_{j+1}(q_1, q_2, q_3) = H_j(q_1, q_2, q_3)$ for all q_2. However, $\sum_{q_2} H_j(q_1, q_2, q_3) = f_{k+1}^{(m)}(q_1, q_3) = \#$, so that $f(q_1, q_3) = \sum_{q_2} H_{j+1}(q_1, q_2, q_3) = \sum_{q_2} H_j(q_1, q_2, q_3) = \#$.

Let $(q_1, q_2) \in S$ such that $f_{k+1}(q_1, q_3) \neq \#$; we have $f_{k+1}^{(m')}(q_1, q_3) = \max(f_{k+1}(q_1, q_3), m')$ for all m'. Also, $f_{k+1}^{(m)}(q_1, q_3) \neq \#$ therefore $\sum_{q_2} H_j(q_1, q_2, q_3) \neq \#$. By definition of H_{j+1}, this also implies $\sum_{q_2} H_{j+1}(q_1, q_2, q_3) \neq \#$ so that $f(q_1, q_3) \neq \#$. For $m' \geqslant f(q_1, q_3)$, we have $f(q_1, q_3) \leqslant f_{k+1}^{(m')}(q_1, q_3)$.

Let m' large enough so that, for every q_1, q_3 such that $f(q_1, q_3) \neq \#$, $m' \geqslant f(q_1, q_3)$. We have proved that $\mathsf{tf}'_{k+1} \preceq \mathsf{tf}_{k+1}^{(m')}$. This implies that $\mathsf{tf}_{k+1}^{(m')} \in \mathsf{tf}_k^{(j+1)} \otimes t$, concluding the induction. □

We now claim that, for all $j \in \mathbb{N}$, $\chi(\mathsf{tf}_k^{(j)}) \cap U_k = \emptyset$. Indeed, suppose by contradiction that this is not the case: let j such that $\chi(\mathsf{tf}_k^{(j)}) \cap U_k \neq \emptyset$. By Lemma 3.21 and Lemma 3.20, this implies that $\mathsf{tf}_k^{(j)} \in \mathcal{T}^{\leqslant k}$. We apply Lemma 3.54 to obtain m such that $\mathsf{tf}_{k+1}^{(m)} \in \mathsf{tf}_k^{(j)} \otimes t$, so that $\mathsf{tf}_{k+1}^{(m)} \in \mathcal{T}^{\leqslant k+1}$. We have $\chi(\mathsf{tf}_{k+1}^{(m)}) \subseteq U_{k+1}$, but by Lemma 3.53 $\chi(\mathsf{tf}_{k+1}^{(m)}) \subseteq I_{k+1}$. Because $\chi(\mathsf{tf}_{k+1})^{(m)}$ is not empty by Lemma 3.20, this contradicts that $I_{k+1} \subseteq D_{k+1}$.

We have proved that $\chi(\mathsf{tf}_k^{(j)}) \subseteq D_k$ for every j. Recall that our objective is to exhibit an ideal I_k proper at step k whose representing vector is equal to ω at any component in E. Let $\vec{v}_{k+1}$ be an arbitrary vector in $\chi(\mathsf{tf}_{k+1})$, and $\vec{v}_k$ be an arbitrary vector of $\chi(\mathsf{tf}_k)$.

Lemma 3.55. *For every $j \in \mathbb{N}$:*

— $\vec{v}_{k+1}^{(j)} \in I_{k+1}$,

— $\vec{v}_k^{(j)} \in U_{k+1} \setminus U_k$.

Proof. Let $j \in \mathbb{N}$. By Lemma 3.52, $\vec{v}_k^{(j)} \in \chi(\mathrm{tf}_k^{(j)})$ and $\vec{v}_{k+1}^{(j)} \in \chi(\mathrm{tf}_{k+1}^{(j)})$. By Lemma 3.53, this directly proves that $\vec{v}_{k+1}^{(j)} \in I_{k+1}$.

We have $\chi(\mathrm{tf}_k) \subseteq U_{k+1}$ therefore $v_k \in U_{k+1}$, thus $\vec{v}_k^{(j)} \in U_{k+1}$ because U_{k+1} is upward-closed. Suppose by contradiction that we have $\vec{v}_k^{(j)} \in U_k$. This would imply that $\chi(\mathrm{tf}_k^{(j)}) \cap U_k \neq \emptyset$; by Lemma 3.21 and Lemma 3.20, this implies that $\mathrm{tf}_k^{(j)} \in \mathcal{T}^{\leqslant k}$. By Lemma 3.54, there is m such that $\mathrm{tf}_{k+1}^{(m)} \in \mathrm{tf}_k^{(j)} \otimes t$, so that $\mathrm{tf}_{k+1}^{(m)} \in \mathcal{T}^{\leqslant k+1}$. This implies that $\chi(\mathrm{tf}_{k+1}^{(m)}) \subseteq U_{k+1}$, which contradicts Lemma 3.53 since $I_{k+1} \subseteq D_{k+1}$. □

Let $\vec{u}_k$ be the vector such that, for all $i \in [\![1, d]\!]$, $\vec{u}_k(i) \coloneqq \omega$ if $i \in E$ and $\vec{u}_k(i) \coloneqq \vec{v}_k(i)$ if $i \notin E$. Let J be the ideal represented by $\vec{u}_k$, *i.e.*, $J \coloneqq \{\vec{u} \in \mathbb{N}^d \mid \vec{u} \leqslant_\times \vec{u}_k\}$. In particular, J contains vector $\vec{v}_k^{(j)}$ for every $j \in \mathbb{N}$, which are all in $U_{k+1} \setminus U_k$ by Lemma 3.55. This implies that $J \not\subseteq D_{k+1}$. We now prove that $J \subseteq D_k$. Let $\vec{u} \in J$, and let $j \coloneqq \|\vec{u}\|$. We have $\vec{u} \leqslant_\times \vec{v}_k^{(j)}$: for all $i \notin E$, $\vec{u}(i) \leqslant \vec{u}_k(i) = \vec{v}_k^{(j)}(i)$, and for $i \in E$, $\vec{u}_k(i) \leqslant \|\vec{u}\| = j \leqslant \vec{v}_k^{(j)}(i)$. Because $\vec{v}_k^{(j)} \in D_k$ by Lemma 3.55, we have proved that $\vec{v} \in D_k$. That being true for all $\vec{v} \in J$, this proves that $J \subseteq D_k$. In particular, J is contained in some ideal I_k in the decomposition of D_k; because $J \not\subseteq D_{k+1}$, I_k is proper at step k. The representing vector of J is equal to ω on every $i \in E$, therefore the same is true for the representing vector of I_k, concluding the proof of Lemma 3.26.

CHAPTER 4

ROUND-BASED ASMS

4.1 Introduction

A classic workaround to impossibility results for asynchronous consensus [FLP85; LA87; DDS87] is to rely on randomization and to require termination with probability 1 but not in every execution. Randomized consensus algorithms typically work with rounds. Rounds correspond to iterations of a `for` loop: each round has non-zero probability to succeed but also a non-zero probability to fail, so that infinitely many rounds are necessary for almost-sure termination. When the systems under consideration are asynchronous, rounds are also asynchronous: processes can be at different rounds and there is no *a priori* bound in the difference of rounds between processes. The first algorithm to work under this paradigm is Ben-Or's algorithm [Ben83] and many others have followed (*e.g.*, [BT85; Bra87; AH90; CR93]).

In this thesis, we are interested in the particular case of round-based shared-memory algorithms [AH90; Asp02; GR07; RS12]. The algorithm of [Asp02] is of particular interest to us: in this algorithm, each round has its own set of registers. Therefore, the overall number of registers is unbounded. We will present this algorithm in details in Section 4.2. We use this algorithm as a motivating example to extend the (roundless) ASMS model to capture round-based algorithms. This extension is needed, because roundless ASMS cannot encode the round values of the processes nor the infinite set of registers. In the round-based ASMS model, each round has its own set of registers, and processes may only interact with registers of nearby rounds. We define round-based PRP, an adaptation of PRP from Chapter 2 to the round-based setting where one allows for quantification over the rounds but where quantifiers cannot be nested. We prove that this problem is PSPACE-complete. The PSPACE membership is not easy, as one cannot guess the execution configuration by configuration. Instead, our procedure guesses the execution footprint by footprint, where *footprints* correspond to the projections of the executions onto small windows of rounds. This relies on the hypothesis that the integers constants of the input are given in unary, a reasonable hypothesis given our motivations. If the integers constants are encoded in binary, then we prove that round-based PRP becomes EXPSPACE-complete.

This chapter is organized as follows. In Section 4.2, we present in detail the algorithm from [Asp02]. In Section 4.3, we define round-based ASMS and round-based PRP. In Section 4.4, we present exponential lower bounds and a PSPACE-hardness result for Cover. In Section 4.5, we provide a non-deterministic polynomial-space procedure for Cover. We extend this result to round-based PRP in Section 4.6. In Section 4.7, we discuss the impact of unary encoding. We conclude the chapter with some perspectives in Section 4.8. Most of the results presented in this chapter have been published in [BMSW22; Wal23].

Related works The formal methods community has studied various approach to verification of asynchronous round-based randomized algorithms, such as *ad hoc* analysis [PSL00], probabilistic model checking with a fixed number of processes [KNS01; KN02] using PRISM [KNP11] or verification in Isabelle [CDM11]. However, there is little work related to parameterized verification of asynchronous round-based algorithms. The closest works are for parameterized verification of round-based Byzantine agreement algorithms such as Ben-Or's algorithm. In [BKLW21], the authors consider a model for such algorithms in which, for non-probabilistic properties, one may in fact consider that processes behave synchronously in rounds, so that all processes move from round $k-1$ to round k consecutively. This simplification, called *round-rigid adversary* in [BKLW21], does not apply to our model. Another related work is [BTW21], where round-based Byzantine agreement algorithms are abstracted into so-called layered threshold automata, yielding an incomplete verification technique.

4.2 Aspnes' Noisy Consensus Algorithm

We here explain in more detail the round-based consensus algorithm from [Asp02], which we use as a guideline to define round-based ASMS. This algorithm is meant to work in a so-called noisy environment, which is why we refer to it as *Aspnes' noisy consensus algorithm*.

The pseudocode of this algorithm is given in Algorithm 2. It proceeds in asynchronous rounds. This means that each process has a private round value, denoted k in Algorithm 2, that starts at 0 and only increases. Because this value is private, processes can be at different rounds and there is no *a priori* bound on the round difference between pairs of processes. Processes communicate via writing to and reading from shared registers. Each round k has its own shared registers: $\mathsf{reg}_0[k]$ and $\mathsf{reg}_1[k]$. All registers are initialized to value $\bot$, and within an execution, their value may only be updated to $\top$. Intuitively, $\mathsf{reg}_i[k] = \top$ if i is a proposed consensus value at round k. The algorithm essentially consists in a race: if a process manages to get to a

round high enough compared to all disagreeing processes, then it imposes its preference as the consensus value.

```
1  int k ← 0, bool p ∈ {0,1}, (reg_b[k])_{b∈{0,1},k∈ℕ} all initialized to ⊥;
2  while true do
3      read from reg_0[k] and reg_1[k];
4      if reg_0[k] = ⊤ and reg_1[k] = ⊥ then p ← 0;
5      else if reg_0[k] = ⊥ and reg_1[k] = ⊤ then p ← 1;
6      write ⊤ to reg_p[k];
7      if k > 0 then
8          read from reg_{1−p}[k−1];
9          if reg_{1−p}[k−1] = ⊥ then return p;
10     k ← k+1;
```

Algorithm 2: Aspnes' noisy consensus algorithm [Asp02].

We now explain the algorithm in more detail. As usual in distributed consensus algorithms, each process starts with a preference value p. At each round, a process starts by reading the value of the shared registers of that round (line 3). If exactly one of them is set to $\top$, the process updates its preference p to the corresponding value (lines 4 and 5). In all cases, it writes $\top$ to the register of the current round that corresponds to its preference p (line 6). Then, it reads the register of the previous round corresponding to the opposite preference $1-p$ (line 8), and if it is equal to $\bot$, the process returns its preference p for the consensus (line 9). To be able to return its current preference value, a process must write to a register of its current round k while no other process has written to the register of round $k-1$ for the opposite value. In other words, to decide, the process must be enough ahead, in terms of rounds, of all disagreeing processes.

In this algorithm, it is assumed that processes are *not Byzantine*, *i.e.*, that they follow the pseudocode above faithfully. There is however, one exception in that processes may *crash* at any point: a process that crashes stops prematurely. A process that does not crash is called *non-faulty*. The expected properties of a consensus algorithm are *validity*, *agreement* and *termination* [PSL80; Asp02]. Validity expresses that if all processes start with the same preference p, then no process can return a value different from p. Agreement expresses that no two processes can return different values. Finally, termination expresses that eventually all non-faulty processes should return a value. Because Aspnes' noisy consensus algorithm consists in a race between processes, one cannot guarantee termination of every execution. The termination of Aspnes' algorithm requires some additional constraints on the scheduler, which is why the algorithm is designed to work in a so-called noisy environment [Asp02] where

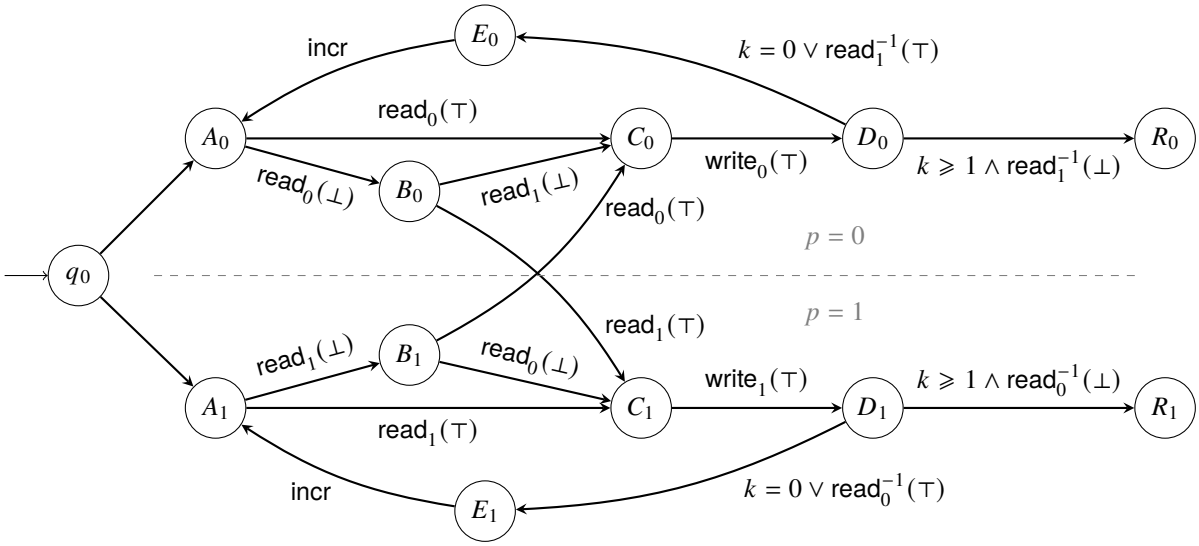


Figure 4.1 – A round-based protocol $\mathcal{P}_{\text{Asp}}$ for Aspnes' noisy consensus algorithm. Since the first round ($k = 0$) slightly differs from the others, to avoid duplication of the state space, we allow for guards related to whether $k = 0$ or $k \geqslant 1$ in the transition labels.

the relative computation speeds of the processes tend to vary. This is abstracted into a stochastic scheduler. In this case, termination is proved to occur almost surely under the assumptions that the stochastic scheduler is continuous and satisfies some reasonable properties [Asp02]. This makes termination a probabilistic property; such properties are the topic of Chapter 5. In the current chapter, we focus on non-probabilistic properties such as validity and agreement, which must hold unconditionally, *i.e.*, regardless of the choices made by the scheduler. The objective of this chapter is to study problems related to the automated verification of such properties.

For a single round – corresponding to one iteration of the while loop of line 2 – the system can be seen as a roundless ASMS so that the techniques from Chapter 2 apply. If the while loop was guarded by a bound on k, for example if it was `while` $k \leqslant K$ for some constant K, then one could model the algorithm as a roundless ASMS with $O(K)$ states and $O(K)$ registers. In general, such an encoding is not possible: the system has unboundedly many rounds and thus unboundedly many shared registers. Therefore, this calls for a new model and new verification techniques.

The model studied here is an extension of roundless ASMS meant to model round-based algorithms. We dub this model round-based ASMS, and we introduce it formally in the next section. In Fig. 4.1, we depict a protocol encoding Aspnes' algorithm in this new model.

4.3 Round-based Asynchronous Shared-Memory Systems

4.3.1 Round-Based Protocols

We start with a formal definition of round-based asynchronous shared-memory systems. In this round-based model, there is a fresh set of dim registers at each round, and each process has its own private round value that starts at 0 and can only be incremented. A process may only write to registers of its current round and read from registers of nearby rounds; this will be made more formal in the upcoming definitions.

Definition 4.1 (Round-based Asynchronous Shared-Memory Systems)**.** A *round-based asynchronous shared-memory system* (round-based ASMS for short) is described by a *round-based protocol* which is a tuple $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \perp, \Delta \rangle$ where

- Q is a finite set of states with a distinguished initial state $q_0 \in Q$;
- $\mathsf{dim} \in \mathbb{N}$ is the number of shared registers per round;
- $\mathbb{D}$ is a finite data alphabet with an *initial symbol* $\perp$;
- $\Delta \subseteq Q \times \mathcal{A} \times Q$ is a finite set of transitions, where $\mathcal{A} = \{\mathsf{read}_r^{-i}(\mathsf{d}) \mid i \in \mathbb{N}, r \in [\![1, \mathsf{dim}]\!], \mathsf{d} \in \mathbb{D}\} \cup \{\mathsf{write}_r(\mathsf{d}) \mid r \in [\![1, \mathsf{dim}]\!], \mathsf{d} \in \mathbb{D} \setminus \{\perp\}\} \cup \{\mathsf{incr}\} \cup \{\circledast\}$ is the set of actions.

Note that this definition is almost identical to the one of roundless protocols (Definition 2.1). There are, however, two differences. The first one is the addition of a fourth type of action: *round increment* transitions, written incr. These transitions allow a process in round k to go to round $k + 1$; this corresponds to line 10 in Algorithm 2. Another difference can be found in the definition of read actions. Indeed, they now have an additional value, written i above, that specifies to what round they refer. This round is given in relative value with respect to the current round of the process. The action $\mathsf{read}_r^{-i}(\mathsf{d})$ can be read as: "read symbol d from register r of round $k - i$" where k denotes the current round of the process. If $i = 0$ then we simply write $\mathsf{read}_r(\mathsf{d})$. Write actions always refer to the registers of the current round of the process.

We denote by $\mathsf{rg}_r[k]$ register r of round k. Let $\mathsf{Reg}_k \coloneqq \{\mathsf{rg}_r[k] \mid r \in [\![1, \mathsf{dim}]\!]\}$ be the set of registers of round k, and let $\mathsf{Reg} \coloneqq \{\mathsf{rg}_r[k] \mid r \in [\![1, \mathsf{dim}]\!], k \in \mathbb{N}\}$ be the set of all registers. We use **reg** to denote a variable in Reg. We let

$$v \coloneqq \max\{i \in \mathbb{N} \mid \exists r, \mathsf{d}, q, q', (q, \mathsf{read}_r^{-i}(\mathsf{d}), q') \in \Delta\}$$

with the convention that $\max \emptyset = 0$. Because Δ is a finite set, $v < \infty$. We call v the *visibility range* of $\mathcal{P}$.

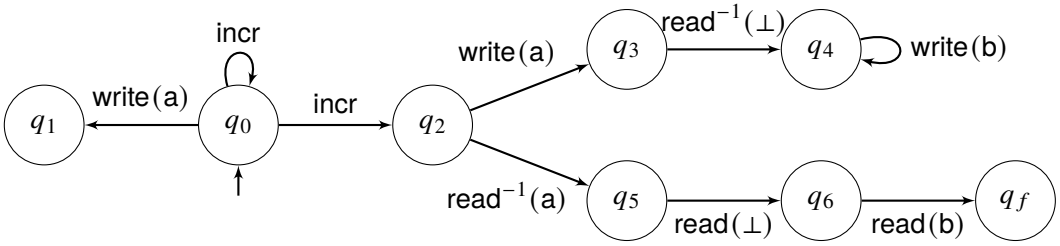


Figure 4.2 – An example of a round-based protocol.

The size of a protocol is $|\mathcal{P}| = |Q| + |\mathbb{D}| + |\Delta| + v + \mathsf{dim}$. Note that v appears in the sum: we assume that i is encoded in unary in the actions $\mathsf{read}_r^{-i}(\mathsf{d})$. We justify this decision by the fact that v is typically small: in round-based algorithms from the literature, one rarely interacts with rounds far away. In particular, in Aspnes' noisy consensus algorithm, we have $v = 1$.

Example 4.2. An example of a round-based protocol is depicted in Fig. 4.2. In this example, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}$, $\mathbb{D} = \{\perp, \mathsf{a}, \mathsf{b}\}$ and $\mathsf{dim} = 1$, so that there is only one register per round and we omit the subscript indicating the index $r \in [\![1, \mathsf{dim}]\!]$. All read and write actions interact with the register of the current round k of the process, with the exception of the read transitions from q_3 to q_4 and from q_2 to q_5, which read from the register of the previous round $k - 1$. Here, $v = 1$: a process in round k can no longer read from registers of rounds 0 to $k - 2$.

Another example of round-based protocol is the one from Fig. 4.1 that encodes Aspnes' noisy consensus algorithm. In the represented protocol, $Q = \{q_0\} \cup \{A_b, B_b, C_b, D_b, E_b, R_b \mid b \in \{0, 1\}\}$. However, the state space should in fact be twice bigger because we must encode in the state space whether $k = 0$ or $k \geqslant 1$. In this protocol, $\mathbb{D} = \{\perp, \top\}$, $\mathsf{dim} = 2$ and $v = 1$.

4.3.2 Semantics

An important difference with Chapter 2 is that the situation of a process at a given point in time is no longer only described by its state, but also by its round value. Formally, the relevant information about a process at a given point in time takes the form of a pair $(q, k) \in Q \times \mathbb{N}$ called *location*. Let $\mathcal{L} := Q \times \mathbb{N}$ be the set of locations. A *configuration* describes the number of processes in each location along with the value of each register. Formally, a *configuration* is a pair $\langle \mu, \vec{d} \rangle$ with $\mu \in \mathcal{M}(\mathcal{L})$ such that $|\mu| > 0$ and $\vec{d} \in \mathbb{D}^{\mathsf{Reg}}$. For $\gamma = \langle \mu, \vec{d} \rangle$, we write $\mathsf{loc}(\gamma) := \mu$, $\mathsf{data}(\gamma) := \vec{d}$ and $\mathsf{supp}(\gamma) := \overline{\mu}$. The *size* of γ is $|\gamma| := |\mu|$. We write Γ for the set of configurations. For every $n \geqslant 1$, we let $\gamma_0(n) := \langle n \cdot (q_0, 0), \perp^{\mathsf{Reg}} \rangle$ be the *initial configuration* with n processes. The set of initial configurations is $\Gamma_0 := \{\gamma_0(n) \mid n \geqslant 1\}$. Observe that configurations of ASMS cannot in general be stored in finite space, because the set

Reg is infinite. Reachable configurations, however, will only have finitely many registers set to values in $\mathbb{D} \setminus \{\bot\}$.

Towards defining the operational semantics of round-based ASMS, we define the notion of *move*. A *move* is a pair $\theta \in \Delta \times \mathbb{N}$: move (δ, k) expresses that transition δ is taken by a process at round k. We write $\mathcal{M} := \Delta \times \mathbb{N}$ for the set of all moves. A move (δ, k) is called *at round* k. A move θ has *effect* on a given round $k \in \mathbb{N}$ if it is related to round k in any way. Formally, a move θ has *effect* on round k when one of the following conditions is satisfied:

— $\theta = (\delta, k)$ for some δ, or

— $\theta = ((q, \mathsf{incr}, q'), k{-}1)$ for some q and q', or

— $\theta = ((q, \mathsf{read}_r^{-i}(\mathsf{d}), q'), \ell)$ for some q, q', r, ℓ, i such that $\ell - i = k$.

We define a step as follows: for $\theta = ((q, a, q'), k) \in \mathcal{M}$, $\gamma \xrightarrow{\theta} \gamma'$ when $(q, k) \in \mathsf{loc}(\gamma)$ and:

— if $a = \mathsf{read}_r^{-i}(\mathsf{d})$, $\mathsf{loc}(\gamma') = (\mathsf{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k)\}$, $\mathsf{data}(\gamma)(\mathsf{rg}_r[k{-}i]) = \mathsf{d}$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$;

— if $a = \mathsf{write}_r(\mathsf{d})$, $\mathsf{loc}(\gamma') = (\mathsf{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k)\}$, $\mathsf{data}(\gamma')(\mathsf{rg}_r[k]) = \mathsf{d}$ and for all $\mathbf{reg} \neq \mathsf{rg}_r[k]$, $\mathsf{data}(\gamma')(\mathbf{reg}) = \mathsf{data}(\gamma)(\mathbf{reg})$;

— if $a = \mathsf{incr}$, $\mathsf{loc}(\gamma') = (\mathsf{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k{+}1)\}$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$,

— if $a = \circledast$, $\mathsf{loc}(\gamma') = (\mathsf{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k)\}$ and $\mathsf{data}(\gamma') = \mathsf{data}(\gamma)$.

Location (q, k) is called the *source location* of the step, while the *destination location* is (q', k) if $a \neq \mathsf{incr}$ and $(q', k + 1)$ if $a = \mathsf{incr}$. A step is *at round* k when the corresponding move is of the form (δ, k). This step is obtained by *applying* move (δ, k). When γ' exists, (δ, k) *can be applied* from γ; in this case, γ' is unique.

In the read steps above, we have not specified what happens if the round of the read register is negative, *i.e.*, if $k - i < 0$. For simplicity, we consider that registers of negative rounds always contain value $\bot$, so that if $k - i < 0$ then action $\mathsf{read}_r^{-i}(\mathsf{d})$ is only possible if $\mathsf{d} = \bot$.

The definition of execution, reachability and coverability are similar to the ones of Chapter 2, except that the notion of transition is replaced with the one of move. An *execution* is a sequence $\rho = \gamma_0, \theta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \theta_\ell, \gamma_\ell$ such that, for all i, $\gamma_i \xrightarrow{\theta_{i+1}} \gamma_{i+1}$. Its *length* $\mathsf{len}(\rho)$ is simply defined as its number ℓ of steps. We write $\gamma_0 \xrightarrow{*} \gamma_\ell$ for the existence of an execution from γ_0 to γ_ℓ. Given an execution $\rho = \gamma_0, \theta_1, \gamma_1, \ldots, \gamma_{\ell-1}, \theta_\ell, \gamma_\ell$, the configurations $\gamma_0, \ldots, \gamma_\ell$ are the configurations *visited* in ρ. A configuration γ' is *reachable from* γ when $\gamma \xrightarrow{*} \gamma'$. Given a set $C \subseteq \Gamma$, we write $\mathsf{Post}^*(C) := \{\gamma' \mid \exists \gamma \in C,\ \gamma \xrightarrow{*} \gamma'\}$. Dually, we write $\mathsf{Pre}^*(C) := \{\gamma \mid \exists \gamma' \in C, \gamma \xrightarrow{*} \gamma'\}$. A configuration is *reachable* when it belongs to $\mathsf{Post}^*(\Gamma_0)$. A location $(q, k) \in \mathcal{L}$ is *coverable*

from γ if there is an execution $\rho : \gamma \xrightarrow{*} \gamma'$ where $\mathsf{loc}(\gamma')(q,k) > 0$. A state q is *coverable from* γ if there is $k \in \mathbb{N}$ such that (q,k) is coverable from γ. In this case, execution ρ *covers* location (q,k) and state q. A state q (or a location (q,k)) is coverable when it is coverable from Γ_0. Given a configuration γ, we call a register $\mathbf{reg} \in \mathsf{Reg}$ *blank* when $\mathsf{data}(\gamma)(\mathbf{reg}) = \bot$.

Example 4.3. Consider again the protocol $\mathcal{P}$ depicted in Fig. 4.2. We give two examples of executions. State q_4 is coverable from $\gamma_0(1)$ with the following execution:

$$\rho_1 : \gamma_0(1) = \left\langle (q_0,0), \begin{smallmatrix}\mathsf{rg}[0]=\bot\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle \xrightarrow{(q_0,\mathsf{incr},q_2),0} \left\langle (q_2,1), \begin{smallmatrix}\mathsf{rg}[0]=\bot\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle \xrightarrow{(q_2,\mathsf{write}(a),q_3),1}$$
$$\left\langle (q_3,1), \begin{smallmatrix}\mathsf{rg}[0]=\bot\\ \mathsf{rg}[1]=a\end{smallmatrix} \right\rangle \xrightarrow{(q_3,\mathsf{read}^{-1}(\bot),q_4),1} \left\langle (q_4,1), \begin{smallmatrix}\mathsf{rg}[0]=\bot\\ \mathsf{rg}[1]=a\end{smallmatrix} \right\rangle.$$

State q_6 is coverable from $\gamma_0(2)$ with the following execution:

$$\rho_2 : \gamma_0(2) = \left\langle 2\cdot(q_0,0), \begin{smallmatrix}\mathsf{rg}[0]=\bot\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle \xrightarrow{(q_0,\mathsf{write}(a),q_1),0} \left\langle (q_0,0)\oplus(q_1,0), \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle$$
$$\xrightarrow{(q_0,\mathsf{incr},q_2),0} \left\langle (q_2,1)\oplus(q_1,0), \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle \xrightarrow{(q_2,\mathsf{read}^{-1}(a),q_5),1} \left\langle (q_5,1)\oplus(q_1,0), \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle$$
$$\xrightarrow{(q_5,\mathsf{read}(\bot),q_6),1} \left\langle (q_6,1)\oplus(q_1,0), \begin{smallmatrix}\mathsf{rg}[0]=a\\ \mathsf{rg}[1]=\bot\end{smallmatrix} \right\rangle.$$

The content of registers $\mathsf{rg}[k]$ with $k > 1$ are not represented as these registers remain blank.

This proves that states q_4 and q_6 can be covered. However, no execution can cover both state q_4 and state q_6 *at the same round*, regardless of the number of processes, thus preventing from covering q_f. Formally, for all $k \in \mathbb{N}$, there is no $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\mathsf{loc}(\gamma)(q_4,k) > 0$ and $\mathsf{loc}(\gamma)(q_6,k)$. Indeed, in order to cover (q_4,k), one must write a to $\mathsf{rg}[k]$ then read $\bot$ from $\mathsf{rg}[k-1]$, while in order to reach (q_6,k), one must read a from $\mathsf{rg}[k-1]$ then read $\bot$ from $\mathsf{rg}[k]$. In other words, covering (q_4,k) requires that $\mathsf{rg}[k]$ is written *before* $\mathsf{rg}[k-1]$, whereas covering (q_6,k) requires that $\mathsf{rg}[k]$ is written *after* $\mathsf{rg}[k-1]$.

4.3.3 Problems of Interest

The two simplest problems are, as in Chapter 2, Cover and Target.

Round-based Cover
Input: A round-based protocol $\mathcal{P}$, $q_f \in Q$
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ and $k \in \mathbb{N}$ such that $\mathsf{loc}(\gamma)(q_f,k) > 0$?

ROUND-BASED TARGET
Input: A round-based protocol $\mathcal{P}$, $q_f \in Q$
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ s.t. for all $q \neq q_f$ and $k \in \mathbb{N}$, $\mathsf{loc}(\gamma)(q,k) = 0$?

We now generalize presence constraints from Chapter 2 to round-based ASMS. To do so, we need to choose how rounds are specified in the formulas. We choose to allow quantification over rounds, because the value k is indeed quantified over $\mathbb{N}$ in the two problems above. However, we forbid nested quantifiers in order to restrict the logical power of the presence constraints and to simplify the analysis.

We start with the definitions of a few auxiliary notions. A *term* is of the form m or $k{+}m$ with $m \in \mathbb{N}$ and k a free variable. An *atomic proposition* is either of the form "$\mathsf{popu}(q,t)$" with t a term and $q \in Q$ or of the form "$\mathsf{cont}(\mathsf{rg}_r[t],\mathsf{d})$" with t a term, $r \in [\![1,\mathsf{dim}]\!]$ and $\mathsf{d} \in \mathbb{D}$. A *proposition* is a Boolean combination of atomic propositions that has at most one free variable. An *atomic presence constraint* is either a *closed* proposition (no free variables), or of the form "$\exists k\ \phi$" or "$\forall k\ \phi$" where ϕ is a proposition with k as a free variable. A closed proposition is of the form (negation of) "$\mathsf{popu}(q,m)$" or (negation of) "$\mathsf{cont}(\mathsf{rg}_r[m],\mathsf{d})$" for some $m \in \mathbb{N}$; such an closed proposition is *related to round* m. A *presence constraint* is a Boolean combination of atomic presence constraints. A presence constraint is interpreted over a configuration $\gamma \in \Gamma$ as follows:

— for all $q \in Q$, $k \in \mathbb{N}$, $\gamma \models \mathsf{popu}(q,k)$ if and only if $\mathsf{loc}(\gamma)(q,k) > 0$;

— $\gamma \models \mathsf{cont}(\mathbf{reg},\mathsf{d})$ if and only if $\mathsf{data}(\gamma)(\mathbf{reg}) = \mathsf{d}$;

— Boolean operators are interpreted as usual;

— $\gamma \models \exists k\ \phi$ if and only if there is $\ell \in \mathbb{N}$ such that $\gamma \models \phi[k \leftarrow \ell]$ where $\phi[k \leftarrow \ell]$ is equal to ϕ where all occurrences of free variable k are replaced by constant term ℓ (and term of the form $k+m$ are replaced with the constant $\ell + m \in \mathbb{N}$);

— $\gamma \models \forall k\ \phi$ if and only if, for every $\ell \in \mathbb{N}$, $\gamma \models \phi[k \leftarrow \ell]$.

Example 4.4. Consider a round-based protocol $\mathcal{P}$ with $Q = \{q_0, q_1\}$, $\mathbb{D} = \{\bot, \mathsf{a}\}$ and $\mathsf{dim} = 1$. The formula $\psi = (\forall k\ \mathsf{popu}(q_0,k) \vee \neg\mathsf{popu}(q_1,k{+}1)) \vee \mathsf{cont}(\mathsf{rg}[1],\mathsf{a}) \vee (\forall k\ \mathsf{cont}(\mathsf{rg}[k{+}2],\mathsf{a}))$ is an example of presence constraint. It is composed of three atomic presence constraints: "$\forall k\ \mathsf{popu}(q_0,k) \vee \neg\mathsf{popu}(q_1,k{+}1)$", "$\mathsf{cont}(\mathsf{rg}[1],\mathsf{a})$" and "$\forall k\ \mathsf{cont}(\mathsf{rg}[k{+}2],\mathsf{a})$". Let $\gamma := ((q_0,0) \oplus (q_1,1), \bot^{\mathsf{Reg}})$. We have $\gamma \not\models \mathsf{cont}(\mathsf{rg}[1],\mathsf{a})$. However, $\gamma \models \forall k\ \mathsf{popu}(q_0,k) \vee \neg\mathsf{popu}(q_1,k+1)$: for $k = 0$ we have $\gamma \models \mathsf{popu}(q_0,k)$ and for all $k \geqslant 1$ we have $\gamma \models \neg\mathsf{popu}(q_1,k+1)$. This proves that $\gamma \models \psi$. Note that we have $\gamma \not\models \forall k\ \mathsf{cont}(\mathsf{rg}[k{+}2],\mathsf{a})$, and

that the same is true for every reachable configuration γ because reachable configurations only have finitely many values in $\mathbb{D} \setminus \{\perp\}$.

We define the *round-based presence reachability problem* (round-based PRP for short):

ROUND-BASED PRP
Input: A round-based protocol $\mathcal{P}$, a presence constraint ψ
Question: Does there exist $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \models \psi$?

As in the roundless case, round-based COVER and round-based TARGET are particular cases of round-based PRP with $\psi \coloneqq \exists k,\, \mathsf{popu}(q,k)$ for COVER and $\psi \coloneqq \forall k \bigwedge_{q \neq q_f} \neg \mathsf{popu}(q,k)$ for TARGET.

Example 4.5. Consider the protocol $\mathcal{P}$ from Fig. 4.2. As proved in Example 4.3, $(\mathcal{P}, q_f)$ is a negative instance of round-based COVER and of round-based TARGET.

Let $\psi \coloneqq \exists k\, \mathsf{popu}(q_3,k) \wedge \mathsf{popu}(q_6,k)$. We have that $(\mathcal{P},\psi)$ is a positive instance of round-based PRP, because $(\mathcal{P}, \psi[k \leftarrow 1])$ is: it suffices to send two processes to $(q_2,1)$, then write a to $\mathsf{rg}[0]$ with a third process and make one process go to $(q_6,1)$ then another one go to $(q_3,1)$. It fact, $(\mathcal{P}, \phi[k \leftarrow \ell])$ is positive if and only if $\ell \geqslant 1$. However, if we let $\psi' \coloneqq \exists k\, \mathsf{popu}(q_4,k) \wedge \mathsf{popu}(q_6,k)$, then the instance $(\mathcal{P},\psi')$ is negative, as proved in Example 4.3.

Example 4.6. Consider now the protocol $\mathcal{P}_{\text{Asp}}$ from Fig. 4.1, which encoded Aspnes' noisy consensus algorithm. Agreement expresses that no two processes may return distinct values. This is equivalent to $(\mathcal{P}_{\text{Asp}},\psi)$ being a negative instance of round-based PRP where $\psi \coloneqq (\exists k\, \mathsf{popu}(R_0,k)) \wedge (\exists k\, \mathsf{popu}(R_1,k))$.

Validity expresses that any value decided by a process must have been the original preference of some process. In other words, for all $p \in \{0,1\}$, validity states that if all processes start with preference p then no process will decide $1-p$. For each $p \in \{0,1\}$, let $\mathcal{P}^{(p)}_{\text{Asp}}$ be the protocol where q_0 is removed and A_p is the initial state instead. Validity is equivalent to $(\mathcal{P}^{(p)}_{\text{Asp}}, R_{1-p})$ being a negative instance of round-based COVER for each $p \in \{0,1\}$.

We have established that the agreement and validity of Aspnes' noisy consensus algorithm can be expressed using round-based PRP. We are now interested in the complexity of this problem. We will establish in the three following sections that round-based PRP is PSPACE-complete.

4.4 Exponential Lower Bounds

Towards our objective to establish the complexity of round-based PRP, we start by understanding why and to what extent the problem is hard. This section is devoted to the construction of complex round-based protocols to highlight why round-based ASMS are more complex than roundless ASMS studied in Chapter 2. This section is split into two parts. In the first part, composed of Section 4.4.1 and Section 4.4.2, we provide family of protocols that highlight the power of round-based PRP. The aim is twofold: to provide the reader with some intuition on our model and, later, to justify our approach towards a polynomial-space algorithm. In the second part, corresponding to Section 4.4.3, we provide a complexity lower bound for round-based PRP, proving that this problem is PSPACE-hard.

4.4.1 Exponential Lower Bound in the Number of Rounds for Cover

The first question is whether there is a polynomial bound on the smallest number of rounds in a witness execution for round-based PRP and in particular for round-based Cover. If the answer was yes, then we could simply compute this polynomial bound and build an equivalent roundless ASMS of polynomial size, so that all the complexity results from Chapter 2 would carry over to their round-based counterparts. However, it is not the case:

Proposition 4.7. *There exists a family $(\mathcal{P}_m)_{m\geqslant 1}$ of round-based protocols with a special state q_f such that $|\mathcal{P}_m| = O(m)$ and the minimum round k at which (q_f, k) can be covered is $\Omega(2^m)$.*

The protocol $\mathcal{P}_m$, depicted in Figure 4.3, encodes a binary counter on m bits. It only uses one register per round ($\mathsf{dim} = 1$) and has visibility range $v = 0$. The counter value is initially equal to 0 and is incremented by one at each round; setting the most significant bit to 1 puts a process in q_f. In order to reach q_f, any execution needs at least m+1 processes: one in q_{tick} ticking every round, and one per bit, in states $\{q_{i,0}, q_{i,1}\}$ to represent the value of the counter's i-th bit. At round k, the value of the i-th least significant bit is 0 if at least one process is at $(q_{i,0}, k)$, and 1 if at least one process is at $(q_{i,1}, k)$. The construction in fact guarantees the following: for every i, for every k, it cannot be that some execution covers $(q_{i,0}, k)$ but that some other execution covers $(q_{i,1}, k)$. Finally, at round 2^{m-1}, setting the m-th least significant bit –of weight 2^{m-1}– to 1 corresponds to $(q_f, 2^{m-1})$ being reached. This round-based ASMS satisfies the following property, that entails Proposition 4.7.

Proposition 4.8. *Let $k \in [\![0, 2^{m-1}]\!]$. Location (q_f, k) is coverable in $\mathcal{P}_m$ iff $k = 2^{m-1}$.*

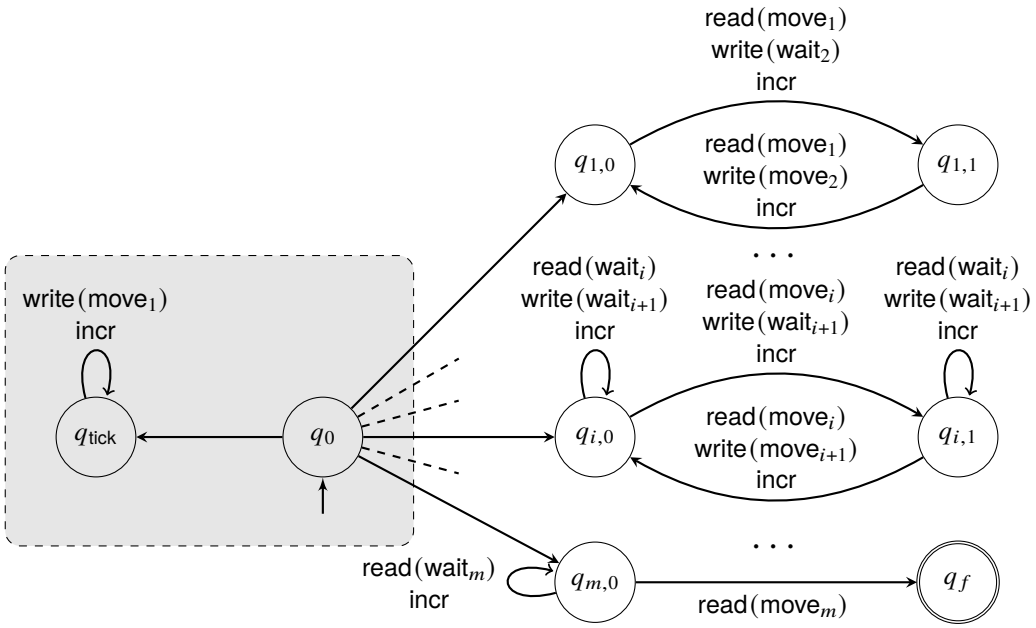


Figure 4.3 – Protocol $\mathcal{P}_m$ where an exponential number of rounds is needed to reach q_f. For the sake of readability, we allow for transitions labeled with sequence of actions. These sequences are performed non-atomically: one should in principles add intermediate states to split the transition into several consecutive transitions. The tick gadget in grey will be modified in Section 4.4.2.

Proof. Let $i \in [\![1, m]\!]$, $k \in [\![0, 2^{m-1}]\!]$ and let r be the remainder of the Euclidean division of k by 2^i. We claim that $(q_{i,0}, k)$ is reachable if and only if $0 \leqslant r \leqslant 2^{i-1} - 1$ and $(q_{i,1}, k)$ is reachable if and only if $2^{i-1} \leqslant r \leqslant 2^i - 1$. The proof is by induction on pairs (k, i), ordered lexicographically.

Observe first that, for all $i \in [\![1, m]\!]$, $(q_{i,0}, 0)$ is coverable and $(q_{i,1}, 0)$ is not. Moreover, for all $k \in [\![0, 2^m]\!]$, $(q_{1,0}, k)$ is coverable exactly for even k, and $(q_{1,1}, k)$ exactly for odd k.

Let $k > 0$, $i \in [\![2, m]\!]$ and suppose that the statement holds for all pairs (k', i') with $k' < k$ or $k' = k$ and $i' < i$. The only way to write move_i to $\mathsf{rg}[k]$ is when a process moves from $(q_{i-1,1}, k-1)$ to $(q_{i-1,0}, k)$. By induction hypothesis, this means that the remainder of the Euclidean division of $k-1$ by 2^{i-1} is in $[\![2^{i-2}, 2^{i-1} - 1]\!]$ and the remainder of the Euclidean division of k by 2^{i-1} is in $[\![0, 2^{i-2}]\!]$, which is equivalent to k being divisible by 2^{i-1}. To sum up, move_i can be written to $\mathsf{rg}[k]$ exactly when k is a multiple of 2^{i-1}. Similarly, wait_i can be written to $\mathsf{rg}[k]$ exactly when k is not divisible by 2^{i-1}.

We distinguish cases according to the remainder r of the Euclidean division of k by 2^i:

— if $r = 0$, then the remainder of $k-1$ by 2^i is in $[\![2^{i-1}, 2^i - 1]\!]$ hence $(q_{i,1}, k-1)$ can be reached and $(q_{i,0}, k-1)$ cannot; since k is divisible by 2^{i-1}, move_i can be written to $\mathsf{rg}[k]$ but wait_i cannot, so that $(q_{i,0}, k)$ can be reached and $(q_{i,1}, k)$ cannot;

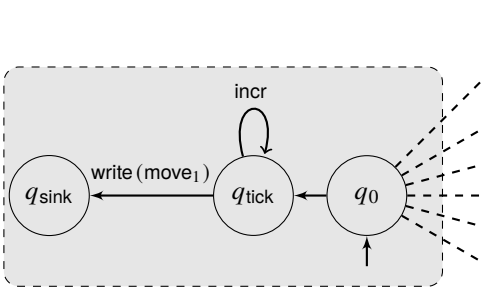
— if $1 \leqslant r \leqslant 2^{i-1} - 1$, then the remainder of $k-1$ by 2^i is in $[\![0, 2^{i-1} - 1]\!]$ hence $(q_{i,0}, k-1)$ can be reached and $(q_{i,1}, k-1)$ cannot; since k is not divisible by 2^{i-1}, wait_i can be written to $\mathsf{rg}[k]$ but move_i cannot, so that $(q_{i,0}, k)$ can be reached and $(q_{i,1}, k)$ cannot;

— if $r = 2^{i-1}$, then the remainder of $k-1$ by 2^i is in $[\![0, 2^{i-1} - 1]\!]$ hence $(q_{i,0}, k-1)$ can be reached and $(q_{i,1}, k-1)$ cannot; since k is divisible by 2^{i-1}, move_i can be written to $\mathsf{rg}[k]$ but wait_i cannot, so that $(q_{i,1}, k)$ can be reached and $(q_{i,0}, k)$ cannot;

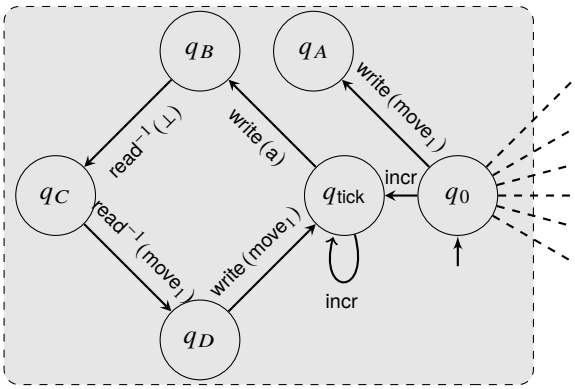
— if $2^{i-1}+1 \leqslant r \leqslant 2^i - 1$, then the remainder of $k-1$ by 2^i is in $[\![2^{i-1}, 2^i - 1]\!]$ hence $(q_{i,1}, k-1)$ can be reached and $(q_{i,0}, k-1)$ cannot; since k is divisible by 2^{i-1}, wait_i can be written to $\mathsf{rg}[k]$ but move_i cannot, $(q_{i,1}, k)$ can be reached and $(q_{i,0}, k)$ cannot.

This concludes the inductive step. Applied to (m, k), this proves that, for all $k \in [\![0, 2^{m-1}]\!]$, $(q_{m,1}, k)$ is coverable if and only if the remainder of the Euclidean division of k by 2^m is at least 2^{m-1}, which is the case if and only if $k = 2^{m-1}$. □

This proves that the number of rounds needed in an execution witnessing Cover may have to be as large as exponential in the size of the protocol.



(a) An exponential number of processes is needed to reach q_f.



(b) An exponential number of active rounds is needed to reach q_f.

Figure 4.4 – Two modifications of the tick mechanism of $(\mathcal{P}_m)_{m\geqslant 1}$ yielding protocols that need respectively an exponential number of processes and an exponential number of active rounds.

4.4.2 Additional Exponential Lower Bounds

Here, we generalize the lower bound from the previous section to other relevant quantities, starting with the number of processes.

Proposition 4.9. *There exists a family $(\mathcal{P}'_m)_{m\geqslant 1}$ of round-based protocols with q_f a special state such that $|\mathcal{P}'_m| = O(m)$ and the minimal number of processes to cover q_f is $\Omega(2^m)$.*

Proof. We modify the tick mechanism of protocol $\mathcal{P}_m$ from Proposition 4.7 so that, at each round, a process must sacrifice itself to provide the tick. This modification is depicted in 4.4(a). □

In the protocol built in Fig. 4.3, although the number of rounds needed is exponential, one can cover q_f with an execution where processes are never more than one round apart. More specifically, we can cover q_f with an executions whose steps are first at round 0, then round increments, then moves at round 1, then round increments, and so on. This makes it possible to store this execution configuration by configuration, because in a given configuration there are at most 2 relevant rounds which must be stored, and all earlier rounds can be forgotten. This gives the hope to design a polynomial-space non-deterministic algorithm that guesses the witness execution in such a manner, under the assumption that few rounds are relevant at a time. We will prove that this assumption cannot however be made in general. To do so, we introduce another quantity, which we dub *number of simultaneously active rounds*. In an execution $\rho = \gamma_0, \theta_1, \ldots, \theta_\ell, \gamma_\ell$, round k is *active* at configuration γ_i if there is $j < i$ and $j' \geqslant i$

such that θ_j and $\theta_{j'}$ have effect on round k (recall that we have defined in Section 4.3.2 what is means for a move to have effect on a round). The *number of simultaneously active rounds* of ρ is the maximum over i of the number of rounds active at γ_i in ρ. If a round is not active, then it is either blank (recall that a register **reg** is blank in γ when $\mathsf{data}(\gamma)(\mathbf{reg}) = \bot$) and with no process or it is irrelevant from this point onwards because it plays no role in the future. Executions with a small number of simultaneously active rounds are therefore easier to store. However, as we will now see, the number of simultaneously active rounds may need to be as large as exponential, even for Cover.

Proposition 4.10. *There exists a family $(\mathcal{P}''_m)_{m\geqslant 1}$ of round-based protocols with q_f a special state such that $|\mathcal{P}''_m| = O(m)$ and the minimal number of simultaneously active rounds in executions covering q_f is in $\Omega(2^m)$.*

Proof. Again, we modify the family of protocols $(\mathcal{P}_m)$ from Proposition 4.7. To do so, we replace the tick mechanism with the gadget presented in Fig. 4.4(b). This increases the value of v to 1, because we use $\mathsf{read}^{-1}(\bot)$ and $\mathsf{read}^{-1}(\mathsf{move}_1)$ actions. The transitions from q_{tick} to q_B and from q_B to q_C ensure that, for all $k \in [\![0, 2^{m-1}]\!]$, a must be written to $\mathsf{rg}[k]$ before it is written to $\mathsf{rg}[k-1]$. The transitions from q_C to q_D and from q_D to q_{tick}, on the contrary, ensure that, for all $k \in [\![1, 2^{m-1}]\!]$, move_1 must be written to $\mathsf{rg}[k-1]$ before it is written to $\mathsf{rg}[k]$. Hence, in an execution covering q_f, when move_1 is first written to $\mathsf{rg}[0]$, all rounds from 1 to 2^{m-1} must be active and the number of simultaneously active rounds is at least 2^{m-1}. □

Remark 4.11. The construction in the proof of Proposition 4.10 requires $v > 0$. More generally, for any positive instance $(\mathcal{P}, q_f)$ of Cover with $v = 0$, there is an execution covering q_f whose number of simultaneous active rounds is at most 2. Indeed, let ρ be an execution of $\mathcal{P}$ that covers q_f. In $\mathcal{P}$, a process do not interact with rounds different from his own, so that we may rearrange ρ as follows: all steps at round 0 first, then all steps at round 1, then all steps at round 2, and so on. The obtained execution covers q_f and its number of simultaneously active rounds is at most 2. Therefore, when $v = 0$, a naive polynomial-space algorithm for Cover consists in computing all coverable states round after round. This algorithm is not satisfying for us, in particular because our motivating example from Section 4.2 requires $v = 1$.

4.4.3 Complexity Lower Bound for PRP

Here, we prove that the round-based presence reachability problem is PSPACE-hard. In comparison with the results from Chapter 2, this highlights an increase in complexity due to

the addition of rounds. We in fact prove that this complexity lower bound already holds for the simplest problem, Cover:

Theorem 4.12. *Round-based Cover is PSPACE-hard, even for round-based ASMS with visibility range $v = 0$ and number of registers per round* $\mathsf{dim} = 1$.

The rest of this section is devoted to proving Theorem 4.12. The proof is by reduction from the validity problem of 3-QBF. We fix an instance ϕ of 3-QBF over the $2m$ variables $\{x_0, \cdots, x_{2m-1}\}$

$$\phi = \forall x_{2m-1} \exists x_{2m-2} \forall x_{2m-3} \exists x_{2m-4} \ldots \forall x_1 \exists x_0 \bigwedge_{1 \leqslant j \leqslant p} a_j \vee b_j \vee c_j \ ,$$

where for every $j \in [\![1, p]\!]$, $a_j, b_j, c_j \in \{x_i, \neg x_i \mid i \in [\![0, 2m-1]\!]\}$ are the *literals*. We write $\psi \coloneqq \bigwedge_{1 \leqslant j \leqslant p} a_j \vee b_j \vee c_j$ for the inner 3-SAT formula.

From ϕ we construct a round-based protocol on the data alphabet

$$\mathbb{D} \coloneqq \{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i \mid i \in [\![0, 2m]\!]\} \cup \{\mathtt{x_i}, \neg \mathtt{x_i} \mid i \in [\![0, 2m-1]\!]\} \cup \{\bot\}.$$

Moreover, we let $v = 0$ and $\mathsf{dim} = 1$. The protocol we construct is represented in Figure 4.5; it contains several gadgets that we detail in the sequel.

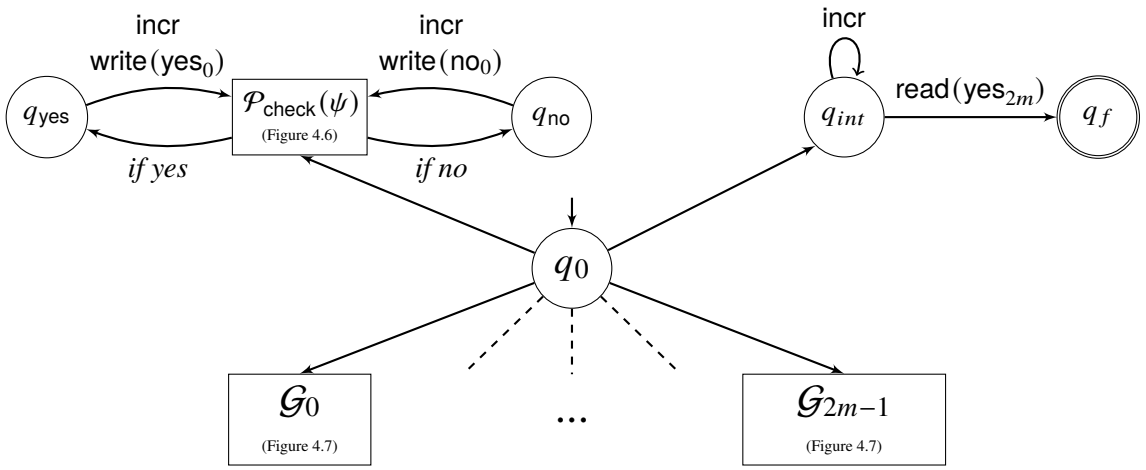


Figure 4.5 – Overview of the protocol $\mathcal{P}_{\mathsf{QBF}}$. All transitions to gadgets go to their initial states.

Before that, we provide a high-level description of $\mathcal{P}_{\mathsf{QBF}}$. In $\mathcal{P}_{\mathsf{QBF}}$, each variable x_i is represented by a subprotocol $\mathcal{G}_i$. At every round; a different valuation is considered and the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ evaluates whether it makes the inner SAT formula true. The gadget $\mathcal{G}_i$ writes

at each round the truth value of x_i in the considered valuation. The protocol enumerates all valuations: a given round k will correspond to one valuation of the variables of ψ, in which variable x is true if $\mathtt{x}$ can be written to $\mathsf{rg}[k]$, and false if $\neg\mathtt{x}$ can be written to $\mathsf{rg}[k]$. This protocol is almost deterministic, in the sense that the enumeration of valuations is done in a deterministic fashion so that, at a given round, for each variable x and round k, it cannot be that some execution has a move writing $\mathtt{x}$ to $\mathsf{rg}[k]$ but that some other execution has a move writing $\neg\mathtt{x}$ to $\mathsf{rg}[k]$. The enumeration of the valuations and corresponding evaluations of ψ are performed so as to take the appropriate decision about the validity of the global formula ϕ.

We start by describing the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$, depicted in Figure 4.6, that checks whether ψ is satisfied by the valuation under consideration. The valuation under consideration at round k corresponds to the symbols that can be written to the register: x is considered true if $\mathtt{x}$ can be written to the register, and considered false if $\neg\mathtt{x}$ can be written to the register. State q_{yes}

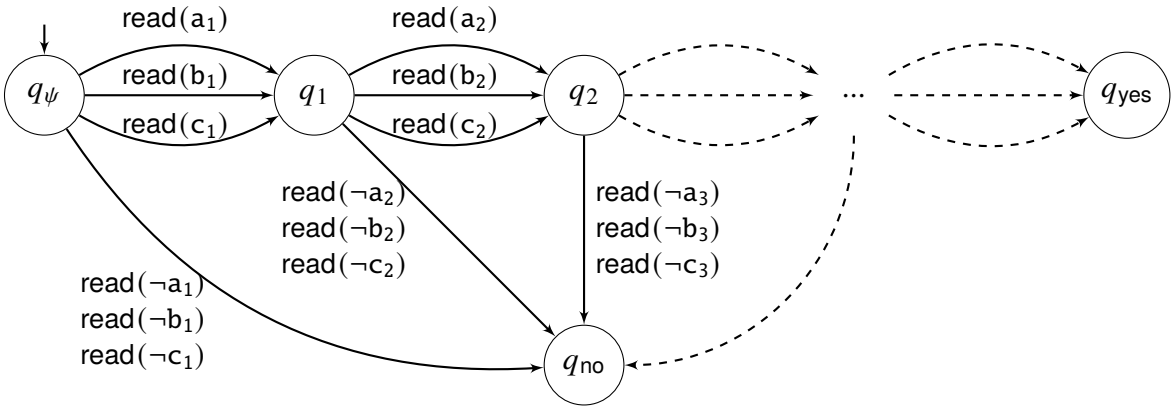


Figure 4.6 – Gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ that checks whether the inner SAT formula ψ is satisfied by the current valuation.

corresponds to ψ evaluated to true and q_{no} corresponding to ψ evaluated to false. Note that we allow transitions labeled by sequences of actions: these sequences of actions are encoded in the model by adding intermediate states and transitions so that each transition is only labeled by one action.

We now explain how valuations are enumerated and how the quantifiers are handled. We define below a procedure next that, given valuation ν, computes the next valuation $\mathsf{next}(\nu)$ that needs to be checked. This next valuation $\mathsf{next}(\nu)$ will depend on whether $\nu \models \psi$ or not, and this information will be propagated from the innermost to the outermost quantifier. An existential quantifier is satisfied with only one positive witness while a universal quantifier needs two positive witnesses. Conversely, an existential quantifier is invalidated with two negative

witnesses while a universal quantifier only needs one. Let ν_0 be the valuation that assigns 0 to all variables. The validity of the formula will be determined by computing, increasingly in k, valuation $\mathsf{next}^k(\nu_0)$ until a value of k is reached for which the outermost quantifier is either satisfied or invalidated.

We now define formally the procedure next. Let ϕ_i be the subformula $Qx_i \ldots \forall x_1 \exists x_0 \psi$ where $Q = \exists$ if i is even, and $Q = \forall$ if i is odd. Variables $x_{2m-1}, \ldots, x_{i+1}$ are free in ϕ_i. The procedure next uses variables $b_i \in \{\mathsf{yes}, \mathsf{no}, \mathsf{wait}\}$ for each $i \in [\![0, 2m]\!]$, whose role is the following. We will set $b_0 = \mathsf{yes}$ if $\nu \models \psi$, and $b_0 = \mathsf{no}$ otherwise. For any $1 \leqslant i \leqslant 2m-1$, $b_i = \mathsf{yes}$ means that we have satisfied ϕ_i with the current values of x_{i+1} to x_{2m-1} (ν was the last check needed for ϕ_i and it was successful); $b_i = \mathsf{no}$ means that we have satisfied $\neg\phi_i$ with the current values of x_{i+1} to x_{2m-1} and $b_i = \mathsf{wait}$ means that more valuations are needed to determine the truth value of ϕ_i with the current values of x_{i+1} to x_{2m-1}. Given a valuation ν, the procedure next computes, at each iteration i, the truth value of x_i in valuation $\mathsf{next}(\nu)$ and the value of b_{i+1}. After $2m$ iterations, this provides the new valuation $\mathsf{next}(\nu)$ against which ψ must be checked. Formally, the iteration is defined as follows:

- If $b_i = \mathsf{wait}$, then $\mathsf{next}(\nu)(x_i) \leftarrow \nu(x_i)$ and $b_{i+1} \leftarrow \mathsf{wait}$.
- Otherwise,
  - if i is even (existential quantifier):
    - if $b_i = \mathsf{yes}$, then $\mathsf{next}(\nu)(x_i) \leftarrow 0$ and $b_{i+1} \leftarrow \mathsf{yes}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) \leftarrow 1$ and $b_{i+1} \leftarrow \mathsf{wait}$,
    - if $b_i = \mathsf{no}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) \leftarrow 0$ and $b_{i+1} \leftarrow \mathsf{no}$;
  - if i is odd (universal quantifier):
    - if $b_i = \mathsf{no}$, then $\mathsf{next}(\nu)(x_i) \leftarrow 0$ and $b_{i+1} \leftarrow \mathsf{no}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 0$, then $\mathsf{next}(\nu)(x_i) \leftarrow 1$ and $b_{i+1} \leftarrow \mathsf{wait}$,
    - if $b_i = \mathsf{yes}$ and $\nu(x_i) = 1$, then $\mathsf{next}(\nu)(x_i) \leftarrow 0$ and $b_{i+1} \leftarrow \mathsf{yes}$.

Example 4.13. Let us illustrate the next operator on a small example. Assume

$$\phi = \forall x_1 \exists x_0 \neg x_1 \wedge (\neg x_1 \vee x_0),$$

which is not a valid formula. In this case, $\psi = \neg x_1 \wedge (\neg x_1 \vee x_0)$. To determine that ϕ is not valid, we start by checking the valuation ν_0, and set $b_0 = \mathsf{yes}$. According to the next operator, we obtain $\mathsf{next}(\nu)(x_0) = 0$, $b_1 = \mathsf{yes}$ in the first iteration (we have satisfied $\exists x_0 \psi$ for $x_1 = 0$); and $\mathsf{next}(\nu)(x_1) = 1$, $b_2 = \mathsf{wait}$ in the second iteration. In fact, even though $\nu \models \psi$, because x_1 is

quantified universally, we cannot yet conclude: we must also check whether ψ holds by setting x_1 to 1. This is what $b_2 = \mathsf{wait}$ means, and this is why $\mathsf{next}(\nu)(x_1)$ is set to 1. Let $\nu' = \mathsf{next}(\nu)$ and consider the computation of next from ν'. We have $\nu' \not\models \psi$ therefore $b_0 = \mathsf{no}$. The first iteration sets $\mathsf{next}(\nu')(x_0) = 1, b_1 = \mathsf{wait}$; and the second iteration leaves x_1 unchanged and sets $b_2 = \mathsf{wait}$. For x_1, we have computed that $x_0 = 0$ does not work; because the quantifier before x_0 is existential, we still have a chance with $x_0 = 1$ and the computation continues. We let $\nu'' = \mathsf{next}(\nu')$: ν'' sets both variables to 1; however, $\nu'' \not\models \psi$. Therefore, we set $b_0 = \mathsf{no}$. Now, having exhausted all possibilities, the first iteration sets $b_1 = \mathsf{no}$, and the second sets $b_2 = \mathsf{no}$. We have determined that ϕ is not valid.

The following lemma formalizes how validity can be checked using next. It is easily proved by induction on the number of variables.

Lemma 4.14. *QBF formula ϕ is valid if and only if, when iterating* next *from valuation ν_0, one eventually obtains a computation of* next *that sets b_{2m} to* yes.

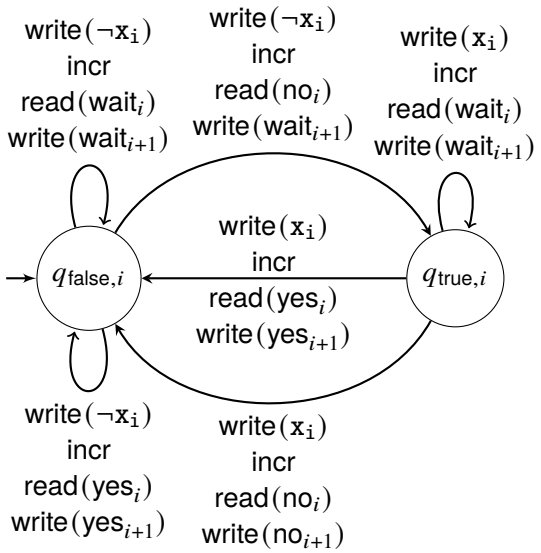
Now, we define, for all $i \in [\![0, 2m-1]\!]$, a gadget $\mathcal{G}_i$ that will play the role of variable x_i. At each round, gadget $\mathcal{G}_i$ receives from gadget $\mathcal{G}_{i-1}$ the value $b_i \in \{\mathsf{wait}_i, \mathsf{yes}_i, \mathsf{no}_i\}$, except for gadget $\mathcal{G}_0$ which receives b_0 from $\mathcal{P}_{\mathsf{check}}(\psi)$. It transmits $b_{i+1} \in \{\mathsf{wait}_{i+1}, \mathsf{yes}_{i+1}, \mathsf{no}_{i+}\}$ to $\mathcal{G}_{i+1}$, computes the value of variable x_i accordingly and communicates it by writing either $\mathtt{x_i}$ or $\neg\mathtt{x_i}$. These gadgets $\mathcal{G}_i$ are given in Figure 4.7(a) if x_i is existentially quantified (*i.e.*, even i), and Figure 4.7(b) if x_i is universally quantified (*i.e.*, odd i). We define the protocol $\mathcal{P}_{\mathsf{QBF}}$ represented in Fig. 4.5 using the gadgets $\mathcal{G}_i$ from Fig. 4.7(a) and Fig. 4.7(b) and the gadget $\mathcal{P}_{\mathsf{check}}(\psi)$ from Fig. 4.6.

Finally, the following lemma justifies the correctness of the construction by formalizing the relation between next and $\mathcal{P}_{\mathsf{QBF}}$.

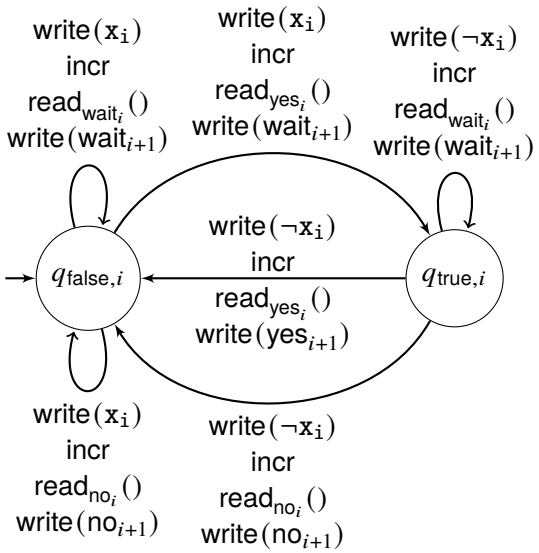
Lemma 4.15. *Let $k \in \mathbb{N}$ and let $\nu_k \coloneqq \mathsf{next}^k(\nu_0)$ be the valuation obtained by applying* next *k times from $\nu_0 \coloneqq 0^{2m}$. For all $i \in [\![0, 2m-1]\!]$:*

- *$(q_{\mathsf{false},i}, k)$ is coverable if and only if $\nu_k(x_i) = 0$,*
- *$(q_{\mathsf{true},i}, k)$ is coverable if and only if $\nu_k(x_i) = 1$,*
- *$\neg\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 0$,*
- *$\mathtt{x_i}$ can be written to $\mathsf{rg}[k]$ if and only if $\nu_k(x_i) = 1$.*

The above lemma holds by construction, because we have designed the protocol so that it follows step by step the computation of next.



(a) Gadget $\mathcal{G}_i$ for existentially quantified variable x_i (*i.e.*, i even).



(b) Gadget $\mathcal{G}_i$ for universally quantified variable x_i (*i.e.*, i odd).

Figure 4.7 – Illustration of the gadgets $\mathcal{G}_i$.

Combining Lemma 4.15 with Lemma 4.14 proves that ϕ is valid if and only if there exists k such that yes_{2m} can be written to the register of round k; by construction of $\mathcal{P}_{\mathsf{QBF}}$, this is also the condition under which q_f can be covered. This proves that $(\mathcal{P}, q_f)$ is a positive instance of Cover if and only if ϕ is a positive instance of 3-QBF. Also, the protocol $\mathcal{P}$ can be built in time polynomial in m. This concludes the proof of Theorem 4.12.

Note that this lower bound already holds for $v = 0$ and $\mathsf{dim} = 1$ and without $\mathsf{read}(\perp)$ transitions, which proves that none of the restrictions considered in Chapter 2 for roundless ASMS would yield a complexity lower than PSPACE for Cover in round-based ASMS.

4.5 A Polynomial-Space Algorithm for Cover

We have proved PSPACE-hardness of round-based Cover and therefore of round-based PRP in the previous section. We now aim to prove that these problems are in PSPACE. For pedagogical reasons, we start by providing a polynomial-space procedure to decide Cover. We will generalize this procedure to PRP in the next section.

Theorem 4.16. *Round-based Cover is PSPACE-complete.*

The rest of this section is devoted to proving Theorem 4.16. We have proved PSPACE-

hardness in Theorem 4.12, we here prove membership in PSPACE. We start by defining in Section 4.5.1 a non-counting abstraction similar to the one developed in Chapter 2 for roundless ASMS. We then define in Section 4.5.2 the notion of *footprint* meant to represent the projection of an abstract execution onto a few rounds. We explain in Section 4.5.3 how we check that several footprints can be merged into an execution. We finally introduce the polynomial-space procedure in Section 4.5.4.

4.5.1 A Non-Counting Abstraction

We here develop a non-counting abstraction very similar to the one developed in Section 2.4 for roundless ASMS. Indeed, we have the same copycat property as in Lemma 2.10.

Lemma 4.17. *Consider γ_1, γ_2, q_2, k_2 such that $\gamma_1 \xrightarrow{*} \gamma_2$ and $(q_2, k_2) \in \mathsf{supp}(\gamma_2)$. There exists $(q_1, k_1) \in \mathsf{supp}(\gamma_1)$ s.t. $\langle \mathsf{loc}(\gamma_1) \oplus (q_1, k_1), \mathsf{data}(\gamma_1)\rangle \xrightarrow{*} \langle \mathsf{loc}(\gamma_2) \oplus (q_2, k_2), \mathsf{data}(\gamma_2)\rangle$.*

This lemma can be proved almost identically to Lemma 2.10. The only new case is the one of incr transitions, which causes no issue because it does not interact with the registers.

This leads to define abstract configurations of round-based ASMS in similar fashion to Section 2.4. An *abstract configuration* is a pair $\sigma = \langle \mathsf{loc}(\sigma), \mathsf{data}(\sigma)\rangle \in \overline{\Gamma} = 2^{Q\times\mathbb{N}} \times \mathbb{D}^{\mathsf{Reg}}$. Given a configuration $\gamma \in \Gamma$, its *abstract projection* is the abstract configuration $\langle \mathsf{supp}(\gamma), \mathsf{data}(\gamma)\rangle$. The *initial abstract configuration* is $\sigma_0 \coloneqq \langle (q_0, 0), \bot\rangle$.

As in the roundless case, we define the abstract semantics by letting $\sigma_1 \xrightarrow{\theta} \sigma_2$ whenever there are $\gamma_1, \gamma_2 \in \Gamma$ such that $\gamma_1 \xrightarrow{\theta} \gamma_2$, $\overline{\gamma_1} = \sigma_1$ and $\overline{\gamma_2} = \sigma_2$. We have the same non-determinism as in the roundless case: given a move θ that can be applied from σ, there are two ways to apply θ, yielding either a deserting step that empties the source location or a non-deserting step that does not. We define as in Chapter 2 the notion of *abstract execution*, the reachability relation $\xrightarrow{*}$, the reachability set $\mathsf{Post}^*(S)$ for $S \subseteq \overline{\Gamma}$ and the notion of coverability.

Given an abstract configuration σ and a presence constraint ψ, we let $\sigma \models \psi$ when there is $\gamma \in \Gamma$ such that $\gamma \models \psi$ and $\overline{\gamma} = \sigma$. With this definition, the interpretations of $\mathsf{popu}(q, k)$ and of $\mathsf{cont}(\mathbf{reg}, \mathsf{d})$ are intuitive: $\sigma \models \mathsf{popu}(q, k)$ whenever $(q, k) \in \mathsf{loc}(\sigma)$ and $\sigma \models \mathsf{cont}(\mathbf{reg}, \mathsf{d})$ whenever $\mathsf{data}(\sigma)(\mathbf{reg}) = \mathsf{d}$. Just like in Proposition 2.15, this abstraction is sound and complete for round-based PRP:

Proposition 4.18. *Let ψ be a presence constraint. There is $\gamma \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \models \psi$ if and only if there is $\sigma \in \mathsf{Post}^*(\sigma_0)$ such that $\sigma \models \psi$.*

Therefore, we work with abstract configurations and abstract executions in the rest of the section. As in Section 2.4, we define a notion of abstract execution in *normal form*. This definition is almost identical to the one from Section 2.4 except that the concept of state (an element of set Q) is replaced by the concept of location (an element of set $Q \times \mathbb{N}$). An abstract execution is in *normal form* when every step either deserts a location, populates a location that was never populated before in the execution, or is a write step whose source and destination locations are distinct and that writes to a register **reg** $\in \mathsf{Reg}$ such that the next step involving **reg**, if it exists, and is a read step. It is easy to adapt the proof of Lemma 2.17 by replacing states with locations:

Lemma 4.19. *For every $\sigma_1, \sigma_2 \in \overline{\Gamma}$, if $\sigma_1 \xrightarrow{*} \sigma_2$ then there is an abstract execution from σ_1 to σ_2 that is in normal form.*

In round-based ASMS, contrarily to the roundless case, there is no general bound on the number of steps in an abstract execution in normal form, because the set of locations is infinite. However, we can bound the number of steps of the executions on a given round.

Lemma 4.20. *An abstract execution has at most $5|Q| + |\mathbb{D}|$ steps at each round.*

Proof. The proof is the same as for Lemma 2.18, with locations of round k instead of states; in addition to the $4|Q| + |\mathbb{D}|$ steps identified in the proof of Lemma 2.18, there might be $|Q|$ round increment steps at round k that populate a new location of round $k + 1$. □

4.5.2 Footprints

While abstract configurations can be stored more concisely than (concrete) configurations, this is far from being sufficient to obtain a polynomial-space procedure. Indeed, there is no general bound on the space needed to store an abstract configuration, and with the exponential lower bounds from Section 4.4, we know that we need to consider configurations that spread across exponentially many rounds. Therefore, it seems hard to store entire configurations of the witness execution in polynomial space. For this reason, instead of representing the execution step by step, we represent it within a short, sliding window of rounds. This leads us to defining *footprints*, which correspond to the projection of executions onto windows of consecutive rounds. Our goal will be to guess the abstract execution covering q_f footprint by footprint. First, we define so-called *restricted configurations*, which correspond to abstract configurations projected onto such a window.

Restricted Configurations We first define *restricted configurations*, which correspond to configurations restricted to a limited number of consecutive rounds.

Let $0 \leqslant j \leqslant k$. We write $\mathcal{L}_{[\![j,k]\!]} \coloneqq Q \times [\![j,k]\!]$ for the set of locations of rounds j to k; similarly, we write $\mathsf{Reg}_{[\![j,k]\!]} \coloneqq \bigcup_{\ell \in [\![j,k]\!]} \mathsf{Reg}_\ell$ for the set of registers of rounds j to k. In this context, we sometimes refer to rounds $\ell < j$ as the rounds *below* j and to the rounds $\ell > k$ as the rounds *above* k.

Definition 4.21. Let $j \leqslant k$. A *restricted configuration* on (rounds) $[\![j,k]\!]$ is an element $\lambda = \langle \mu, \vec{d} \rangle \in \Gamma_{[\![j,k]\!]} \coloneqq 2^{\mathcal{L}_{[\![j,k]\!]}} \times \mathbb{D}^{\mathsf{Reg}_{[\![j,k]\!]}}$. We write $\mathsf{loc}(\lambda) \coloneqq \mu$ and $\mathsf{data}(\lambda) \coloneqq \vec{d}$. Given $\sigma \in \overline{\Gamma}$, we define the restricted configuration $\mathsf{restr}_{[\![j,k]\!]}(\sigma)$ by $\mathsf{loc}(\mathsf{restr}_{[\![j,k]\!]}(\gamma)) \coloneqq \mathsf{loc}(\sigma) \cap \Gamma_{[\![j,k]\!]}$ and $\mathsf{data}(\mathsf{restr}_{[\![j,k]\!]}(\gamma))(\mathbf{reg}) \coloneqq \mathsf{data}(\sigma)(\mathbf{reg})$ for all $\mathbf{reg} \in \mathsf{Reg}_{[\![j,k]\!]}$.

In words, the restricted configuration $\mathsf{restr}_{[\![j,k]\!]}(\sigma)$ is obtained by removing from σ all information that is not about rounds j to k.

Restricted Semantics We now define *restricted semantics*. Let $\lambda, \lambda' \in \Gamma_{[\![j,k]\!]}$. Given θ a move, we let $\lambda \xrightarrow{\theta} \lambda'$, which we call a *(restricted) step*, when there exist two configurations γ and γ' such that $\gamma \xrightarrow{\theta} \gamma'$, $\mathsf{restr}_{[\![j,k]\!]}(\gamma) = \lambda$ and $\mathsf{restr}_{[\![j,k]\!]}(\gamma') = \lambda'$. For clarity, we explicit under which conditions $\lambda \xrightarrow{\theta} \lambda'$ holds:

- if θ is a move with no effect on rounds j to k, then $\lambda \xrightarrow{\theta} \lambda'$ if and only if $\lambda = \lambda'$;
- if $\theta = ((q, \mathsf{write}_r(\mathsf{d}), q'), \ell)$ with $\ell \in [\![j,k]\!]$ then either $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \cup \{(q', \ell)\}$ or $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \setminus \{(q, \ell)\} \cup \{(q', \ell)\}$, $\mathsf{data}(\lambda')(\mathsf{rg}_r[\ell]) = \mathsf{d}$ and $\mathsf{data}(\lambda')(\mathbf{reg}) = \mathsf{data}(\lambda)(\mathbf{reg})$ for all $\mathbf{reg} \in \mathsf{Reg}_{[\![j,k]\!]} \setminus \{\mathsf{rg}_r[\ell]\}$;
- if $\theta = ((q, \mathsf{read}_r^{-i}(\mathsf{d}), q'), \ell)$ with $\ell \in [\![j,k]\!]$ then either $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \cup \{(q', \ell)\}$ or $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \setminus \{(q, \ell)\} \cup \{(q', \ell)\}$, $\mathsf{data}(\lambda) = \mathsf{data}(\lambda')$ and if $\ell - i \in [\![j,k]\!]$ then $\mathsf{data}(\lambda)(\mathsf{rg}_{\ell-i}[r]) = \mathsf{d}$ (if the register read is below round j then there is no condition on the content of the registers);
- if $\theta = ((q, \mathsf{read}_r^{-i}(\mathsf{d}), q'), \ell)$ with $\ell \notin [\![j,k]\!]$ and $\ell - i \in [\![j,k]\!]$ then $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda)$ (the round of the process is outside $[\![j,k]\!]$), $\mathsf{data}(\lambda) = \mathsf{data}(\lambda')$ and $\mathsf{data}(\lambda)(\mathsf{rg}_{\ell-i}[r]) = \mathsf{d}$;
- if $\theta = ((q, \mathsf{incr}, q'), \ell)$ then $\lambda \xrightarrow{\theta} \lambda'$ if and only if $\mathsf{data}(\lambda') = \mathsf{data}(\lambda)$ and:
  - if $\ell = j-1$ then $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \cup \{(q', k)\}$ (there is no condition related to $(q, j-1)$ since $j-1$ is outside of $[\![j,k]\!]$);
  - if $\ell \in [\![j, k-1]\!]$ then $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \setminus \{(q, \ell)\} \cup \{(q', \ell+1)\}$ or $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \cup \{(q', \ell+1)\}$,
  - if $\ell = k$ then $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda) \setminus \{(q, k)\}$ or $\mathsf{loc}(\lambda') = \mathsf{loc}(\lambda)$.

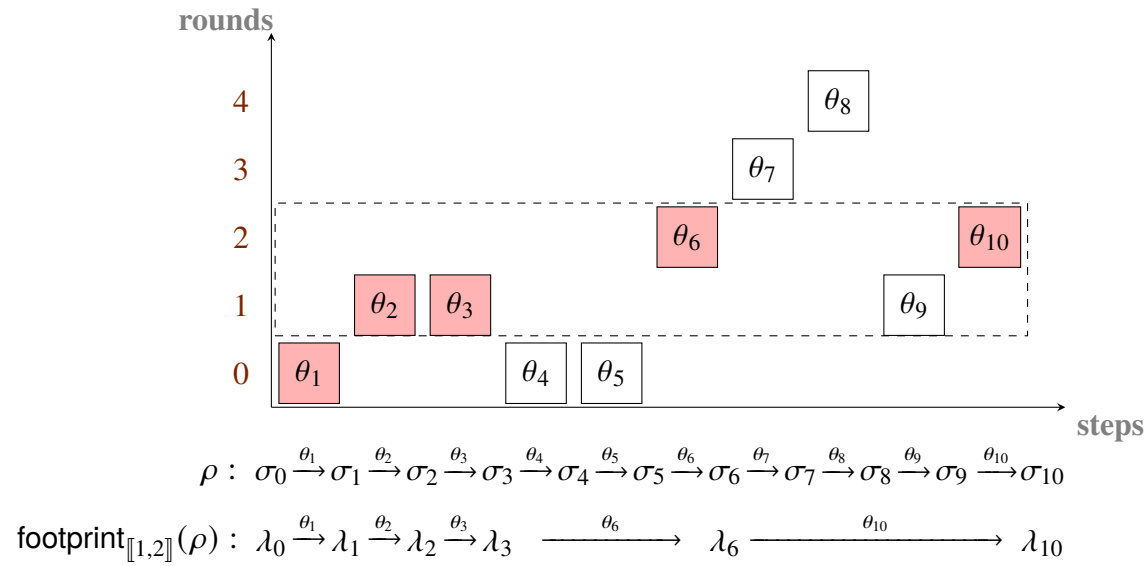


Figure 4.8 – An example of footprint of an abstract execution. A move (δ, k) of ρ is represented at y-coordinate k. For every $i \in [\![0, 10]\!]$, we let $\lambda_i \coloneqq \mathsf{restr}_{[\![1,2]\!]}(\gamma_i)$. We assume that $\lambda_3 = \lambda_4 = \lambda_5$ and $\lambda_6 = \lambda_7 = \lambda_8 = \lambda_9$. The moves that remain in $\mathsf{footprint}_{[\![1,2]\!]}(\rho)$ are colored in red.

Footprints A *footprint* on rounds $[\![j, k]\!]$ is an alternating sequence $\tau = \lambda_0, \theta_0, \lambda_1, \ldots, \theta_{m-1}, \lambda_m$ where $\lambda_i \in \Gamma_{[\![j,k]\!]}$ for all $i \in [\![0, m]\!]$ and for all $i \leqslant m-1$, $\lambda_i \xrightarrow{\theta_i} \lambda_{i+1}$ and $\lambda_i \neq \lambda_{i+1}$. It is meant to correspond to the projection of an execution on rounds $[\![j, k]\!]$. The *length* of the footprint τ is defined by its number of steps m.

Let $\rho = \gamma_0, \theta_0, \gamma_1, \ldots, \theta_{m-1}, \gamma_m$ be an execution. The *footprint of ρ on (rounds) $[\![j, k]\!]$*, written $\mathsf{footprint}_{[\![j,k]\!]}(\rho)$, is the footprint on $[\![j, k]\!]$ obtained from ρ by replacing γ_i by $\lambda_i = \mathsf{restr}_{[\![j,k]\!]}(\gamma_i)$ and then removing all steps $\lambda_i \xrightarrow{\theta} \lambda_{i+1}$ with $\lambda_i = \lambda_{i+1}$. In words, we take the restricted configurations and we remove useless steps[1]; this means that the length of $\mathsf{footprint}_{[\![j,k]\!]}(\rho)$ can be much smaller than the one of ρ.

Example 4.22. An example of footprint projection of an abstract execution can be found in Fig. 4.8. In this example, ρ is an abstract execution of length 10 and $\mathsf{footprint}_{[\![1,2]\!]}(\rho)$ is a footprint on rounds $[\![1, 2]\!]$ of length 6. Here, we suppose that $\lambda_3 = \lambda_4 = \lambda_5$ and $\lambda_6 = \lambda_7 = \lambda_8 = \lambda_9$. The moves θ_i of ρ that do not appear in $\mathsf{footprint}_{[\![1,2]\!]}(\rho)$ are those such that $\lambda_{i-1} = \lambda_i$. The only moves that can remain in $\mathsf{footprint}_{[\![1,2]\!]}(\rho)$ are those on rounds 1 and 2 and round increments

1. In the projection of an execution, we forbid that the same restricted configuration is visited twice consecutively, which matches the condition that $\lambda_i \neq \lambda_{i+1}$ in the definition of footprints. This may seem surprising, as executions are allowed to visit the same configuration several times in a row. This choice, however, guarantees unicity of the projection while also guaranteeing that a footprint on rounds $[\![j, k]\!]$ is equal to its projection onto $[\![j, k]\!]$. This will be in particular convenient for the statements and proofs of Lemma 4.23 and Lemma 4.24.

on round 0. This condition is however not sufficient: for example, it can be that θ_4 is a round increment but that its destination location is already populated in σ_3. Even moves at rounds 1 or 2 can be removed, as it is here the case of θ_9.

Let $[\![j', k']\!] \subseteq [\![j, k]\!]$. We extend the projection operator in a natural way to define, given $\lambda \in [\![j, k]\!]$, the restricted configuration $\mathsf{restr}_{[\![j',k']\!]}(\lambda)$ by $\mathsf{loc}(\mathsf{restr}_{[\![j',k']\!]}(\lambda)) \coloneqq \mathsf{loc}(\lambda) \cap \Gamma_{[\![j',k']\!]}$ and $\mathsf{data}(\mathsf{restr}_{[\![j',k']\!]}(\lambda))(\mathbf{reg}) \coloneqq \mathsf{data}(\lambda)(\mathbf{reg})$ for all $\mathbf{reg} \in \mathsf{Reg}_{[\![j',k']\!]}$. Similarly, given a footprint τ on $[\![j, k]\!]$, we define the projected footprint $\mathsf{footprint}_{[\![j',k']\!]}(\tau)$ of τ on $[\![j', k']\!]$ by projecting configurations onto $[\![j', k']\!]$ and removing useless steps.

It will be convenient to define $\Gamma_{[\![j,k]\!]}$ for j and k negative as well. In this case, we let $\mathcal{L}_{[\![j,k]\!]} = \{(q, \ell) \mid j \leqslant \ell \leqslant k, \ell \in \mathbb{N}\}$ so that only locations of positive rounds are considered. We consider that registers of negative rounds exist, and that they always hold value $\perp$. This will allow us to assume that $\mathsf{Reg}_{[\![j,k]\!]}$ always contains the symbols of $\mathsf{dim}(k - j + 1)$ registers. In particular, if $j, k < 0$ then $\Gamma_{[\![j,k]\!]} = \{\langle \emptyset, \perp^{\mathsf{dim}(k-j+1)} \rangle\}$. With this convention, we allow ourselves to consider footprints on $[\![j, k]\!]$ for $j, k \in \mathbb{Z}$.

4.5.3 Combining Footprints

Our goal is to design a non-deterministic polynomial-space algorithm that guesses an execution covering q_f footprint by footprint. Therefore, we need a sufficient condition for a sequence of overlapping footprints to be the projections of a single common execution. As we will see, there are essentially two conditions: that the footprints coincide on overlapping rounds, and that the overlap between footprints represents at least $\max(v, 1)$ rounds. This last condition corresponds to the fact that the overlap must be high enough in terms of rounds that we can check that all moves are valid: we need at least v common rounds to check read actions and at least 1 common round to check round increments.

We start by giving a condition allowing us to combine two overlapping footprints into one:

Lemma 4.23. *Let $w \geqslant \max(v, 1)$, $k \in \mathbb{Z}$, τ_- a footprint on $[\![k-w, k]\!]$ and τ_+ a footprint on $[\![k-w+1, k+1]\!]$ such that $\mathsf{footprint}_{[\![k-w+1,k]\!]}(\tau_-) = \mathsf{footprint}_{[\![k-w+1,k]\!]}(\tau_+)$. There exists T a footprint on $[\![k-w, k+1]\!]$ such that $\mathsf{footprint}_{[\![k-w,k]\!]}(T) = \tau_-$ and $\mathsf{footprint}_{[\![k-w+1,k+1]\!]}(T) = \tau_+$.*

Proof. We start by explaining the high-level idea of the proof. We build T in a naive manner, where we essentially start with the common part between τ_- and τ_+ and add all steps at rounds $k - w$ from τ_- and all steps at round $k + 1$ from τ_+, in a way such that the relative order of steps in τ_- is preserved and same for τ_+. This will yield a footprint because, since $w \geqslant v$, steps at

round $k+1$ may not depend on what happens on rounds $k-w$ and below, and steps at round $k-w$ may not depend on what happens on rounds above $k-w+1$ and in particular on round $k+1$.

More formally, let $\tau_{\text{com}} := \mathsf{footprint}_{[\![k-w+1,k]\!]}(\tau_-) = \mathsf{footprint}_{[\![k-w+1,k]\!]}(\tau_+)$. We proceed by induction on the number of steps in τ_{com}.

First, assume that τ_{com} has length 0. It contains no steps and exactly one restricted configuration $\lambda_{\text{com}} \in \Gamma_{[\![k-w,k+1]\!]}$. This implies that all steps in τ_- are at round $k-w$ and that all steps in τ_+ are at round $k+1$. Let $\tau_+ =: \lambda_0^+, \theta_1^+, \lambda_1^+, \dots, \theta_p^+, \lambda_p^+$ and $\tau_- =: \lambda_0^-, \theta_1^-, \lambda_1^-, \dots, \theta_m^-, \lambda_m^-$. Let $\tilde{\tau_+} := \tilde{\lambda_0^+}, \theta_1^+, \tilde{\lambda_1^+}, \dots, \theta_p^+, \tilde{\lambda_p^+}$ where, for all $i \in [\![0,p]\!]$, $\tilde{\lambda_i^+} \in \Gamma_{[\![k-w,k+1]\!]}$, $\mathsf{restr}_{[\![k-w+1,k+1]\!]}(\tilde{\lambda_i^+}) = \lambda_i^+$ and $\mathsf{restr}_{[\![k-w,k-w]\!]}(\tilde{\lambda_i^+}) = \mathsf{restr}_{[\![k-w,k-w]\!]}(\lambda_0^-)$. In words, $\tilde{\tau_+}$ corresponds to τ_+ where the information from λ_0^- about round $k-w$ has been added to all restricted configurations. Because moves in τ_+ are at round $k+1$ and $w \geqslant v$, these moves have no effect on round $k-w$ therefore $\tilde{\tau_+}$ is a footprint on rounds $[\![k-w, k+1]\!]$. Similarly, we build a footprint $\tilde{\tau_-} = \tilde{\lambda_0^-}, \theta_1^-, \tilde{\lambda_1^-}, \dots, \theta_m^-, \tilde{\lambda_m^-}$ that corresponds to τ_- where the information about round $k+1$ from λ_p^+ has been added to all restricted configurations. This is possible because moves on round $k-w$ has no effect on round $k+1$. Indeed, moves on round $k-w$ may only have effect on rounds below $k-w$ and on round $k-w+1$ for round increments (this last case is why we require that $w \geqslant 1$ so that $k-w+1 < k+1$). Observe that $\tilde{\lambda_p^+} = \tilde{\lambda_0^-}$: they coincide on rounds $k-w+1$ to k where they are equal to λ_{com}, on round $k-w$ by construction of λ_p^+ and on round $k+1$ by construction of λ_0^-. Let T be the footprint obtained by concatenating $\tilde{\tau_+}$ and $\tilde{\tau_-}$. We have $\mathsf{footprint}_{[\![k-w+1,k+1]\!]}(\tilde{\tau_-}) = \lambda_p^+ = \mathsf{restr}_{[\![k-w+1,k+1]\!]}(\tilde{\lambda_p^+})$ (footprint of length 0 with one restricted configuration) and $\mathsf{footprint}_{[\![k-w,k]\!]}(\tilde{\tau_+}) = \lambda_0^- = \mathsf{restr}_{[\![k-w,k]\!]}(\tilde{\lambda_0^-})$. Therefore all restricted configurations in $\tilde{\tau_-}$ merge into one single restricted configuration where we project onto $[\![k-w+1, k+1]\!]$, and the same is true for $\tilde{\tau_+}$ when we project onto $[\![k-w, k]\!]$. This proves that $\mathsf{footprint}_{[\![k-w,k]\!]}(T) = \tau_-$ and $\mathsf{footprint}_{[\![k-w+1,k+1]\!]}(T) = \tau_+$.

Assume that the property is true if τ_{com} has m steps, and suppose that τ_{com} has $m+1$ steps. We decompose $\tau_- = t_-, \theta, s_-$ and $\tau_+ = t_+, \theta, s_+$ where t_- and t_+ coincide on rounds $k-w+1$ to k, their projection on these rounds has exactly m steps, θ is the $m+1$-th move of τ_{com} and s_- and s_+ have no move with effect on rounds $k-w+1$ to k. By induction hypothesis, there exists a footprint t on $[\![k-w, k+1]\!]$ such that $\mathsf{footprint}_{[\![k-w,k]\!]}(t) = t_-$ and $\mathsf{footprint}_{[\![k-w+1,k+1]\!]}(t) = t_+$. By the base case of the induction, there exists s such that $\mathsf{footprint}_{[\![k-w,k]\!]}(s) = s_-$ and $\mathsf{footprint}_{[\![k-w+1,k+1]\!]}(s) = s_+$. We claim that $T := t, \theta, s$ is a footprint. Let λ_t be the last restricted configuration of t and λ_s the first restricted configuration of s; we prove that $\lambda_t \xrightarrow{\theta} \lambda_s$. All conditions about rounds $k-w+1$ to k are satisfied because of τ_{com}. Conditions about round $k-w$ are satisfied because of τ_- and conditions about round $k+1$ are satisfied because of τ_+. Finally, T projects to τ_- on $[\![k-w, k]\!]$ and to τ_+ on $[\![k-w+1, k+1]\!]$, which concludes the inductive step. □

We now provide a sufficient condition to merge footprints into a common execution:

Lemma 4.24. *Let $K \in \mathbb{N}$, $w \in \mathbb{N}$ such that $w \geqslant v-1$, $(\tau_k)_{0\leqslant k\leqslant K}$ and $(T_k)_{1\leqslant k\leqslant K}$ such that:*

— *for all $k \in [\![0, K]\!]$, τ_k is a footprint on $[\![k-w, k]\!]$,*

— *for all $k \in [\![1, K]\!]$, T_k is a footprint on $[\![k-w-1, k]\!]$,*

— *for all $k \in [\![1, K]\!]$, $\mathsf{footprint}_{[\![k-w-1,k-1]\!]}(T_k) = \tau_{k-1}$,*

— *for all $k \in [\![1, K]\!]$, $\mathsf{footprint}_{[\![k-w,k]\!]}(T_k) = \tau_k$.*

There exists an abstract execution ρ such that, for all $k \in [\![0, K]\!]$, $\mathsf{footprint}_{[\![k-w,k]\!]}(\rho) = \tau_k$.

Proof. The high-level idea of this proof is to merge footprints two by two using Lemma 4.23, and to repeat the operation until we have only one large footprint on rounds 0 to K, which can be seen as an abstract execution.

For the formal proof, we proceed by induction on K. First, if $K = 0$, footprint τ_0 only has moves at round 0 and may be seen as an abstract execution. Suppose that the property is true for K, and consider $(\tau_k)_{k\in[\![0,K+1]\!]}$, $(T_k)_{k\in[\![1,K+1]\!]}$ satisfying the hypothesis. For all $k \in [\![1, K]\!]$, T_k and T_{k+1} both have projection τ_k on rounds $[\![k-w, k]\!]$, hence thanks to Lemma 4.23 applied with $w' := w+1$ and $k' := k+1$, there exists U_k on rounds $[\![k-w-1, k+1]\!]$ that projects to T_k and T_{k+1} on $[\![k-w-1, k]\!]$ and $[\![k-w, k+1]\!]$ respectively. By applying the induction hypothesis on (T_k) and (U_k) with $K' := K-1$ and $w' := w + 1$, there exists an abstract execution ρ such that, for all $k \in [\![1, K + 1]\!]$, $\mathsf{footprint}_{[\![k-w-1,k]\!]}(\rho) = T_k$; this implies that, for all $k \in [\![0, K+1]\!]$, $\mathsf{footprint}_{[\![k-w,k]\!]}(\rho) = \tau_k$, concluding the proof of Lemma 4.24. □

Finally, we prove that, over a window of rounds of height $v + 1$, a footprint of an abstract execution in normal form can be stored in polynomial space. Of course, this is not possible if we store the value of the rounds explicitely, because their is no a priori bound on these values. It is however possible if we store the values of the rounds relatively to the highest round of the window, which is itself not stored explicitely.

Lemma 4.25. *Let ρ be an abstract execution in normal form, let $k \in \mathbb{N}$. The footprint $\mathsf{footprint}_{[\![k-v-1,k]\!]}(\rho)$ has length at most $(v + 3)(5|Q| + |\mathbb{D}|)$ and it can be stored in polynomial space $O((v + 1)^2|Q|^2|\Delta||\mathbb{D}|\mathsf{dim})$, assuming that round values are stored relatively to k.*

Proof. Let $\tau := \mathsf{footprint}_{[\![k-v-1,k]\!]}(\rho)$. Any step of τ corresponds to a step of ρ at a round between $k-v-2$ and k. By Lemma 4.20, there are at most $(v + 3)(5|Q| + |\mathbb{D}|)$ such steps in ρ, so that τ has length in $O((v + 1)|Q||\mathbb{D}|)$. Each move of τ can be stored in $O(|\Delta|(v + 1))$ and each restricted configuration of τ in $O((v+1)|Q| + |\mathbb{D}|(v+1)\mathsf{dim})$ if the round values are stored relatively to k. □

The space bound above is expressed as a function of $v+1$ because v may be equal to 0. The bound from Lemma 4.25 could be made much smaller with a more thorough analysis; however, this bound is sufficient to prove that such footprints can be stored in polynomial space. Note that the hypothesis that v is encoded in unary in the input is required to argue that the above bound is polynomial.

4.5.4 A Polynomial-Space Algorithm for Cover

We now prove Theorem 4.16 by providing a polynomial-space procedure to decide Cover. The pseudocode of the algorithm can be found in Algorithm 3. Thanks to Savitch's theorem [Sav70], we may design a non-deterministic algorithm. The high-level idea of the algorithm is to guess the execution footprint by footprint until state q_f is seen. We may look for an abstract execution in normal form, which gives a bound of the length of the footprints to consider. The guessed footprints are of height $v+1$ (*i.e.*, they are on rounds $[\![k-v-1, k]\!]$ for $k \in \mathbb{N}$). In fact, one could work with footprints of height v except when $v = 0$, in this case we need footprints of height at least 1 to be able to check that round increment steps are valid. Note that Algorithm 3 does not always terminate, because the `for` loop at line 3 allows for infinite computations. We start by analyzing this non-terminating algorithm to argue that it has an accepting computation whenever the instance is positive; we will later discuss how to guarantee termination.

1 **Input:** A Cover instance $(\mathcal{P}, q_f)$
2 $\tau \leftarrow \langle \emptyset, \bot^{\dim(v+1)} \rangle$; `// footprint on` $[\![-v-1,-1]\!]$ `with one config, no step`
3 **for** *k from* 0 *to* $+\infty$ **do**
4 Guess $T = \lambda_0, \theta_1, \ldots, \theta_\ell, \lambda_\ell$ with $\lambda_i \in \Gamma_{[\![k-v-1,k]\!]}$ and $\ell \leqslant (v+3)(5|Q| + |\mathbb{D}|)$;
5 **if** *there is i for which we do not have* $\lambda_i \xrightarrow{\theta_{i+1}} \lambda_{i+1}$ **then** Reject;
6 **if** $\mathsf{footprint}_{[\![k-v-1,k-1]\!]}(T) \neq \tau$ **then** Reject;
7 **if** $\lambda_0 \neq \mathsf{restr}_{[\![k-v-1,k]\!]}(\sigma_0)$ **then** Reject;
8 $\tau \leftarrow \mathsf{footprint}_{[\![k-v,k]\!]}(T)$;
9 **if** q_f *appears in* τ **then** Accept ;

Algorithm 3: Non-deterministic algorithm for round-based Cover

Lemma 4.26. *Algorithm 3 has an accepting computation on input $(\mathcal{P}, q_f)$ if and only if $(\mathcal{P}, q_f)$ is a positive instance of Cover.*

Proof. We start with the high-level idea of the proof. If there is an execution covering q_f, then there is an abstract execution in normal form that does so, and it is not difficult build a

computation that guesses all footprints of this execution and eventually accepts. Conversely, from an accepting computation of the algorithm, we use Lemma 4.24 to build an execution covering q_f that projects to the footprints guessed by the computation.

Suppose first that there is an accepting computation of Algorithm 3. For all $k \in [\![0, K]\!]$, let τ_k be the footprint on $[\![k-v, k]\!]$ assigned to τ at line 8 during iteration k, and let T_k be the footprint on $[\![k-v-1, k]\!]$ guessed for T at line 4 during iteration k (both τ_k and T_k are footprints because of the test at line 5). By the definition of τ at line 8, we have $\mathsf{footprint}_{[\![k-v,k]\!]}(T_k) = \tau_k$, and because of the test at line 6, we have $\mathsf{footprint}_{[\![k-v-1,k-1]\!]}(T_k) = \tau_{k-1}$. By applying Lemma 4.24 on $(\tau_k)_{k\in[\![0,K]\!]}$ and $(T_k)_{k\in[\![1,K]\!]}$, there exists an abstract execution $\rho : \sigma_s \xrightarrow{*} \sigma$ such that, for all $k \leqslant K$, $\mathsf{footprint}_{[\![k-v+1,k]\!]}(\rho) = \tau_k$. Because all tests at line 7 have passed, we have that, for all $k \leqslant K$, $\mathsf{restr}_{[\![k-v,k]\!]}(\sigma_s) = \mathsf{restr}_{[\![k-v,k]\!]}(\sigma_0)$. Without loss of generality, in σ_s, all locations on rounds $\ell > K$ are empty and all registers of rounds $> K$ are blank, so that $\sigma_s = \sigma_0$. Because q_f appears in τ_K, it appears in some configuration of ρ. By Proposition 4.18, $(\mathcal{P}, q_f)$ is a positive instance of Cover.

Conversely, suppose that $(\mathcal{P}, q_f)$ is a positive instance of Cover. By Proposition 4.18, there is an abstract execution $\rho : \sigma_0 \xrightarrow{*} \sigma$ and $K \in \mathbb{N}$ such that $(\sigma, K) \in \mathsf{loc}(\sigma)$. By Lemma 4.19, we may assume that ρ is in normal form. For every $k \in [\![-1, K]\!]$, let $\tau_k \coloneqq \mathsf{footprint}_{[\![k-v,k]\!]}(\rho)$, and for all $k \in [\![0, K]\!]$, let $T_k \coloneqq \mathsf{footprint}_{[\![k-v-1,k]\!]}(\rho)$. We prove, by induction on $k \leqslant K$, there is a computation of Algorithm 3 that does the following. It starts with $\tau = \tau_{-1}$ and for every $k' \leqslant k$, if the computation gets to iteration k' then:

(i) it starts iteration k' with $\tau = \tau_{k'-1}$, and

(ii) it guesses $T = T_{k'}$ and then sets $\tau = \tau_{k'}$, and

(iii) it does not reject at lines 5, 6 and 7.

First, $\tau_{-1} = \langle \emptyset, \perp^{\mathsf{dim}(v+1)} \rangle$ because all registers of negative rounds always hold value $\perp$, so that iteration 0 starts with $\tau = \tau_{-1}$. Let $k \leqslant K$, and suppose that the statement is true for $k-1$. By induction hypothesis, we have a computation of Algorithm 3 that respects the conditions for all iterations before k. By hypothesis (iii), the computation did not reject until then; if the computation accepted before iteration k, then we are done. Suppose now that this is not the case, *i.e.*, that the considered computation gets to iteration k. Applying (ii) for $k' = k-1$ proves (i) for $k' = k$. By definition, T_k is a footprint on $[\![k-v-1, k]\!]$ of length bounded by $(v+3)(5|Q| + |\mathbb{D}|)$ by Lemma 4.20, so that T_k can be guessed for T at line 4 and passes the test at line 5. Also, $\mathsf{footprint}_{[\![k-v-1,k-1]\!]}(T_k) = \tau_{k-1}$ which is the value of τ at the beginning of iteration k by (i), so that iteration k passes the test at line 6. The first restricted configuration λ_0 of T_k is equal to $\mathsf{restr}_{[\![k-v,k]\!]}(\sigma_0)$ by definition of T_k, so that iteration k passes the test at line 7. We have proved

that iteration k does not reject so that (iii) holds. Also, this means that the value of τ is set to τ_k at line 8 during iteration k, satisfying (ii). This concludes the inductive step. Lastly, the computation accepts at some iteration $k \leqslant K$. Indeed, by (iii), the computation never rejects, therefore it either accepts at some iteration $k < K$ or gets to iteration K. Suppose that it gets to iteration K; in this case, it sets $\tau = \tau_K$. By definition, $\tau_K = \mathsf{footprint}_{[\![K-\nu,K]\!]}(\rho)$ therefore q_f appears in τ_K and the computation accepts at iteration K. □

We have proved that the algorithm accepts if and only if the instance is positive. There are two remaining tasks: to argue that the algorithm works in polynomial space and to provide a termination criterion. By Lemma 4.25, the footprint T can be stored in $O((\nu+1)^2|Q|^2|\Delta||\mathbb{D}|)$, and the same is true for footprint τ. This relies on the idea that k is not stored explicitly in T and τ, and that all round values are stored relatively to k. Indeed, although the instructions inside the `for` loop of Algorithm 3 are written with variable k, they can be implemented without using the explicit value of k [2]. If the same value of T appears twice in two differents iterations of the same computation, then the computation has looped. By the pigeonhole principle, there is no reason to let the computation perform more iterations than there are possible values for T. Let B be the total number of values for T; B is exponential in the size of the system and can be computed. Therefore, one can change the `for` loop at line 3 to make it stop when $k = B$. This guarantees termination of the algorithm. The values of k and B are exponential in the size of the system therefore can be stored in polynomial space. This proves that Algorithm 3 works in polynomial space, concluding the proof of Theorem 4.16.

4.6 Handling Presence Constraints

In the previous section, we have proved that the problem of Cover can be solved in polynomial space. The techniques developed and the algorithm provided, however, are far from being specific to Cover: the fact that the reachability objective is a coverability objective actually only appears at line 9 in Algorithm 3. In particular, this algorithm can be directly extended to round-based Target by simply changing the test at line 9. Extending the result of Theorem 4.16 to round-based PRP requires more work and is the topic of this section.

Theorem 4.27. *Round-based PRP is PSPACE-complete.*

2. Except for the iteration $k = 0$ where the test at line 7 is different for the one performed in iterations $k > 0$, because $(q_0, 0)$ may be in λ_0 while (q_0, k) may not for all $k > 1$. This does not affect the pigeonhole argument because the value of T with $k = 0$ cannot be equal to a value of T of a later iteration.

The PSPACE-hardness comes from Theorem 4.12. We here prove membership in PSPACE. To do so, we use the same idea as in Section 4.5, which is to guess the witness execution footprint by footprint. There are however many new technical considerations, because we need to check whether the constructed abstract execution satisfies the presence constraint. Unlike for Cover, this cannot be tested on each footprint separately, because atomic propositions of a presence constraint refer to various rounds which are not all in the same footprint. For this reason, we need to store intermediate information about what parts of the presence constraint have already been satisfied, and what parts have not.

Example 4.28. Recall that agreement of Aspnes' noisy consensus algorithm can be stated as the fact that, in $\mathcal{P}_{\text{Asp}}$, no configuration can be reached that satisfies $\psi \coloneqq (\exists k_1\, \mathsf{popu}(R_0, k_1)) \wedge (\exists k_2\, \mathsf{popu}(R_1, k_2))$. To solve the corresponding instance $(\mathcal{P}_{\text{Asp}}, \psi)$ of PRP, we guess, footprint by footprint, an abstract execution whose last configuration satisfies ψ. There is no a priori bound on the difference between k_1 and k_2, so that the truth value of ψ cannot be evaluated on each footprint separately. We need to remember, as we are guessing the witness execution, whether we have already found a witness for k_1 or not, and same for k_2.

4.6.1 Description of the Algorithm

The pseudocode can be found in Algorithm 4. The `for` loop on k relies on the same principles as in Algorithm 3, which is to guess the execution footprint by footprint. This corresponds to line 6; the checks performed on T are the same as in Algorithm 3. The function `NDInit`, `NDComputeIteration` and `TestPresenceConstraint` are in charge of handling the presence constraint. The algorithm is non-deterministic; whenever the algorithm must "check" something, it rejects if the condition is not met.

To handle the presence constraint ψ, we split it into smaller formulas. This is mostly done in some initial phase performed by `NDInit` (line 10). Recall that atomic propositions are of the form $\mathsf{popu}(q, t)$ or $\mathsf{cont}(\mathsf{rg}_r[t], \mathsf{d})$ with t a term, which is either of the form $k + m$ with k a free variable and $m \in \mathbb{N}$ or simply equal to a constant $m \in \mathbb{N}$. Let $\mathsf{Cons}(\psi)$ be the set of closed atomic propositions in ψ whose term is constant, along with the negations of such propositions. Atomic propositions in $\mathsf{Cons}(\psi)$ must be checked at a fixed round. We start by deciding which atomic propositions in $\mathsf{Cons}(\psi)$ are true and which are false (line 11). We put in a set C all closed atomic propositions that have to be true, to check them later. For each atomic proposition ϕ in $\mathsf{Cons}(\psi)$, either ϕ is in C or $\neg\phi$ is in C (but not both). We then simplify ψ according to the truth value of closed atomic propositions of $\mathsf{Cons}(\psi)$: a closed atomic proposition ϕ in ψ is

Input: A PRP instance $(\mathcal{P}, \psi)$
$E, U, C \leftarrow \emptyset$;
$\tau \leftarrow \langle \emptyset, \bot^{\mathsf{dim}(v+1)} \rangle$;
`NDInit(`E, U, C`)` ;
for k *from* 0 *to* $+\infty$ **do**
  Guess $T = \lambda_0 \theta_1 \dots \theta_\ell \lambda_\ell$ a footprint on $[\![k-v-1, k]\!]$ with $\ell \leqslant (v+2)(5|Q| + |\mathbb{D}|)$ such that $\mathsf{footprint}_{[\![k-v-1,k-1]\!]}(T) = \tau$ and $\lambda_0 = \mathsf{restr}_{[\![k-v,k]\!]}(\sigma_0)$;
  $\tau \leftarrow \mathsf{footprint}_{[\![k-v,k]\!]}(T)$;
  `NDComputeIteration(`E, U, C, λ_ℓ, k`)` ;
  if `TestPresenceConstraint(`E, U, C`)` **then** Accept ;
Function `NDInit(`E, U, C`)` **:**
  `/* Puts in` U`,` E `and` C `the APC to check: universal in` U`, existential in` E`, closed in` C`. Modifies` ψ`. */`
  Guess $C \subseteq \mathsf{Cons}(\psi)$ and simplify ψ by setting all atomic propositions in C to true ;
  Guess $X \subseteq \mathsf{APC}(\psi)$ s.t. ψ is true when all APC in X are true ;
  Add all universal APC in X to E and all existential APC in X to U ;
Function `NDComputeIteration(`E, U, C, λ, k`)` **:**
  for *"*$\forall \ell\ \phi$*" in* U **do**
    Guess $L \subseteq \mathsf{AP}(\phi[\ell \leftarrow k])$ s.t. $\phi[\ell \leftarrow k]$ is true when formulas in L are true ;
    Add all formulas in L to C ;
  for *"*$\exists \ell\ \phi$*" in* E **do**
    if $\phi[\ell \leftarrow k]$ *guessed to be true* **then**
      Guess $L \subseteq \mathsf{AP}(\phi[\ell \leftarrow k])$ s.t. $\phi[\ell \leftarrow k]$ is true when formulas in L are true ;
      Add all formulas in L to C ; Remove "$\exists \ell\ \phi$" from E ;
  for ϕ *in* C *related to round* k **do**
    Check that ϕ is satisfied by λ ; Remove ϕ from C ;
Function `TestPresenceConstraint(`E, U, C`)` **:**
  if $E \neq \emptyset$ **then return** false ;
  for $\phi \in C$ *or "*$\forall \ell\ \phi$*" in* U **do**
    if $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \not\models \phi$ **then return** false ; `// Exec. cannot stop at round` k
  return true ;

Algorithm 4: Non-deterministic algorithm for round-based PRP

simplified to true if $\phi \in C$, and to false if $\neg\phi \in \psi$. After this transformation, there are no closed atomic propositions left in ψ.

This means that, after the above step performed at line 11, all atomic presence constraints in ψ start with a quantifier. Let $\mathsf{APC}(\psi)$ be the set of atomic presence constraints (APC for short) that are either present in ψ or whose negation is present in ψ. We guess a subset $X \subseteq \mathsf{APC}(\psi)$ such that ψ is satisfied when all APC in X are evaluated to true and their negations are evaluated to false (line 12). We split atomic presence constraints in X into two sets: we put universal APC of X (of the form $\forall \ell\ \phi$) in U and existential APC of X (of the form $\exists \ell\ \phi$) in E (line 13).

Overall, we have selected a number of smaller formulas to check. Formulas in C refer to constant rounds and are checked at these rounds only. Formulas in U are checked at every round. For each formula in E, the algorithm will guess at which round the formula is true.

The algorithm guesses the execution footprint by footprint. At a given round k, after guessing the footprint as in the algorithm for Cover, there are three tasks to perform:

(i) check that atomic propositions in C that are related to round k are satisfied,

(ii) check that all universal APC in U are satisfied,

(iii) guess which existential APC in E are satisfied and check that they are.

These tasks are performed in `NDComputeIteration`. The last two tasks, (ii) and (iii), are performed first in `NDComputeIteration`, because they add new APC to check in C. While universal APC are checked at every round, for existential APC, we guess at line 19 which ones are satisfied at round k. There are, however, other non-deterministic choices performed for both (ii) and (iii). Indeed, (ii) and (iii) require to guess, in the APC, which atomic propositions are satisfied and which are not; we put those that have to be true in C, so that they are checked at line 23 of this iteration or of subsequent ones. More formally, given an APC $\forall \ell\ \phi$ or $\exists \ell\ \phi$ that must be satisfied at round k, we denote by $\mathsf{AP}(\phi)$ the set of atomic propositions appearing in ϕ and of negations of atomic propositions appearing in ϕ. We guess (at line 16 for U and at line 20 for E) $L \subseteq \mathsf{AP}(\phi)$ so that setting atomic propositions in L to true makes ϕ true. Because atomic propositions in L may be of the form $\ell + m$, they may need to be checked at higher rounds. For this reason, we simply add all atomic propositions in L to C (line 17 for U and line 21 for E), to be checked at round k or at later rounds. Finally, `NDComputeIteration` checks all atomic propositions in C related to round k at line 23. Because such an atomic proposition ϕ is related to round k, it makes sense to test whether it is satisfied by λ, because all information about round k can be found in λ. At iteration k, we only add to C atomic propositions related to rounds $\geqslant k$, so that, after `NDComputeIteration`, all atomic propositions in C are related to rounds $> k$.

It remains to explain under which conditions the algorithm accepts. This is the role of function `TestPresenceConstraint`. In this function, we check whether we can stop the execution at round k, leaving all rounds $\geqslant k{+}1$ untouched. First, we check that E is empty (line 25). This means that a round has been guessed for every existential formula that needed to be checked. Moreover, we check that remaining formulas in C and U would be satisfied at rounds $\geqslant k{+}1$ if these rounds are left untouched by the execution, which is done in lines 26 and 27. The test is expressed under the condition $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \models \phi$, and is implemented as follows. Any formula ϕ that is in C at this stage is about some round $\ell \geqslant k+1$ and must be either of the form $\neg\mathsf{popu}(q, \ell)$ or of the form $\mathsf{cont}(\mathsf{rg}_r[\ell], \bot)$. A universal APC $\forall \ell\, \phi$ must be satisfied on arbitrarily large rounds $\geqslant k{+}1$, hence must be true when evaluating in ϕ all $\mathsf{popu}(q, t)$ to false, all $\mathsf{cont}(\mathsf{rg}_r[t], \bot)$ to true and all $\mathsf{cont}(\mathsf{rg}_r[t], \mathsf{d})$ to false for $\mathsf{d} \neq \bot$. Note that if the universal APC does not evaluate to true during this test, then it will in fact never be satisfied and the computation has no hope of accepting at further iterations. We could have detected this issue in `NDInit` and rejected directly, but for the sake of simplicity we leave it as part of the test for termination.

Example 4.29. Consider $\phi_1 := \forall k\, \mathsf{popu}(q, k) \vee \mathsf{cont}(\mathsf{rg}_r[k], \bot)$ and $\phi_2 := \forall k\, \mathsf{cont}(\mathsf{rg}_r[k], \mathsf{d})$ with $\mathsf{d} \neq \bot$. One has $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \models \phi_1$, but $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \not\models \phi_2$. There is no hope of finding $\sigma \in \mathsf{Post}^*(\sigma_0)$ such that $\sigma \models \phi_2$.

4.6.2 Correctness of the Algorithm

We now prove that the algorithm is correct, which is expressed using the following lemma:

Lemma 4.30. *$(\mathcal{P}, \psi)$ is a positive instance of round-based PRP if and only if there exists an accepting computation of Algorithm 4 on input $(\mathcal{P}, \psi)$.*

First, consider a computation of the algorithm that accepts at round $K \in \mathbb{N}$. For all $k \in [\![0, K]\!]$, let τ_k be the footprint on $[\![k{-}v{+}1, k]\!]$ guessed by the algorithm during iteration k. By Lemma 4.24, with the same reasoning as in the proof of Lemma 4.26, there exist $\sigma_0 \in \Gamma_0$ and an execution $\rho : \sigma_0 \xrightarrow{*} \sigma$ such that, for all $k \leqslant K$, $\mathsf{footprint}_{[\![k-v+1,k]\!]}(\rho) = \tau_k$. Moreover, ρ leaves rounds $\geqslant K$ untouched.

We claim that, for every formula P that is in U, E or C at some point in the computation, $\sigma \models P$. Let L be a closed atomic proposition that has been in C at some point. If it was removed from C at line 23, then C is satisfied by λ hence by σ. If it has remained in C until the end, then it is related to round $\ell \geqslant K{+}1$ and $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \models L$, hence $\sigma \models L$.

Consider $\exists \ell\, \phi$ that has appeared in E at some point in the computation. At some iteration k, $\exists \ell\, \phi$ is removed from E at line 21. All closed atomic propositions guessed at line 20 are added to C at line 21 hence are satisfied by σ, thus $\sigma \models \phi[\ell \leftarrow k]$ and $\sigma \models \exists \ell\, \phi$.

Similarly, consider $\forall \ell\, \phi$ that has appeared in U at some point. By the same argument, for all $k \leqslant K$, $\sigma \models \phi[\ell \leftarrow k]$. Also, thanks to the verification at lines 26 and 27, for all $k \geqslant K{+}1$, $\sigma \models \phi[\ell \leftarrow k]$, which proves that $\sigma \models \forall \ell\, \phi$.

We have proved that all formula that are in U, E and C at some point in the computation are satisfied by σ. This proves that all atomic propositions guessed at line 11 and all APC guessed at line 12 are satisfied by σ. Because ψ is satisfied when all these subformulas are true, $\sigma \models \psi$ and $(\mathcal{P}, \psi)$ is a positive instance of PRP.

We now prove the converse implication: suppose that there exists $\rho : \sigma_0 \xrightarrow{*} \sigma$ with $\sigma \models \psi$. Since ρ is an abstract execution, it has finitely many steps and there exists K such that ρ has no move with effect on rounds $> K$. We build an accepting computation of the algorithm as follows. First, the computation of the algorithm guesses σ_0 as initial configuration. At line 11, it guesses all atomic propositions ϕ in $\mathsf{AP}(\psi)$ such that $\sigma \models \phi$. This modifies ψ, let ψ' be the new formula. By construction of ψ', we have $\sigma \models \psi$ if and only if $\sigma \models \psi'$. At line 12, the computation guesses all APC $\phi \in \mathsf{AP}(\psi')$ such that $\sigma \models \phi$. This means that all formulas added to C, E and U in `NDInit` are satisfied by σ.

At iteration k, the computation guesses $T = \mathsf{footprint}_{[\![k-v-1,k]\!]}(\rho)$, so that the restricted configuration λ obtained is equal to $\mathsf{restr}_{[\![k-v,k]\!]}(\sigma)$. For every $\forall \ell\, \phi$ in U, it guesses for L at line 16 exactly the formulas in $\mathsf{AP}(\phi[\ell \leftarrow k])$ that are satisfied by σ; $\phi[\ell \leftarrow k]$ is true when all formulas in L are true because $\sigma \models \phi[\ell \leftarrow k]$. At line 19, it guesses exactly the formulas $\exists \ell\, \phi$ in E for which $\sigma \models \phi[\ell \leftarrow k]$. In this case, it guesses for L at line 20 exactly the formulas in $\mathsf{AP}(\phi[\ell \leftarrow k])$ that are satisfied by σ. Either way, formulas added to C are satisfied by σ.

We have proved that all formulas added to C, E and U at any point in the computation are satisfied by σ. It is clear that the computation never rejects, *i.e.*, that it passes all "check" instructions. We now argue that the computation accepts at some iteration $k \leqslant K + 1$. By contradiction, suppose that the computation finishes iteration $K + 1$ without accepting. This means that the call to `TestPresenceConstraint` at iteration $K + 1$ returns false. We perform a case analysis on the possible reasons. If $E \neq \emptyset$, then there is $\exists \ell\, \phi$ in E that was guessed true in no iteration of the computation. By definition of the computation, $\sigma \not\models \phi[\ell \leftarrow k]$ for all $k \leqslant K$. We have however $\sigma \models \exists \ell\, \phi$, hence there is $k' > K+1$ such that $\sigma \models \phi[\ell \leftarrow k']$. However, σ has all rounds above K untouched, so that $\sigma \models \phi[\ell \leftarrow k']$ implies $\sigma \models \phi[\ell \leftarrow K+1]$, so that $\exists \ell\, \phi$ is removed from ϕ at iteration $K+1$. Moreover, if we have ϕ in C at the end of iteration $K+1$, then

ϕ is related to some round $k' > K + 1$, but round k' is untouched in σ so that $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \models \phi$. Lastly, for each $\forall \ell\, \phi$ in U, we have in particular $\sigma \models \phi[\ell \leftarrow K + 1]$ so that $\langle \emptyset, \bot^{\mathsf{Reg}} \rangle \models \phi$. This proves that, if the computation gets to iteration $K + 1$, then it accepts at this iteration. We thus have constructed an accepting computation of the algorithm. This concludes the proof of Proposition 4.30.

4.6.3 Space Complexity and Termination

For termination, we proceed as we did in Section 4.5 for Cover: we prove that the algorithm works in polynomial space, so that the algorithm loops after an exponential number of iterations and we can add a counter that stops the algorithm and rejects once the number of iterations has reached this exponential bound. Therefore, it remains to prove that Algorithm 4 works in polynomial space.

Regarding the footprints τ and T, the argument is the same as for the algorithm for Cover in Section 4.5. It therefore remains to prove that the sets E, U and C can always be stored in polynomial space. Let N be the number of atomic propositions in ψ; also, let M be the largest value of m such that a term of the form m or $k + m$ appears in ψ. By hypothesis, such values of m are given in unary, so that M is polynomial in the size of the input. As before, we store round values relatively to k; because round values considered at round k cannot exceed $k + M$, all round values considered are storable in polynomial space. The number of atomic presence constraints put to E and U is in $O(N)$ and E and U can be stored in polynomial space. The same is true for the number of elements added to C in `NDInit`. However, the size of C may in fact become larger, because more atomic propositions are added to C at lines 17 and 21 and these atomic propositions do not necessarily appear in ψ initially. However, the number of elements added to C at a given iteration may not exceed $|\mathsf{AP}(\psi)| = O(N)$ and a given element does not stay in C for more than $M + 1$ iterations. This bounds the maximal number of elements in C at a given point of the computation by $O(N + NM)$, so that C can be stored in polynomial space. This proves that Algorithm 4 works in polynomial space.

We have provided a non-deterministic polynomial-space algorithm solving round-based PRP, which proves that the problem is in PSPACE, concluding the proof of Theorem 4.27.

4.7 About Unary Encoding

In the previous sections, we proved that round-based Cover and round-based PRP are both PSPACE-complete. This result, however, required the hypothesis that we use unary encoding for the integer constants in the input, namely the visibility range v and the constants in the terms of the presence constraint. This choice of unary encoding makes a lot of sense with respect to the motivations of the model: because no practical example is known with large integer constants, we do not want them to have a large influence on the complexity results. Nonetheless, what happens when they are encoded in binary instead remains an interesting theoretical question. As we will now see, this choice drastically increases the complexity, which is another argument in favor of our choice of unary encoding.

Proposition 4.31. *Round-based PRP and round-based Cover are both EXPSPACE-complete when the visibility range v is given in binary or, in the case of round-based PRP, when the integer constants in the presence constraint are encoded in binary.*

Proof. To obtain membership in EXPSPACE, we use Algorithm 4 again. The analysis of the space complexity provides a bound polynomial in $|Q|, \mathsf{dim}, v, |\mathbb{D}|, |\Delta|, |\psi|$ and M where M is the largest constant in ψ. With binary encoding, this bound is now exponential in the size of the input. This means that the result from Theorem 4.27 translates to an EXPSPACE membership when all integers in the input are encoding in binary.

We now prove that this complexity upper bound is tight by providing a matching complexity lower bound. We prove EXPSPACE-hardness for two distinct problems: round-based Cover with v encoded in binary, and round-based PRP with the constants in the presence constraint encoded in binary (but v encoded in unary).

First, round-based Cover is EXPSPACE-hard when v is encoded in binary. We proceed by reduction from the halting problem for a deterministic Turing machine with a tape of exponential size $2^n - 1$. The idea is, for every i, to use rounds $i\,2^n$ to $(i+1)2^n - 2$ to encode the i-th configuration of the Turing machine, and to use round $(i+1)2^n - 1$ as a separator between configurations. As in the proof of Theorem 4.12, we design the protocol so that two different executions cannot write different values to the same register, which is possible because the Turing machine is deterministic. We set $\mathsf{dim} = 3$: each round has one register that plays the role of a cell of the Turing machine, along with another register that is set to the state of the machine if the head is at this cell. The third register per round is used to encode a binary counter in the style of the protocols of Proposition 4.7 in order to write the initial configuration of the Turing machine along with the separator symbol. To write subsequent configurations of the Turing machine,

processes proceed round by round. A process at round $k = i\,2^n + j$ with $i \geqslant 1$ reads the values of the registers $k - 2^n - 1$, $k - 2^n$ and $k - 2^n + 1$ to compute, in configuration i, the symbol in the cell number j, whether the head is at cell number j and, if yes, what the state of the machine is. If a process computes that the Turing machine halts, it goes to state q_f.

For round-based PRP, if v is encoded in unary but the constants in the presence constraint ψ are encoded in binary, then we can proceed similarly. In this case, processes will simply write symbols to the registers without checking that they form valid configurations and that these configurations are the ones of the Turing machine. The first 2^n rounds, however, will contain the initial configuration of the Turing machine thanks to the binary counter. In ψ, using universal quantification over the rounds, we enforce that the values written respect the rules of the Turing machine. This takes the form of implications such as "for every k, if the content of the cell at round $k+1$ is symbol a and the head is at round $k+1$ with state q, then the cell at round $k+1+2^n$ contains symbol b and the head is at the cell of round $k + 2^n$ with state q'". Implications of this shape allow us to enforce that the execution of the round-based ASMS encodes a run of the Turing machine. This corresponds to enforcing that the head of the Turing machine follows the rules of the Turing machine and that the contents of the cells do not change except when the head is there and performs a write transition. In ψ, when a halting state of the Turing machine has been written, the round above must contain a special round symbol. Rounds are allowed to remain blank only above the round where the end symbol has been written. This is expressed by: "for all k, round $k + 1$ is blank if and only if k is blank or k contains the end symbol". This means that an abstract configuration $\sigma \in \mathsf{Post}^*(\sigma_0)$ such that $\sigma \models \psi$ must faithfully encode the run of the Turing machine. Also, because σ must have blank rounds, it must have the end symbol in some register so that σ with $\sigma \models \psi$ exists if and only if the run of the Turing machine halts, concluding the proof. □

4.8 Perspectives

We defined round-based ASMS, a new model of shared-memory systems which aims to capture round-based shared-memory distributed algorithms such as Aspnes' noisy consensus algorithm. In this model, we defined a general problem, namely the round-based presence reachability problem, and established that this problem is decidable and PSPACE-complete.

Although the complexities of these problems are settled, the algorithm provided is non-deterministic and its stopping criterion relies on an exponential number of iterations, so that its interest is mostly theoretical. There are several lines of attack to make this algorithm applicable

in practice. Non-deterministic choices would be implemented by branching instructions whose number must be limited (some ideas to limit non-deterministic choices for COVER may be found in [BMSW22] with so-called *first-write orders*) and for which we would have to find heuristics to start with the most likely candidates. Also, instead of making the algorithm run exponentially many iterations on every positive instance, one could provide a termination criterion so that we may accept at earlier iterations. An inspiration for such a termination criterion could be dynamic cutoff detection [KKW10], where one provides conditions, in a distributed system, under which increasing the number of processes will not allow to cover more states. In our case, one would want to detect that the next round will not offer better chances to satisfy the property that earlier rounds. More specifically for COVER, this could mean to detect that we have already obtained all coverable states because some fixpoint condition is satisfied.

There remain round-based shared-memory algorithms that cannot be captured by the round-based ASMS model. We illustrate this with two examples of round-based shared-memory algorithms. The first one is the algorithm from [RS12], that relies on failure detection and so-called closure sets, where processes may deposit values and collect values deposited by others. There is one closure set by round, and a process may deposit to the closure set of its current round and collect values from the closure set one round below. Another famous algorithm is the one from [GR07]; like Aspnes' noisy consensus algorithm, this algorithm relies on a race between processes, but this race is circular in similar fashion to a stadium. This algorithm relies on so-called weak counters; the number of weak counters depends on the number of processes, and each weak counter is implemented using an unbounded number of binary registers. Finding parameterized models for these algorithms as we did Aspnes' noisy consensus algorithms appears challenging, because they are significantly more complex. In order to retain decidability, one would have to choose wisely the set of primitives in the model.

On the contrary, one could study whether one can reduce the expressive power of the model to obtain better complexity while retaining, *e.g.*, the ability to model Aspnes' noisy consensus algorithm. The latter ambition seems, however, hard to achieve. Indeed, having an unbounded number of rounds generally gives the ability to explore exponential sets such as the set of valuations over n variables, which naturally leads to PSPACE-hardness.

Another family of open problems is related to the study of more general properties on round-based ASMS. For example, one could define a notion of generalized reachability expression akin to the one introduced in Chapter 3, but using presence constraints as basic blocks, and study the emptiness problem of such expressions. Also, in line with Chapter 3, one could study the problem of stuttering LTL verification on round-based ASMS.

Finally, there is an important remaining task related to the automated verification of Aspnes' noisy consensus algorithm. While we provided an algorithm that allows to decide properties such as validity and agreement, we have no such tools for termination. As mentioned in Section 4.2, the termination of Aspnes' noisy consensus algorithm relies on the hypothesis that the scheduler is stochastic, in which case, under reasonable assumptions on the probability distribution, termination occurs almost surely. The techniques developed in this chapter are unapplicable when it comes to verification of probabilistic properties under stochastic schedulers. This is the topic of the next chapter.

Chapter 5

Round-Based ASMS under Stochastic Schedulers

5.1 Introduction

In Chapter 2 and Chapter 4, we studied existential and universal properties over the set of executions. Implicitly, we restricted our study of roundless and round-based ASMS to non-deterministic (and often adversarial) scheduling. However, many distributed algorithms rely on randomization for termination, as it is the case for Aspnes' noisy consensus algorithm (Algorithm 2). This algorithm is deterministic at the level of the processes, but relies on randomized scheduling for termination. This randomization at the level of the scheduler models the natural difference of speed between processes. In this chapter, we study round-based ASMS from Chapter 4 with this type of randomization; we formalize it with so-called *stochastic schedulers* where, at each step, the next process to act is selected uniformly at random among all processes. The objective is to design techniques for automated verification of almost-sure termination of round-based algorithms such as Aspnes' noisy consensus algorithm. In this model, we are interested more specifically in the properties of almost-sure coverability (the final state is covered with probability 1) and almost-sure termination (with probability 1, all processes end up in the final state). However, as we will see, when combining the stochastic setting with round-based systems, even simple questions may hide complex mathematic problems arising from random walks. In [BMRSS16], the authors turn almost-sure coverability into a non-probabilistic property by proving its equivalence with the inclusion $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$. Because of random-walk behaviors, this property does not hold in round-based ASMS. This means that, in round-based ASMS, almost-sure coverability and termination may rely on fundamentally probabilistic behaviors, which is bad news for our ambitions. We explore a few natural restriction to prevent random-walk behaviors, but without success. In particular, random walks still occur when all processes are required to progress in round at the same average pace, and also under another choice of scheduler more similar to the one considered

in [Asp02]. We therefore consider a very restrictive property, *almost-sure obstruction-freedom*. This property is inspired by a similar notion used in the area of distributed algorithms [Asp24, Chapter 27] that is typically satisfied by consensus algorithm. This property requires that, from every reachable configuration, a process left playing in isolation terminates with probability 1. This property implies almost-sure termination, and deciding whether a protocol is almost surely obstruction-free is a PSPACE-complete problem.

This chapter is organized as follows. In Section 5.2, we introduce our stochastic schedulers. In Section 5.3, we present the probabilistic problems that we are interested in. In Section 5.4, we highlight results from the literature related to roundless ASMS under stochastic schedulers [BMRSS16]. In Section 5.5, we exhibit a first protocol with random walk behaviors. We discuss some ideas to prevent these behaviors in Section 5.6 and Section 5.7.In Section 5.8, we introduce and study *almost-sure obstruction-freedom*. We conclude the chapter with some perspectives in Section 5.9.

Related Works

The impossibility results for asynchronous consensus [FLP85; DDS87; LA87; FR03] have lead the distributed algorithms community to consider other solutions. One such solution is to rely on probabilistic algorithms, a concept first introduced by Rabin [Rab76]. Probabilities can be used in a distributed setting as a symmetry-breaker, as first done in [LR81] for mutual exclusion. This idea is particularly well-suited for consensus algorithms. Indeed, the impossibility result from [FLP85] proves the existence of non-terminating executions, but can be circumvented by requiring termination with probability 1 and not for all executions. The first randomized consensus algorithm was suggested by Ben-Or [Ben83]. The randomization approach was then applied to shared-memory systems, notably in [Abr88; AH90]; see [Asp03] for a survey.

In asynchronous distributed systems, one does not assume that processes perform their computations at the same speed. This allows, *a priori*, all consistent interleaving of actions. Previously in this thesis, we implicitly worked under the assumption that any difference of speed is possible, so that the system must be correct for all interleavings. In practice, one expects that some of the behaviors considered are very unlikely, for example because they require than some processes are recurrently much faster than others. It is therefore reasonable to rule out such behaviors. One classic approach consists in enforcing some fairness conditions, as we briefly mentioned in Chapter 3. Many notions of fairness exist, see [GH19] for a survey. Another approach consists in modeling the noise from the environment as a stochastic process, which we call here stochastic scheduler. In most of the work with stochastic schedulers, they are seen as

a way to give the adversary less power, and one evaluates how distributed algorithms perform against this randomized opponent [GM01; GM14; ASV15].

From the viewpoint of designers of distributed algorithms, stochastic schedulers may also be seen as allies. Indeed, they may be used as symmetry-breakers for consensus problems. This concept, first introduced in [BT85], makes use of deterministic, simple algorithms that rely on the probabilistic noise of the environment to guarantee termination with probability 1. This approach has since been used in various settings to design consensus algorithms that do not always terminate, but do so with probability 1 under the right assumptions about the scheduler [Asp00; Asp02; GMMB15]. These algorithms implicitly exploit the irregularity of speed of the processes; this irregularity has been exploited in other ways, *e.g.*, to generate random numbers [ABGS18]. Another distributed computing model where the scheduler is often considered to be stochastic is the one of population protocols. In the original work on population protocols [AADFP04], the authors indeed consider that the scheduler selects the next interacting pair uniformly at random, and are interested in almost-sure termination and termination with high probability (*i.e.*, probability that tends to 1 when the number of participating processes tends to infinity). They also consider quantitative probability questions such as the expected number of steps until termination. A finer approach consists in attempting to quantify how much randomness an algorithm needs, as done in [BBBG15] in the setting of population protocols. See [AR09] for a survey of works on population protocols from the distributed algorithms community, and in particular see [AR09, Section 1.6.] which is devoted to random interactions.

Reasoning about distributed randomized systems is a complex task; to quote Lehmann and Rabin [LR81], "proofs of correctness for probabilistic distributed systems are extremely slippery". This calls for tools allowing formal verification of such systems, which is in fact a very challenging objective. As an example, a round-based randomized consensus algorithm due to Aspnes and Herlihy [AH90], where the processes are allowed to use coin tosses, was formally proved correct in an *ad hoc* proof of about 30 pages in [PSL00]. This proof notably uses involved probabilistic notions such as random walks. Since designing automated model-checking tool is typically harder than *ad hoc* proofs, this gives an idea of how big of a challenge it could be. Most of the work from the literature focuses on algorithm where the randomization is at the level of process, and where the behavior of the scheduler is specified in a non-probabilistic manner. For example, [LR16] studies liveness in systems where the processes are randomized but the scheduler is arbitrary (all executions are possible). In [LLMR17], the problem of interest is termination and the scheduler satisfies a finitary fairness condition: in a finitary-fair (infinite) execution, there is k such that an enabled process is always picked after at most k steps. In

[BKLW21], the authors are interested in almost-sure termination in randomized round-based systems where the scheduler is assumed to be round-rigid, which means that processes move from round to round in a synchronous manner. See [Ber20] for a survey on formal verification of randomized algorithms.

In this thesis, we are interested in another family of randomized models, where randomization is at the level of the scheduler. This model is common in the distributed algorithms community, less so in the formal verification community. In the parameterized setting where the number of processes is arbitrary, it is not possible to abstract the system into a finite Markov chain or Markov decision process, which calls for new techniques. An example of study under stochastic scheduler in a parameterized setting is [BMRSS16], which we will describe in more detail in Section 5.4. The model considered in [BMRSS16] is, except for a few details, our roundless ASMS model from Chapter 2. The authors consider the problem of almost-sure coverability of a state, and they reduce this probabilistic problem to a non-probabilistic one using what we will call decisiveness in Section 5.4.

In [BKL14], a study is made of so-called probabilistic basic parallel processes under stochastic scheduler. In this setting, a selected process chooses its next action at random depending on its state only (there is no communication), and may generate other processes. The first part of the paper studies stochastic schedulers, and establishes that the exact choice ofprobability distribution the of the scheduler is not important. Despite the fact that processes do not communicate, the authors prove that the problem of almost-sure coverability of an upward-closed set, while decidable, is non-elementary. When working with population protocols, the distributed computing community oftentimes relies on stochastic schedulers. Classic model checking was used to prove almost-sure termination of population protocols when fixing the initial configuration [PLD08; CDFS11]. By contrast, the numerous works about parameterized verification of population protocols (*i.e.*, when the initial configuration is not fixed) tend to abstract the random character of the scheduler under the notion of fairness, see [Esp17; BEJK18] for surveys. There are, however, a few works on verification of population protocols under stochastic schedulers. In [EGLM16], population protocols are considered under a uniform scheduler. The authors establish that quantitative model checking (for example, whether the probability to cover a given state is at least $\frac{1}{2}$) is undecidable. However, qualitative model checking is decidable: one can decide almost-sure satisfiability of linear-time properties over actions, but the problem is Ackermann-complete. Again with a uniform scheduler, the expected number of steps until convergence of a population protocol is studied in [BEK18] where the authors provide an algorithm that, given a protocol, gives a reasonable asymptotic upper bound on this expected number

of steps. In [BEHKM20], the authors consider stochastic schedulers in replicated systems, a generalization of population protocols. They only assume that the scheduler is memoryless and always gives non-zero probability of selecting a given process, as the exact scheduler does not matter. [BEHKM20] proves that qualitative model checking of linear-time properties is decidable but non-elementary, and they provide an incomplete procedure that is more efficient on practical instances.

5.2 Stochastic Schedulers

In this section, we formally introduce the stochastic schedulers used in this chapter. We start by presenting the underlying idea, which should suffice to follow the remainder of this chapter. The stochastic scheduler selects an execution at random, step by step. From a given configuration, the scheduler picks the next configuration in the following fashion. First, the scheduler selects, uniformly at random among all processes, which process acts in the next step. It then selects the transition performed by the process, uniformly at random among all transitions that can be applied by the process. The obtained stochastic process is Markovian, in the sense that the choice of the next step only depends on the current configuration and not on previous configurations or steps.

This stochastic scheduler gives all processes the same chance of being selected at each step, regardless of their number of applicable transitions. This requires ruling out the possibility that a process has no applicable transition: if a process with no applicable transition is selected, we consider that it performs a dummy transition that has no effect.

> In all the following, we assume that all processes always have at least one applicable transition.

Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \bot, \Delta \rangle$ be a round-based ASMS protocol. For all $\gamma \in \Gamma$, $q \in Q$ and $k \in \mathbb{N}$, let $T_\gamma(q,k) \subseteq \mathcal{M}$ be the set of moves whose source location is (q,k) and that can be applied from γ. By the above hypothesis, for all γ, q, k such that $\mathsf{loc}(\gamma)(q,k) > 0$, $|T_\gamma(q,k)| > 0$. We define, given a configuration $\gamma = \langle \mu, \vec{d} \rangle \in \Gamma$ and a move $\theta = ((q, \mathsf{act}, q'), k) \in \mathcal{M}$, the probability $\mathsf{P}(\gamma \xrightarrow{\theta})$ that θ is applied from γ by:

$$\mathsf{P}(\gamma \xrightarrow{\theta}) \coloneqq \frac{\mu(q,k)}{|\mu| \ |T_\gamma(q,k)|}.$$

Observe that, for a given γ, we have

$$\sum_{\theta\in\mathcal{M}} \mathsf{P}(\gamma \xrightarrow{\theta}) = \sum_{(q,k)\in\mathcal{L}} \sum_{\theta\in T_\gamma(q,k)} \frac{\mu(q,k)}{|\mu|\ |T_\gamma(q,k)|} = \sum_{(q,k)\in\mathcal{L}} \frac{\mu(q,k)}{|\mu|} \sum_{\theta\in T_\gamma(q,k)} \frac{1}{|T_\gamma(q,k)|} = 1.$$

With this definition, all processes has the same chance of acting at each step. Indeed, from $\gamma = \langle \mu, \vec{d} \rangle$, the probability that the move selected has source location (q,k) is $\frac{\mu(q,k)}{|\mu|}$.

Let $\gamma \in \Gamma$. We aim to assign a probability $\mathbb{P}_\gamma$ to (suffixes of) executions starting from γ. The rigorous way to do so would be to define a probabilistic space and a measure over the set of infinite executions (see, *e.g.*, [Par11]). Again, to save ourselves this tedious effort, we proceed less rigorously and rely on random variables instead. To do so, we consider two sequences of random variables: $(C_i)_{i\in\mathbb{N}}$ with values in Γ and $(T_i)_{i\geqslant 1}$ with values in $\mathcal{M}$. We define $C_0 \coloneqq \gamma$ and for all $i \geqslant 1$:

— for all $\theta \in \mathcal{M}$, $\mathbb{P}_\gamma(T_i = \theta) \coloneqq \mathsf{P}(C_{i-1} \xrightarrow{\theta})$;

— C_i is the configuration such that $C_{i-1} \xrightarrow{T_i} C_i$.

We also require that the stochastic process is *Markovian*, in the sense that T_i only depends on the past insofar as it depends on C_{i-1}. Formally, for all $i \geqslant 1$, for all $\theta_1, \ldots, \theta_i \in \mathcal{M}$, for all $\gamma_{i-1} \in \Gamma$,

$$\mathbb{P}_\gamma(T_i = \theta_i \mid C_{i-1} = \gamma_{i-1}, T_{i-1} = \theta_{i-1}, \ldots, T_1 = \theta_1) = \mathbb{P}_\gamma(T_i = \theta_i \mid C_{i-1} = \gamma_{i-1}) = \mathsf{P}(\gamma_{i-1} \xrightarrow{\theta_i}).$$

Implicitly, this defines a probability measure over executions starting from γ. We will refer to the probability that a given event occurs *from configuration* γ to refer to this probability measure. In particular, the event that a set of configurations U is *reached from* γ is the event $\bigcup_{i\in\mathbb{N}}(C_i \in U)$. A convenient observation is that this scheduler is memoryless so that, when $C_i = \gamma_i$, the probability distribution of configurations $(C_j)_{j\geqslant i}$ corresponds to the probability distribution $\mathbb{P}_{\gamma_i}$, *i.e.*, the probability distribution from configuration γ_i.

Note that, from a given initial configuration, the stochastic process is in fact a Markov chain. However, the state space of the Markov chain is infinite even for a fixed number of processes.

Remark 5.1. An interpretation of this definition is that we have randomization at two distinct levels: at the scheduler level to select the process, but also at the level of the processes to select the transition among all possible transitions. This second level of randomization is necessary when the protocol is non-deterministic, but is not our main topic of interest. In fact, if the protocol is deterministic, which is the case in Aspnes' noisy consensus algorithm (Algorithm 2), then randomization lies only at the level of the scheduler.

5.3 Stochastic Properties

In this section, we introduce the stochastic properties studied in this chapter. They correspond to the probabilistic qualitative versions of the Cover and Target problems from Chapter 4. Here, qualitative means that we are interested in whether a given event occurs with probability 1 or not (or, dually, whether it occurs with probability 0 or not). By contrast, quantitative properties would refer to specific values; examples of quantitative properties would be whether a given set is reached with probability $\frac{1}{2}$ or whether the expected number of steps to reach it is greater than some value. In this thesis, we will only study qualitative properties.

Let $\mathcal{P} = \langle Q, q_0, \mathsf{dim}, \mathbb{D}, \perp, \Delta \rangle$ be a round-based ASMS protocol with a special state q_f. For all $n \geqslant 1$, we use $\mathbb{P}_n$ as a shorthand for $\mathbb{P}_{\gamma_0(n)}$, *i.e.*, the probability measure of the stochastic process starting on the initial configuration of size n.

Fix an arbitrary starting configuration $\gamma \in \Gamma$. Let $(C_i)_{i \in \mathbb{N}}$ be the random variables defined in Section 5.2 from γ. For every $U \subseteq \Gamma$, we denote by $\Diamond U$ the event $\bigcup_{i \in \mathbb{N}} C_i \in U$. The value $\mathbb{P}_\gamma(\Diamond U)$ therefore represents the probability that, starting from γ, U is eventually visited. In the system with n processes, we call *almost-sure reachability* of U the fact that $\mathbb{P}_\gamma(\Diamond U) = 1$. We are interested in two families of sets U. To define them, we assume that the protocol has a special state q_f which is a *stalemate state*, *i.e.*, all transitions with source state q_f are self-loops with internal actions, so that processes in q_f have terminated and may not longer influence the rest of the system.

The first one is when U is a *coverability objective*, *i.e.*, the set of all configurations that cover state q_f. Formally, the coverability objective associated to state q_f is $\uparrow q_f \coloneqq \{\gamma \in \Gamma \mid \exists k \in \mathbb{N}, \mathsf{st}(\gamma)(q_f, k) > 0\}$. State q_f is *covered almost surely* from γ when $\mathbb{P}_\gamma(\Diamond \uparrow q_f) = 1$. In the system with n processes, we call *almost-sure coverability* the fact that $\mathbb{P}_n(\Diamond \uparrow q_f) = 1$.

The other family of sets U considered in this chapter is the set $\mathsf{Cons}(q_f)$ of configurations where all processes are in state q_f. Formally, $\mathsf{Cons}(q) \coloneqq \{\gamma \mid \forall k \in \mathbb{N}, \forall q' \neq q, \mathsf{st}(\gamma)(q', k) = 0\}$. $\mathsf{Cons}(q_f)$ can be interpreted as the set of configurations where all processes have terminated. For this reason, in the system with n processes, we call *almost-sure termination* the property $\mathbb{P}_n(\Diamond \mathsf{Cons}(q_f)) = 1$.

Both properties are more restrictive that there non-probabilistic counterparts, because almost-sure reachability of a set U implies that U is reachable, but the converse is not true. Moreover, as before, almost-sure termination implies almost-sure coverability. Note that almost-sure coverability and almost-sure termination are properties and not problems, because they apply for a given value of n. Corresponding problems could be, *e.g.* whether they hold for large values

of n. However, in this chapter, we will mostly analyze the properties for fixed values of n, in the hope to rephrase them into equivalent, non-probabilistic properties.

5.4 Literature on ASMS under Stochastic Schedulers

Almost-sure coverability in *roundless* ASMS under stochastic schedulers has been studied in [BMRSS16]. We present here their approach, which we will use as a guideline in the next sections when considering the same question for round-based ASMS.

The approach from [BMRSS16] leverages the following crucial observation. Assume that q_f is a stalemate state. Fix $n \geqslant 1$: the set of configurations with n processes is finite, therefore there are $\lambda > 0$, $M \in \mathbb{N}$ such that, for all $\gamma \in \mathsf{Pre}^*(\uparrow q_f)$, there is probability at least λ that q_f is covered within the next M steps. This implies that $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$ if and only if one can reach from $\gamma_0(n)$ a configuration from which q_f may no longer be covered. This property was not named in [BMRSS16]. We dub it *decisiveness*, and call a system that satisfies this property *decisive*. We borrow this terminology from a study of infinite Markov chains where an infinite Markov chain is called *decisive* if, with probability 1, it reaches either a final state or a state from which no final state may be reached [AHM07]. For a given $\mathcal{P}, q_f$ and n, decisiveness is formally stated as follows:

$$\mathbb{P}_n(\Diamond \uparrow q_f) = 1 \text{ if and only if } \mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f). \qquad \text{(Decisiveness)}$$

Decisiveness is convenient because it turns almost-sure coverability into a non-probabilistic property so that probabilistic reasoning is not needed. If $\mathcal{P}$ is a roundless ASMS with a stalemate state, then the system is decisive for every $n \geqslant 1$. Based on this observation, the authors of [BMRSS16] prove the following results.

Theorem 5.2 ([BMRSS16])**.** *In roundless ASMS:*

- *there is a* cutoff $N \geqslant 1$ *such that either* $\mathbb{P}_n(\Diamond \uparrow q_f) = 1$ *for all* $n \geqslant N$ *(*positive cutoff*) or* $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$ *for all* $n \geqslant N$ *(*negative cutoff*),*
- *there is such a cutoff of value at most doubly-exponential in the size of the protocol,*
- *whether the cutoff is positive or negative can be decided in* EXPSPACE.

The previous results have been extended to more general reachability objective, and in particular to almost-sure termination, in [Sta17].

Remark 5.3. Decisiveness in roundless ASMS relies on the fact that, from a given configuration $\gamma_0(n)$, the set $\mathsf{Post}^*(\gamma_0(n))$ is finite. In particular, this equivalence is not specific to the stochastic scheduler defined in Section 5.2. In fact, it applies to any scheduler that is (weakly) fair, *i.e.*, in which, almost surely, if a configuration γ is visited infinitely often and δ can be applied from γ then, with probability 1, there is eventually a step $\gamma \xrightarrow{\delta} \gamma'$. Weak fairness is a very standard property that is satisfied by all reasonable stochastic schedulers. This essentially implies that, in the roundless model, the choice of the scheduler is irrelevant.

In the rest of this chapter, we study almost-sure coverability and almost-sure termination in the round-based ASMS model. To do so, we first ask whether the round-based model is decisive. Decisiveness would indeed be of great help; however, as we will see, it does not hold for round-based ASMS. In fact, the round-based model under a stochastic scheduler possesses complex mathematical behaviors which make analysis a challenging task.

5.5 Round-based ASMS are not Decisive

In this section, we will provide a round-based protocol that disproves decisiveness. This protocol will rely on probabilistic behaviors closely related to random walks. Somehow, this will suggest that, in round-based ASMS under stochastic schedulers, almost-sure coverability is fundamentally probabilistic.

The main theorem of this section is the following.

Theorem 5.4. *There is a round-based ASMS $\mathcal{P}$ with a state q_f such that, for all $n \geqslant 2$:*

— $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$*, but*

— $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$.

The rest of this section is devoted to proving Theorem 5.4. The organization of the proof is as follows. First, we introducing the protocol $\mathcal{P}$ and provide some intuition in Section 5.5.1. We will prove in Section 5.5.2 that this protocol satisfies the desired conditions, under the assumption that some initial gadget works as expected. The description of this initial gadget and the associated proof are postponed to Section 5.5.3.

5.5.1 Some Intuition about the Protocol

We first provide an overall introduction of the protocol as well as some intuition. The protocol $\mathcal{P}$ is partially depicted in Fig. 5.1. Not depicted in the picture is the initial gadget

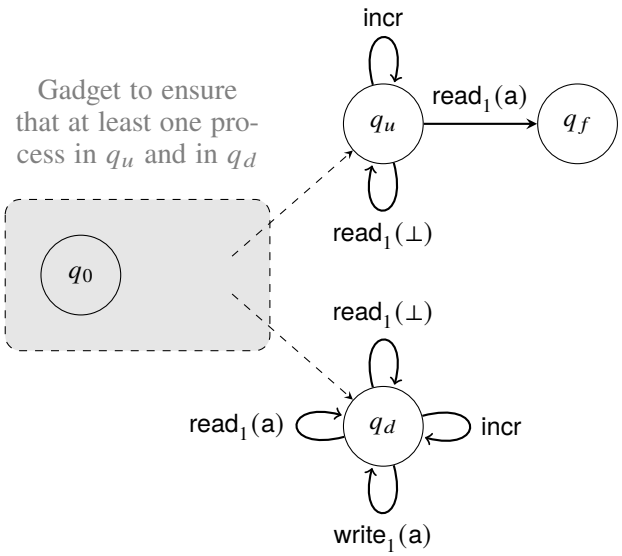


Figure 5.1 – The protocol $\mathcal{P}$ used in the proof of Theorem 5.4.

which guarantees that, almost surely, if $n \geqslant 2$ then at least one process visits $(q_u, 1)$ and at least one process visits $(q_d, 1)$. The details of this gadget are provided in Section 5.5.3; until then, we assume that this gadget works as intended. Under this assumption, we have, for all $n \geqslant 2$, $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$.

We explain why this protocol satisfies $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$. In order to go to (q_f, k), a process must be in (q_u, k) and read a, so that register 1 of round k must contain symbol a. This symbol must have been written by a process in q_d, which is only possible if there is a process in q_d at a round higher than k. In particular, this is not possible if all processes in q_u always remain at higher rounds than all processes in q_d. A process in q_u always has two possible actions, among which one incr, so that it increments its round every two actions on average. By contrast, a process in q_d always has three possible actions, among which one incr, so that it needs on average to be selected three times by the scheduler to increment its round once.

Consider a configuration where n_u processes are in $(q_u, 1)$ and n_d processes are in $(q_d, 1)$. First, there is a non-zero probability that all processes in q_u go in $(q_u, 2)$ while processes in q_d remain idle. We aim at proving that there is a non-zero probability that, forever after, the lowest process in q_u (in terms of rounds) remains strictly higher than the highest process in q_d. The full proof is presented in Section 5.5.2; we explain here the case where $n_u = n_d = 1$. For the sake of simplicity, we deanonymize processes; we denote by p_u the process in q_u and by p_d the process in q_d. Almost surely, infinitely many increment steps are overall performed. For all $i \in \mathbb{N}$, let s_i be the random variable giving the index of the i-th increment step overall

(the i-th step of the execution that is an increment step) and let $R_i(p_u)$ (resp. $R_i(p_d)$) be the random variable giving the round of p_u (resp. p_d) in C_{s_i}, *i.e.*, after the i-th increment step. In particular, we have $R_0(p_u) = 2$ and $R_0(p_d) = 1$. The random variable $R_i(p_u) - R_i(p_d) - 1$ is a 1-dimensional random walk of parameter $p = \frac{3}{5}$. Indeed, at each step, p_u has probability $\frac{1}{2}$ of acting and, when it does, it has probability $\frac{1}{2}$ of performing an increment step, which gives an overall probability of $\frac{1}{4}$. With the same reasoning, at each step, p_d has probability $\frac{1}{6}$ of performing a round increment. Overall, when a round increment is performed, it has probability $\frac{\frac{1}{4}}{\frac{1}{4}+\frac{1}{6}} = \frac{3}{5}$ of being performed by p_u. This proves that the difference of rounds behaves like a positively biased 1-dimensional random walk. Hence, by Proposition 1.3, there is a non-zero probability that p_u always remains at a higher round than p_d and that q_f is never covered. We now extend this observation to the general case.

5.5.2 Processes may Diverge in Rounds

We prove the following result:

Lemma 5.5. *Let $n_u, n_d > 0$, let $\gamma \coloneqq \langle n_u \cdot (q_u, 2) \oplus n_d \cdot (q_d, 1), \bot^{\mathsf{Reg}} \rangle$ be the configuration with n_u processes in q_u at round 2, n_d processes in q_d at round 1 and where all registers are blank. Let $\Gamma_{\leqslant} \coloneqq \{\gamma \mid \exists k \leqslant \ell,\ (q_u, k), (q_d, \ell) \in \mathsf{st}(\gamma)\}$. We have $\mathbb{P}_\gamma(\Diamond\, \Gamma_{\leqslant}) < 1$. In words, from configuration γ, there is a non-zero probability that all processes in q_u always remain at strictly higher rounds than all processes in q_d.*

The rest of Section 5.5.2 is devoted to proving Lemma 5.5. Let $n_u, n_d > 0$, and let $\gamma \coloneqq \langle n_u \cdot (q_u, 2) \oplus n_d \cdot (q_d, 1), \bot^{\mathsf{Reg}} \rangle$. If q_f is covered then $\Gamma_{\leqslant}$ has been visited. For this reason, we equivalently prove the property in the protocol $\mathcal{P}'$ where q_f is removed and the $\mathsf{read}_1(\mathsf{a})$ transition is a self-loop on q_u. This choice makes it more convenient to reason about infinite executions, because it means that a process in q_u stays in q_u forever.

Again, we deanonymize processes. Let P_u be the set of all processes in q_u in γ and P_d be the set of processes in q_d in γ. For every $p_u \in P_u, p_d \in P_d$, for every $i \in \mathbb{N}$, let $s_i(p_u, p_d)$ be the index of the i-th increment performed by a process among $\{p_u, p_d\}$ (where we consider the probability distribution from γ). Almost surely, all $s_i(p_u, p_d)$ are well-defined. Let $\mathrm{Eq}_i(p_u, p_d)$ be the event: "the round numbers of p_u and p_d are equal after step $s_i(p_u, p_d)$".

Lemma 5.6. *Almost surely, finitely many events in $(\mathrm{Eq}_i(p_u, p_d))_{p_u \in P_u, p_d \in P_d, i \in \mathbb{N}}$ occur.*

Proof. Thanks to the Borel-Cantelli lemma (Lemma 1.2), it suffices to prove that

$$\sum_{p_u \in P_u} \sum_{p_d \in P_d} \sum_{i \in \mathbb{N}} \mathbb{P}(\mathrm{Eq}_i(p_u, p_d)) < \infty.$$

Because the two leftmost sums are over finite sets, this amounts to proving that, for every $p_u \in P_u$ and $p_d \in P_d$, $\sum_i \mathbb{P}(\mathrm{Eq}_i(p_u, p_d)) < \infty$. Let $p_u \in P_u, p_d \in P_d$. In γ, p_u is at round 2 and p_d is at round 1. For all $i \in \mathbb{N}$, $\mathrm{Eq}_i(p_u, p_d)$ occurs if, among the first i increment steps performed by the two processes, $\frac{i+1}{2}$ are performed by p_d and $\frac{i-1}{2}$ are performed by p_u. This is only possible when i is odd, let $j \coloneqq \frac{i-1}{2}$. An increment step performed by a process among $\{p_u, p_d\}$ has probability $\frac{3}{5}$ to be performed by p_u and probability $\frac{2}{5}$ to be performed by p_d. Therefore,

$$\begin{aligned}\mathbb{P}(\mathrm{Eq}_{2j+1}(p_u, p_d)) &= \binom{2j+1}{j} (\frac{2}{5})^{j+1} (\frac{3}{5})^j \\ &= \frac{2(2j+1)!\,6^j}{5\,j!\,(j+1)!\,25^j}\end{aligned}$$

By Stirling's formula,

$$\begin{aligned}\frac{2(2j+1)!\,6^j}{5\,j!(j+1)!\,25^j} &\sim \frac{\sqrt{2\pi(2j+1)}\,(2j+1)^{2j+1}\,e^{j+(j+1)}\,6^j}{5\pi\,\sqrt{j(j+1)}\,j^j\,(j+1)^{j+1}\,25^j\,e^{2j+1}} \\ &\sim O(j)(\frac{6(2j+1)^2}{25j\,(j+1)})^j \sim O(j)(\frac{24j^2+24j+6}{25j^2+25j})^j\end{aligned}$$

Asymptotically, $\frac{24j^2+24j+6}{25j^2+25j}$ converges to $\frac{24}{25}$ thus is eventually smaller than, *e.g.*, $\frac{49}{50}$. This asymptotically bounds $\mathbb{P}(\mathrm{Eq}_i(p_u, p_d))$ by a geometric progression of ratio strictly less than 1, hence the sum of all $\mathbb{P}(\mathrm{Eq}_i(p_u, p_d))$ converges. □

Let $\Gamma_= \coloneqq \{\gamma_= \in \Gamma \mid \exists k \in \mathbb{N}, (q_u, k), (q_d, k) \in \mathsf{st}(\gamma_=)\}$. We argue that, almost surely (from γ), $\Gamma_=$ is visited finitely many times. This is not directly proved by Lemma 5.6: it only proves that the set $\Gamma_=$ is *entered* finitely many times. Formally, Lemma 5.6 proves that, almost surely, there are finitely many indices i such that $C_i \notin \Gamma_=$ but $C_{i+1} \in \Gamma_=$. To conclude that $\Gamma_=$ is visited finitely many times, we need to argue that there is probability 0 to stay in $\Gamma_=$ forever. From every configuration $\gamma_= \in \Gamma_=$, there is an execution of length at most n^2 to a configuration in $\Gamma \setminus \Gamma_=$: indeed, it suffices to send, one by one, each process to the nearest round where no other process is. This proves that, from any configuration in $\Gamma_=$, there is probability at least $p = \frac{1}{(3n)^{n^2}}$ to escape $\Gamma_=$ within the next n^2 steps. For all m, the probability to remain in $\Gamma_=$ for mn^2 steps is

less than $(1-p)^m \underset{m\to+\infty}{\longrightarrow} 0$, so that the probability to eventually remain in $\Gamma_=$ forever is zero. Combined with Lemma 5.6, this proves that, almost surely, $\Gamma_=$ is visited finitely often.

By contradiction, if we had $\mathbb{P}_{\gamma'}(\Diamond\,\Gamma_=) = 1$ for all $\gamma' \in \mathsf{Post}^*(\gamma)$, then, by iteratively applying this property, we would have probability 1 of visiting $\Gamma_=$ infinitely often from γ. Hence there exists $\gamma' \in \mathsf{Post}^*(\gamma)$ such that $\mathbb{P}_{\gamma'}(\Diamond\,\Gamma_=) < 1$. We now claim that $\gamma' \notin \Gamma_\leqslant$, using the following argument.

Lemma 5.7. *For every $\gamma_\leqslant \in \Gamma_\leqslant$, $\mathbb{P}_{\gamma_\leqslant}(\Diamond\,\Gamma_=) = 1$.*

Proof. Because $\gamma_\leqslant \in \Gamma_\leqslant$, there is $k \leqslant \ell$ such that $\gamma_\leqslant$ has a process p_u in (q_u, k) and a process p_d in (q_d, ℓ). Almost surely, infinitely many increment steps are performed by p_u and p_d. For all $i \in \mathbb{N}$, let s_i be the random variable giving the index of the i-th increment step performed by a process in $\{p_u, p_d\}$, and let $R_i(p_u)$ (resp. $R_i(p_d)$) be the random variable giving the round of p_u (resp. p_d) in C_{s_i}, *i.e.*, after the i-th increment step. As before, the random variable $R_i(p_u) - R_i(p_d) - k + \ell$ is a 1-dimensional random walk of parameter $p = \frac{3}{5}$. Hence, by Proposition 1.3, it visits $\ell - k$ with probability 1 and $\Gamma_=$ is reached almost surely from $\gamma_\leqslant$. □

This proves that $\gamma' \notin \Gamma_\leqslant$. Any execution from $\Gamma \setminus \Gamma_\leqslant$ to $\Gamma_\leqslant$ must visit $\Gamma_=$, therefore this also proves that $\mathbb{P}_{\gamma'}(\Diamond\,\Gamma_\leqslant) < 1$. Of course, it could still be that the execution ρ from γ to γ' goes through $\Gamma_\leqslant$. However, we can rearrange ρ as follows. In a first phase, we make processes in q_u perform the increment transitions that they performed in ρ. In the second phase, we make processes in q_d perform the increment and write transitions that they performed in ρ. This yields an execution ρ' from γ to γ' which does not visit any configuration in $\Gamma_\leqslant$. Because $\mathbb{P}_{\gamma'}(\Diamond\,\Gamma_\leqslant) < 1$, this proves that $\mathbb{P}_{\gamma}(\Diamond\,\Gamma_\leqslant) < 1$, concluding the proof of Lemma 5.5.

5.5.3 Details about the Initial Part of the Protocol

We denote by Γ_{ok} the set of reachable configurations γ such that, for some $n_d, n_u > 0$, the multiset of states of γ is $n_u \cdot (q_u, 2) \oplus n_d \cdot (q_d, 1)$. Thanks to Lemma 5.5, we have that, for all $\gamma \in \Gamma_{\mathrm{ok}}$, $\mathbb{P}_\gamma(\Diamond{\uparrow}q_f) < 1$. We now aim to design the initial gadget so that, for all $n \geqslant 2$:

— $\mathbb{P}_n(\Diamond\,\Gamma_{\mathrm{ok}}) > 0$, and

— $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$.

Regarding the first point, it suffices to make sure that Γ_{ok} is reachable from $\gamma_0(n)$ for every $n \geqslant 2$. Regarding the second point, it suffices to ensure that, from every configuration reachable from $\gamma_0(n)$, one can still reach a configuration where at least one process is sent to q_u and at least one process is sent to q_d. Indeed, if there is one process p_u in (q_u, k) and a process p_d in

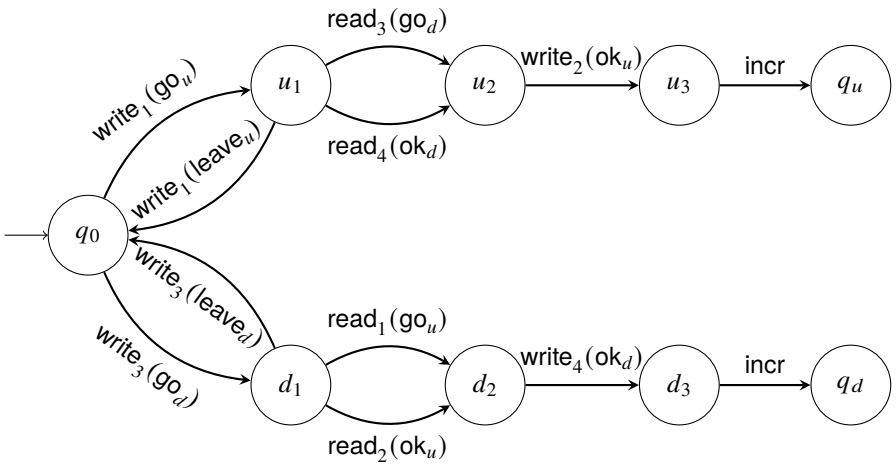


Figure 5.2 – The gadget omitted in Fig. 5.1, allowing to guarantee that at least one process is sent to $(q_u, 1)$ and at least one process is sent to $(q_d, 1)$.

(q_d, ℓ), it is possible to cover q_f: the two processes get to round $\max(k, \ell)$, the process in q_d writes a and then the process in q_u reads a.

The initial gadget is represented in Fig. 5.2. This gadget does not have incr transitions therefore processes in the gadget are at round 0. It also uses four registers and six new symbols: $\{\mathsf{go}_u, \mathsf{leave}_u, \mathsf{ok}_u, \mathsf{go}_d, \mathsf{leave}_d, \mathsf{ok}_d\}$. Because these symbols may be written to the registers of round 0 only, this gadget may not interfere with the rest of the protocol.

For all $n \geqslant 2$, Γ_{ok} can be reached from $\gamma_0(n)$ with the following execution. We first send one process to $(u_1, 0)$ and $n-1$ processes to $(d_1, 0)$. We make the process at u_1 go to $(u_2, 0)$ by reading go_d, then to $(q_u, 1)$, and finally to $(q_u, 2)$ using the incr loop on q_u. We finally make the $n-1$ processes in $(d_1, 0)$ read ok_u from register 2 to get to $(d_2, 0)$ and then to $(q_d, 1)$. This proves, using Lemma 5.5, that $\mathbb{P}_n(\Diamond{\uparrow}q_f) < 1$.

We now prove that $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*({\uparrow}q_f)$. It suffices to prove that, for every $\gamma \in \mathsf{Post}^*(\gamma_0(n))$ that has no process on q_f, there is $\gamma' \in \mathsf{Post}^*(\gamma)$ and k, ℓ such that $(q_u, k), (q_d, \ell) \in \mathsf{st}(\gamma')$. Let $\gamma \in \mathsf{Post}^*(\gamma_0(n))$ that has no process in q_f. We make a case disjunction on γ.

— If γ has a process in $\{u_2, u_3, q_u\}$ and a process in $\{q_d, d_2, d_3\}$, then it can trivially reach a configuration γ' satisfying the specifications.

— Assume that γ has a process in $\{u_2, u_3, q_u\}$, none in $\{d_2, d_3, q_d\}$ and at least one in $\{u_1, q_0, d_1\}$. If there is a process in u_2, we first make it go to u_3, so that we are guaranteed to have a process in $\{u_3, q_u\}$. Such a process must have taken the $\mathsf{write}_2(\mathsf{ok}_u)$ transition. Because no other transition can write to register 2 of round 0, this register must contain ok_u in γ. As a consequence, any process in $\{u_1, q_0, d_1\}$ can get to $\{d_2, d_3, q_d\}$.

— Assume that all processes are in $\{q_0, u_1, d_1\}$. We can put all processes back to q_0, then send some to q_u and some to q_d using the sequence of transitions described above.

— The last two remaining cases is the one where all processes are in $\{u_2, u_3, q_u\}$ and the one where all processes are in $\{d_2, d_3, q_d\}$. We treat the first case, the other case follows by symmetry. We prove by contradiction that this case is in fact not possible. Consider the last process to go in u_2. It cannot have read ok_d because that would imply that some process went to $\{d_3, q_d\}$; therefore it must have read go_d from register 3. This contradicts the fact that the last process to leave d_1 must have written leave_d to register 3.

Overall, we have proved that, for all $n \geqslant 2$, $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ but $\mathbb{P}_n(\Diamond \uparrow q_f)$, concluding the proof of Theorem 5.4.

Remark 5.8. This also proves that the choice of the scheduler might impact whether q_f is almost surely covered. For example, if we modify the scheduler so that processes in q_d are twice more likely to be selected than processes in q_u, then, in the protocol $\mathcal{P}$ built above, q_f would be almost surely covered for all $n \geqslant 2$. This contrasts with the roundless case where, as highlighted in Remark 5.3, the choice of the stochastic scheduler is unimportant.

The protocol built above also hints that some protocols, under stochastic schedulers, have behaviors akin to random walks, which are mathematically challenging to study even on fixed protocols, let alone automatically. This is bad news considering our ambitions to automatically verify almost-sure termination of round-based protocols.

5.6 Regular Increments

In this section and the next one, we discuss some restrictions in the hope to prevent random-walk behaviors from occurring. The first and main idea to restrict random-walk behaviors is to limit the *drift*, *i.e.*, to make it unlikely that processes end up many rounds apart from one another. Indeed, in the protocol built in Section 5.5, the reason why we have $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$ even though $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ is that processes in q_u increment their rounds faster on average than processes in q_d. This means that the gap in terms of rounds between processes in q_u and q_d is likely to become large and tend to infinity, preventing processes from communicating. For this reason, a natural attempt would be to enforce that all processes increment their rounds at the same average rate.

Definition 5.9. Let $m \geqslant 1$, let $\mathcal{P}$ be a protocol with a stalemate state q_f. The protocol $\mathcal{P}$ has *m-regular increments* when, for each cycle π in the underlying graph of $\mathcal{P}$ such that π does not

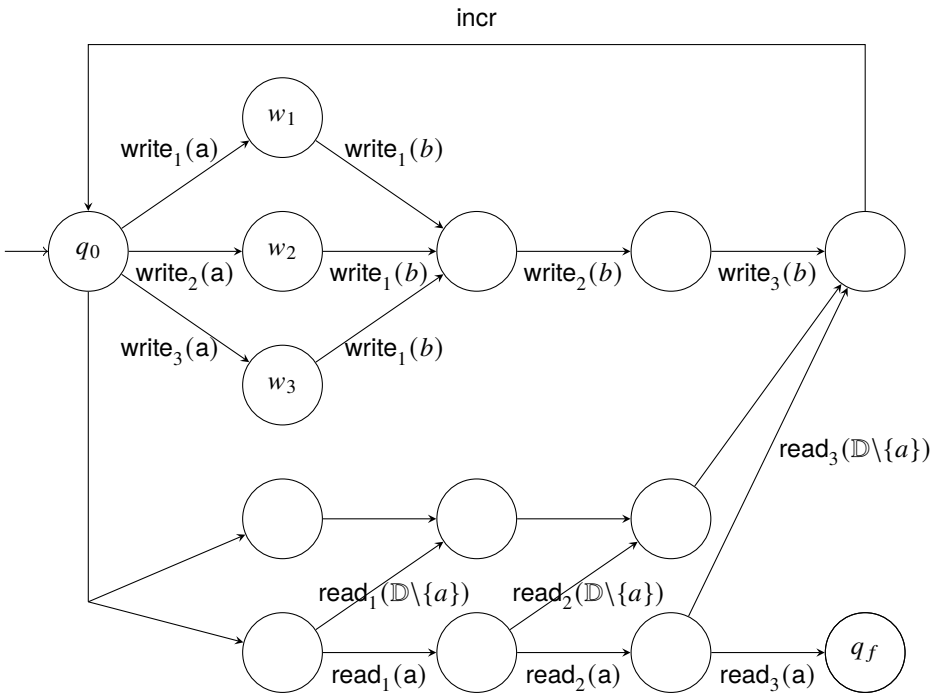


Figure 5.3 – The protocol built in the proof of Proposition 5.10.

visit q_f, π has length exactly m and includes exactly one increment transition.

Hence, each process increments its round once every m actions that it performs (except for processes already in q_f). On average, all processes increment their rounds at the same speed. This, however, does not prevent complex random-walk behaviors from occurring.

Proposition 5.10. *There is a round-based ASMS $\mathcal{P}$ with a special state q_f and 5-regular increments such that, for all $n \geqslant 4$:*

— $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$*, but*

— $\mathbb{P}_n(\Diamond \uparrow q_f) < 1$.

We now prove Proposition 5.10. The protocol is represented in Fig. 5.3. This protocol has 3 registers per round and its data alphabet is $\mathbb{D} \coloneqq \{\bot, \mathsf{a}, \mathsf{b}\}$. At each round, a process may take the top branch to write a to one of the three registers; however, its next three actions will write symbol b to all three registers of its round. In order to reach q_f, a process needs to read a from all three registers of its round. Therefore, when a process gets to (q_f, k), 4 different process are at round k.

Let $n \geqslant 4$. We first observe that we indeed have $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$, because it suffices that 4 processes get at the same round, that one goes in w_i for each $i \in \{1, 2, 3\}$ and

that one last process goes in the bottom branch and reaches q_f. We now argue that there is a non-zero probability that q_f is never covered. It suffices to prove that there is a non-zero probability that, after a few initial steps where processes spread across the first rounds, there are never 4 processes on the same round again. The intuition is that the relative rounds of 4 distinct processes have three degrees of freedom and are similar to a balanced random walk in dimension 3, which is known to have a non-zero probability of never visiting a given point. However, this intuition is hard to formalize because the round differences between pairs of processes are not independent from one another. We instead proceed in similar fashion to Section 5.5. One may work in the protocol where q_f is removed, so that processes never stop performing increments. In this protocol, it suffices to prove that the event "four processes are at the same round" occurs finitely often with probability one. Fix a 4-tuple of processes (p_1, p_2, p_3, p_4); almost surely, this 4-tuple overall performs infinitely many increments. Let Eq_i be the event that all four processes are at the same round once the 4 processes have overall performed i steps. In order to use the Borel-Cantelli lemma (Lemma 1.2), we need to prove that $\sum_i \mathbb{P}(\mathrm{Eq}_i)$ is finite. Fix $i \in \mathbb{N}$, let $i = 20k + m$ with $m < 20$ be the Euclidean division of i by 20; we consider what happens in the first i steps of the execution. We are interested in the probability that, after i steps overall, all 4 processes have performed a number of steps in $[\![5k, 5k + 4]\!]$, so that they all are at round k. For all $j \in \{1, 2, 3, 4\}$, let R_j be the random variable giving the remainder of the Euclidean division by 5 of the number of steps performed by p_j after i overall steps. For all $r_1, r_2, r_3, r_4 \in [\![0, 4]\!]$,

$$\mathbb{P}(\mathrm{Eq}_i, R_1 = r_1, R_2 = r_2, R_3 = r_3, R_4 = r_4) = \frac{(20k + r_1 + r_2 + r_3 + r_4)!}{(5k + r_1)!(5k + r_2)!(5k + r_3)!(5k + r_4)!} \left(\frac{1}{4}\right)^{20k+r_1+r_2+r_3+r_4}.$$

Indeed, having all $\mathrm{Eq}_i, R_1 = r_1, R_2 = r_2, R_3 = r_3$ and $R_4 = r_4$ is equivalent to having, for all $j \in [\![1, 4]\!]$, that the number of steps performed by p_j is equal to $5k + r_j$. Therefore, a simple counting argument allows to prove the equality above.

We now prove that the sum of all i, $\sum_i \mathbb{P}(\mathrm{Eq}_i)$, is finite. By summing over all values of k and of r_1, r_2, r_3, r_4, we obtain:

$$\sum_{i \in \mathbb{N}} \mathbb{P}(\mathrm{Eq}_i) = \sum_{k \in \mathbb{N}} \sum_{r_1, r_2, r_3, r_4 \in [\![0,4]\!]} \frac{(20k + r_1 + r_2 + r_3 + r_4)!}{(5k + r_1)!(5k + r_2)!(5k + r_3)!(5k + r_4)!} \left(\frac{1}{4}\right)^{20k+r_1+r_2+r_3+r_4}$$

By Stirling's approximation, we have that when $k \to +\infty$:

$$\frac{(20k + r_1 + r_2 + r_3 + r_4)!}{(5k + r_1)!(5k + r_2)!(5k + r_3)!(5k + r_4)!}(\frac{1}{4})^{20k+r_1+r_2+r_3+r_4}$$
$$\sim O(1)k^{-\frac{3}{2}} \frac{(20k + r_1 + r_2 + r_3 + r_4)^{20k+r_1+r_2+r_3+r_4}}{(5k + r_1)^{5k+r_1}(5k + r_2)^{5k+r_2}(5k + r_3)^{5k+r_3}(5k + r_4)^{5k+r_4}}(\frac{1}{4})^{20k+r_1+r_2+r_3+r_4}$$
$$\sim O(1)k^{-\frac{3}{2}} \frac{(20k)^{20k+r_1+r_2+r_3+r_4}}{(5k)^{20k+r_1+r_2+r_3+r_4}4^{20k+r_1+r_2+r_3+r_4}} \frac{(1 + \frac{r_1+r_2+r_3+r_4}{20k})^{20k+r_1+r_2+r_3+r_4}}{(1 + \frac{r_1}{5k})^{5k+r_1}(1 + \frac{r_2}{5k})^{5k+r_2}(1 + \frac{r_3}{5k})^{5k+r_3}(1 + \frac{r_4}{5k})^{5k+r_4}}$$
$$\sim O(1)k^{-\frac{3}{2}} \frac{(20k)^{20k+r_1+r_2+r_3+r_4}}{(20k)^{20k+r_1+r_2+r_3+r_4}} \sim k^{-\frac{3}{2}}O(1)$$

The last line of equivalence uses the fact that $(1 + \frac{K}{m})^{m+p} \underset{m\to\infty}{\longrightarrow} e^K = O(1)$. This proves that the sum converges, completing the proof of Proposition 5.10.

This example puts an end to our hope. Not only does the restriction to regular increments fail to restore decisiveness, but it also does not prevent random-walk behaviors. Such complex mathematical behaviors are already hard to analyze on fixed protocols. For example, one could change the protocol above and adapt it with two registers per round, so that 3 processes only are needed at the same round to cover q_f. In this case, the rounds of a 3-tuple of processes has two degrees of freedom and is therefore similar to a 2-dimensional random walk, which almost surely visits all points infinitely often. Therefore, we would have almost-sure coverability of q_f for all $n \geqslant 3$. The fact that, to analyze this simple example, one must invoke rather involved random walk results is a bad sign for our ambitions to perform automated analysis.

5.7 Another Choice of Scheduler

In this section, we mention one last line of attack in the hope to prevent random walks from occurring and to make automated analysis feasible. With the current definition of stochastic schedulers, at each step, all processes have the same probability of acting, regardless of whether they just performed a transition or if they have been idle for a while. It would arguably be more realistic that the scheduler gives greater probability of being selected to processes which have not acted recently. Intuitively, such processes are more likely to be done with their internal computation and to be ready for their next read or write action. A priori, this choice might also limit random-walk behaviors by limiting the drift in rounds.

We model this idea with an alternative notion of stochastic scheduler, which we call *waiting-time scheduler*. This scheduler relies on continuous time and works as follows. Whenever a

process performs an action, the waiting time until its next action is drawn according to some continuous probability distribution. It is important that all distributions are continuous: this way, the probability that two events occur at the same time is zero and the semantics are sequential. After this waiting time, the process performs an action selected randomly among all its possible actions. This is the type of scheduler considered in [Asp02], where the probability distribution of the waiting time is allowed to depend on the last operation performed by the process. Here, we focus on a simpler case, where all waiting time are independent identically distributed.

Definition 5.11 (Waiting-time schedulers)**.** Let P be the set of all processes of the system. A *waiting-time scheduler* uses positive continuous random variables $(T_i(p))_{i\geqslant 1, p\in P}$ that are independent identically distributed of finite mean value $\mu > 0$ and finite variance σ^2 with $\sigma > 0$. For all $j \in \mathbb{N}$ and $p \in P$, the time of the j-th action of process p is $\sum_{i=1}^{j} T_i(p)$. Almost surely, no two actions take place at the same time. The execution is obtained by iteratively selecting the process whose next action takes place at the nearest time in the future, and selecting the transition uniformly at random among all possible transitions of the process.

We prove that such schedulers do not prevent random-walk behaviors from occurring. To do that, we use a generalized version of the protocol from Proposition 5.10. For every $m \geqslant 1$, we build a protocol $\mathcal{P}_m$ with $(m+2)$-regular increments in which, in order to cover q_f, one must have m processes in the same round. We will prove that, for m large enough and under a waiting-time scheduler, we do not have almost-sure coverability of q_f in $\mathcal{P}_m$.

Theorem 5.12. *Let $m \geqslant 6$. Under a waiting-time scheduler, $(\mathcal{P}_m, q_f)$ violates decisiveness.*

We now prove Theorem 5.12. Fix a m-tuple of processes $(p_1, \ldots, p_m)$. As in Sections 5.5 and 5.6, we prove that, almost surely, all m processes are at the same round in finitely many configurations of the executions, using the Borel-Cantelli lemma (Lemma 1.2). We need to argue that (in the protocol where q_f is removed) the sum over $r \in \mathbb{N}$ of probabilities of the following events is finite: "at some point, all processes in the m-tuple have performed exactly r round increments". This event occurs when, at some point, each process of the tuple has performed a number of actions in $[\![(m+2)r, (m+2)r+m+1]\!]$. Fix $r \geqslant 1$, let $a_r \coloneqq (m+2)r$ and $b_r \coloneqq (m+2)r+m+1$. For all $p \in \{p_1, \ldots, p_m\}$ and $N \geqslant 1$, let $S_N(p) \coloneqq \sum_{i=1}^{N} T_i(p)$ be the random variable giving the time of the N-th action of process p. At a given time t, a process p has performed a number of actions between a_r and b_r when $S_{a_r}(p) < t < S_{b_r}(p)$ [1]. Therefore,

1. Because all random variables are continuous, whether we use strict or non-strict inequalities is irrelevant.

there exists a time t at which all processes of the m-tuple are at round r if and only if

$$\max_{j\in[\![1,m]\!]} S_{a_r}(p_j) \leqslant \min_{j\in[\![1,m]\!]} S_{b_r}(p_j). \tag{5.1}$$

We now prove that the sum over r of the probabilities of the event defined in (5.1) converges. Let $r \geqslant 1$ and suppose that (5.1) holds. Fix a pair (p, p') of distinct process of the tuple. We have $\max(S_{a_r}(p), S_{a_r}(p')) \leqslant \min(S_{b_r}(p), S_{b_r}(p'))$. We claim that this implies the following inequality:

$$|S_{a_r}(p) - S_{a_r}(p')| \leqslant \max(S_{b_r}(p) - S_{a_r}(p), S_{b_r}(p') - S_{a_r}(p')). \tag{5.2}$$

Indeed, if $|S_{a_r}(p) - S_{a_r}(p')| > S_{b_r}(p) - S_{a_r}(p)$ then $S_{a_r}(p') \notin [S_{a_r}(p), S_{b_r}(p)]$, which combined with $S_{a_r}(p') \leqslant S_{b_r}(p)$ gives that $S_{a_r}(p') < S_{a_r}(p)$. If we also have $|S_{a_r}(p) - S_{a_r}(p')| > S_{b_r}(p') - S_{a_r}(p')$, then we have $S_{a_r}(p) > S_{b_r}(p')$, a contradiction. This proves that inequality implies (5.1) implies inequality (5.2) for every pair (p, p') of processes.

Let $D_r \coloneqq S_{a_r}(p) - S_{a_r}(p')$ and $Z_r \coloneqq S_{b_r}(p) - S_{a_r}(p) + S_{b_r}(p') - S_{a_r}(p')$. Because $\max(a, b) \leqslant a + b$, it suffices to bound $\mathbb{P}(|D_r| \leqslant Z_r)$ with respect to r. Note that $D_r = \sum_{i=1}^{a_r} T_i(p) - T_i(p')$ and $Z_r = \sum_{i=a_r+1}^{b_r} T_i(p) + T_i(p')$, so that D_r and Z_r are independent. For every i, let $D_i \coloneqq T_i(p) - T_i(p')$. The random variable D_r is equal to the sum $\sum_{i=1}^{a_r} D_i$ where the random variables $D_i \coloneqq T_i(p) - T_i(p')$ are i.i.d. of mean value 0 and of variance $2\sigma^2$. Because $a_r = (m + 2)r$, by central limit theorem, when r goes to infinity the distribution of $\frac{D_r}{\sigma\sqrt{2(m+2)\,r}}$ converges to a normal distribution of mean value 0 and of variance 1. Therefore, for every $z > 0$,

$$\mathbb{P}(|D_r| \leqslant z) = \mathbb{P}(\frac{D}{\sigma\sqrt{(m+2)\,r}} \leqslant \frac{z}{\sigma\sqrt{(m+2)\,r}}) \underset{r\to+\infty}{\longrightarrow} \frac{1}{\sqrt{2\pi}} \frac{z}{\sigma\sqrt{(m+2)\,r}}.$$

By contrast, the random variable $Z_r = \sum_{i=a_r+1}^{b_r} T_i(p)+T_i(p')$ is the sum of $2m+2$ i.i.d. independent variables of fixed distribution, hence the probability distribution of Z_r does not depend on r. By conditioning over possible values for Z_r, this proves that $\mathbb{P}(|D_r| \leqslant Z_r)$ is asymptotically equivalent, when r goes to infinity, to a term in $\frac{O(1)}{\sqrt{r}}$. This bounds the probability of the event defined by (5.2) by a term in $\frac{O(1)}{\sqrt{r}}$.

We apply this reasoning to three disjoint pairs of distinct processes in $\{p_1, \ldots, p_m\}$, which is possible because $m \geqslant 6$. For disjoint pairs, the events of (5.2) are independent, which bounds the probability of the event described by (5.1) by a term in $r^{-\frac{3}{2}}$. Summing these probabilities over r gives a finite sum. By using the same reasoning as in Sections 5.5 and 5.6, this proves

that, for all $n \geqslant 6$, there is a non-zero probability that q_f is never covered. Because we have $\mathsf{Post}^*(\gamma_0(n)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ for all $n \geqslant 6$, this violates decisiveness.

We have proved that waiting-time schedulers do not allow us to retrieve decisiveness and that they do not prevent random-walk behaviors. This is another negative result, on top of the ones from Section 5.5 and Section 5.6. This forces us to lower our ambitions; to do so in a meaningful way, we come back to our motivations and therefore to round-based consensus algorithms.

5.8 Almost-sure Obstruction-Freedom

Round-based consensus algorithms as in [Asp00; GR07; RS12] are meant to be robust with respect to failure of any number of processes. This in particular implies that, in these algorithm, if all processes suddenly crash except one, this process will eventually terminate. We abstract this observation into the notion of *almost-sure obstruction-freedom*.

5.8.1 Definition

Let $\mathcal{P} = (Q, \mathsf{dim}, \mathbb{D}, \bot, \Delta)$ be a round-based ASMS of visibility range v and a special stalemate state q_f. Given two configurations $\gamma = \langle \mu, \vec{d} \rangle, \gamma' = \langle \mu', \vec{d'} \rangle \in \Gamma$, we let $\gamma \preceq \gamma'$ when $\vec{d} = \vec{d'}$ and $\mu \subseteq \mu'$. In words, $\gamma \preceq \gamma'$ when γ can be obtained from γ' by removing some processes. Given a configuration $\gamma = \langle \mu, \vec{d} \rangle$, let $\mathsf{Iso}(\gamma) \coloneqq \{\gamma' \mid \gamma' \preceq \gamma, |\gamma'| = 1\}$ be the set of configurations obtained from γ by removing all processes except one. Given a set $S \subseteq \Gamma$, we let $\mathsf{Iso}(S) \coloneqq \bigcup_{\gamma \in S} \mathsf{Iso}(\gamma)$. Given a configuration $\gamma \in \Gamma$ and $(q, k) \in \mathcal{L}$ such that $\mathsf{st}(\gamma)(q, k) > 0$, the *local view* of a process at (q, k) in γ is the tuple $(q, (\mathsf{data}(\gamma)(\mathsf{rg}_{k-i}[r]))_{i \in [\![0,v]\!], r \in [\![1,\mathsf{dim}]\!]}) \in Q \times \mathbb{D}^{\mathsf{dim}(v+1)}$. In words, the local view of a process at round k includes its state and the content of all registers of rounds $k - v$ to k. Recall that v is the visibility range, so that registers of rounds $k - v$ to k are those that the process can read from round k. To save us some case disjunctions, we assume that registers of negative rounds always contain value $\bot$, so that the local view of a process at round $k < v$ is still an element of $Q \times \mathbb{D}^{\mathsf{dim}(v+1)}$.

In particular, when γ has size 1, the *local view of* γ is the local view of the only process in γ, *i.e.*, the local process at (q, k) where $(q, k) \in \mathsf{st}(\gamma)$.

Definition 5.13 (Almost-sure Obstruction-Freedom)**.** A protocol $\mathcal{P}$ with a special stalemate state q_f is *almost surely obstruction-free*, or *Asof* for short, when for every $n \geqslant 1$, for every $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\gamma_0(n)))$, we have $\mathbb{P}_\gamma(\Diamond \uparrow q_f) = 1$.

In words, in any reachable configuration, if one process is selected and only this process is allowed to act in the remainder of the execution, then this process reaches q_f with probability 1. This is a loosened version of *obstruction-freedom*, a classic notion in distributed algorithm which states that, in any reachable configuration, any process finishes in a bounded number of steps if all the other processes stop. In fact, if all processes are deterministic then the two notions coincide. Obstruction-freedom was first defined in [HLM], see *e.g.* [Asp24, Chapter 27] for an introduction.

5.8.2 Link with Almost-Sure Termination

It is clear that almost-sure obstruction-freedom is a strong property. In fact, it is stronger than almost-sure coverability and even than almost-sure termination for every n.

Theorem 5.14. *If a protocol $\mathcal{P}$ with a special stalemate state q_f is almost surely obstruction-free, then for all $n \geqslant 1$, $\mathbb{P}_n(\Diamond \mathsf{Cons}(q_f)) = 1$.*

The rest of this section is devoted to proving Theorem 5.14. We start with the observation that, from configurations with only one process, there is a convenient bound in the number of steps needed to cover q_f. This bound depends on the highest round of the configuration that is not blank. Recall that, in a configuration γ, round k is blank if, in γ, all registers of round k have symbol $\perp$. The *highest non-blank round* in γ is the maximal k such that round k is not blank in γ; this value is finite for all reachable configurations. In a reachable configuration, one also has a process whose round is at least the highest non-blank round. This is however not true for configurations in $\mathsf{Iso}(\mathsf{Post}^*(\gamma_0(n)))$, because it could be that the process that wrote to the highest non-blank round was removed.

The following lemma bounds the number of steps needed to cover q_f from a configuration of size 1. This bound depends on the distance in terms of rounds that the process has to perform to get above the highest non-blank round. Indeed, if the process is far below the highest non-blank round and it needs to get above this round to get to q_f, then it must perform a large number of steps to get to q_f.

Lemma 5.15. *Let $\gamma \in \Gamma$ of size 1, let K be the highest non-blank round in γ and let k be the round of the process in γ. If one can cover q_f from γ then one can do so with an execution of less than $(\max(K-k,0)+1)|Q||\mathbb{D}|^{\mathsf{dim}(v+1)}$ steps.*

Proof. Let ρ be the shortest execution covering q_f from γ. By the pigeonhole principle, for all ℓ, this execution has at most $|Q||\mathbb{D}|^{\mathsf{dim}}$ configurations where the process is at round ℓ. Indeed,

such configurations only differ in the state of the process and the content of the registers of round ℓ. Therefore, at most $|Q||\mathbb{D}|^{\mathsf{dim}} \max(K-k,0)$ steps in ρ are at rounds strictly less than K. Moreover, when the process is at round $\ell \geqslant K$, the only register contents that matter are the ones in rounds $\ell - v$ to ℓ, *i.e.*, those in the local view of the process; registers of rounds below $\ell - v$ may not be read by the process and registers of rounds above ℓ are all blank. Assume that there are $q \in Q$ and two configurations γ_1, γ_2 appearing in this order in ρ such that γ_1 and γ_2 have the same local view. We build a shorter execution by removing all steps between γ_1 and γ_2 and shifting the rounds of all steps after γ_2 by $k_1 - k_2$ rounds; the obtained execution covers q_f from γ and is shorter than ρ, which contradicts minimality of ρ. By the pigeonhole principle, in ρ, once the process is at round at least K, the process covers q_f within the next $|Q||\mathbb{D}|^{\mathsf{dim}(v+1)}$ steps of ρ. This proves that q_f can be covered from γ in less than $(\max(K-k,0)+1)|Q||\mathbb{D}|^{\mathsf{dim}(v+1)}$ steps overall. □

Another easy observation is that, for all $n \geqslant 1$, any execution with one process can be translated to an execution with n processes where $n-1$ processes remain idle.

Lemma 5.16. *Let $\gamma \in \Gamma$ and $\gamma_1 \in \mathsf{Iso}(\gamma)$. Let ρ_1 be an execution from γ_1 to some configuration γ_1'. There is ρ from γ to some configuration γ' such that $\gamma_1' \in \mathsf{Iso}(\gamma')$. Moreover, ρ has the same underlying sequence of moves as ρ_1.*

Proof. The intuition is that it suffices, in ρ, to leave all processes idle except one. The proof is by induction on the length of ρ_1. The initialization is trivial. For the induction step, it suffices to split ρ_1 before its last step, perform the induction hypothesis on the prefix of length $\mathsf{len}(\rho_1) - 1$ and prove that, from the configuration obtained, one can apply the move performed in the last step of ρ_1. □

Lemma 5.17. *For all $\gamma \in \Gamma$, $\mathsf{Post}^*(\mathsf{Iso}(\gamma)) \subseteq \mathsf{Iso}(\mathsf{Post}^*(\gamma))$.*

Proof. Let $\gamma_1' \in \mathsf{Post}^*(\mathsf{Iso}(\gamma))$; there is $\gamma_1 \in \mathsf{Iso}(\gamma)$ and ρ_1 an execution from γ_1 to γ_1'. There is only one process in ρ. Because this process appears with the same state and round in γ and in γ_1, and because the content of the registers is the same in γ and in γ_1, it is possible to build, from ρ_1, an execution ρ' where only this process acts and where it performs the same moves as in ρ. By letting γ' the end configuration of ρ', we have $\gamma' \in \mathsf{Post}^*(\gamma)$ and $\gamma_1' \in \mathsf{Iso}(\gamma')$. □

We now prove Theorem 5.14. Suppose that $(\mathcal{P}, q_f)$ is AsoF. For all $\gamma \in \Gamma$, we prove the following property by induction on $|\gamma|$: if there is $\gamma_u \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \preceq \gamma_u$, then $\mathbb{P}_\gamma(\Diamond \mathsf{Cons}(q_f)) = 1$. This is true for configurations of size 1 thanks to the AsoF property. Let

$\gamma \in \Gamma, \gamma_u \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \preceq \gamma_u$. Assume that the property is true for configurations of size strictly less than $|\gamma|$. We first prove that $\mathbb{P}_\gamma(\Diamond\uparrow q_f) = 1$, *i.e.*, that q_f is covered almost surely from γ.

Let K be the highest non-blank round in γ and let $M \coloneqq (K+1)|Q||\mathbb{D}|^{\mathsf{dim}(\nu+1)}$. We prove that, from every $\gamma' \in \mathsf{Post}^*(\gamma)$, q_f can be covered from γ' in less than M steps. Let $\gamma' \in \mathsf{Post}^*(\gamma)$. Consider a process whose round k' is maximal among the rounds of all processes in γ', and let $\gamma'_1 \in \mathsf{Iso}(\gamma')$ be the configuration obtained from γ by deleting all processes except this one. Let K' be the highest non-blank round in γ'; this is also the highest non-blank round in γ'_1. Because $\gamma' \in \mathsf{Post}^*(\gamma)$, we have $K \leqslant K'$. Moreover, we have $K' - k' \leqslant K$. Indeed, if $K' = K$ then this is trivially true, and if $K' > K$ then a register of round K' has been written in the execution from γ to γ', hence the process in γ' has round greater than K' and $k' \geqslant K'$. Because $(\mathcal{P}, q_f)$ is Asof and $\gamma'_1 \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$, q_f can be covered from γ'_1. Thanks to Lemma 5.15, q_f can be covered from γ'_1 in less than M steps. By Lemma 5.16, the same is true for γ'.

Let $p \coloneqq \frac{1}{|\gamma|\cdot|\Delta|}$. From any $\gamma' \in \mathsf{Post}^*(\gamma)$, there is probability at least p^M to cover q_f within the next M steps. This proves that the probability to cover q_f from γ within $m \cdot B$ steps is at least $1 - (1-p)^m \underset{m\to\infty}{\longrightarrow} 1$, hence that $\mathbb{P}_\gamma(\Diamond\uparrow q_f) = 1$.

We have proved that $\mathbb{P}_\gamma(\Diamond\uparrow q_f) = 1$; we now must prove that $\mathbb{P}_\gamma(\Diamond\mathsf{Cons}(q_f)) = 1$. Let C_f be the random variable giving the first configuration, in executions from γ, that has a process in q_f. With probability 1, C_f is in $\mathsf{Post}^*(\gamma)$. We have that

$$\mathbb{P}_\gamma(\Diamond\uparrow q_f) = 1 = \sum_{\gamma_f \in \mathsf{Post}^*(\gamma)} \mathbb{P}(C_f = \gamma_f)\mathbb{P}_{\gamma_f}(\Diamond\mathsf{Cons}(q_f)).$$

It therefore suffices to prove that, for all $\gamma_f \in \mathsf{Post}^*(\gamma)$ with exactly one process in q_f, we have $\mathbb{P}_{\gamma_f}(\Diamond\mathsf{Cons}(q_f)) = 1$. Fix such a configuration γ_f. Recall that, by hypothesis, there is $\gamma_u \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma \preceq \gamma_u$. By mimicking the execution from γ to γ_f but from γ_u, we obtain $\gamma_{u,f} \in \mathsf{Post}^*(\Gamma_0)$ such that $\gamma_f \preceq \gamma_{u,f}$. Let γ'_f is the configuration of size $|\gamma| - 1$ obtained from γ_f by removing all processes in q_f (if all processes in γ_f are in q_f then we already have $\gamma_f \in \mathsf{Cons}(q_f)$). We have $\gamma'_f \preceq \gamma_{u,f}$ hence the induction hypothesis applies and $\mathbb{P}_{\gamma'_f}(\Diamond\mathsf{Cons}(q_f)) = 1$. Executions from γ_f are isomorphic to executions from γ'_f: the only difference are the steps performed by the process in q_f, but such steps have no influence on the rest of the system because q_f is a stalemate state. The means that the probability distribution of executions from γ_f is equal, up to dismissing the process on q_f, to the probability distribution of executions from γ'_f; therefore, $\mathbb{P}_{\gamma_f}(\Diamond\mathsf{Cons}(q_f)) = 1$. This concludes the induction. Applying the induction hypothesis to each $\gamma_0(n)$, $n \geqslant 1$, concludes the proof of Theorem 5.14.

This proves that if a protocol is Asof then, for every number of participating processes, all processes almost surely terminate. Therefore, we can use the Asof property to argue of the almost-sure termination of round-based consensus algorithms. Of course, this is only relevant if one can check that a given protocol is Asof.

5.8.3 Deciding Almost-Sure Obstruction-Freedom

We now provide the complexity of this verification problem.

Theorem 5.18. *The following problem is PSPACE-complete:*

Almost-sure Obstruction-Freedom
Input*: A round-based protocol $\mathcal{P}$ with a special stalemate state q_f*
Question*: Is $(\mathcal{P}, q_f)$ Asof?*

To prove Theorem 5.18, we first observe that the Asof property can be phrased in similar fashion to decisiveness, under a weaker form where only configurations in $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ are considered. The intuition is that, given a configuration $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$, if q_f can be covered from all configurations in $\mathsf{Post}^*(\gamma)$, then there is a lower bound (depending on γ) on the probability to cover q_f within a bounded number of steps. By iterating the argument, this gives probability 1 to eventually cover q_f from γ. This is not true in general for $\gamma \in \mathsf{Post}^*(\Gamma_0)$: in the examples from Section 5.5 and Section 5.6, there is no such lower bound.

Lemma 5.19. *$(\mathcal{P}, q_f)$ is Asof if and only if $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$.*

Proof. First, if $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)) \not\subseteq \mathsf{Pre}^*(\uparrow q_f)$ then there is $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ such that $\mathbb{P}_\gamma(\Diamond \uparrow q_f) = 0$ which contradicts $(\mathcal{P}, q_f)$ being Asof.

Conversely, assume that $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$ Let $\gamma_u \in \mathsf{Post}^*(\Gamma_0)$ and let $\gamma \in \mathsf{Iso}(\gamma_u)$; by hypothesis, q_f can be covered from γ. We prove that $\mathbb{P}_\gamma(\Diamond \uparrow q_f) = 1$. Let K be the highest non-blank round in γ and let $M \coloneqq (K+1)|Q||\mathbb{D}|^{\mathsf{dim}(\nu+1)}$. Let $\gamma' \in \mathsf{Post}^*(\gamma) \subseteq \mathsf{Post}^*(\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)))$. By Lemma 5.16, $\gamma' \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ and, by hypothesis, q_f can be covered from γ'. Using the same reasoning as in the proof of Theorem 5.14, in γ', the process is not more than K rounds below the highest non-blank round. Thanks to Lemma 5.15, q_f can be covered in less than M steps from γ'. Therefore, for every $\gamma' \in \mathsf{Post}^*(\gamma)$, this is probability at least $\frac{1}{|\Delta|^M}$ of covering q_f in the next M steps, which proves that $\mathbb{P}_\gamma(\Diamond \uparrow q_f) = 1$. This being true for every $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$, $(\mathcal{P}, q_f)$ is Asof. □

We therefore aim at deciding whether $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)) \subseteq \mathsf{Pre}^*(\uparrow q_f)$. Among configurations of size 1, we highlight a convenient subset: the one where the process is at least as high as the highest non-blank round. In such configurations, the sequences of moves that the processes will be able to perform (up to translation) only depends on its local view, because rounds that are lower than the local view are irrelevant and rounds that are above the local view are blank. We denote by $\Gamma_{\mathsf{h}} \subseteq \mathsf{Iso}(\Gamma)$ the set of configurations of size 1 where the round of the process is greater than or equal to the highest non-blank round. Note that this set is stable by reachability, *i.e.*, $\mathsf{Post}^*(\Gamma_{\mathsf{h}}) \subseteq \Gamma_{\mathsf{h}}$. Indeed, processes may never decrease their rounds, hence if a round k is blank in γ but not blank in some configuration $\gamma' \in \mathsf{Post}^*(\gamma)$, then in γ' the process is at a round greater than or equal to k.

Because of the argument above, configurations in Γ_{h} can be abstracted by their local view; therefore, our aim is to build a reachability graph for local views. Given a protocol $\mathcal{P}$, let $\mathcal{G}_{\mathcal{P}}$ be the directed graph whose set of vertices $V \coloneqq Q \times \mathbb{D}^{[\![0,v]\!]\times\mathsf{dim}}$ is the set of local views and such that, for all $\ell_1, \ell_2 \in V$, there is an edge from ℓ_1 to ℓ_2 whenever there exists $\gamma_1, \gamma_2 \in \Gamma_{\mathsf{h}}$ and $\theta \in \mathcal{M}$ such that $\gamma_1 \xrightarrow{\theta} \gamma_2$, ℓ_1 is the local view of γ_1 and ℓ_2 is the local view of γ_2.

Enforcing that $\gamma_1, \gamma_2 \in \Gamma_{\mathsf{h}}$ may seem useless: if $\ell_1 \to \ell_2$ holds, why would it matter if higher rounds are blank or not? In fact, this condition is needed when the move allowing the step from γ_1 to γ_2 is a round increment, in which case the fact that $\gamma_1 \in \Gamma_{\mathsf{h}}$ guarantees that the round of the process in γ_2 is blank.

Let Q_{incr} be the set of states q such that there is $q' \in Q$ with $(q, \mathsf{incr}, q') \in \Delta$, *i.e.*, the set of states from which there is a round increment transition.

Lemma 5.20. *Let $\gamma_1 \in \Gamma$ of size 1, let ℓ_1 be the local view of γ_1, and let ℓ_2 be another local view such that there is a path π in $\mathcal{G}_{\mathcal{P}}$ from ℓ_1 to ℓ_2. If one of the following two conditions is satisfied, then there is an execution from γ_1 to some configuration $\gamma_2 \in \Gamma$ such that γ_2 has local view ℓ_2:*

(i) $\gamma_1 \in \Gamma_{\mathsf{h}}$, *or*

(ii) π *visits no local view with state in* Q_{incr}, *except possibly for* ℓ_2.

Proof. We first prove the result in the case where the path from ℓ_1 to ℓ_2 is composed of one edge only. By definition of $\mathcal{G}_{\mathcal{P}}$, there are $\gamma_1', \gamma_2' \in \Gamma_{\mathsf{h}}$ and $\theta \in \mathcal{M}$ such that $\gamma_1' \xrightarrow{\theta} \gamma_2'$ and the local view of γ_1' (resp. γ_2') is ℓ_1 (resp. ℓ_2). Because the local views of γ_1 and γ_1' are the same, θ can be applied from γ_1: let γ_2 be the configuration such that $\gamma_1 \xrightarrow{\theta} \gamma_2$. If θ is not a round increment, then we directly have that γ_2 and γ_2' have the same local view. This is in particular the case if (ii) is satisfied. If (i) is satisfied, then $\gamma_1, \gamma_1' \in \Gamma_{\mathsf{h}}$, so that γ_2 and γ_2' have the same local view even if θ is a round increment, because the round immediately above the round of the process is blank

in both γ_1 and γ'_1. We generalize this to paths π of $\mathcal{G}_\mathcal{P}$ by direct induction on the length of π. In case (i), the induction requires the observation that $\mathsf{Post}^*(\Gamma_\mathsf{h}) \subseteq \Gamma_\mathsf{h}$, so that the configurations obtained are also in Γ_h. □

We now provide a characterization of the AsoF property in terms of local views. A local view ℓ is *reachable* when there is $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ with local view ℓ.

Lemma 5.21. *$(\mathcal{P}, q_f)$ is not AsoF if and only if there is a reachable local view ℓ whose strongly connected component in $\mathcal{G}_\mathcal{P}$ is bottom and does not have vertices with state q_f.*

Proof. Thanks to Lemma 5.19, $(\mathcal{P}, q_f)$ is not AsoF if and only if $\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0)) \not\subseteq \mathsf{Pre}^*(\uparrow q_f)$.

First, suppose that there is such a local view ℓ. Because ℓ is reachable, there is $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ with local view ℓ. Let k be the round of the process in γ. Let $\rho : \gamma_0(n) \xrightarrow{*} \gamma_u$ such that $\gamma \preceq \gamma_u$. Consider ρ' the execution identical to ρ except that all steps taking place at rounds $> k$ are removed. Because steps from rounds below k may not depend on the content of registers above k, ρ' is a valid execution, let γ'_u be its last configuration. By letting (q, k) the location of the process in γ, there is a process in (q, k) in γ'_u; let $\gamma' \in \mathsf{Iso}(\gamma'_u)$ obtained by deleting all processes except this one. We have that $\gamma' \in \Gamma_\mathsf{h}$ and that γ' has the same local view as γ. We have $\mathsf{Post}^*(\gamma') \subseteq \Gamma_\mathsf{h}$, so that the local views of configurations reachable from γ' are exactly those that can be reached from ℓ in $\mathcal{G}_\mathcal{P}$. Since the strongly connected component of ℓ in $\mathcal{G}_\mathcal{P}$ is bottom and has no vertex with state q_f, $\gamma' \notin \mathsf{Pre}^*(\uparrow q_f)$ and $(\mathcal{P}, q_f)$ is not AsoF.

Suppose now that $(\mathcal{P}, q_f)$ is not AsoF. By Lemma 5.19, there is $\gamma \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ such that $\gamma \notin \mathsf{Pre}^*(\uparrow q_f)$. Note, however, that we do not necessarily have $\gamma \in \Gamma_\mathsf{h}$, so that paths from the local view of γ in $\mathcal{G}_\mathcal{P}$ do not automatically yield executions from γ. We make a case disjunction based on the following assertion:

$$\begin{gathered}\text{for each } \gamma' \in \mathsf{Post}^*(\gamma),\ \text{there is a path in } \mathcal{G}_\mathcal{P} \\ \text{from the local view of } \gamma' \text{ to a local view with state in } Q_{\mathsf{incr}}.\end{gathered} \tag{5.3}$$

Assume first that (5.3) does not hold. Let $\gamma' \in \mathsf{Post}^*(\gamma)$ that contradicts the assertion, let ℓ' be the local view of γ'. We know that, in $\mathcal{G}_\mathcal{P}$, there is no path from ℓ' to a local view with state in Q_{incr}. There exists a local view ℓ_{bot} that is in a bottom strongly connected component of $\mathcal{G}_\mathcal{P}$ and such that there is a path from ℓ' to ℓ_{bot}; moreover, no local view along this path has state in Q_{incr}, hence by Lemma 5.20, there is $\gamma_{\mathsf{bot}} \in \mathsf{Post}^*(\gamma)$ with local view ℓ_{bot}. Again by Lemma 5.20, a path in $\mathcal{G}_\mathcal{P}$ from ℓ_{bot} to a local view with state q_f would contradict that $\gamma \notin \mathsf{Pre}^*(\uparrow q_f)$. Therefore, the strongly connected component of ℓ_{bot} is bottom and contains no local view with

state q_f; moreover, ℓ_{bot} is reachable because $\gamma_{\mathsf{bot}} \in \mathsf{Post}^*(\mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))) \subseteq \mathsf{Iso}(\mathsf{Post}^*(\gamma))$ by Lemma 5.17, concluding this case.

Assume now that (5.3) holds. By Lemma 5.20, this implies that, for all $\gamma' \in \mathsf{Post}^*(\gamma)$, one can reach a configuration where the process is in a state in Q_{incr}, and therefore there is an execution from γ' that increases the round by 1. By iterating the argument, $\mathsf{Post}^*(\gamma)$ contains configurations of arbitrarily large rounds. For $\gamma' \in \mathsf{Post}^*(\gamma)$, if the round of the process in γ' is greater than the highest non-blank round in γ then $\gamma' \in \Gamma_{\mathsf{h}}$. This proves the existence of $\gamma' \in \mathsf{Post}^*(\gamma) \cap \Gamma_{\mathsf{h}}$. Let ℓ' be its local view. Because $\mathsf{Post}^*(\Gamma_{\mathsf{h}}) \subseteq \Gamma_{\mathsf{h}}$, any path in $\mathcal{G}_{\mathcal{P}}$ from ℓ' can be translated to an execution from γ' using Lemma 5.20. There exists a local view ℓ_{bot} that is in a bottom strongly connected component of $\mathcal{G}_{\mathcal{P}}$ and such that there is a path from ℓ' to ℓ_{bot}; using Lemma 5.20 and Lemma 5.17, we obtain $\gamma_{\mathsf{bot}} \in \mathsf{Iso}(\mathsf{Post}^*(\Gamma_0))$ of local view ℓ_{bot}, and ℓ_{bot} is reachable. Moreover, a path from ℓ_{bot} to a local view with state q_f would contradict that $\gamma \notin \mathsf{Pre}^*(\uparrow q_f)$. This concludes the proof. □

Given a local view ℓ, one can easily explore its SCC in polynomial space. Indeed, although $\mathcal{G}_{\mathcal{P}}$ has size exponential in the size of $\mathcal{P}$, its vertices are storable in polynomial space (thanks to the fact that v is assumed to be encoded in unary in the input). Moreover, a given vertex has at most $|\Delta|$ many successors, and this set can be computed efficiently. It remains to argue that we can check, given a local view ℓ, whether it is reachable, *i.e.*, whether there is $\gamma \in \mathsf{Post}^*(\Gamma_0)$ with local view ℓ.

Lemma 5.22. *Whether a local view ℓ is reachable is decidable in PSPACE.*

Proof. The local view of a process depends only on its state and on the content of the registers. Therefore, for all $\gamma' \in \Gamma$, there is a process with local view ℓ in γ' if and only if there is $\gamma \in \mathsf{Iso}(\gamma')$ with local view ℓ. The set of configurations that have a process with local view ℓ can be expressed as a presence constraint (in the sense of Chapter 4) of polynomial size. Membership in PSPACE is directly given by Theorem 4.27. □

We are now ready to prove that one can decide whether a protocol is Asof in polynomial space. By Lemma 5.21 and because PSPACE is stable by complement, it suffices to be able to detect whether there is a reachable local view in $\mathcal{G}_{\mathcal{P}}$ that is in a bottom strongly connected component where q_f does not appear. We cannot directly build $\mathcal{G}_{\mathcal{P}}$ because its size is exponential in the size of the protocol; each vertex, however, can be stored in polynomial space, and the set of successors of a given vertex can be computed efficiently. The polynomial-space procedure guesses the local view ℓ and checks that it is reachable using Lemma 5.22. To check that its

strongly connected component is bottom and that it contains no occurrence of q_f, it suffices to perform a (non-deterministic) exploration of the strongly connected component vertex by vertex. This procedure works in polynomial space, and is correct thanks to Lemma 5.21.

This proves that deciding whether a protocol is ASOF is a PSPACE problem. It remains to argue that the problem is PSPACE-hard.

Proposition 5.23. *The almost-sure obstruction-freedom problem is PSPACE-hard.*

Proof. We proceed by reduction from the halting problem of a deterministic Turing machine with a tape of size n. The idea, in line with the hardness proofs of Theorem 4.12 and Proposition 4.31, is to design the protocol so that no two executions write different symbols to a register. If the Turing machine halts, the protocol must be ASOF: any process, if left in isolation, must be able to carry out the simulation of the Turing machine. To do that, we use $n + 2$ registers per round: one for each cell of the tape, one for the position of the head and one for the state. Registers of a round can thus store an entire configuration of the Turing machine. At round 0, every process goes through an initial gadget where it writes the initial configuration of the Turing machine. A process that gets to round $k+1$ reads the registers from round k and writes the next configuration of the machine (this requires $v = 1$). A process at round k writes to every register of round k before incrementing its round. Because the Turing machine is deterministic, all write actions performed to a given register write the same symbol. Processes go to q_f upon writing the halting state of the Turing machine. In this protocol, reachable configurations correspond to (partial) simulations of runs of the Turing machines. From a reachable configuration, a process left in isolation will eventually simulate the run of the Turing machine and go to q_f; this process may write to registers that were already written, which is not an issue. This protocol is ASOF if and only if the Turing machine halts. The protocol is polynomial in the size of the machine, which concludes the proof. □

5.9 Perspectives

In this section, we studied round-based ASMS under stochastic schedulers. One of our initial ambitions was to transpose to the round-based setting the study of almost-sure coverability from [BMRSS16]. In particular, [BMRSS16] relies on an equivalence, called here decisiveness, that turns almost-sure coverability into a non-probabilistic property. In the round-based model, however, the set of rounds is infinite which allows for random-walk behaviors. Such behaviors entail that decisiveness does not hold in round-based ASMS. Random walks are mathematically

complex phenomena, therefore we tried to find reasonable restrictions that exclude such behaviors. We considered enforcing that processes progress in terms of rounds at the same average pace. Also, inspired by the stochastic scheduler model from [Asp02], we considered another type of scheduler based on continuous time, where each process draws independently the time until its next action. Both attempts fail to forbid random-walk behaviors. This led us to consider a stronger restriction, called almost-sure obstruction-freedom. It states that, from any reachable configuration, if a process is left playing in isolation, then it eventually decided with probability 1. This property is typically satisfied by randomized round-based consensus algorithms, and it is in particular satisfied by Aspnes' noisy consensus algorithm [Asp02]. Almost-sure obstruction-freedom implies almost-sure termination and that deciding whether a protocol satisfies it is a PSPACE-complete problem.

While this last fact constitutes a much-needed positive result after all the negative ones from the rest of this chapter, it remains a bit unsatisfactory. Most of the questions associated with round-based ASMS under stochastic schedulers remain unsolved. In particular, we have not performed a cutoff analysis in the style of [BMRSS16]. An important result would therefore be the following: "for every round-based ASMS protocol $\mathcal{P}$ with a stalemate state q_f, there is $N \in \mathbb{N}$ such that either $\mathbb{P}_n(\uparrow q_f) = 1$ for all $n \geqslant N$ (positive cutoff) or $\mathbb{P}_n(\uparrow q_f) < 1$ for all $n \geqslant N$ (negative cutoff)". A similar question can be asked for almost-sure termination, and potentially for more general properties such as almost-sure reachability of sets defined by presence constraints. If such a cutoff exists, then the next question would be what the order of magnitude of this cutoff. Can it be bounded by a primitive-recursive function? By a doubly-exponential one? A related question would be the decidability status and complexity of the associated decision problem: given $(\mathcal{P}, q_f)$, does it have a positive cutoff or a negative one? All of these questions are potentially very challenging; in our experience, even the analysis of simple protocols may raise questions that random walk specialists cannot easily answer [2].

2. We thank Mathias Rousset, Frédéric Cérou and Bruno Sericola for the interesting discussions on this challenging task.

CONCLUSION

Summary

This thesis is dedicated to the study of parameterized verification of distributed systems and in particular of shared-memory systems. We defined and studied several models, notably inspired by shared-memory consensus algorithms such as Aspnes' noisy consensus algorithm [Asp02]. While we used distributed algorithms as a guideline, our contributions are theoretical and mostly related to the decidability and complexity of the considered decision problems. We recall here the main results from this thesis.

Chapter 2 We studied a slightly extended version of the classic model of asynchronous shared-memory systems [EGM13; EGM16]. On this model, we define presence reachability problem (PRP), a generic reachability problem of which COVER and TARGET are two particular cases. We proved that the generic problem is NP-complete, and that NP-hardness already holds for COVER. Assuming that the registers are not initialized makes COVER solvable in polynomial time, but this does not extend to more sophisticated questions such as TARGET. However, if the system has a single shared register then TARGET is also PTIME, and this can be extended to PRP as long as the presence constraint is in disjunctive normal form. When considering parameterized complexity with respect to the number of registers, COVER is FPT but TARGET is W[2]-hard.

Chapter 3 We defined and studied copycat systems, a general model meant to capture parameterized systems composed of many identical processes that satisfy the copycat property, such as asynchronous shared-memory systems. We defined a generic mathematical notion, called *transfer flows*, to describe the possibilities offered by transitions of the system. We defined a compositional product on transfer flows, which expresses the possibilities offered by several transitions applied in a row. Thanks to this concept and to a bound from the literature on the length of descending chains of $\mathbb{N}^d$ [LS21; SS24], we obtained a general-purpose doubly-exponential bound. This proves that any execution can be rearranged into one of at most doubly-exponential length in which only at most doubly-exponentially many processes play an important role. We presented a few applications of this result, notably to decide emptiness of generalized reachability expressions and for LTL verification. It also allowed us to provide, with little overhead,

most of the results of [BMRSS16; Sta17] on almost-sure reachability in ASMS.

Chapter 4 We generalized the ASMS model from Chapter 2 to capture round-based algorithms such as Aspnes' noisy consensus algorithm. We extended the presence reachability problem PRP from Chapter 2 to round-based PRP by allowing non-nested quantification over round values. We established that this problem is PSPACE-complete; to obtain membership in PSPACE, one cannot guess the execution naively configuration by configuration, and we instead relied on the notion of footprint, which is the projection of an execution onto a few rounds. This PSPACE result relies on the hypothesis that the integer constants of the input are given in unary; if they are given in binary then the problem becomes EXPSPACE-complete.

Chapter 5 We defined a stochastic scheduler where the next process to act is selected uniformly among all processes, and we considered the round-based ASMS model under such schedulers. Unfortunately, the occurrence of random-walk behaviors make the analysis very complex, and it does not seem that one can prevent such behaviors easily. Nonetheless, we defined almost-sure obstruction-freedom, a property that round-based consensus algorithms typically satisfy. We proved that this property implies almost-sure termination and that the problem of deciding whether a protocol satisfies this property is PSPACE-complete.

Future Works

We have highlighted, at the end of each chapter of this thesis, several open questions and possible extensions. We here mention two future works that are, in our eyes, the most interesting and important tasks:

Gap on the structural bound in copycat systems The structural bound (Theorem 3.19) depends doubly-exponentially in the number of states of the system. While examples are known where one cannot cover a state with less than exponentially many steps (see, *e.g.*, PSPACE-hardness in [BMRSS16]), it is not known whether this doubly-exponential bound is tight or not. In other words: can the bound from Theorem 3.19 be improved to a bound whose dependency in the number of states is only exponential and not doubly-exponential? This is closely related to the complexity gap on almost-sure coverability for ASMS left open in [BMRSS16] that was wrongly claimed to be solved in [BGW22; BGW23]. If one can prove that the structural bound on copycat systems can be improved to exponential in the number of states, then this would have

direct implications to ASMS and to RBN and would constitute a valuable contribution. In our view, this is the most important question left open by this thesis.

Round-based ASMS under stochastic schedulers. Many questions related to round-based ASMS under stochastic schedulers remain open. The most interesting ones are the following. First, given $(\mathcal{P}, q_f)$, does there exist a cutoff for almost-sure coverability, *i.e.*, a number N of processes such that the answer to almost-sure coverability remains the same for all $n \geqslant N$? How large can this cutoff be? What are the decidability status and complexity of the problem of deciding whether $(\mathcal{P}, q_f)$ has a positive cutoff? All these questions seem very interesting but also, given the presence of random walks in the model, very challenging.

BIBLIOGRAPHY

[AADFP04] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta, *Computation in networks of passively mobile finite-state sensors*, In: *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004*, 2004, pp. 290–299, URL: https://doi.org/10.1145/1011767.1011810.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert, *The computational power of population protocols*, In: *Distributed Comput.* 20.*4* (2007), pp. 279–304, URL: https://doi.org/10.1007/s00446-007-0040-2.

[AAR20] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Rojin Rezvan, *Parameterized verification under TSO is PSPACE-complete*, In: *POPL'2020.* 4 (2020), 26:1–26:29, URL: https://doi.org/10.1145/3371094.

[ABGS18] Karolos Antoniadis, Peva Blanchard, Rachid Guerraoui, and Julien Stainer, *The entropy of a distributed computation random number generation from memory interleaving*, In: *Distributed Comput.* 31.*5* (2018), pp. 389–417, URL: https://doi.org/10.1007/s00446-017-0311-5.

[Abr88] Karl Abrahamson, *On achieving consensus using a shared memory*, In: *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, USA, 1988, pp. 291–302, URL: https://doi.org/10.1145/62546.62594.

[ACJT96] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay, *General Decidability Theorems for Infinite-State Systems*, In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, 1996, pp. 313–321, URL: https://doi.org/10.1109/LICS.1996.561359.

[AH90] James Aspnes and Maurice Herlihy, *Fast Randomized Consensus Using Shared Memory*, In: *J. Algorithms* 11.*3* (1990), pp. 441–461, URL: https://doi.org/10.1016/0196-6774(90)90021-6.

[AHM07] Parosh Aziz Abdulla, Noomene Ben Henda, and Richard Mayr, *Decisive Markov Chains*, 2007, URL: http://arxiv.org/abs/0706.2585.

[AJK15] Simon Außerlechner, Swen Jacobs, and Ayrat Khalimov, *Tight Cutoffs for Guarded Protocols with Fairness*, In: *CoRR* abs/1505.03273 (2015), URL: http://arxiv.org/abs/1505.03273.

[AJKR14] Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin, *Parameterized Model Checking of Token-Passing Systems*, In: *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014*, vol. 8318, 2014, pp. 262–281, URL: https://doi.org/10.1007/978-3-642-54013-4%5C_15.

[AK86] Krzysztof R. Apt and Dexter Kozen, *Limits for Automatic Verification of Finite-State Concurrent Systems*, In: *Inf. Process. Lett.* 22.*6* (1986), pp. 307–309, URL: https://doi.org/10.1016/0020-0190(86)90071-2.

[AKRSV14] Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith, *Parameterized Model Checking of Rendezvous Systems*, In: *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014*, vol. 8704, 2014, pp. 109–124, URL: https://doi.org/10.1007/978-3-662-44584-6%5C_9.

[AR09] James Aspnes and Eric Ruppert, *An Introduction to Population Protocols*, In: *Middleware for Network Eccentric and Mobile Applications*, 2009, pp. 97–120, URL: https://doi.org/10.1007/978-3-540-89707-1%5C_5.

[Asp00] James Aspnes, *Fast deterministic consensus in a noisy environment*, In: Portland, Oregon, USA, 2000, pp. 299–308, URL: https://doi.org/10.1145/343477.343631.

[Asp02] James Aspnes, *Fast deterministic consensus in a noisy environment*, In: *Journal of Algorithms* 45.*1* (2002), pp. 16–39, URL: https://www.sciencedirect.com/science/article/pii/S0196677402002201.

[Asp03] James Aspnes, *Randomized protocols for asynchronous consensus*, en, In: *Distributed Comput.* 16.*2-3* (2003), pp. 165–175, URL: http://link.springer.com/10.1007/s00446-002-0081-5 (visited on 04/14/2022).

[Asp24] James Aspnes, *Notes on Theory of Distributed Systems*, Lecture notes, 2024, URL: http://www.cs.yale.edu/homes/aspnes/classes/465/notes.pdf.

[ASV15] Dan Alistarh, Thomas Sauerwald, and Milan Vojnovic, *Lock-Free Algorithms under Stochastic Schedulers*, In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, 2015, pp. 251–260, URL: https://doi.org/10.1145/2767386.2767430.

[Bal18] A. R. Balasubramanian, *Parameterized Verification of Coverability in Well-Structured Broadcast Networks*, In: *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018*, vol. 277, 2018, pp. 133–146, URL: https://doi.org/10.4204/EPTCS.277.10.

[BBBG15] Joffroy Beauquier, Peva Blanchard, Janna Burman, and Rachid Guerraoui, *The Benefits of Entropy in Population Protocols*, In: *19th International Conference on Principles of Distributed Systems, OPODIS 2015*, vol. 46, 2015, 21:1–21:15, URL: https://doi.org/10.4230/LIPIcs.OPODIS.2015.21.

[BBM18] A. R. Balasubramanian, Nathalie Bertrand, and Nicolas Markey, *Parameterized Verification of Synchronization in Constrained Reconfigurable Broadcast Networks*, In: *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018*, vol. 10806, 2018, pp. 38–54, URL: https://doi.org/10.1007/978-3-319-89963-3%5C_3.

[BBM21] Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar, *Reconfiguration and Message Losses in Parameterized Broadcast Networks*, In: *Log. Methods Comput. Sci.* 17.*1* (2021), URL: https://lmcs.episciences.org/7280.

[BDGG17] Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, and Hugo Gimbert, *Controlling a Population*, In: *28th International Conference on Concurrency Theory, CONCUR 2017*, vol. 85, 2017, 12:1–12:16, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2017.12.

[BEHKM20] Michael Blondin, Javier Esparza, Martin Helfrich, Antonín Kucera, and Philipp J. Meyer, *Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling*, In: *Computer Aided Verification - 32nd International Conference, CAV 2020*, vol. 12225, 2020, pp. 372–397, URL: https://doi.org/10.1007/978-3-030-53291-8%5C_20.

[BEJK18] Michael Blondin, Javier Esparza, Stefan Jaax, and Antonín Kucera, *Black Ninjas in the Dark: Formal Analysis of Population Protocols*, In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, 2018, pp. 1–10, URL: https://doi.org/10.1145/3209108.3209110.

[BEK18] Michael Blondin, Javier Esparza, and Antonín Kucera, *Automatic Analysis of Expected Termination Time for Population Protocols*, In: *29th International Conference on Concurrency Theory, CONCUR 2018*, vol. 118, 2018, 33:1–33:16, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2018.33.

[Ben83] Michael Ben-Or, *Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols*, In: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983, pp. 27–30, URL: https://doi.org/10.1145/800221.806707.

[Ber20] Nathalie Bertrand, *Model checking randomized distributed algorithms*, In: *ACM SIGLOG News* 7.*1* (2020), pp. 35–45, URL: https://doi.org/10.1145/3385634.3385638.

[BFS14] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier, *Playing with Probabilities in Reconfigurable Broadcast Networks*, In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, vol. 8412, 2014, pp. 134–148, URL: https://doi.org/10.1007/978-3-642-54830-7%5C_9.

[BFS15] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier, *Distributed Local Strategies in Broadcast Networks*, In: *26th International Conference on Concurrency Theory, CONCUR 2015*, vol. 42, 2015, pp. 44–57, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2015.44.

[BGKMWW24] Steffen van Bergerem, Roland Guttenberg, Sandra Kiefer, Corto Mascle, Nicolas Waldburger, and Chana Weil-Kennedy, *Verification of Population Protocols with Unordered Data*, In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, vol. 297, 2024, 156:1–156:20, URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.156.

[BGS14] Benedikt Bollig, Paul Gastin, and Jana Schubert, *Parameterized Verification of Communicating Automata under Context Bounds*, In: *Reachability Problems - 8th International Workshop, RP 2014*, vol. 8762, 2014, pp. 45–57, URL: https://doi.org/10.1007/978-3-319-11439-2%5C_4.

[BGW22] A. R. Balasubramanian, Lucie Guillou, and Chana Weil-Kennedy, *Parameterized Analysis of Reconfigurable Broadcast Networks*, In: *Foundations of Software Science and Computation Structures - 25th International Conference, FoSSaCS 2022*, vol. 13242, 2022, pp. 61–80, URL: https://doi.org/10.1007/978-3-030-99253-8%5C_4.

[BGW23] A. R. Balasubramanian, Lucie Guillou, and Chana Weil-Kennedy, *Erratum to Parameterized Analysis of Reconfigurable Broadcast Networks*, 2023, URL: https://chana-wk.github.io/erratum-fossacs22.pdf.

[BH14] Joe Blitzstein and Jessica Hwang, *Introduction to Probability*, 2014, URL: https://ia803404.us.archive.org/6/items/introduction-to-probability-joseph-k.-blitzstein-jessica-hwang/Introduction%20to%20Probability-Joseph%20K.%20Blitzstein%2C%20Jessica%20Hwang.pdf.

[BJKKRVW15] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder, *Decidability of Parameterized Verification*, 2015, URL: https://doi.org/10.2200/S00658ED1V01Y201508DCT013.

[BK08] Christel Baier and Joost-Pieter Katoen, *Principles of Model Checking*, 2008, URL: https://is.ifmo.ru/books/_principles_of_model_checking.pdf.

[BKL14] Rémi Bonnet, Stefan Kiefer, and Anthony Widjaja Lin, *Analysis of Probabilistic Basic Parallel Processes*, In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, vol. 8412, 2014, pp. 43–57, URL: https://doi.org/10.1007/978-3-642-54830-7%5C_3.

[BKLW21] Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder, *Verification of randomized consensus algorithms under round-rigid adversaries*, In: *Int. J. Softw. Tools Technol. Transf.* 23.*5* (2021), pp. 797–821, URL: https://doi.org/10.1007/s10009-020-00603-x.

[BMRSS16] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan, *Reachability in Networks of Shared-memory Protocols under Stochastic Schedulers*, In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, vol. 55, 2016, 106:1–106:14, URL: https://doi.org/10.4230/LIPIcs.ICALP.2016.106.

[BMSW22] Nathalie Bertrand, Nicolas Markey, Ocan Sankur, and Nicolas Waldburger, *Parameterized Safety Verification of Round-Based Shared-Memory Systems*, In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, vol. 229, 2022, 113:1–113:20, URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.113.

[Bor09] Émile Borel, *Les probabilités dénombrables et leurs applications arithmétiques*, In: *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 27 (1909), pp. 247–271, URL: https://api.semanticscholar.org/CorpusID:184479669.

[Bra87] Gabriel Bracha, *Asynchronous Byzantine Agreement Protocols*, In: *Inf. Comput.* 75.*2* (1987), pp. 130–143, URL: https://doi.org/10.1016/0890-5401(87)90054-X.

[BRS21] Benedikt Bollig, Fedor Ryabinin, and Arnaud Sangnier, *Reachability in Distributed Memory Automata*, In: *29th EACSL Annual Conference on Computer Science Logic, CSL 2021*, vol. 183, 2021, 13:1–13:16, URL: https://doi.org/10.4230/LIPIcs.CSL.2021.13.

[BT85] Gabriel Bracha and Sam Toueg, *Asynchronous consensus and broadcast protocols*, In: *J. ACM* 32.*4* (1985), pp. 824–840, URL: https://doi.org/10.1145/4221.214134.

[BTW21] Nathalie Bertrand, Bastien Thomas, and Josef Widder, *Guard Automata for the Verification of Safety and Liveness of Distributed Algorithms*, In: *32nd International Conference on Concurrency Theory, CONCUR 2021*, vol. 203, 2021, 15:1–15:17, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2021.15.

[BW21] A. R. Balasubramanian and Chana Weil-Kennedy, *Reconfigurable Broadcast Networks and Asynchronous Shared-Memory Systems are Equivalent*, In: *Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2021*, vol. 346, 2021, pp. 18–34, URL: https://doi.org/10.4204/EPTCS.346.2.

[CDFS11] Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu, *Guidelines for the Verification of Population Protocols*, In: *2011 International Conference on Distributed Computing Systems, ICDCS 2011*, 2011, pp. 215–224, URL: https://doi.org/10.1109/ICDCS.2011.36.

[CDM11] Bernadette Charron-Bost, Henri Debrat, and Stephan Merz, *Formal Verification of Consensus Algorithms Tolerating Malicious Faults*, In: *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011*, vol. 6976, 2011, pp. 120–134, URL: https://doi.org/10.1007/978-3-642-24550-3%5C_11.

[CFO20] Thomas Colcombet, Nathanaël Fijalkow, and Pierre Ohlmann, *Controlling a Random Population*, In: *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020*, vol. 12077, 2020, pp. 119–135, URL: https://doi.org/10.1007/978-3-030-45231-5%5C_7.

[CLSW24] Dmitry Chistikov, Jérôme Leroux, Henry Sinclair-Banks, and Nicolas Waldburger, *Invariants for One-Counter Automata with Disequality Tests*, In: (2024), URL: https://drops.dagstuhl.de/storage/00lipics/lipics-vol311-concur2024/LIPIcs.CONCUR.2024.17/LIPIcs.CONCUR.2024.17.pdf.

[CMS19] Peter Chini, Roland Meyer, and Prakash Saivasan, *Complexity of Liveness in Parameterized Systems*, In: *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019*, vol. 150, 2019, 37:1–37:15, URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2019.37.

[CO21] Wojciech Czerwinski and Lukasz Orlikowski, *Reachability in Vector Addition Systems is Ackermann-complete*, In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, 2021, pp. 1229–1240, URL: https://doi.org/10.1109/FOCS52979.2021.00120.

[CR93] Ran Canetti and Tal Rabin, *Fast asynchronous Byzantine agreement with optimal resilience*, In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computings*, 1993, pp. 42–51, URL: https://doi.org/10.1145/167088.167105.

[CTTV04] Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith, *Verification by Network Decomposition*, In: *CONCUR 2004 - Concurrency Theory, 15th International Conference*, vol. 3170, 2004, pp. 276–291, URL: https://doi.org/10.1007/978-3-540-28644-8%5C_18.

[Cyg+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh, *Parameterized Algorithms*, 2015, URL: https://doi.org/10.1007/978-3-319-21275-3.

[DDS87] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer, *On the minimal synchronism needed for distributed consensus*, In: *J. ACM* 34.*1* (1987), pp. 77–97, URL: https://doi.org/10.1145/7531.7533.

[DEGM15] Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar, *Model Checking Parameterized Asynchronous Shared-Memory Systems*, In: *Computer Aided Verification - 27th International Conference, CAV 2015*, vol. 9206, 2015, pp. 67–84, URL: https://doi.org/10.1007/978-3-319-21690-4%5C_5.

[DFGSS17] S. Demri, A. Finkel, J. Goubault-Larrecq, S. Schmitz, and Ph. Schnoebelen, *Well-Quasi-Orders for Algorithms*, Lecture Notes, MPRI Course 2.9.1 – 2017/2018, 2017, URL: https://wikimpri.dptinfo.ens-cachan.fr/lib/exe/fetch.php?media=cours:upload:poly-2-9-1v02oct2017.pdf.

[DP08] Rayna Dimitrova and Andreas Podelski, *Is Lazy Abstraction a Decision Procedure for Broadcast Protocols?*, In: *Verification, Model Checking, and Abstract Interpretation, 9th International Conference, VMCAI 2008*, vol. 4905, 2008, pp. 98–111, URL: https://doi.org/10.1007/978-3-540-78163-9%5C_12.

[DST13] Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso, *Parameterized Verification of Broadcast Networks of Register Automata*, In: *Reachability Problems - 7th International Workshop, RP 2013*, vol. 8169, 2013, pp. 109–121, URL: https://doi.org/10.1007/978-3-642-41036-9%5C_11.

[DSTZ12] Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro, *On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks*, In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012*, vol. 18, 2012, pp. 289–300, URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2012.289.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro, *Parameterized Verification of Ad Hoc Networks*, In: *Concurrency Theory, 21th International Conference, CONCUR 2010*, vol. 6269, 2010, pp. 313–327, URL: https://doi.org/10.1007/978-3-642-15375-4%5C_22.

[DSZ11] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro, *On the Power of Cliques in the Parameterized Verification of Ad Hoc Networks*, In: *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011*, vol. 6604, 2011, pp. 441–455, URL: https://doi.org/10.1007/978-3-642-19805-2%5C_30.

[EFM99] Javier Esparza, Alain Finkel, and Richard Mayr, *On the Verification of Broadcast Protocols*, In: *14th Annual IEEE Symposium on Logic in Computer Science*, 1999, pp. 352–359, URL: https://doi.org/10.1109/LICS.1999.782630.

[EGLM16] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar, *Model Checking Population Protocols*, In: *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, vol. 65, 2016, 27:1–27:14, URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2016.27.

[EGM13] Javier Esparza, Pierre Ganty, and Rupak Majumdar, *Parameterized Verification of Asynchronous Shared-Memory Systems*, In: *Computer Aided Verification - 25th International Conference, CAV 2013*, vol. 8044, 2013, pp. 124–140, URL: https://doi.org/10.1007/978-3-642-39799-8%5C_8.

[EGM16] Javier Esparza, Pierre Ganty, and Rupak Majumdar, *Parameterized Verification of Asynchronous Shared-Memory Systems*, In: *J. ACM* 63.*1* (2016), 10:1–10:48, URL: https://doi.org/10.1145/2842603.

[EJW24] Paul Eichler, Swen Jacobs, and Chana Weil-Kennedy, *Parameterized Verification of Systems with Precise (0,1)-Counter Abstraction*, 2024, URL: https://arxiv.org/abs/2408.05954.

[EK03] E. Allen Emerson and Vineet Kahlon, *Model Checking Guarded Protocols*, In: *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, 2003, pp. 361–370, URL: https://doi.org/10.1109/LICS.2003.1210076.

[EKS18] Javier Esparza, Jan Kretínský, and Salomon Sickert, *One Theorem to Rule Them All: A Unified Translation of LTL into ω-Automata*, In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, 2018, pp. 384–393, URL: https://doi.org/10.1145/3209108.3209161.

[EN03] E. Allen Emerson and Kedar S. Namjoshi, *On Reasoning About Rings*, In: *Int. J. Found. Comput. Sci.* 14.*4* (2003), pp. 527–550, URL: https://doi.org/10.1142/S0129054103001881.

[ERW19] Javier Esparza, Mikhail A. Raskin, and Chana Weil-Kennedy, *Parameterized Analysis of Immediate Observation Petri Nets*, In: *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019*, vol. 11522, 2019, pp. 365–385, URL: https://doi.org/10.1007/978-3-030-21571-2%5C_20.

[Esp14] Javier Esparza, *Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk)*, In: *31st International Symposium on Theoretical Aspects of Computer Science, STACS 2014*, vol. 25, 2014, pp. 1–10, URL: https://doi.org/10.4230/LIPIcs.STACS.2014.1.

[Esp17] Javier Esparza, *Advances in Parameterized Verification of Population Protocols*, In: *Computer Science - Theory and Applications - 12th International Computer Science Symposium in Russia, CSR 2017*, vol. 10304, 2017, pp. 7–14, URL: https://doi.org/10.1007/978-3-319-58747-9%5C_2.

[FL02] Alain Finkel and Jérôme Leroux, *How to Compose Presburger-Accelerations: Applications to Broadcast Protocols*, In: *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference*, vol. 2556, 2002, pp. 145–156, URL: https://doi.org/10.1007/3-540-36206-1%5C_14.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson, *Impossibility of Distributed Consensus with One Faulty Process*, In: *J. ACM* 32.*2* (1985), pp. 374–382, URL: https://doi.org/10.1145/3149.214121.

[FMW17] Marie Fortin, Anca Muscholl, and Igor Walukiewicz, *Model-Checking Linear-Time Properties of Parametrized Asynchronous Shared-Memory Pushdown Systems*, In: *Computer Aided Verification - 29th International Conference, CAV 2017*, vol. 10427, 2017, pp. 155–175, URL: https://doi.org/10.1007/978-3-319-63390-9%5C_9.

[Fou15] Paulin Fournier, *Parameterized verification of networks of many identical processes*, PhD thesis, University of Rennes 1, France, 2015, URL: https://tel.archives-ouvertes.fr/tel-01355847.

[FR03] Faith E. Fich and Eric Ruppert, *Hundreds of impossibility results for distributed computing*, In: *Distributed Comput.* 16.*2-3* (2003), pp. 121–163, URL: https://doi.org/10.1007/s00446-003-0091-y.

[FS01] Alain Finkel and Philippe Schnoebelen, *Well-structured transition systems everywhere!*, In: *Theor. Comput. Sci.* 256.*1-2* (2001), pp. 63–92, URL: https://doi.org/10.1016/S0304-3975(00)00102-X.

[GH19] Rob van Glabbeek and Peter Höfner, *Progress, Justness, and Fairness*, In: *ACM Comput. Surv.* 52.*4* (2019), 69:1–69:38, URL: https://doi.org/10.1145/3329125.

[GM01] Eli Gafni and Michael Mitzenmacher, *Analysis of Timing-Based Mutual Exclusion with Random Times*, In: *SIAM J. Comput.* 31.*3* (2001), pp. 816–837, URL: https://doi.org/10.1137/S0097539799364912.

[GM14] Alexander Gogolev and Lucio Marcenaro, *Efficient binary consensus in randomized and noisy environments*, In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Pro-*

cessing (ISSNIP), 2014, pp. 1–6, URL: https://doi.org/10.1109/ISSNIP.2014.6827594.

[GMMB15] Alexander Gogolev, Nikolaj Marchenko, Lucio Marcenaro, and Christian Bettstetter, *Distributed Binary Consensus in Networks with Disturbances*, In: *ACM Trans. Auton. Adapt. Syst.* 10.*3* (2015), 19:1–19:17, URL: https://doi.org/10.1145/2746347.

[GMW24] Lucie Guillou, Corto Mascle, and Nicolas Waldburger, *Parameterized Broadcast Networks with Registers: from NP to the Frontiers of Decidability*, In: *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024*, vol. 14575, 2024, pp. 250–270, URL: https://doi.org/10.1007/978-3-031-57231-9%5C_12.

[GR07] Rachid Guerraoui and Eric Ruppert, *Anonymous and fault-tolerant shared-memory computing*, In: *Distributed Comput.* 20.*3* (2007), pp. 165–177, URL: https://doi.org/10.1007/s00446-007-0042-0.

[GS92] Steven M. German and A. Prasad Sistla, *Reasoning about Systems with Many Processes*, In: *J. ACM* 39.*3* (1992), pp. 675–735, URL: https://doi.org/10.1145/146637.146681.

[GSS23] Lucie Guillou, Arnaud Sangnier, and Nathalie Sznajder, *Safety Analysis of Parameterised Networks with Non-Blocking Rendez-Vous*, In: *34th International Conference on Concurrency Theory, CONCUR 2023*, vol. 279, 2023, 7:1–7:17, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2023.7.

[GSWW24] Pierre Ganty, Cesar Sanchez, Nicolas Waldburger, and Chana Weil-Kennedy, *Temporal Hyperproperties on Population Protocols*, in preparation, 2024.

[Hag11] Matthew Hague, *Parameterised Pushdown Systems with Non-Atomic Writes*, In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, vol. 13, 2011, pp. 457–468, URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2011.457.

[Haw+17] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael Lowell Roberts, Srinath T. V. Setty, and Brian Zill, *IronFleet: proving safety and liveness of practical distributed systems*, In: *Commun. ACM* 60.*7* (2017), pp. 83–92, URL: https://doi.org/10.1145/3068608.

[HLM] Maurice Herlihy, Victor Luchangco, and Mark Moir, *Obstruction-Free Synchronization: Double-Ended Queues as an Example*, In: *23rd International Conference on Distributed Computing Systems, ICDCS 2003*.

[HS20] Florian Horn and Arnaud Sangnier, *Deciding the Existence of Cut-Off in Parameterized Rendez-Vous Networks*, In: *31st International Conference on Concurrency Theory, CONCUR 2020*, vol. 171, 2020, 46:1–46:16, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2020.46.

[Joh11] Derek Johnston, *An Introduction to Random Walks*, 2011, URL: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Johnston.pdf.

[JS18] Swen Jacobs and Mouhammad Sakr, *Analyzing Guarded Protocols: Better Cutoffs, More Systems, More Expressivity*, In: *Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018*, vol. 10747, 2018, pp. 247–268, URL: https://doi.org/10.1007/978-3-319-73721-8%5C_12.

[KKW10] Alexander Kaiser, Daniel Kroening, and Thomas Wahl, *Dynamic Cutoff Detection in Parameterized Concurrent Programs*, In: *Computer Aided Verification, 22nd International Conference, CAV 2010*, vol. 6174, 2010, pp. 645–659, URL: https://doi.org/10.1007/978-3-642-14295-6%5C_55.

[KKW18] Jure Kukovec, Igor Konnov, and Josef Widder, *Reachability in Parameterized Systems: All Flavors of Threshold Automata*, In: *29th International Conference on Concurrency Theory, CONCUR 2018*, vol. 118, 2018, 19:1–19:17, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2018.19.

[KLSW23] Igor Konnov, Marijana Lazic, Ilina Stoilkovska, and Josef Widder, *Survey on Parameterized Verification with Threshold Automata and the Byzantine Model Checker*, In: *Log. Methods Comput. Sci.* 19.*1* (2023), URL: https://doi.org/10.46298/lmcs-19(1:5)2023.

[KN02] Marta Z. Kwiatkowska and Gethin Norman, *Verifying Randomized Byzantine Agreement*, In: *Formal Techniques for Networked and Distributed Systems - FORTE 2002*, vol. 2529, 2002, pp. 194–209, URL: https://doi.org/10.1007/3-540-36135-9%5C_13.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker, *PRISM 4.0: Verification of Probabilistic Real-Time Systems*, In: *Computer Aided Verification - 23rd International Conference, CAV*, vol. 6806, 2011, pp. 585–591, URL: https://doi.org/10.1007/978-3-642-22110-1%5C_47.

[KNS01] Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala, *Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM*, In: *Computer Aided Verification, 13th International Conference, CAV 2001*, vol. 2102, 2001, pp. 194–206, URL: https://doi.org/10.1007/3-540-44585-4%5C_17.

[KVW14] Igor Konnov, Helmut Veith, and Josef Widder, *On the Completeness of Bounded Model Checking for Threshold-Based Distributed Algorithms: Reachability*, In: *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014*, vol. 8704, 2014, pp. 125–140, URL: https://doi.org/10.1007/978-3-662-44584-6%5C_10.

[LA87] M. C. Loui and H. H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, In: 1987, pp. 163–183.

[Lad75] Richard E. Ladner, *The circuit value problem is log space complete for P*, In: *SIGACT News* 7.*1* (1975), pp. 18–20, URL: https://doi.org/10.1145/990518.990519.

[Lam06] Leslie Lamport, *Checking a Multithreaded Algorithm with* $^{+}$*CAL*, In: *Distributed Computing, 20th International Symposium, DISC 2006*, vol. 4167, 2006, pp. 151–163, URL: https://doi.org/10.1007/11864219%5C_11.

[Lam19] Leslie Lamport, *The mutual exclusion problem: part II - Statement and solutions*, In: *Concurrency: the Works of Leslie Lamport*, 2019, pp. 247–276, URL: https://doi.org/10.1145/3335772.3335938.

[Lam83] Leslie Lamport, *What Good is Temporal Logic?*, In: *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress*, 1983, pp. 657–668, URL: https://lamport.azurewebsites.net/pubs/what-good.pdf.

[Lei10] K. Rustan M. Leino, *Dafny: An Automatic Program Verifier for Functional Correctness*, In: *Logic for Programming, Artificial Intelligence, and Rea-*

soning - 16th International Conference, vol. 6355, 2010, pp. 348–370, URL: https://doi.org/10.1007/978-3-642-17511-4%5C_20.

[LLMR17] Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer, *Fair Termination for Parameterized Probabilistic Concurrent Systems*, In: *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*, vol. 10205, 2017, pp. 499–517, URL: https://doi.org/10.1007/978-3-662-54577-5%5C_29.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi, *UPPAAL in a Nutshell*, In: *Int. J. Softw. Tools Technol. Transf.* 1.*1-2* (1997), pp. 134–152, URL: https://doi.org/10.1007/s100090050010.

[LR16] Anthony W. Lin and Philipp Rümmer, *Liveness of Randomised Parameterised Systems under Arbitrary Schedulers*, In: *Computer Aided Verification - 28th International Conference, CAV 2016*, vol. 9780, 2016, pp. 112–133, URL: https://doi.org/10.1007/978-3-319-41540-6%5C_7.

[LR81] Daniel Lehmann and Michael O. Rabin, *On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem*, In: *Conference Record of the Eighth Annual ACM Symposium on Principles of Programming Languages*, 1981, pp. 133–138, URL: https://doi.org/10.1145/567532.567547.

[LS21] Ranko Lazic and Sylvain Schmitz, *The ideal view on Rackoff's coverability technique*, In: *Inf. Comput.* 277 (2021), p. 104582, URL: https://doi.org/10.1016/j.ic.2020.104582.

[Lyn96] Nancy A. Lynch, *Distributed Algorithms*, 1996.

[Pan01] Prakash Panangaden, *Measure and probability for concurrency theorists*, In: *Theor. Comput. Sci.* 253.*2* (2001), pp. 287–309, URL: https://doi.org/10.1016/S0304-3975(00)00096-7.

[Par11] Dave Parker, *Lectures - Probabilistic Model Checking*, 2011, URL: https://www.prismmodelchecker.org/lectures/pmc/.

[PLD08] Jun Pang, Zhengqin Luo, and Yuxin Deng, *On Automatic Verification of Self-Stabilizing Population Protocols*, In: *Second IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE 2008*, 2008, pp. 185–192, URL: https://doi.org/10.1109/TASE.2008.8.

[Pnu77] Amir Pnueli, *The Temporal Logic of Programs*, In: *18th Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57, URL: https://doi.org/10.1109/SFCS.1977.32.

[PSL00] Anna Pogosyants, Roberto Segala, and Nancy A. Lynch, *Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study*, In: *Distributed Comput.* 13.*3* (2000), pp. 155–186, URL: https://doi.org/10.1007/PL00008917.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport, *Reaching Agreement in the Presence of Faults*, In: *Journal of the ACM* 27.*2* (1980), pp. 228–234, URL: https://doi.org/10.1145/322186.322188.

[PW97] Doron A. Peled and Thomas Wilke, *Stutter-Invariant Temporal Properties are Expressible Without the Next-Time Operator*, In: *Inf. Process. Lett.* 63.*5* (1997), pp. 243–246, URL: https://doi.org/10.1016/S0020-0190(97)00133-6.

[Rab76] M.O. Rabin, *Probabilistic Algorithms*, 1976.

[Ras21] Mikhail A. Raskin, *Population Protocols with Unreliable Communication*, In: *Algorithms for Sensor Systems - 17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2021*, vol. 12961, 2021, pp. 140–154, URL: https://doi.org/10.1007/978-3-030-89240-1%5C_10.

[Ras23] Mikhail A. Raskin, *Protocols with constant local storage and unreliable communication*, In: *Theor. Comput. Sci.* 940.*Part* (2023), pp. 269–282, URL: https://doi.org/10.1016/j.tcs.2022.11.006.

[RS12] Michel Raynal and Julien Stainer, *A Simple Asynchronous Shared Memory Consensus Algorithm Based on Omega and Closing Sets*, In: *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012*, 2012, pp. 357–364, URL: https://doi.org/10.1109/CISIS.2012.198.

[Sav70] Walter J. Savitch, *Relationships Between Nondeterministic and Deterministic Tape Complexities*, In: *J. Comput. Syst. Sci.* 4.*2* (1970), pp. 177–192, URL: https://doi.org/10.1016/S0022-0000(70)80006-X.

[SS24] Sylvain Schmitz and Lia Schütze, *On the Length of Strongly Monotone Descending Chains over N^d*, In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*, vol. 297, 2024, 153:1–153:19, URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.153.

[Sta17] Daniel Stan, *Randomized strategies in concurrent games. (Stratégies randomisées dans les jeux concurrents)*, PhD thesis, Université de Paris-Saclay, France, 2017, URL: https://tel.archives-ouvertes.fr/tel-01519354.

[tea04] The Coq development team, *The Coq proof assistant reference manual*, Version 8.0, LogiCal Project, 2004, URL: http://coq.inria.fr.

[TMW15] Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz, *Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable*, In: *26th International Conference on Concurrency Theory, CONCUR 2015*, vol. 42, 2015, pp. 72–84, URL: https://doi.org/10.4230/LIPIcs.CONCUR.2015.72.

[Wal23] Nicolas Waldburger, *Checking Presence Reachability Properties on Parameterized Shared-Memory Systems*, In: *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023*, vol. 272, 2023, 88:1–88:15, URL: https://doi.org/10.4230/LIPIcs.MFCS.2023.88.

[Wei23] Chana Weil-Kennedy, *Observation Petri Nets*, PhD thesis, Technical University of Munich, Germany, 2023, URL: https://nbn-resolving.org/urn:nbn:de:bvb:91-diss-20230320-1691161-1-3.

[WWPTWEA15] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson, *Verdi: a framework for implementing and formally verifying distributed systems*, In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015, pp. 357–368, URL: https://doi.org/10.1145/2737924.2737958.

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Vérification paramétrée de systèmes distribués à mémoire partagée

Mot clés : vérification paramétrée, systèmes infinis, algorithmes distribués

Résumé : Les systèmes distribués sont constitués de plusieurs composantes informatisés (que nous appelons processus) qui interagissent pour accomplir une tâche commune. Un exemple de tâche est le consensus, où tous les processus doivent se mettre d'accord sur une valeur commune. Dans cette thèse, nous nous intéressons aux systèmes à mémoire partagée, où les processus interagissent en lisant et en écrivant dans une mémoire partagée. Nous ne travaillons pas directement sur des systèmes distribués, mais plutôt sur des modèles de ces systèmes, où nous considérons des questions de vérification automatique. Nos modèles sont paramétrés : le nombre de processus n'est pas fixé à l'avance et peut être arbitrairement grand, ce qui nous permet de vérifier le système pour tout nombre de participants. Cette hypothèse permet également des propriétés de monotonicité qui simplifient l'analyse. Notre modèle, inspiré par des algorithmes de consensus de la littérature, est à ronde : chaque processus évolue de manière incrémentale en un nombre appelé ronde, et où chaque ronde a sa propre mémoire partagée. Nous étudions de plus l'impact d'un ordonnanceur stochastique sur ce modèle à rondes. Notre approche est théorique et nous nous intéressons principalement à l'analyse de nos modèles et à la classification de nos problèmes en termes de classes de complexité.

Title: Parameterized verification of distributed shared-memory systems

Keywords: parameterized verification, infinite-state systems, distributed algorithms

Abstract: Distributed systems consist in several computerized components (called *processes*) interacting with one another to perform a common task. An example of such a task is consensus, where all processes must agree on a common value. In this thesis, we are interested in shared-memory systems where the processes interact via reading from and writing to a shared memory. We do not work directly on distributed systems, but rather on models of such systems, and we consider questions related to the automated verification of these models. Our models are parameterized: the number of processes is not fixed beforehand and may be arbitrarily large, so that we are able to verify the system regardless of the number of participants. They enjoy some monotonicity property, called copycat property, which simplifies the analysis. Motivated by consensus algorithms from the literature, we consider a model where each process evolves incrementally in a number called round and where each round has its own set of registers. Also, we study the impact of a stochastic scheduler on this round-based model. Our approach is theoretical and we are mostly interested in analyzing our models and in classifying our problems of interest in terms of complexity.