Theoretical complexity of reasoning in dynamicepistemic logic and study of a symbolic approach

Tristan Charrier

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

Ecole Doctorale N°601
Mathématique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Tristan CHARRIER »

« Complexité théorique du raisonnement en logique épistémique dynamique et étude d'une approche symbolique »

Thèse présentée et soutenue à Rennes, le 5 décembre 2018
Unité de recherche : IRISA

Rapporteurs avant soutenance :

DIMA Catalin, Professeur, Université Paris-Est Créteil
MURANO Aniello, Associate Professor, University of Naples, Italie

Composition du jury :

Président :	RIDOUX Olivier, Professeur, Université de Rennes 1
Examinateurs	VAN DITMARSCH Hans, Directeur de Recherche, LORIA (Nancy)
	BOLANDER Thomas, Associate Professor, Technical University of Denmark
	DIMA Catalin, Professeur, Université Paris-Est Créteil
	MURANO Aniello, Associate Professor, University of Naples, Italie
	SCHNOEBELEN Philippe, Directeur de Recherche, ENS Paris-Saclay
Dir. de thèse :	PINCHINAT Sophie, Professeur, Université de Rennes 1
Co-enc. de thèse :	SCHWARZENTRUBER François, Maître de Conférence, ENS Rennes

Remerciements

Tout d'abord, je tiens à remercier chaleureusement Sophie Pinchinat pour toutes ces années passées dans l'équipe LogicA. Son enthousiasme et sa passion pour son métier ont été une grande inspiration durant ma thèse, et sa grande vision du domaine m'a beaucoup aidé à aborder mon sujet sous d'autres angles. Sophie a été également d'un soutien moral indéfectible durant mes périodes chargées, et je suis très reconnaissant et honoré d'avoir pu préparer mon doctorat dans son équipe. Je remercie également de tout cœur François Scwharzentruber, qui m'a très bien encadré pendant ma thèse. C'est un chercheur qui aime aussi beaucoup son métier, et c'était un plaisir de travailler avec lui. Sa sympathie et son humour étaient très agréables à vivre au quotidien. En outre, et quoi qu'il en dise, c'est un excellent enseignant, qui désire réellement que ses étudiants réussissent et qui sait faire preuve de beaucoup de pédagogie, et j'ai beaucoup appris à ses côtés. Son énergie entrainante m'a motivé jusqu'au bout de ma thèse, et pour cela merci.

I would also like to thank the jury who participated in my PhD defense. Many thanks to Catalin Dima and Aniello Murano for reviewing my manuscript, your comments were truly insightful. I thank too the other members of the jury: Thomas Bolander, Olivier Ridoux, Philippe Schnoebelen and Hans van Ditmarsch for accepting the examination of my defense and for all your great comments during the defense. I also am honored to have the MAFTEC workshop co-organized with my PhD defense, and I thank a lot Frédéric Maris and François Schwarzentruber for the organization.

I am also truly grateful to have had the opportunity to spend three months in Barcelona in the team of artificial intelligence lead by Anders Johnson. My main contact, Hector Geffner, is a warmful and nice person, who welcomed me with open arms, and gave me the opportunity to learn more about planning techniques. I also thank the other members of the research team in Barcelona. In particular I thank Filippos Kominis for all our interesting discussions about epistemic planning, and Guillem Frances for your help using your classical planner.

Je remercie tous les chercheurs et personnels de l'IRISA/INRIA que j'ai cotoyés pendant ces années, en particulier notre ancienne assistante Tifenn Donguy et notre nouvelle assistante Fanny Banor pour leur disponibilité et leur sympathie, les autres membres de

l'équipe LogicA, Maxime, Sébastien, Arthur et Florence, qui animent au quotidien la vie de l'équipe. Ce laboratoire est vraiment un endroit merveilleux pour travailler, et c'était un réel plaisir et côtoyer également les autres membres des autres équipes, en particulier Sumo et Celtique.

Évidemment je remercie également beaucoup mes proches pour m'avoir soutenu pendant ces trois ans, en particulier mon père, ma mère, ma sœur Marie et mon frère Gabriel. Je remercie aussi Anthony, qui me supporte au quotidien et qui m'a toujours encouragé à aller plus loin et viser plus haut dans tout ce que j'entreprends. J'ai énormément de chance d'avoir une famille comme vous ! Je remercie également mes amis, en particulier ceux de Tours, Lola, Elric, Victor, Maxence, Clara et Julie, ceux de Rennes et entre autres ceux du club jeux, Hugo, Elisa, Engel, Laura, Gwezheneg, Casey, Enora, Bastien, Jean-Joseph, Yann, Simon, Camille, Antoine, Gurvan, Gaetan, et enfin mes autres amis anciens de l'ENS, Antonin, Théo, Thomas et Mathieu.

Sommaire

Résumé étendu en français

Le corps du document est écrit en anglais. Ce résumé présente le contexte, l'état de l'art et la contribution développée dans le reste du document.

Contexte

Le concept de *système multi-agents* est communément accepté comme une représentation d'exemples de systèmes réels où des entités autonomes, les *agents*, agissent conformément à leur *connaissance* du monde courant afin d'atteindre des buts individuels ou collectifs.

Considérons des exemples de systèmes multi-agents.

- Dans les systèmes multi-robots, les agents sont les robots qui accumulent de la connaissance par l'environnement qu'ils perçoivent depuis leurs capteurs et par l'information obtenue via des canaux de communication [Lem+14]. Ils doivent également prendre en compte les autres agents, en particulier les agents humains, pour remplir leurs objectifs. Par exemple, Lemaignan et al. [Lem+10] donnent un exemple de gestion de la connaissance dans les systèmes multi-robots où les interactions avec l'humain sont prises en compte. Leur système met en jeu une base de connaissance sur l'environnement que le robot maintient. Le domaine de l'intelligence distribuée est dédiée précisément à ce problème, avec des applications directes pour les systèmes multi-robots [Par08].

- En sécurité, les agents sont l'attaquant et le propriétaire du système. Par exemple en sécurité réseau [PKS16], le propriétaire désire que l'attaquant ait le moins de connaissance possible sur le système. En effet, si l'attaquant obtient de l'information, il pourrait être alors capable de compromettre ce système, comme par exemple dans les attaques d'élévation des privilèges. En outre en cryptographie [MOV96], le propriétaire désire que l'information transmise ne soit pas décryptée par l'attaquant. En effet, l'information peut être aisément interceptée par l'attaquant car le canal de communication est ouvert, mais s'il ne possède pas la clé de cryptage, il ne sera pas capable de lire le contenu du message. Nous pouvons également mentionner

les détections d'attaque dans les bâtiments, généralement modélisées par des arbres d'attaque-défense [Kor+11], où l'attaquant veut accéder par exemple à une ressource cachée, et doit infiltrer le bâtiment sans être repéré. Dans ce cas, il s'agit du point du vue opposé à la sécurité réseau, puisque c'est l'attaquant qui veut que le propriétaire ait le moins d'information possible sur ses agissements.

- Dans les jeux de société, comme les jeux de carte, la bataille navale, Cluedo, etc., les agents sont les joueurs. Par exemple dans les jeux de carte classiques, la main d'un joueur est la seule information qu'il possède au début de la partie. Au fur et à mesure que le jeu progresse, les agents acquièrent de la connaissance supplémentaire de part les actions et déclarations des autres joueurs. Il existe une panoplie de jeux de société qui sont pertinents car ils couvrent le champ des interactions possibles entre les joueurs. Par exemple *Hanabi* [1] est un jeu où la connaissance mutuelle entre les agents est centrale, car un joueur ne voit que les mains des autres joueurs, et les joueurs doivent collaborer pour gagner ensemble un maximum de points. *Tragedy Looper* [2] met également en jeu une notion de collaboration, mais les joueurs doivent ici collaborer contre un joueur isolé, appelé le mastermind. En effet, dans ce jeu, les joueurs ont une vue incomplète du scénario, et doivent s'assurer que le mastermind, qui lui a une vision complète du scénario, ne déclenche pas une condition de défaite. Ils peuvent jouer le scénario plusieurs fois (par une mécanique de voyage dans le temps dans le jeu) afin de gagner de plus en plus de connaissance sur le contenu du scénario. Le mastermind lui doit s'assurer que les joueurs déclenchent toujours une condition de défaite et obtiennent le moins de connaissance possible. Enfin, beaucoup de jeux existent autour de la communication, comme le jeu *Loups-Garous de Thiercelieux* [3], où les joueurs ont tous un rôle caché, soit loup-garou soit villageois. Les villageois doivent démasquer les loups-garous pendant les phases de jour et les loups-garous doivent manger les villageois pendant les phases de nuit. Dans ce jeu, la connaissance est obtenue par les interactions entre les joueurs mais également par les rôles d'information, comme la voyante qui peut regarder le rôle d'un joueur chaque nuit.

- Dans les jeux vidéos, les agents sont les joueurs et les personnages non-jouables (PNJ). Les PNJs ont une connaissance et une capacité de raisonnement limitées,

1. Créé par Antoine Bauza en 2010, édité par Cocktail Games et XII Singes.
2. Créé par Bakafire en 2014, édité by Z-Man Games et BakaFire Party.
3. Créé par Philippe des Pallières et Hervé Marly en 2001, édité par Lui-Même.

mais il est pertinent de considérer les jeux vidéos comme des systèmes multi-agents lorsque les PNJs interagissent avec le joueur. Par exemple, dans les jeux de rôle massivement multi-joueur en ligne, les boss ont un mécanisme de menace qui leur permet de toujours attaquer le joueur qui représente la plus grande menace à leurs yeux. Il existe également des mécaniques plus sophistiquées, comme le boss notoire Dark Link tiré du jeu *"The Legend of Zelda: Ocarina of Time"*[4]. Sa vie se cale sur celle du joueur et il a une intelligence artificielle qui s'adapte à la façon d'attaquer du joueur. Par exemple si le joueur ne donne que des coups d'épée basiques, Dark Link va principalement bloquer les attaques.

Dans tous ces exemples, la notion de connaissance est centrale, en particulier la connaissance d'ordre supérieur, c'est à dire la connaissance que les agents ont à propos de la connaissance des autres agents. En effet, la connaissance d'ordre supérieur représente la façon que les agents ont de raisonner sur les autres agents. C'est donc une notion cruciale dans beaucoup des exemples cités au dessus, comme en sécurité où chaque agent doit raisonner sur la connaissance de l'autre agent, ou dans les jeux de carte pour concevoir des stratégies pertinentes. De plus, la *dynamique* est également importante, puisque les agents communiquent et agissent sur le système. Quelques exemples d'actions dynamiques seraient l'envoi d'un message, l'ouverture d'une porte, le déplacement d'un objet, le jeu d'une carte, le lancer d'un sort dans un jeu vidéo, etc. C'est pourquoi nous voulons exprimer les actions des agents dans les systèmes multi-agents.

Tout au long du document, nous nous focalisons sur la vérification de propriétés mettant en jeu de la connaissance dans les systèmes multi-agents. Puisque nous avons besoin d'exprimer la connaissance et des actions dynamiques, nous choisissons d'utiliser la *logique épistémique dynamique* qui exprime les deux. Nous étudions le problème de *vérification de modèles*, mais également le problème d'existence d'un modèle, appelé problème de *satisfiabilité*. Nous étudions également leurs variantes *symboliques*. Nous allons aussi plus loin en considérant des variantes stratégiques de la logique épistémique dynamique, comme la *planification épistémique* ou les *actions arbitraires*.

Notions introduites dans le document

Logique épistémique Dans le document, nous mettons en avant l'utilisation de la logique pour exprimer des propriétés dans les systèmes multi-agents. En effet, la logique

4. Édité par Nintendo.

donne un langage de spécification, et est utile pour définir des procédures de vérification, puisque sa définition claire permet d'écrire naturellement ces procédures.

Nous considérons la *logique épistémique* introduite par Hintikka [Hin65]. Elle étend la logique propositionnelle avec des opérateurs de connaissance, comme K_a pour "L'agent a sait que". Toutes les valeurs considérées sont booléennes, ce qui veut dire que les deux seules valeurs possibles sont vrai ($\top$) ou faux ($\bot$). Nous pouvons ainsi exprimer toute variable avec une valeur sur un domaine fini, comme les entiers bornés par exemple en utilisant plusieurs variables booléennes. En revanche, nous ne pouvons pas exprimer les valeurs dans des domaines infinis, comme les rationnels ou les réels.

L'opérateur K_a est appelé une modalité. En effet, la logique épistémique est un type spécial de logique modale (cf par exemple [BBW06] pour plus d'information sur la logique modale) où les seules modalités sont K_a. La différence principale entre la logique propositionnelle et la logique modale réside dans l'interprétation des formules. Alors que les formules propositionnelles sont évaluées sur les interprétations des variables (appelées *valuations*), les formules modales sont évaluées sur ce que l'on appelle des *modèles de Kripke*. Ces modèles peuvent être vus comme des graphes où chaque noeud (appelé *monde*) possède une valuation et représente une configuration possible, par exemple le contenu des mains des joueurs. Les arcs représentent quels mondes les agents imaginent comme possibles, de telle manière que K_a est interprété comme " pour tous les mondes imaginés comme possibles par l'agent a, ... ". Les systèmes multi-agents sont alors représentés par des modèles de Kripke et les propriétés sont des formules de la logique épistémique.

Pour plus d'information à propos de la logique épistémique, le lecteur peut se référer aux livres [Fag+95; MV04] qui décrivent des travaux classiques sur la logique épistémique, ou à l'article [vV02]. Plus récemment, le livre [Dit+15] couvre également la logique épistémique et plusieurs de ses extensions.

Néanmoins la logique épistémique a plusieurs limitations. En effet, les agents sont supposés rationnels et ont une connaissance parfaite du modèle de Kripke. Plusieurs extensions de la logique épistémique existent pour résoudre ces problèmes. Par exemple l'article [FH87] introduit une extension de la logique épistémique où les agents peuvent être ignorants de l'existence de propositions atomiques, et ont des capacités de raisonnements limitées, ce qui signifie qu'ils ne peuvent pas déduire toutes les implications logiques. De plus, la logique épistémique ne se focalise par sur *pourquoi* les agents savent une propriété. La logique de justification [Art08] définit un mécanisme de justification de la connaissance des agents. Pour citer un article plus récent, l'article [SW18] propose

une logique où les noms des agents ne sont pas connaissance commune, faisant ainsi une distinction entre "savoir que l'agent a a commis un crime sans connaître son nom" et "savoir qu'un agent nommé a a commis un crime sans savoir qui c'est."

Dans le document, nous ne considérons pas ces extensions pour garder une logique simple et considérons la définition usuelle de la logique épistémique.

Connaissance commune La connaissance commune a d'abord été définie formellement dans [Aum76]. On dit qu'une formule est connaissance commune si elle est vraie dans chaque monde possible. On peut également définir la connaissance commune par une conjonction de formules de la forme "tout le monde sait que tout le monde sait que ...".

La notion de connaissance commune en logique épistémique est très proche de l'opérateur AG de CTL (cf [BK08] pour plus d'information sur les logiques temporelles). Les deux opérateurs peuvent être vus comme des opérateurs de point fixe comme ceux du μ-calcul [Pra81].

Dans le document, tous les chapitres excepté le Chapitre 4 autorisent l'opérateur de connaissance commune dans la logique.

Vérification de modèles et satisfiabilité Quand on définit une logique, il est classique d'étudier ses propriétées computationnelles. Ici, nous considérons une approche similaire à [HV91] en définissant le problème de *vérification de modèles*. Ce problème prend en entrée un modèle et uen formule et vérifie si la formule est vraie dans ce modèle. Nous prouvons d'ailleurs en Annexe C que le problème est dans P pour la logique épistémique, avec une preuve très similaire à celle pour CTL. Pour un rappel sur la complexité computationnelle, le lecteur peut se référer à l'Annexe A.

Un autre problème classique est de vérifer la *satisfiabilité* d'une formule, c'est à dire trouver un modèle de Kripke satisfaisant la formule. En réalité, le problème de satisfiabilité est dual au problème de preuve de théorème, puisqu'une formule φ est valide si et seulement si $\neg\varphi$ n'est pas satisfiable. Pour la logique épistémique, le problème de satisfiabilité est PSPACE-complet sans connaissance commune, avec au moins deux agents et pour des modèles S5 [5] [HM92].

Il est également classique de définir *l'axiomatisation* de la logique. Il s'agit d'un ensemble d'axiomes et de règles qui illustrent la construction de formules prouvables (cf

5. c'est à dire des modèles de Kripke avec des arcs qui forment une relation d'équivalence.

[Fag+95] pour la logique épistémique). Nous établissons généralement si cette axiomatisation est correcte (chaque formule prouvable est valide) et complète (chaque formule valide est prouvable). Lorsqu'une axiomatisation correcte et complète est proposée, il est parfois possible de définir alors un algorithme de *preuve de théorème* qui vérifie la validité de formules en utilisant l'axiomatisation. La différence principale avec la vérification de modèles est que l'on prouve alors la validité de la formule en entrée au lieu de sa satisfaction dans un modèle donné. Si nous voulons alors utiliser la preuve de théorème pour vérifier une formule φ dans un modèle, nous devons caractériser le modèle par une formule $\varphi_{\mathcal{M}}$ et prouver la validité de $\varphi_{\mathcal{M}} \to \varphi$. La preuve de théorème étant duale à la satisfiabilité, la vérification de modèles est en général un problème plus simple. C'est pourquoi nous n'utilisons pas la preuve de théorème ici, car ce serait un problème trop difficile pour l'utilisation que l'on compte en faire.

Modèles symboliques Lorsque l'on souhaite utiliser la vérification de modèles et la satisfiabilité en pratique, nous nous heurtons généralement à l'explosion combinatoire des modèles de Kripke. Par exemple dans les jeux de carte, quand il y a k joueurs et n cartes, le nombre de propositions atomiques est généralement $k \times n$. Si l'on considère un jeu de tarot classique avec 5 joueurs, le nombre de mondes possibles serait alors environ 6×10^{48}, ce qui est du même ordre de grandeur que le nombre d'atomes sur Terre. Il paraît donc déraisonnable de manipuler les modèles explicitement.

Pour cette raison, nous proposons une représentation symbolique des modèles de Kripke, ce qui évite de les définir à la main. De telles représentations existent déjà pour d'autres formalismes, comme les circuits booléens pour les chemins Hamiltoniens dans les graphes [Pap03]. Généralement, une représentation symbolique définit un modèle par des formules booléennes au lieu de définir le modèle de façon explicite. Par exemple l'article [Bur+90] donne une représentation symbolique des modèles pour les logiques temporelles. Il est alors possible d'utiliser des structures de données appelées des diagrammes de décision binaires ordonnés et réduits [DB98] (ROBDDs), afin d'implémenter les représentations symboliques.

Cela peut paraître surprenant de considérer des procédures de vérification avec des modèles symboliques en entrée, car cela peut paraître très similaire à la méthode de preuve de théorème mentionnée précédemment. La différence principale est que la description des modèles (donnée par des ROBDDs par exemple) n'est pas écrite dans le même langage que la formule (par exemple en logique épistémique), cela aurait donc peu de sens d'essayer

de prouver $\varphi_{\mathcal{M}} \to \varphi$ avec $\varphi_{\mathcal{M}}$ la représentation symbolique du modèle de Kripke $\mathcal{M}$.

Pour la logique épistémique, plusieurs approches symboliques existent déjà, qui sont décrites plus en détail dans le Chapitre 3. La première approche définie associe les mondes à des valuations et les agents observent un ensemble de propositions atomiques [HTW11; HIW12; Ågo+13]. Dans ce cas, un modèle de Kripke est seulement défini par l'ensemble des mondes et les propositions que chaque agent observe, les relations épistémiques étant alors inférées des ensembles d'observation. En outre, l'article [GGS15] étend cette idée en considérant des agents dans un espace de dimension 2 qui peuvent seulement voir un cône devant eux. Enfin, l'article [HLM15] considère une représentation des relations épistémiques avec des variables de la forme $S_a p$ pour "l'agent a perçoit la valeur de p" qui peuvent être d'ordre supérieur comme $S_a S_b p$ pour "l'agent a perçoit la valeur de $S_b p$".

Ici, nous utilisons une approche symbolique qui utilise les programmes de la logique dynamique avec des affectations propositionnelles (DL-PA [BHT13]), qui sont des programmes particuliers de la logique dynamique propositionnelle (PDL [FL79]) où les programmes atomiques sont des affectations propositionnelles. Même si ce ne sont pas directement des formules booléennes, ces programmes peuvent être traduits en formules booléennes classiques afin d'utiliser des techniques ROBDDs [6]. Cette approche a les avantages habituels des modèles symboliques : ils peuvent exprimer n'importe quel modèle, ils sont succincts, et ils permettent une représentation simple des exemples habituels de la logique épitémique.

Dynamique dans la logique épistémique Comme souligné dans le paragraphe sur la logique épistémique, on ne peut pas y exprimer la dynamique dans les modèles. Nous considérons ici la *logique épistémique dynamique*. C'est un terme parapluie pour toutes les extensions de la logique épistémique avec des opérateurs dynamiques. Les extensions de la logique épistémique avec de la dynamique sont fortement inspirées de la logique dynamique comme définie dans [FL79].

Initialement, la première occurrence d'opérateurs dynamiques dans la logique épistémique reposait sur les annonces publiques (la logique associée est nommée PAL) [Pla07] [7]. Cela correspond à des diffusions de propriétés vraies à tous les agents. Par exemple dans un jeu vidéo, si un joueur dit "le boss est au milieu attention !", tous les joueurs sauront maintenant que le boss est au milieu. Dans ce cas, si le boss est également considéré comme

6. En fait, le Chapitre 6 donne une réduction vers la logique du premier ordre, ce qui permet également d'utiliser les ROBDDs par la suite.

7. Le papier cité est une réédition de 2007 d'un papier de 1989.

un agent, nous pouvons considérer que le boss sait maintenant que les joueurs savent sa position, donc il peut agir en conséquence en se cachant par exemple.

La logique épistémique dynamique comme référéee dans le reste du document (appelée **DEL**) étend PAL en considérant des événements plus riches (annonces privées par exemple), l'article fondateur étant [BMS98]. Un événement est maintenant représenté par un graphe, similairement aux modèles de Kripke, et est appelé un *modèle d'événement*. Dans ce modèle, chaque noeud est un événement possible et les arcs, étiquetés par des agents, représentent les événements que les agents imaginent comme possible. Par exemple, dans l'exemple du boss, le modèle d'événement pour "il y a une annonce que le boss n'entend pas" contiendrait deux événements : l'événement "annoncer que le boss est au milieu" et l'événement "rien n'est dit", et le boss penserait que le deuxième événement est celui exécuté. De plus, les événements peuvent modifier le monde (on dit alors qu'ils sont *ontiques*). Par exemple, jouer une carte est un événement ontique, car la carte n'est plus dans la main du joueur qui l'a jouée. La logique **DEL** étend alors la logique épistémique avec l'opérateur "après l'exécution du modèle d'événement $\mathcal{E}$ ... ".

Comme pour la logique épistémique, on peut définir les problèmes de vérification de modèles et de satisfiabilité pour **DEL**. Il est déjà prouvé que sans connaissance commune, le problème de vérification de modèles est PSPACE-complet et le problème de satisfiabilité est NEXPTIME-complet [AS13]. Toutefois, ce résultat n'est pas pour **DEL** en soi mais pour une variante légèrement plus riche où les modèles d'événement sont *multi-pointés*. Cela signifie que lorsque l'on applique un modèle d'événement, le choix de l'événement appliqué dans le monde courant est non déterministe. Par exemple le modèle d'événement "lancer un dé" aurait six événements pointés, un pour chaque valeur possible. En réalité, l'article [HP18] prouve que le problème de vérification de modèles est PSPACE-dur pour des modèles d'événement multi-pointés, S5, non-ontiques, et avec deux agents.

Pour plus d'information sur la logique épistémique dynamique, le lecteur peut se référer à [BMS98], qui est le livre de référence sur **DEL**. Comme cité aupravant, le livre [Dit+15] traite également de **DEL**.

Pour citer des variantes de **DEL**, il existe la variante probabiliste de PAL [Koo03], ainsi que la logique épistémique dynamique temporelle [RSY09] qui étend **DEL** avec des opérateurs temporels. De plus, il a été prouvé récemment que la logique de description de jeux III (GDL-III [Thi16]), une logique pour exprimer des jeux avec des aspects épistémiques, est aussi expressive que **DEL** [Eng+18].

Planification épistémique La planification est un domaine en intelligence artificielle où le problème de décision habituel est, donnés une situation initiale, des actions et un but, de déterminer si l'on peut trouver une séquence de ces actions pour atteindre le but.

La forme la plus connue de la planification est appelée la planification classique, en particulier le langage de résolution de problème de Stanford Research Institute (STRIPS [FN71]). Dans STRIPS, la situation initiale est représentée par une valuation, les actions ont seulement des préconditions/effets propositionnels et le but est une formule propositionnelle. Une vaste littérature existe à ce sujet, et un langage de spécification commun a été défini pour les planificateurs, appelé langage de description de domaine de planification (PDDL [McD+98]).

Ici, on s'intéresse à la *planification épistémique*. Cela signifie que les problèmes de planification doivent prendre en compte des aspects épistémiques. La planification épistémique est une notion vaste, et a une communauté active. Nous résumons ici le groupe de discussion du séminaire Dagstuhl sur les méthodes pour la planification épistémique ([Bar+17] Section 4.4) que j'ai coordonné. Nous laissons de côté les aspects mono-agents.

La planification épistémique considérée dans le document est celle basée sur **DEL**. Cela signifie que nous considérons en entrée un modèle de Kripke, des modèles d'événements pour les actions, et une formule épistémique pour le but. Malheureusement, la planification épistémique basée sur **DEL** est indécidable dans le cas général [AB13], y compris lorsqu'il n'y a pas de connaissance d'ordre supérieur dans les formules des modèles d'événement. [BA11]. Néanmoins, quand les modèles d'événement ont seulement des formules propositionnelles, la planification épistémique devient décidable [YWL13] (plus précisément non-élémentaire [DPS18]). En outre, lorsque les modèles d'événements sont propositionnels et non-ontiques, il a été prouvé que la planification épistémique est dans EXPSPACE [BJS15]. Nous prouvons en réalité dans [CMS16] que la planification épistémique est PSPACE-complète dans ce cas, et expliquerons ce résultat en détail dans le Chapitre 4.

Pour contourner le problème de complexité, d'autres approches ont été considérées pour la planification épistémique basée sur **DEL**. Elles utilisent principalement des compilations syntaxiques vers la planification classique, utilisant en général PDDL. Le point commun de ces approches est que l'expressivité est restreinte afin d'être applicable. Le travail de [Mui+15] considère par exemple des modèles d'événement avec un seul événement et des effets conditionnels, ce qui produit une traduction exponentielle vers la planification classique. Dans [KG15], les auteurs considèrent des annonces publiques, des affectations

publiques et des annonces semi-privées [8], et ont une réduction polynomiale à la planification classique [9] . Dans [Coo+16], les auteurs considèrent les variables $S_i p$ pour décrire les modèles.

D'autres approches existent pour résoudre la planification épistémique qui ne se basent pas sur **DEL**. Par exemple, il est possible de faire de la planification épistémique en étendant la logique temporelle alternante (ATL [AHK02]) avec des opérateurs épistémiques (AETL). ATL est elle même une extension de CTL [BK08] avec des modalités dynamiques pour exprimer l'existence de stratégies dans les jeux. Avec cette modélisation, on peut exprimer qu'un groupe d'agents a une stratégie pour atteindre une formule φ, ce qui correspond bien à la planification quand il n'y a pas d'adversaire. Cela permet d'exprimer également des propriétés plus fortes que juste la planification, comme l'existence de stratégies gagnantes par exemple. Le vérificateur MCMAS [LQR17] permet de résoudre AETL, mais a de fortes limitations : les modèles sont nécessairement S5, et la connaissance n'a pas de rappel parfait (puisque sinon cela devient indécidable [DT11]), ce qui signifie que les agents oublient une partie de l'historique des actions. En particulier, on ne peut pas exprimer la planification épistémique basée sur **DEL** ainsi, puisqu'elle a intrinsèquement un rappel parfait. De plus, le modèle doit être régulier (décrit par un automate à états finis), ce qui n'est pas toujours le cas pour **DEL**.

Il est également possible d'exprimer la planification épistémique en utilisant des techniques par automate pour utiliser la vérification de modèles de $\mathrm{CTL}^*\mathrm{K}_n$ [BMP15]. Cela est possible par exemple lorsque les modèles sont propositionnels [AMP14a; DPS18]. Malheureusement, cette approche ne va pas plus loin.

D'autres techniques existent mais sont trop éloignées par rapport à la planification épistémique considérée dans ce document. Le lecteur peut se référer à [Bar+17] pour plus d'information, puisque cet article sonde les techniques existantes pour la planification épistémique.

Annonces publiques arbitraires A cause de l'indécidabilité de la planification épistémique, nous étudions une classe plus restreinte de modèles d'événements. Des extensions de PAL existent pour exprimer l'existence d'une annonce publique. La logique correspondante est APAL [Bal+08], l'extension de PAL avec l'opérateur d'existence d'annonce (appelé *an-*

8. Annonce à un groupe d'agents d'une formule φ, les autres agents sachant que φ ou $\neg\varphi$ a été annoncée sans savoir la valeur.

9. Ce qui n'est pas surprenant, dans le Chapitre 4 nous prouvons que leur article s'exprime en planification épistémique avec des événements séparables.

nonce arbitraire). Une autre variante a été considérée : la logique d'annonce de groupe [Ågo+10] (GAL), où l'on ne quantifie pas sur toutes les annonces possibles mais seulement sur celles qui peuvent être faites par un certain groupe d'agents.

Malheureusement, les problèmes de satisfiabilité pour APAL et GAL sont indécidables [FD08; Ågo+10]. Ce n'est pas un problème si l'on veut voir APAL comme une autre façon de faire de la planification épistémique, comme le modèle de Kripke est donné en entrée.

Toutefois, APAL et GAL sont loin de la planification épistémique basée sur **DEL**, comme ils permettent seulement d'exprimer des annonces publiques, et ne permettent pas d'exprimer l'existence d'une séquence d'annonces.

Contribution

Dans ce document, nous étudions les problèmes de décision pour la logique épistémique dynamique mentionnés précédemment : la vérification de modèles et la satisfiabilité. Nous étudions également le problème de planification épistémique basé sur la logique épistémique dynamique, et des extensions de la logique épistémique avec quantification sur les annonces publiques. Enfin, nous définissons une représentation symbolique des modèles de Kripke et des modèles d'événements, et étudions son impact sur l'expressivité et sur la complexité algorithmique. Tout au long du document, nous donnons des exemples de jeux et de puzzles car ils sont intuitifs et mettent en jeu les problèmes que nous souhaitons discuter, comme par exemple l'interaction entre les agents. Nous décrivons maintenant la contribution de la thèse et le contenu de chaque chapitre de la partie en anglais.

- **Le Chapitre 1** définit la syntaxe et la sémantique de la logique épistémique dynamique (**DEL**). Nous définissons d'abord la logique épistémique, avec la notion de connaissance commune, et introduisons après les opérateurs dynamiques, définis par les modèles d'événement. Nous rappelons également quelques axiomes définissant les classes de modèles usuelles. En particulier, nous définissons S5 pour les modèles pour la connaissance et KD45 pour les modèles pour la croyance. Nous prouvons ensuite des résultats d'expressivité pour **DEL** : toute formule de **DEL** avec des modèles d'événement sans connaissance commune peut être exprimée en une formule équivalente où tous les modèles d'événements ont des formules sans connaissance d'ordre supérieur.

- **Le Chapitre 2** définit les problèmes de vérification de modèles et de satisfiabilité pour **DEL** avec connaissance commune (**DELCK**). Nous étendons les résultats de [AS13] pour **DELCK** en prouvant que le problème de vérification de modèles est Pspace-complet (tout comme **DEL**) et le problème de satisfiabilité est 2-Exptime-complet (alors qu'il est NExptime-complet pour **DEL**). Toutes les preuves utilisent les classes de complexité alternantes, définies par des algorithmes/machines de Turing enrichis par des choix existentiels et universels. En effet, comme rappelé en Annexe A, ces classes sont directement corrélées avec les classes de complexité classiques et permettent d'exprimer très intuitivement les complexités des problèmes de décision. Nous appliquons le résultat de vérification de modèles pour exprimer l'existence d'une stratégie uniforme dans le jeu du morpion fantôme, c'est à dire le morpion où les joueurs ne voient pas directement les coups de l'adversaire.

- **Le Chapitre 3** introduit les représentations symboliques des modèles de Kripke et des modèles d'événements, en utilisant des programmes de DL-PA. Nous montrons que nous ne perdons pas en expressivité en introduisant ces modèles symboliques et que la représentation est exponentiellement plus succincte dans les cas usuels. Ensuite, nous montrons que les complexités pour les problèmes de vérification de modèles et de satisfiabilité restent les mêmes que dans le cas non symbolique. Nous appliquons le résultat pour montrer que l'existence de stratégies uniformes dans le jeu de bridge peut se décider avec un algorithme Pspace.

- **Le Chapitre 4** est dédié à la planification épistémique. Nous prouvons en premier des résultats de décidabilité lorsque les modèles sont séparables, c'est à dire que les préconditions des événements sont disjointes. Dans ce cas, la planification épistémique est NP-complète si les événements sont non-ontiques et Pspace-complète sinon. Nous prouvons ensuite que la planification épistémique est Pspace-complète lorsque tous les modèles d'événement en entrée sont propositionnels (mais pas forcément séparables). Enfin, nous montrons que la planification épistémique est indécidable lorsque les modèles d'événement en entrée sont non-ontiques et ont des préconditions avec des formules de profondeur modale 2.

- **Le Chapitre 5** considère le problème de vérification de modèle pour APAL et GAL, introduisant une logique qui étend les deux, nommée la logique de protocoles d'annonces publiques (PAPL). Cette logique considère des *protocoles d'annonce* qui combinent des annonces publiques et des annonces arbitraires via des programmes

Référence	Partie concernée
[CS18] Complexity of Dynamic Epistemic Logic with Common Knowledge *Tristan Charrier and François Schwarzentruber, AiML2018*	Vérification de modèles et satisfiabilité des Chapitres 2 et 3
[CS17] A Succinct Language for Dynamic Epistemic Logic, *Tristan Charrier and François Schwarzentruber, AAMAS2017*	Définition des modèles symboliques, résultats d'expressivité et de concision du Chapitre 3
[CMS16] On the Impact of Modal Depth in Epistemic Planning, *Tristan Charrier and Bastien Maubert and François Schwarzentruber, IJCAI2016*	Chapitre 4
[CS15] Arbitrary Public Announcement Logic with Mental Programs *Tristan Charrier and François Schwarzentruber, AAMAS2015*	APAL symbolique, étendue dans le Chapitre 5
Soumission dans le Journal of Logic and Computation (pas encore publiée)	Chapitre 5
[CPS17] Model Checking Against Arbitrary Public Announcement Logic: A First-Order-Logic Prover Approach for the Existential Fragment, *Tristan Charrier and Sophie Pinchinat and François Schwarzentruber, DALI2017*	Chapitre 6

Table 1: Publications durant ma thèse.

similaires à ceux de PDL, ce qui permet d'étendre la planification épistémique dans le cas où toutes les actions sont des annonces publiques. Nous prouvons que le problème de vérification de modèles symboliques pour PAPL est $\mathrm{A}_{\mathrm{pol}}$Exptime-complet (voir Annexe A), qui est une classe entre Exptime et Expspace. Ce résultat de complexité est différent du cas non symbolique, qui est Pspace-complet.

- **Le Chapitre 6** décrit une réduction depuis le problème de vérification de modèles symboliques vers la vérification de modèles de MMSO : la logique du second-ordre monadique où tous les prédicats sont monadiques. Nous considérons le fragment existentiel de PAPL et montrons que nous pouvons implémenter la vérification de modèles sur ce fragment en utilisant la logique du premier ordre monadique (MFO). Nous détaillons ensuite des résultats expérimentaux en utilisant le solveur du premier ordre Iprover [Kor08].

La Table 1 résume les publications réalisées pendant ma thèse et les parties concernées du document.

Introduction

Context

The concept of *multi-agent system* is commonly accepted as a representation of many examples of real-life systems where autonomous entities, the *agents*, act according to their *knowledge* of the actual world in order to achieve individual or collective goals.

Let us consider examples of multi-agent systems.

- In multi-robot systems, the agents are robots that gather knowledge from the environment they perceive with their sensors and from the information obtained via communication channels [Lem+14]. They also need to take into account other agents, in particular human agents, to fulfill their objectives. For instance, Lemaignan et al. [Lem+10] give an example of knowledge management in multi-robot systems to take into account interactions with a human agent. Their system contains a notion of knowledge base the robot has about the environment. The field of distributed intelligence is dedicated precisely to this matter, having direct applications in multi-robot systems [Par08].

- In security, the agents are the attacker and the owner of a system. For instance in web security [PKS16], the owner wants the attacker to have the least possible knowledge. Indeed, if the attacker obtains information about the system, he may be able to compromise it, for instance in privilege escalation attacks. Also in cryptography [MOV96], the owner usually wants to ensure that the transmitted information is not decrypted by the attacker. Indeed, the information can be easily intercepted by the attacker, but if he does not own the encryption key, he cannot find the content of the message. We may also mention here attack detection, usually done with attack-defense trees [Kor+11], where for instance the attacker wants to access some hidden resource in a building (a document typically), and has to infiltrate the building without being seen. In this case, it is quite the contrary of the web security example. Indeed, the attacker is the one who wants to leak the least possible information, and the defender must gain knowledge of the attacker's whereabouts.

- In board games, like card games, Battleship, Clue, etc., the agents are the players. For instance in classical card games, the hand of a player is the only information he perceives at first. When the game goes on, the agents also acquire additional knowledge by other players' statements and actions. There exists a variety of board games that are relevant because they cover the spectrum of possible interactions between players. For instance in *Hanabi* [10], there is a heavy focus on mutual knowledge between agents, because a player only sees the hands of the other players, and the players must collaborate to collectively earn a maximum of points. *Tragedy Looper* [11] also features a notion of collaboration, but the players act together against an isolated player, the mastermind. Indeed, in this game, the players have an incomplete view about the scenario, and have to ensure that they do not let the mastermind trigger a defeat condition. They can play the scenario several times (in the game by a time travel mechanic) and collaborate to gain knowledge about the content of the scenario. The mastermind, on the other side, must ensure that the players always trigger a defeat condition and gain as little knowledge as possible. Finally, a lot of communication games exist, like the *werewolf game*, where all players have a hidden role and can be either a werewolf or a villager. The villagers must find the werewolves during the day phase and the werewolves must eat the villagers during the night phase. In this game, the knowledge is obtained by interactions between players but also by information roles, like the fortune teller who can see a role each turn.

- In video games, the agents are the players and the non-playable characters (NPC). The NPCs may have limited knowledge and capacities of reasoning, but it is relevant to consider the game as a multi-agent system when the NPCs interact with the players. For instance, in many massively multi-player online role-playing games (MMORPGs), the bosses have a threat mechanism that allows them to attack the player that represents the biggest threat. Also, there exist more sophisticated behaviors for NPCs, as the notorious boss Dark Link from the game *"The Legend of Zelda: Ocarina of Time"*[12]. His health scales to the player's and he has a AI that adapts to the way the player is attacking. For instance, if the player only does basic sword attacks, Dark Link will mainly block attacks.

10. Created by Antoine Bauza in 2010, edited by Cocktail Games and XII Singes.
11. Created by Bakafire in 2014, edited by Z-Man Games and BakaFire Party.
12. Edited by Nintendo.

In all these examples, the notion of knowledge is central, in particular nested knowledge i.e. knowledge agents have about the knowledge of other agents. Indeed, nested knowledge represents the way agents reason about the reasoning of other agents. It is central in many examples cited above, as in security where each agent needs to reason about the other agent's knowledge, or card games in order to design meaningful strategies. Furthermore, *dynamics* is also important, since agents communicate and act on a system. Some examples of dynamic actions are sending a message, opening a door, moving an object, playing a card, casting a spell in a video game, etc. Therefore, we aim at expressing the actions of agents in a multi-agent system.

Through the manuscript, we focus on the verification of properties involving knowledge in multi-agent systems. We need to express knowledge and dynamic actions, therefore we choose to use *dynamic epistemic logic* featuring both. It is an extension of *epistemic logic* with dynamic operators. We study the verification problem, called *model checking* and also the existence of a model, called the *satisfiability* problem, and their *symbolic* counterparts. We also go further by studying strategic variants of dynamic epistemic logic, like *epistemic planning* or *arbitrary actions*.

Notions introduced in the manuscript

Epistemic logic In this manuscript, we advocate for the use of logic to express properties in multi-agent systems. Indeed, logic provides a clear specification language, and is handy to define verification procedures, since its clear definition allows to write verification algorithms.

We consider *epistemic logic* as introduced by Hintikka [Hin65]. It extends propositional logic with knowledge operators, such as K_a for "Agent a knows that". All considered variables will be boolean, meaning that the only two values are true ($\top$) or false ($\bot$). We can express every variable on a finite domain, as bounded integers, by using several variables. We cannot express infinite domains though, as rational or real numbers.

The operator K_a is called a modality. Indeed, epistemic logic is a special type of modal logic (read for instance [BBW06] for more information about modal logic) where the only modalities are K_a. The main difference between propositional logic and modal logic is the interpretation of formulas. As a propositional formula (also called boolean formula) is evaluated over interpretations for variables (called *valuations*), modal formulas are evaluated over so-called Kripke models. Such models can be seen as graphs where each

node (called a *world*) has a valuation and represent a possible setting, for instance which cards players own in a card game. The edges represent which worlds agents imagine as possible, so that K_a is interpreted as "for all worlds imagined as possible by Agent a, ...". Multi-agent systems are then represented by Kripke models and the properties are formulas in epistemic logic.

For more information about epistemic logic, the reader may refer to any the books [Fag+95; MV04] that browse classical works about epistemic logic, or the survey [vV02]. More recently, the book [Dit+15] also covers epistemic logic and many of its extensions.

However epistemic logic has some limitations. Indeed, agents are supposed to be rational and to have a perfect knowledge of the Kripke model. Some extensions of epistemic logic exist to solve these issues. For instance, the paper [FH87] introduces an extension of epistemic logic where agents may be unaware of the existence of atomic propositions, and have limited capacities of reasoning, meaning that they cannot deduce all logical validities. Also, epistemic logic does not focus on *why* the agents know a fact. Justification logic [Art08] is a way of defining a justification of their knowledge when agents know a fact. To cite a more recent work, the paper [SW18] proposes a logic where agents' names are not known by all agents, making a distinction between "knowing that a committed a crime and not knowing his name" and "knowing that an agent named a committed a crime without knowing who he is".

In the manuscript, we do not consider all these extensions to keep a simple logic and stick to the usual definition of epistemic logic.

Common knowledge Common knowledge has been first defined formally in [Aum76]. A formula is common knowledge if it is true in every possible world. It can be defined as an infinite conjunction of formulas of the form "everyone knows that everyone knows that ..." which is an equivalent definition.

The notion of common knowledge in epistemic logic is very close to the AG operator in CTL (see [BK08] for more information about temporal logics). Both operators can be seen as fixpoint operators as seen in the μ-calculus [Pra81].

In the manuscript, all the chapters, except Chapter 4, allow the common knowledge operator in the logic.

Model checking and satisfiability When a logic is defined, it is common to study its computational properties. Here, we take a similar approach to [HV91] by defining the

verification problem called the *model checking* problem. It takes as input a Kripke model and a formula and verifies whether the formula is true in the current model. We prove in Appendix C that the model checking problem for epistemic logic with common knowledge is in P, which is very similar to the proof for the model checking of CTL. For a reminder on computational complexity, the reader may refer to Appendix A.

Another classical problem is to check the *satisfiability* of a formula, meaning finding a Kripke model satisfying a formula. In fact, the satisfiability problem is dual to the theorem proving problem, since a formula φ is valid if and only if $\neg\varphi$ is not satisfiable. For epistemic logic, the satisfiability without common knowledge is known to be PSPACE-complete, for at least two agents and for S5 models [13] [HM92].

It is also common to define the *axiomatization* of the logic. It consists in a set of axioms and rules that illustrate how to construct provable formulas (see [Fag+95] for epistemic logic). We establish whether it is sound (any provable formula is valid) and complete (any valid formula is provable). When a sound and complete axiomatization is provided, it may lead to a *theorem proving* algorithm, that checks the validity of the formula using the axiomatization. The main difference with model checking is that we prove the validity of a formula instead of verifying it in a given model. Therefore, if we would like to use theorem proving to verify a formula φ in a model, we would need to characterize the model by a formula $\varphi_{\mathcal{M}}$ and prove the validity of $\varphi_{\mathcal{M}} \to \varphi$. Theorem proving is dual to satisfiability, so in fact model checking is a simpler problem. That is why we do not use theorem proving instead of model checking because it would be a sledgehammer.

Symbolic models When we want to use model checking and satisfiability approaches in practice, we have to deal with combinatorial explosion in the size of Kripke models. For instance in card games, when there is k players and n cards, the number of atomic propositions is usually $k \times n$. If we consider a classical tarot card game with 5 players, the number of possible worlds would be approximately 6×10^{48}, which is the same order of magnitude than the number of atoms in Earth. It thus seems rather unrealistic to define the input Kripke models explicitly.

For this reason, we propose to use a symbolic representation of the Kripke models, that avoids defining models completely by hand. Such representations already exist for other formalisms, such as boolean circuits for searching for Hamiltonian paths in graphs [Pap03]. Usually, a symbolic representation of models consists in describing the input model with

13. I.e. edges in Kripke models are equivalence relations.

boolean formulas instead of describing it explicitly. For example, the paper [Bur+90] gives a symbolic representation of models for temporal logics. For symbolic representations, it is possible to use data structures known as reduced and ordered binary decision diagrams [DB98] (ROBDDs) to implement symbolic representations.

It may seem surprising to consider model checking procedures with symbolic inputs, as it looks very similar to theorem proving as mentioned before. Yet, the difference is that the description of the model (given with ROBDDs, etc.) is not written in the same language as the formula (e.g. in epistemic logic), so it would not make sense to try to prove $\varphi_{\mathcal{M}} \to \varphi$ with $\varphi_{\mathcal{M}}$ the symbolic representation of the Kripke model $\mathcal{M}$.

For epistemic logic, many symbolic approaches already exist, described in more depth in Chapter 3. The first approach defines worlds as valuations and agents observe a set of atomic propositions [HTW11; HIW12; Ågo+13]. In this case, only the set of worlds is needed, so a Kripke model is only the data of the set of worlds, and for each agent the set of propositions he observes, the edges being inferred from the observation set. Also the paper [GGS15] extends this idea sightly by considering agents in a two-dimensional space that can only see inside a cone in front of them. Finally, [HLM15] considers a representation of the epistemic relations with "seeing"-variables of the form $S_a p$ for "Agent a perceives the value of p" that may be of higher-order like $S_a S_b p$ for "Agent a perceives the value of $S_b p$".

Here, we use a symbolic approach using programs of Dynamic Logic with Propositional Assignments (DL-PA [BHT13]), that are particular programs of Propositional Dynamic Logic (PDL [FL79]) where atomic programs are propositional assignments. While not directly boolean formulas, they can be translated into classical boolean formulas so that BDD techniques may be used [14]. The advantage of this approach is that it has the usual properties of symbolic models: they can express any model, they are succinct, and they allow for a simple representation of usual examples of epistemic logic.

Dynamics in epistemic logic As highlighted in the epistemic logic paragraph, epistemic logic is not sufficient to express the dynamics in models. We consider here the so-called *dynamic epistemic logic*. It is an umbrella term for all extensions of epistemic logic with dynamic operators. The extensions of epistemic logic with dynamics are strongly inspired from dynamic logic [FL79].

14. Actually, Chapter 6 gives the reduction to first-order formulas, which gives a way to define BDDs afterwards.

Originally, the first occurrence of dynamic operators for epistemic logic relied on public announcements (the logic is called PAL for Public Announcement Logic) [Pla07] [15]. It corresponds to broadcasting true properties to all agents. For instance, in a video game, if a player says "the boss is in the middle watch out!", all players will now know that the boss is in the center. In that case, if the boss is also considered as an agent, we would consider that the boss also knows that players know his position, so he may act accordingly by hiding for instance.

The dynamic epistemic logic as referred in the rest of the manuscript (called **DEL**) extends PAL by considering richer types of events (private announcements, etc.), the paper [BMS98] being the milestone. An event is now represented as a graph, alike Kripke models, and is called *event model*. In this model, each node is a possible event and the edges, labeled by agents, represent events agents imagine as possible. For instance, in the broadcast example, the event model for "There is an announcement not heard by the boss" would contain two events: the event "Announcing that the boss is in the middle" and the event "Nothing is said", and the boss would think that the second event is the actual one happening. Furthermore, events may modify the world, and usually are called *ontic* events when they do. For instance, playing a card modifies the world, because the card is no longer in the player's hand. The logic **DEL** thus extends epistemic logic with the operator "after applying the event model $\mathcal{E}$...".

As for epistemic logic, we can define the model checking and satisfiability problems for **DEL** (so without common knowledge). It has been proven that without common knowledge the model checking problem is PSPACE-complete and the satisfiability problem is NEXPTIME-complete [AS13]. Notice that the result in [AS13] does not hold for dynamic epistemic logic per se, but for a slightly richer version where event models are *multi-pointed*. It simply means that when applying an event model, the choice of the actual event occurring is non-deterministic. For instance, the event model for "role a dice" would have 6 events, one for each value, and all would be pointed. Actually, in [HP18], it is proven that the model checking problem is PSPACE-hard for non-ontic, multi-pointed S5 event models with only one agent, or non-ontic, single-pointed S5 event models with two agents.

For more information about dynamic epistemic logic, the reader may refer to [BMS98], which is the reference book on **DEL**. As cited before, the book [Dit+15] also covers **DEL** to some extent.

To cite variants of **DEL**, we may cite probabilistic **DEL** [Koo03] which extends PAL

15. The cited paper is 2007 re-edition of a 1989 paper to our knowledge.

with probabilities and dynamic epistemic temporal logic [RSY09] that extends **DEL** with temporal operators. Also, it has been proven recently that Game Description Logic III (GDL-III [Thi16]), a logic for expressing games with epistemic aspects, is as expressive as **DEL** [Eng+18].

Epistemic planning Planning is a field in Artificial Intelligence where the usual decision problem is, given an initial situation and actions, find a sequence of actions to reach the goal.

The well known form of planning is called classical planning, in particular the language of the Stanford Research Institute problem solver (STRIPS [FN71]), where the initial situation is represented with a valuation, the actions only feature propositional preconditions/effects and the goal is also propositional. A vast literature exists on the subject, leading to a specification language called Planning Domain Description Language (PDDL [McD+98]), that aims at defining a common language for designers and users of planners.

Here, we are interested in *epistemic planning*. It means that the planning problems must consider epistemic aspects. Epistemic planning is a broad notion, and has an active community. We summarize here the group discussion at Dagstuhl about methods for epistemic planning ([Bar+17] Section 4.4) that I coordinated. We leave apart the single-agent approaches.

The way of doing epistemic planning we consider in the manuscript is epistemic planning based on **DEL**. It means that we consider as input a Kripke model, event models for actions, and an epistemic formula as a goal. Unfortunately, epistemic planning based on **DEL** is undecidable [AB13], even when there is no nested knowledge in event models' formulas [BA11]. Yet, when event models only feature propositional formulas, the epistemic planning problem is decidable [YWL13], more precisely non-elementary [DPS18]. Furthermore, when all event models feature non-ontic and propositional events, the epistemic planning problem has been proven to be in Expspace [BJS15]. Actually, we prove in [CMS16] that it is Pspace-complete. We will explain this result in detail in Chapter 4.

To circumvent the complexity problem, some other approaches have been considered to do epistemic planning based on **DEL**. They mainly revolve around syntactic compilation to classical planning, usually using PDDL. The common point between the approaches is the restricted expressivity to be applicable with classical planners. The work of [Mui+15]

considers event models with one event with additional conditional effects, that leads to an exponential reduction to classical planning. In [KG15], they consider public announcements, public assignments and semi-private announcements[16], and have a polynomial reduction to classical planning[17]. In [Coo+16], the authors consider the "seeing"-variables to describe the models.

Other approaches exist to solve epistemic planning that do not rely on **DEL**. For instance, it is possible to do epistemic planning by extending Alternating Temporal Logic (ATL) [AHK02] with knowledge (AETL). ATL is itself an extension of computation-tree logic [BK08] with dynamic modalities to express the existence of strategies. With this model, we can express that a group of agents have a strategy to reach a formula φ, which corresponds to planning. This allows to express stronger properties than just planning, such as winning strategies for instance. The model checker MCMAS [LQR17] can solve AETL, but has strong limitations: only S5 models are allowed and the knowledge has no perfect recall (it is undecidable with perfect recall [DT11]), meaning that each agent forgets a part of the actions history. In particular, this setting does not express epistemic planning based on **DEL** since it is intrinsically perfect recall. Furthermore, the model must be regular (described by a finite-state automaton), which is not the case for **DEL** because the epistemic planning problem would be decidable otherwise.

It is also possible to express epistemic planning using automata techniques to solve the model checking of $\mathrm{CTL}^*\mathrm{K}_n$ [BMP15], when it is possible to describe with automata the infinite structure generated by the event models. It is possible, for instance, when event models are propositional [AMP14a; DPS18]. Unfortunately, this approach does not work further.

Other techniques exist but are further from epistemic planning as considered in the manuscript. The reader can refer to [Bar+17] for more information, since it surveys currently existing techniques for epistemic planning.

Arbitrary public announcements Because epistemic planning is undecidable, we study a more restrictive class of event models. Some extensions of PAL already exist to express the existence of an announcement. The corresponding logic is called APAL [Bal+08], the extension of PAL with the existential operator (called *arbitrary announce-*

16. Announcement to a group of agents of the value of a formula φ, the other agents being uncertain of the announced value.

17. Which is not surprising, in Chapter 4 we will see that their paper falls into epistemic planning with separable events.

ment). Another variant was considered: Group Announcement Logic [Ågo+10] (GAL), where we do not quantify on all announcements but only on announcements doable by a certain group of agents.

Unfortunately, the satisfiability problems of both APAL and GAL are undecidable [FD08; Ågo+10]. It is not a problem if we want to see APAL as a different way of studying epistemic planning, since the Kripke model is given in input.

Yet, by themselves, APAL and GAL are pretty far from epistemic planning. They just express the existence of public announcements once, and do not consider a sequence of announcements.

Contribution

In this document, we study the mentioned decision problems for dynamic epistemic logic, namely model checking and satisfiability. We also study the epistemic planning problem based on dynamic epistemic logic, and extensions of epistemic logic that quantify over public announcements. Finally, we define a symbolic representation of Kripke models and event models and study its impact on expressivity and algorithmic complexity. Through the manuscript, we give games and puzzles as examples since they are intuitive and feature the matters we wish to discuss, as interaction of knowledge between agents. We now detail the content of each chapter.

- **Chapter 1** defines the syntax and the semantics of Dynamic Epistemic Logic (**DEL**). We first define epistemic logic, with the notion of common knowledge, and then introduce dynamic operators, defined with event models. We also recall some axioms leading to classical classes of models. In particular, we define S5 models to talk about knowledge and KD45 to talk about beliefs. We then prove expressivity results for **DEL**, namely that for any formula of **DEL** where event models do not contain common knowledge, we can define an equivalent formula of **DEL** where event model contain no nested knowledge.

- **Chapter 2** defines the model checking and satisfiability problems for **DEL** with common knowledge (**DELCK**). We extend the results of [AS13] for **DEL** by proving that the model checking problem against **DELCK** is Pspace-complete (as **DEL**) and the satisfiability problem is 2-Exptime-complete (compared to NExptime-complete for **DEL**). All proofs strongly rely on alternating complexity classes, de-

fined by algorithms/Turing machines enriched by existential and universal choices. Indeed, as explained in Appendix A, such classes are directly correlated with standard complexity classes and they provide a very intuitive way of establishing complexities of decision problems. We apply our result on the model checking to express the existence of a uniform strategy in the game blind tic tac toe, which is tic tac toe where players do not see the moves of their opponents.

- **Chapter 3** introduces the symbolic representation of Kripke and event models using DL-PA-relations. We show that we do not lose any expressivity by introducing the symbolic models and that the representation is exponentially more succinct. Then, we show that the model checking and satisfiability problems for **DELCK** with symbolic models have the same complexities than the non-symbolic case, Pspace-complete and 2-Exptime-complete respectively. We apply the result for symbolic model checking to express the existence of a uniform strategy in the bridge card game, which shows that with symbolic inputs, expressing the existence of winning uniform strategies in classical card games is in Pspace.

- **Chapter 4** is dedicated to epistemic planning. We first prove decidability results when event models are separable, i.e. events have disjoint conditions. In that case, the epistemic planning is NP-complete when all events are non-ontic, and Pspace-complete when events are ontic. We next prove that epistemic planning is also Pspace-complete when event models are propositional but not necessarily separable and all events are non-ontic. Finally, we show that undecidability holds when event models with modal depth 2 are allowed and non-ontic events.

- **Chapter 5** considers symbolic model checking of APAL and GAL, introducing a unifying logic called Public Announcement Protocol Logic (PAPL). This logic features *announcement protocols* that combine public announcements and arbitrary announcements in a PDL fashion, which allows to extend epistemic planning in the case where events are announcements. We show that the symbolic model checking for PAPL is $\mathrm{A_{pol}}$Exptime-complete (see Appendix A), which is a class between Exptime and Expspace. The complexity result is noticeably different from the Pspace-completeness of the model checking of PAPL in the non-symbolic case.

- **Chapter 6** describes a reduction from the symbolic model checking of APAL into the model checking of MMSO: second-order logic where all predicates occurring in

Reference	Concerned part
[CS18] Complexity of Dynamic Epistemic Logic with Common Knowledge *Tristan Charrier and François Schwarzentruber, AiML2018*	Model checking and satisfiability of Chapters 2 and 3
[CS17] A Succinct Language for Dynamic Epistemic Logic, *Tristan Charrier and François Schwarzentruber, AAMAS2017*	Definition of symbolic **DEL**, expressivity and succinctness results in Chapter 3
[CMS16] On the Impact of Modal Depth in Epistemic Planning, *Tristan Charrier and Bastien Maubert and François Schwarzentruber, IJCAI2016*	Chapter 4
[CS15] Arbitrary Public Announcement Logic with Mental Programs *Tristan Charrier and François Schwarzentruber, AAMAS2015*	Symbolic APAL, extended in Chapter 5
Submission in the Journal of Logic and Computation (not yet published)	Chapter 5
[CPS17] Model Checking Against Arbitrary Public Announcement Logic: A First-Order-Logic Prover Approach for the Existential Fragment, *Tristan Charrier and Sophie Pinchinat and François Schwarzentruber, DALI2017*	Chapter 6

Table 2: Publications during my PhD.

the formulas considered are monadic, as opposed to MSO where only the predicates under quantification must be monadic. We exhibit the existential fragment of PAPL and show how to implement its symbolic model checking with the satisfiability of Monadic First-Order Logic (MFO). We then give some experimental results using the first-order solver Iprover [Kor08].

Table 2 summarizes the publications during my PhD and the concerned parts in the manuscript.

Chapter 1

Dynamic Epistemic Logic

In this chapter, we describe dynamic epistemic logic (**DEL** in short). The chapter is constructed as follows.

- First, we give the definition of **DEL** to model interactions of knowledge between agents in a multi-agent system and complex dynamic actions.
- Second, we show that in actions, it is always possible to gather the knowledge complexity either in the conditions or in the effects. We go even further by showing also that actions with no nesting of knowledge are sufficient to capture the whole expressive power of **DEL**.
- Finally we conclude.

The expressivity results are the contribution of this chapter, and have not been published yet.

1.1 Background on DEL

We define epistemic logic (without dynamic actions) and then introduce dynamic epistemic logic.

1.1.1 Epistemic logic

Syntax

We consider a countable set of atomic propositions AP and a finite set of agents Ag. The syntax of the language of epistemic logic (**ELCK**), denoted by $\mathcal{L}_{\mathbf{ELCK}}$, is defined as follows.

Definition 1 (Syntax of epistemic logic)**.** $\mathcal{L}_{\mathbf{ELCK}}$ *is defined by the following grammar.*

Operator	**Intuitive meaning**
$\top$	True.
p	Atomic proposition p is true.
$\neg\varphi$	Formula φ is false.
$\varphi \vee \psi$	Formula φ or formula ψ is true.
$K_a\varphi$	Agent a knows that φ is true.
$C_G\varphi$	φ is common knowledge among the agents of G.

Table 1.1: Intuitive meaning of the operators of $\mathcal{L}_{\mathbf{ELCK}}$.

Abbreviation	**Meaning**
$\bot$	$\neg\top$
$(\varphi_1 \wedge \varphi_2)$	$\neg(\neg\varphi_1 \vee \neg\varphi_2)$
$\varphi_1 \to \varphi_2$	$\neg\varphi_1 \vee \varphi_2$
$\varphi_1 \leftrightarrow \varphi_2$	$(\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$
$\hat{K}_a\varphi$	$\neg K_a \neg\varphi$
$\hat{C}_G\varphi$	$\neg C_G \neg\varphi$

Table 1.2: Abbreviations for **ELCK**.

$$\varphi \ ::= \ \top \mid p \mid \neg\varphi \mid (\varphi \vee \varphi) \mid K_a\varphi \mid C_G\varphi$$

with $p \in AP$, $a \in Ag$, $G \subseteq Ag$.

Intuitive meaning of the operators of $\mathcal{L}_{\mathbf{ELCK}}$ are given in Table 1.1. Other operators appearing afterwards are defined in Table 1.2, either with the De Morgan's law or by duality.

Epistemic logic without the common knowledge operator (C_G) is noted **EL** and its language $\mathcal{L}_{\mathbf{EL}}$.

Example 1. *Formula "$K_a(K_bp \vee K_cp)$" reads "Agent a knows that either Agent b knows p or Agent c knows p".*

Semantics

We know define the semantics of **ELCK**. A multi-agent system is formalized by a *Kripke model*, defined as follows.

Definition 2. *A* Kripke model $\mathcal{M} = (W, (R_a)_{a\in Ag}, V)$ *is defined by a non-empty set W of epistemic worlds, epistemic relations $R_a \subseteq W \times W$ for all agents $a \in Ag$, and a valuation function $V : W \to 2^{AP}$.*

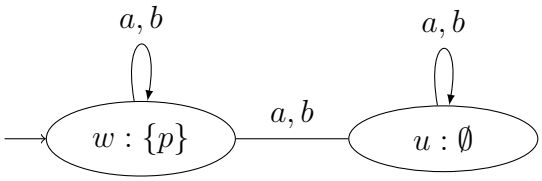


Figure 1.1: A simple Kripke model.

We write $R_a^{\mathcal{M}}$ for the epistemic relation for agent a in model $\mathcal{M}$. We may also write $w \xrightarrow{a}_{\mathcal{M}} u$ for $(w,u) \in R_a^{\mathcal{M}}$ or $w \xrightarrow{a} u$ when $\mathcal{M}$ is clear. Later on, if we need to define a Kripke model but do not need to use W, R_a or V outside of the definition, we may write the tuple $(W, (R_a)_{a\in Ag}, V)$ even if the names are already used.

A pair $(\mathcal{M}, w)$ is called a *pointed Kripke model*, where w denotes the real world, the other worlds being only imagined as possible by the agents.

Intuitively, a world represents a possible configuration, and the Kripke model represents all possible configurations agents imagine as possible. The epistemic relations R_a represent the agents' capacities of reasoning. We have $w \xrightarrow{a} u$ if Agent a thinks that u may be the real world when the real world is actually w.

Example 2. *Figure 1.1 shows a simple Kripke model* $\mathcal{M} = (W, R_a, R_b, V)$ *with* $AP = \{p\}$, $Ag = \{a, b\}$ *and:*

- $W = \{w, u\}$;
- $R_a = R_b = W \times W$;
- $V(w) = \{p\}$; $V(u) = \emptyset$.

Worlds in Kripke models are drawn with ellipses. Inside each ellipse, the label corresponds to the name of the world followed by a colon and its valuation. The edges represent the relations R_a and R_b: there is an a-edge from w to u if and only if $(w,u) \in R_a$. If an edge goes both ways, we do not draw the arrow but just a line instead (as between w and u in Figure 1.1). The pointed world is represented by an incoming edge coming from no world. For instance Example 1.1 represents the pointed Kripke model $(\mathcal{M}, w)$.

When each valuation is unique in the Kripke model, we may name a world by its valuation. For instance in the previous example, world w would be world $\{p\}$ and world u would be world $\emptyset$. In this case, we do not include the name of the world in the ellipse as in Figure 1.2.

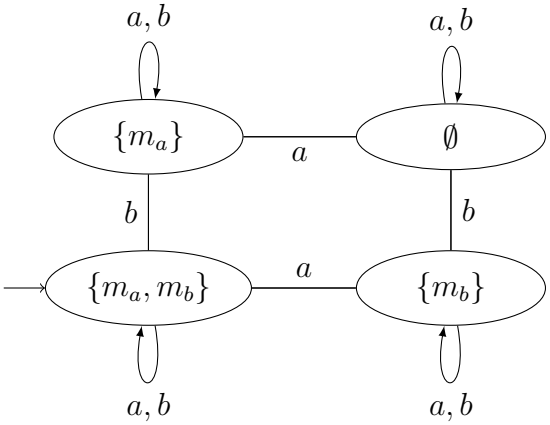


Figure 1.2: The Kripke model for the muddy children puzzle for two children a, b.

Example 3 (Muddy children [McC87; DK15])**.** *We now model the famous muddy children puzzle. In this puzzle, n children are playing in a garden, and some of them become muddy. They are supposed clever and honest. Their father comes and say "At least one of you is muddy". They do not see whether they are muddy or not, but see the other children muddiness. The father then asks several times "Does any one of you know whether he is muddy?". The puzzle consists in determining how many times the father needs to ask the question before any of the children answers "I know that I am muddy" or "I know that I am not muddy".*

We leave aside the answer for now and only model the initial situation. The initial situation is modeled by the Kripke model of Figure 1.2 in the case of 2 agents, and is modeled in general by the Kripke model $\mathcal{M} = (W, \{R_a\}_{a \in Ag}, V)$ *with Ag the set of children,* $AP = \{m_a, a \in Ag\}$ *and:*

- $W = 2^{AP}$
- $V(w) = w$.
- $R_a = \{(w, u) \mid w \setminus \{m_a\} = u \setminus \{m_a\}\}$;

We now give the semantics of **ELCK** on Kripke models.

Definition 3 (Semantics of **ELCK**)**.** *The semantics of* **ELCK** *is defined on a pointed Kripke model. We note* $\mathcal{M}, w \models \varphi$ *for "Formula* φ *is true in the pointed Kripke model* $(\mathcal{M}, w)$*". It is defined by induction on* φ *as follows.*

- $\mathcal{M}, w \models \top$;
- $\mathcal{M}, w \models p$ *if* $p \in V(w)$;

- $\mathcal{M}, w \models \neg\varphi$ *if* $\mathcal{M}, w \not\models \varphi$;
- $\mathcal{M}, w \models (\varphi \vee \psi)$ *if* $\mathcal{M}, w \models \varphi$ *or* $\mathcal{M}, w \models \psi$;
- $\mathcal{M}, w \models K_a\varphi$ *if for all* $u \in W, (w, u) \in R_a$ *implies* $\mathcal{M}, u \models \varphi$;
- $\mathcal{M}, w \models C_G\varphi$ *if for all* $u \in W, (w, u) \in R_G$ *implies* $\mathcal{M}, u \models \varphi$ *where* R_G *is the transitive closure of* $\bigcup_{a\in G} R_a$.

Example 4. *We consider again the Kripke model of Figure 1.1. Let* $\varphi = C_{\{a,b\}}\neg K_a p$. *If we want to check whether* $\mathcal{M}, w \models \varphi$, *we first compute the set of worlds satisfying* p. *Here, there is only* w. *For* $K_a p$, *we verify that all worlds* a*-accessible satisfy* p, *which is the case for no world here. Then both* w *and* u *satisfy* $\neg K_a p$. *Finally for* φ, *we must check that all worlds that are accessible by a combination of* a *and* b *satisfy* $\neg K_a p$, *so both* w *and* u *satisfy* φ.

Axioms and Types of Kripke models

In the rest of the manuscript, we will refer to types of Kripke models, as S5, KD45, K. They correspond to Kripke models where a certain set of axioms is true. Such axioms are summarized in Table 1.3. The models we are going to work on are described in the following definition.

Definition 4 (K, KD45 and S5)**.** *The classes of models K, KD45 and S5 are defined in the following way.*

- *K is the set of Kripke models satisfying the axiom K. It corresponds to all Kripke models.*
- *KD45 is the set of Kripke models satisfying the axioms K, D, 4 and 5.*
- *S5 is the set of Kripke models satisfying the axioms K, T, 4 and 5.*

Other axioms and types of Kripke models exist, but we omit them here since they are not used later.

The usual class of Kripke models considered in the literature are S5 models. They correspond to models where the epistemic relations are equivalence relations and thus represent knowledge. On the contrary, KD45 models usually represent belief, because in particular the current world is not always considered as possible by the agents.

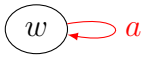
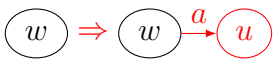
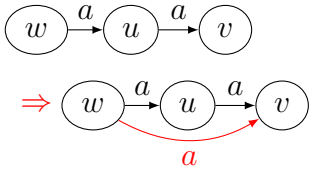
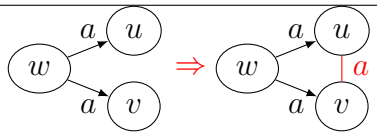
	Axiom	Condition on Kripke models	Drawing of the condition
K	$K_a(\varphi_1 \to \varphi_2) \to (K_a\varphi_1 \to K_a\varphi_2)$	*None*	
T	$K_a\varphi \to \varphi$	*Reflexivity*: for all w, $(w,w) \in R_a$	
D	$K_a\varphi \to \hat{K}_a\varphi$	*Serial*: for all w, there exists u such that $(w,u) \in R_a$	
4	$K_a\varphi \to K_aK_a\varphi$	*Transitivity*: for all w,u,v, $(w,u) \in R_a$ and $(u,v) \in R_a$ implies $(w,v) \in R_a$	
5	$\hat{K}_a\varphi \to K_a\hat{K}_a\varphi$	*Euclidean*: for all w,u,v, $(w,u) \in R_a$ and $(w,v) \in R_a$ implies $(u,v) \in R_a$	

Table 1.3: Axioms for epistemic logic

Example 5. *Both Kripke models of Figures 1.1 and 1.2 are S5 models.*

For more information about axioms and epistemic logic, the reader may refer for instance to [MV04].

1.1.2 Dynamic Epistemic Logic

Dynamic Epistemic Logic (**DELCK**), initially proposed in [BMS98], extends epistemic logic by adding a new operator. The book [DHK07] published later on is an extensive reference on **DELCK**, so the reader may feel free to browse this book for more information about **DELCK**. For a more recent reference with additional extensions of epistemic logic, the reader may refer to [Dit+15].

Its language $\mathcal{L}_{\mathbf{DELCK}}$ is defined in the following way.

Definition 5 (Syntax of DEL)**.** $\mathcal{L}_{\mathbf{DELCK}}$ *is defined by the following grammar.*

$$\varphi \quad ::= \quad \top \mid p \mid \neg\varphi \mid (\varphi \vee \varphi) \mid K_a\varphi \mid C_G\varphi \mid \langle \mathcal{E}, \mathsf{E}_0 \rangle \varphi$$

with $p \in AP$, $a \in Ag$, $G \subseteq Ag$.

The language without the C_G operator is written $\mathscr{L}_{\mathbf{DEL}}$ and the logic **DEL**. The operator $\langle \mathcal{E}, \mathsf{E}_0 \rangle \varphi$ is read "After the execution of the multi-pointed event model $(\mathcal{E}, \mathsf{E}_0)$, φ

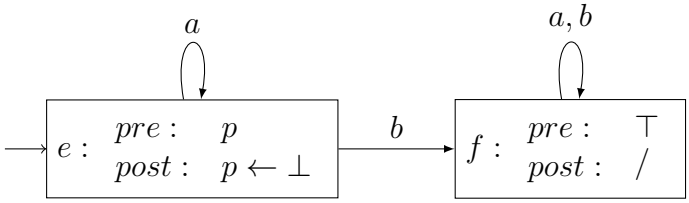


Figure 1.3: Example of an event model

is true". We define its dual $[\mathcal{E}, \mathsf{E}_0]\varphi = \neg\langle\mathcal{E}, \mathsf{E}_0\rangle\neg\varphi$. Event models are a semantic object that appear in the syntax of **DELCK**. The main reason is that **DELCK** aims at reasoning about knowledge when the actions are fixed. Therefore, we need the event models to be fixed in the formula. It is possible to only consider the $\langle\mathcal{E}, \mathsf{E}_0\rangle$ operator as a symbol that is interpreted in the semantics, but it is not done this way in the literature, and changes decision problems later on, in particular the satisfiability problem where the event models are fixed in the input. Event models are defined as follows.

Definition 6 (Event model)**.** *An* event model $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$ *is defined by a non-empty set of* events E*, epistemic relations* $(R_a^{\mathcal{E}})_{a\in Ag} \subseteq \mathsf{E}\times\mathsf{E}$*, a precondition function* $\text{pre} : \mathsf{E} \to \mathcal{L}_{\mathbf{ELCK}}$ *and a postcondition function* $\text{post} : \mathsf{E}\times AP \to \mathcal{L}_{\mathbf{ELCK}}$*.*

An event model is similar to a Kripke model but epistemic worlds are now replaced by events labeled by a precondition and a postcondition. The precondition $\text{pre}(e)$ of an event e is the necessary condition to execute the event. An event e is said *executable* in a world w if and only if its precondition $\text{pre}(e)$ holds in w. The postcondition $\text{post}(e, p)$ represents the effect the event has on atomic proposition p, meaning whether p is now $\top$ or $\bot$. Events can be seen as PDDL [McD+98] actions where formulas in preconditions and postconditions/effects can be any formula of **ELCK** instead of propositional. As for Kripke models, if outside definitions we do not use the content of the tuple, we may write event models with the tuple $(\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$ even if the names are already used. We may write $e \in \mathcal{E}$ for $e \in \mathsf{E}$.

A pair $(\mathcal{E}, e)$ with $e \in \mathcal{E}$ is called a *pointed event model*, where e represents the actual event. A pair $(\mathcal{E}, \mathsf{E}_0)$ with $\mathsf{E}_0 \subseteq \mathsf{E}$ is called a *multi-pointed event model*, where E_0 represents the set of possible actual events. Pointed event models correspond to deterministic actions and multi-pointed event models correspond to non-deterministic actions. We may confuse $(\mathcal{E}, e)$ and $(\mathcal{E}, \{e\})$ as they are equivalent.

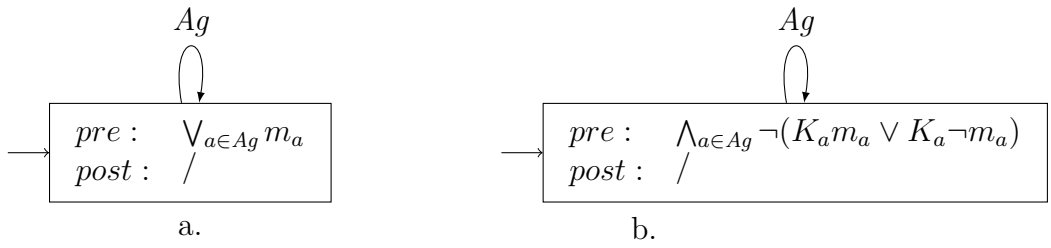


Figure 1.4: Event models in the muddy children puzzle

Example 6. *Figure 1.3 shows an example of a pointed event model* $(\mathcal{E}, e)$ *with* $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$*,* $Ag = \{a, b\}$*,* $AP = \{p\}$ *and:*

- $\mathsf{E} = \{e, f\}$*;*
- $R_a = \{(e, e), (f, f)\}$*;* $R_b = \{(e, f), (f, f)\}$*;*
- $\text{pre}(e) = p$*;* $\text{pre}(f) = \top$*;*
- $\text{post}(e, p) = \bot$*;* $\text{post}(f, p) = p$

It has two events e *and* f*. Event* e *is executable only when* p *is true and assigns* p *to false. Event* f *is always executable and does not perform any assignment. Agent* a *has exact knowledge of the event and agent* b *always thinks that the actual event is* f*.*

By convention, we draw the events of event models with rectangles (whereas worlds in Kripke models are ellipses). In each rectangle, the name of the event is written followed by the precondition and the postcondition. For the postconditions, all atomic propositions not mentioned in the figure are assigned to their previous value. For instance, $\text{post}(f, p) = p$. In this case, we say that the postcondition of f is *trivial* for p. If the postcondition of an event f is trivial for all atoms of AP, we simply say that the postcondition of f is trivial and represent the postcondition with a bar (/) in the figures.

Example 7. *In the muddy children example, two types of event models are relevant.*

- *The announcement by the father, represented in Figure 1.4.a, is an event model with one event whose precondition is true if and only if at least one of the children is muddy and whose postcondition is trivial.*

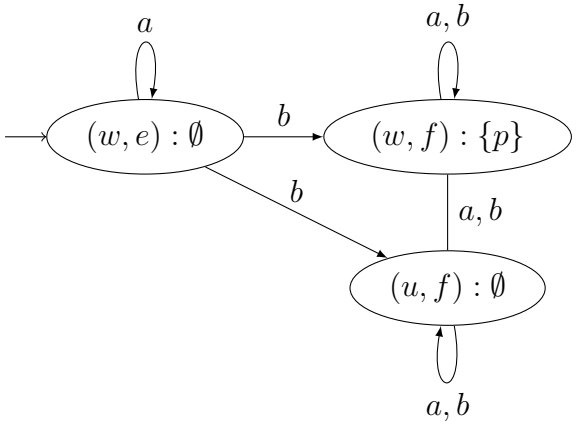


Figure 1.5: Example of the product of the Kripke model of Figure 1.1 page 37 and the event model of the Figure 1.3 page 41.

- *The fact that children do not speak after the question of the father is also modeled by an announcement, represented in Figure 1.4.b. The event model also has only one event with a trivial postcondition and the precondition is no child knows whether he is muddy.*

Both event models are *public announcements*, that will be central in Chapters 5 and 6. They are event models with only one event with a trivial postcondition. If the postcondition is not trivial, we usually called such an event model a *public action*.

The effect an event model has on a Kripke model is characterized by the synchronous product operation, also called product update.

Definition 7. *Let* $\mathcal{M} = (W, (R_a)_{a\in Ag}, V)$ *be a Kripke model. Let* $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *be an event model. The* product *of* $\mathcal{M}$ *and* $\mathcal{E}$ *is* $\mathcal{M} \otimes \mathcal{E} = (W', (R_a)', V')$ *where:*

- $W' = \{(w, e) \in W \times \mathsf{E} \mid \mathcal{M}, w \models \mathrm{pre}(e)\}$*;*
- $((w, e), (w', e')) \in R'_a$ *iff* $(w, w') \in R_a$ *and* $(e, e') \in R_a^{\mathcal{E}}$*;*
- $V'((w, e)) = \{p \in AP \mid \mathcal{M}, w \models \mathrm{post}(e, p)\}$.

Example 8. *Figure 1.5 shows the product operation of the Kripke model of Figure 1.1 and the event model of the Figure 1.3.*

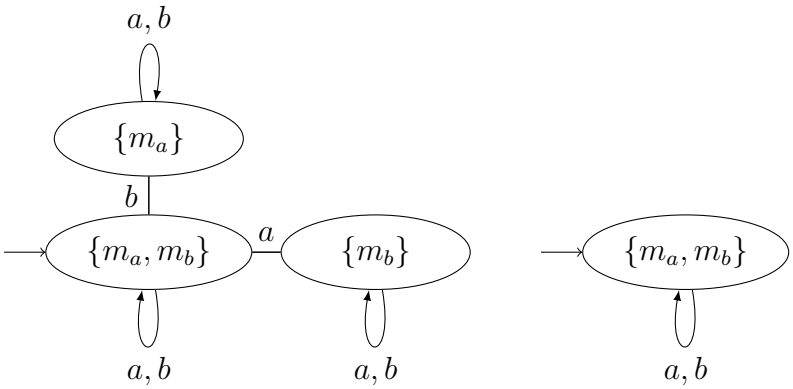


Figure 1.6: Kripke models in the execution of muddy children puzzle with 2 agents a, b.

Example 9. *Figure 1.6 shows the effect of the actions drawn in Figure 1.4 on the Kripke model in 1.2. After the father speaks, the world with valuation $\emptyset$ is removed. After the father asks the question and no child answers, all worlds except the one with valuation $\{m_a, m_b\}$ are removed. In this Kripke model, both children a and b now know that they are muddy.*

In fact, it can be proven that when k children are muddy, the father needs to ask the question $k-1$ times, and all the muddy children then answer that they know they are muddy.

We now define the semantics of the operator $\langle \mathcal{E}, \mathsf{E}_0 \rangle$.

Definition 8 (Semantics of **DELCK**)**.** *We extend the definition $\mathcal{M}, w \models \varphi$ to $\mathcal{L}_{\mathbf{DELCK}}$ with the following clause:*

- $\mathcal{M}, w \models \langle \mathcal{E}, \mathsf{E}_0 \rangle \varphi$ *if there exists* $e \in \mathsf{E}_0$ *s.t.* $\mathcal{M}, w \models \mathrm{pre}(e)$ *and* $\mathcal{M} \otimes \mathcal{E}, (w, e) \models \varphi$.

In the sequel, we take the abbreviation $(w, e_1, \ldots, e_n)$ for $(((w, e_1), \ldots), e_n)$ and further abbreviate it by $w\overrightarrow{e}$. We also write $\mathcal{M}\overrightarrow{\mathcal{E}}$ for $\mathcal{M}, \mathcal{E}_1, \ldots, \mathcal{E}_n$. We write $\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e} \models \varphi$ instead of $\mathcal{M} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n, (((w, e_1), \ldots), e_n) \models \varphi$. A sequence of events $\overrightarrow{e}$ is *executable* in w if $w\overrightarrow{e}$ is in $\mathcal{M}\overrightarrow{\mathcal{E}}$. The empty sequence of events is denoted by ϵ (so $w\epsilon = w$). The empty sequence is of course executable in all worlds.

We will also need a notion of size of formulas and models, noted $|\varphi|$, $|\mathcal{M}|, |\mathcal{E}|$.

Definition 9 (Size of formulas and models)**.** *The size of formulas φ and models $\mathcal{M}, \mathcal{E}$ are mutually defined by induction:*

- $|\top| = |p| = 1$;
- $|\neg\varphi| = 1 + |\varphi|$;
- $|\varphi_1 \vee \varphi_2| = 1 + |\varphi_1| + |\varphi_2|$;
- $|K_a\varphi| = 1 + |\varphi|$;
- $|C_G\varphi| = 1 + |\varphi|$;
- $|\langle \mathcal{E}, \mathsf{E}_0\rangle\varphi| = 1 + |\mathcal{E}| + |\mathsf{E}_0| + |\varphi|$;
- $|\mathcal{M}| = |AP| \times |W|$;
- $|\mathcal{E}| = |\mathsf{E}| + \sum_{e\in\mathsf{E}} \Big(|\operatorname{pre}(e)| + \sum_{p\in AP} |\operatorname{post}(e,p)| \Big)$.

The notion of size may change in the analysis later on, the differences will be explicitly given if it is the case.

1.2 Expressivity of event models in DEL

We now show expressivity results for **DEL**. Indeed, the whole class of event models is not needed to capture the whole expressivity of **DEL**. In fact, it is already proven that **DEL** has the same expressivity than **EL** [DHK07], but it is not the case with **DELCK** and **ELCK**. Yet, the translation from a formula of **DEL** to a formula of **EL** is exponential. Here we exhibit a sublanguage of **DEL** that captures the whole expressivity of **DEL** with a polynomial translation.

Before announcing the outline of the subsections, let us define two notions: modal depth of an epistemic formula without common knowledge and classes of event models.

Definition 10 (Modal depth)**.** *The modal depth of a formula $\varphi \in \mathcal{L}_{\mathbf{EL}}$, noted $d(\varphi)$ is defined by induction on φ:*

- $d(p) = d(\top) = 0$;
- $d(\neg\varphi) = d(\varphi)$;
- $d(\varphi_1 \vee \varphi_2) = \max(d(\varphi_1), d(\varphi_2))$;
- $d(K_a\varphi) = 1 + d(\varphi)$.

Intuitively, the modal depth corresponds to the maximal nesting of knowledge operators. Common knowledge is not included in the definition of modal depth because it intrinsically introduces an unbounded nesting of knowledge operators.

We now define the classes of event models for the section.

Definition 11. *The class of event models $\mathcal{C}_i^j$ with $i \geq 0$ or $i = \infty$, and $j \geq -1$ or $j = \infty$, is the set of event models where the events have preconditions of modal depth at most i and*

postconditions of modal depth at most j. If i or j is ∞ the modal depth is not constrained, and if $j = -1$, the event models must have trivial postconditions.

Example 10. *The event model from Figure 1.3 page 41 is in $\mathcal{C}_0^0$ because all formulas are propositional, but it is not in $\mathcal{C}_0^{-1}$because the postcondition of e is not trivial. Notice that it is also in $\mathcal{C}_i^j$ for any $i, j \geq 0$ and $i, j = \infty$.*

We trivially have that $\mathcal{C}_i^j \subseteq \mathcal{C}_{i'}^{j'}$ whenever $i \geq i'$ and $j \geq j'$.

Let us now define the notion of equivalence of formulas.

Definition 12 (Equivalent formulas)**.** *Let φ_1 and φ_2 be two formulas. We say that they are* equivalent *if $\mathcal{M}, w \models \varphi_1$ if and only if $\mathcal{M}, w \models \varphi_2$ for all pointed Kripke models $(\mathcal{M}, w)$.*

We prove here that for any formula $\varphi \in \mathscr{L}_{\mathbf{DEL}}$ on AP (supposed finite) whose event models are all in a certain $\mathcal{C}_i^j$, it is possible to polynomially construct an equivalent formula $tr(\varphi)$ on a certain $AP' \supseteq AP$ where the event models are in $\mathcal{C}$, where $\mathcal{C}$ can be any of the following: $\mathcal{C}_0^{\max(i,j)}$, $\mathcal{C}_{\max(i,j)}^0$, $\mathcal{C}_0^1$.

It will also prove directly that the result holds for $\mathcal{C}_1^0$. The $\mathcal{C}_0^{\max(i,j)}$ case is not directly needed, but we present the construction because it is fairly easy and allows us to introduce some notions needed for the next two.

The formulas φ and $tr(\varphi)$ do not contain the same set of atomic propositions, but if we evaluate them on Kripke models on AP', we will indeed have $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}, w \models tr(\varphi)$. If we want to be rigorous on the notion of equivalence, they would need to be defined in the same set of atomic propositions. We decide to distinguish AP and AP' to avoid defining both formulas with atomic propositions of the infinite set AP, with $tr(\varphi)$ containing fresh atomic propositions of AP not appearing in φ. Indeed, such a definition would make the proofs more difficult to read.

We dedicate one subsection per class and then discuss the case of common knowledge briefly.

1.2.1 Removing the knowledge operators from the preconditions

This section is dedicated to the $\mathcal{C}_0^{\max(i,j)}$ case.

General idea

In a formula φ, we replace the modality $\langle \mathcal{E}, \mathsf{E}_0 \rangle$ by two nested modalities $\langle \mathcal{E}', \mathsf{E}'_0 \rangle \langle \mathcal{E}_p, \mathsf{E}_{0p} \rangle$. The pointed event model $(\mathcal{E}', \mathsf{E}'_0)$ is $(\mathcal{E}, \mathsf{E}_0)$ where the preconditions are pushed in post-conditions by assigning the value of a fresh atomic proposition, p_{prune}, to $\top$ whenever the precondition is true. The event model $(\mathcal{E}_p, \mathsf{E}_{0p})$ removes any world where p_{prune} is false and resets the value of p_{prune}.

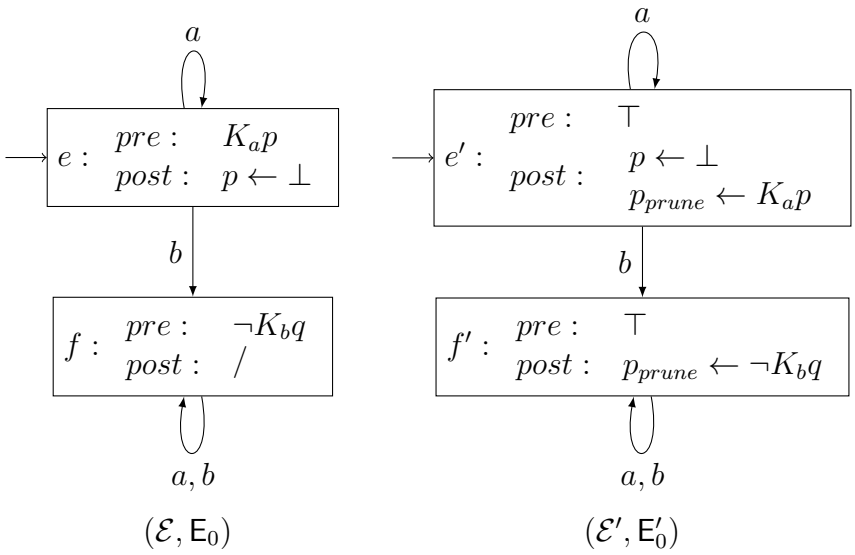


Figure 1.7: Example of an event model $(\mathcal{E}', \mathsf{E}'_0)$ for the $\mathcal{C}_0^{\max(i,j)}$ construction.

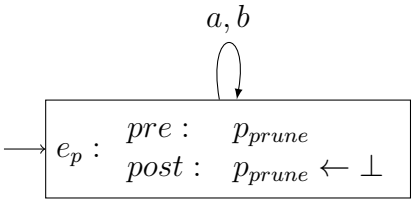


Figure 1.8: Event model $(\mathcal{E}_p, \mathsf{E}_{0p})$ for the $\mathcal{C}_0^{\max(i,j)}$ construction.

An example of $\mathcal{E}'$ is drawn in Figure 1.7 and the event model $\mathcal{E}_p$ is drawn on Figure 1.8.

Formal definition

Let $AP' = AP \cup \{p_{prune}\}$ and $(\mathcal{E}, \mathsf{E}_0)$ an event model on AP with $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a \in Ag}, \text{pre}, \text{post})$. We define the event models $(\mathcal{E}', \mathsf{E}'_0)$ and $(\mathcal{E}_p, \mathsf{E}_{0p})$.

Definition 13 (Event model $(\mathcal{E}', \mathsf{E}'_0)$)**.** *The event model* $\mathcal{E}' = (\mathsf{E}', (R_a^{\mathcal{E}'})_{a \in Ag}, \text{pre}', \text{post}')$ *and set* E'_0 *are defined as follows:*

- $\mathsf{E}' = \{e', e \in \mathsf{E}\}$*;*
- $R_a^{\mathcal{E}'} = \{(e', f'), (e, f) \in R_a^{\mathcal{E}}\}$ *for all* $a \in Ag$*;*
- $\text{pre}'(e') = \top$ *for all* $e' \in \mathsf{E}'$*;*
- $\text{post}'(e', p) = \text{post}(e, p)$ *for all* $e' \in \mathsf{E}'$ *and* $p \in AP$*;*
- $\text{post}'(e', p_{prune}) = \text{pre}(e)$ *for all* $e' \in \mathsf{E}'$*;*
- $\mathsf{E}'_0 = \{e', e \in \mathsf{E}_0\}$.

If $\mathcal{E} \in \mathcal{C}_i^j$ then $\mathcal{E}' \in \mathcal{C}_0^{\max(i,j)}$ because the preconditions in $\mathcal{E}'$ are now propositional and the postconditions contain formulas both from the preconditions and the postconditions of $\mathcal{E}$, thus are of modal depth at most $\max(i, j)$.

Definition 14 (Event model $(\mathcal{E}_p, \mathsf{E}_{0p})$)**.** *The event model* $\mathcal{E}_p = (\mathsf{E}_p, (R_a^{\mathcal{E}_p})_{a \in Ag}, \text{pre}_p, \text{post}_p)$ *and set* E_{0p} *are defined as follows:*

- $\mathsf{E}_p = \{e_p\}$*;*
- $R_a^{\mathcal{E}_p} = \{(e_p, e_p)\}$*;*
- $\text{pre}_p(e_p) = p_{prune}$*;*
- $\text{post}_p(e_p, p_{prune}) = \bot$*;* post_p *is trivial for other atomic propositions;*
- $\mathsf{E}_{0p} = \{e_p\}$.

The event model $\mathcal{E}_p$ is in $\mathcal{C}_0^0$ because both the precondition and the postcondition of e_p are propositional.

We now prove that executing $(\mathcal{E}', \mathsf{E}'_0)$ then $(\mathcal{E}_p, \mathsf{E}_{0p})$ has the same effect than executing $(\mathcal{E}, \mathsf{E}_0)$ for any pointed Kripke model $(\mathcal{M}, w)$. Before stating the theorem, we need to define a notion of equivalence between Kripke models , generally called AP-bisimulation.

Definition 15 (AP-bisimulation)**.** *Let $\mathcal{M} = (W, (R_a)_{a\in Ag}, V)$ and $\mathcal{M}' = (W', (R'_a)_{a\in Ag}, V')$ two Kripke models on $AP_1 \supseteq AP$ and $AP_2 \supseteq AP$. An AP-bisimulation is a relation $B \subseteq W \times W'$ such that:*

- *AP-conservation: For all $(w, w') \in B$, for all $p \in AP$, $p \in V(w)$ if and only if $p \in V'(w')$;*
- *Zig: For all $(w, w') \in B$, for all $a \in Ag$ and $u \in W$, if $(w, u) \in R_a$ then there exists $u' \in W'$ such that $(w', u') \in R'_a$ and $(u, u') \in B$.*
- *Zag: For all $(w, w') \in B$, for all $a \in Ag$ and $u' \in W'$, if $(w', u') \in R'_a$ then there exists $u \in W$ such that $(w, u) \in R_a$ and $(u, u') \in B$.*

If there exists an AP-bisimulation between $\mathcal{M}$ and $\mathcal{M}'$ then we say that both Kripke model are *AP-bisimilar*. Bisimulation is also defined on pointed Kripke models $(\mathcal{M}, w)$ and $(\mathcal{M}, w')$ by adding the constraint $(w, w') \in B$. Interestingly, if two pointed Kripke models $(\mathcal{M}, w)$ and $(\mathcal{M}', w')$ are AP-bisimilar then $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}', w' \models \varphi$ for any formula $\varphi \in \mathcal{L}_{\mathbf{DELCK}}$ on AP.

Theorem 1. *Let $(\mathcal{E}, \mathsf{E}_0)$ be an event model on AP and $(\mathcal{E}', \mathsf{E}'_0)$, $(\mathcal{E}_p, \mathsf{E}_{0p})$ be the two event models defined respectively in Definitions 13 and 14. Then for any pointed Kripke model $(\mathcal{M}, w)$ on AP, $e \in \mathsf{E}_0$, $(\mathcal{M} \otimes \mathcal{E}, w)$ and $(\mathcal{M} \otimes \mathcal{E}' \otimes \mathcal{E}_p, (w, e', e_p))$ are AP'-bisimilar.*

Proof. We define the relation $B = \{((w, e), (w, e', e_p)), (w, e) \in \mathcal{M} \otimes \mathcal{E}\}$. Before proving that B is a AP'-bisimulation, we first need to prove that it is well defined, meaning that (w, e) exists if and only if the world (w, e', e_p) exists.

The world (w, e) exists	if and only if $\mathcal{M}, w \models \mathrm{pre}(e)$
	if and only if $\mathcal{M}, w \models \mathrm{post}(e', p_{prune})$
	if and only if $p_{prune} \in V(w, e')$
	if and only if $\mathcal{M} \otimes \mathcal{E}', (w, e') \models \mathrm{pre}_p(e_p)$
	if and only if (w, e', e_p) exists.

We now prove that B is a AP'-bisimulation.

- AP'-conservation: Let $p \in AP$. Then $p \in V(w, e)$ if and only if $\mathcal{M}, w \models \mathrm{post}(e, p)$. In the construction of (w, e', e_p), the value of p is only modified on the execution of e', then $p \in V(w, e', e_p)$ if and only if $\mathcal{M}, w \models \mathrm{post}'(e', p)$ if and only if $\mathcal{M}, w \models \mathrm{post}(e, p)$ by definition of e'. Therefore, $p \in V(w, e)$ if and only if $p \in V(w, e', e_p)$. For p_{prune}, we necessarily have $p_{prune} \notin V(w, e)$ and $p_{prune} \notin V(w, e', e_p)$.

- Zig and Zag: we have $((w,e),(u,f)) \in R_a^{\mathcal{M}\otimes(\mathcal{E},\mathsf{E}_0)}$ if and only if $((w,e',e_p),(u,f',e_p)) \in R_a^{\mathcal{M}\otimes(\mathcal{E}',\mathsf{E}'_0)\otimes(\mathcal{E}_p,\mathsf{E}_{0p})}$ by definition of $\mathcal{E}'$ and $\mathcal{E}_p$ so the Zig and Zag properties are direct.

□

Therefore, we have $\mathcal{M}, w \models \langle\mathcal{E},\mathsf{E}_0\rangle\varphi$ if and only if $\mathcal{M}, w \models \langle\mathcal{E}',\mathsf{E}'_0\rangle\langle\mathcal{E}_p,\mathsf{E}_{0p}\rangle\varphi$ for any formula φ on AP. Also, we have also proven that p_{prune} is false everywhere after having applied $\mathcal{E}'$ and $\mathcal{E}_p$, thus we can reuse p_{prune}. In particular, we have proven that the following translation is correct.

Definition 16. *Let φ be a formula on AP with event models all in $\mathcal{C}_i^j$. We construct an AP-equivalent formula $tr(\varphi)$ on $AP \cup \{p_{prune}\}$, with events models all in $\mathcal{C}_0^{\max(i,j)}$ by induction on φ:*

- $tr(\top) = \top$;
- $tr(p) = p$;
- $tr(\neg\varphi) = \neg tr(\varphi)$;
- $tr(\varphi_1 \vee \varphi_2) = tr(\varphi_1) \vee tr(\varphi_2)$;
- $tr(K_a\varphi) = K_a tr(\varphi)$
- $tr(C_G\varphi) = C_G tr(\varphi)$;
- $tr(\langle\mathcal{E},\mathsf{E}_0\rangle\varphi) = \langle\mathcal{E}',\mathsf{E}'_0\rangle\langle\mathcal{E}_p,\mathsf{E}_{0p}\rangle tr(\varphi)$.

1.2.2 Removing the knowledge from the postconditions.

This section is dedicated to the $\mathcal{C}^0_{\max(i,j)}$ case.

General idea

Let $(\mathcal{E},\mathsf{E}_0)$ be an event model in $\mathcal{C}_i^j$. To simulate the execution of this event model, we introduce one atomic proposition p_e for each atomic proposition $p \in AP$ and event $e \in \mathcal{E}$, and create one event model $(\mathcal{E}^{e,p},\mathsf{E}_0^{e,p})$ per p and e that will assign the value of $\mathrm{post}(e,p)$ to p_e. We furthermore introduce an event model $(\mathcal{E}',\mathsf{E}'_0)$ that is $(\mathcal{E},\mathsf{E}_0)$ where postcondition of event e become $p \leftarrow p_e$ for every $p \in AP$.

Figure 1.9 gives an example.

Formal definition

Let $(\mathcal{E},\mathsf{E}_0)$ be an event model with $\mathcal{E} = (\mathsf{E},(R_a^{\mathcal{E}})_{a\in Ag},\mathrm{pre},\mathrm{post})$. We define the set$AP' = AP \cup \{p_e, p \in AP, e \in \mathsf{E}\}$.

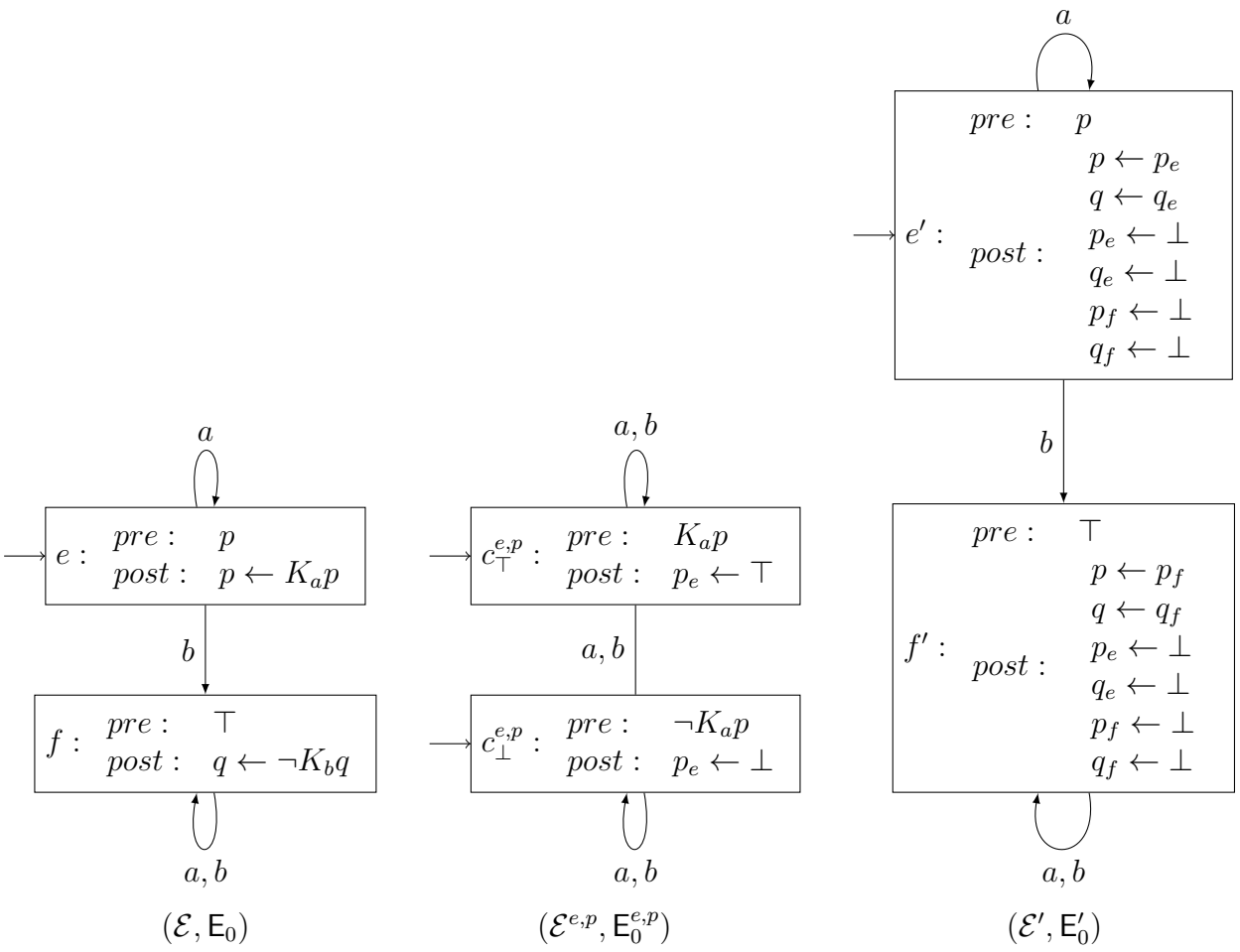


Figure 1.9: Example of an event model $(\mathcal{E}^{e,p}, \mathsf{E}_0^{e,p})$ and $(\mathcal{E}', \mathsf{E}_0')$ for the $\mathcal{C}^0_{\max(i,j)}$ construction.

Definition 17 (Event model $(\mathcal{E}^{e,p}, \mathsf{E}_0^{e,p})$)**.** *The event model* $\mathcal{E}^{e,p} = (\mathsf{E}^{e,p}, (R_a^{e,p})_{p\in AP}, \mathrm{pre}^{e,p}, \mathrm{post}^{e,p})$ *is defined as follows:*

- $\mathsf{E}^{e,p} = \{c_\top^{e,p}, c_\bot^{e,p}\}$;
- $R_a^{e,p} = \mathsf{E}^{e,p} \times \mathsf{E}^{e,p}$ *for all* $a \in Ag$;
- $\mathrm{pre}^{e,p}(c_\top^{e,p}) = \mathrm{post}(e,p)$; $\mathrm{pre}^{e,p}(c_\bot^{e,p}) = \neg\,\mathrm{post}(e,p)$;
- $\mathrm{post}^{e,p}(c_\top^{e,p}, p_e) = \top$; $\mathrm{post}^{e,p}(c_\bot^{e,p}, p_e) = \bot$; $\mathrm{post}^{e,p}$ *is trivial for all other atomic propositions of* AP';
- $\mathsf{E}_0^{e,p} = \mathsf{E}^{e,p}$.

Notice that the following proposition holds.

Proposition 1. *For any Kripke model $\mathcal{M}$ on AP', $\mathcal{M} \otimes \mathcal{E}^{e,p}$ is isomorphic to $\mathcal{M}$ with the only difference that p_e is in the new valuation of w if and only if $\mathcal{M}, w \models \text{post}(e,p)$.*

Proof sketch Each world of $\mathcal{M}$ matches exactly to one event of $\mathsf{E}^{e,p} = \{c_\top^{e,p}, c_\bot^{e,p}\}$, so $\mathcal{M} \otimes \mathcal{E}^{e,p}$ has the same number of worlds. Furthermore, the epistemic relation form a complete graph in $(\mathcal{E}^{e,p}, \mathsf{E}_0^{e,p})$, so all pre-existing edges in $\mathcal{M}$ remain. Therefore, $\mathcal{M}$ and $\mathcal{M} \otimes \mathcal{E}^{e,p}$ are isomorphic.

For the second point, it is direct by definition of $\mathcal{M} \otimes \mathcal{E}^{e,p}$. ■

Definition 18 (Event model $(\mathcal{E}', \mathsf{E}_0')$)**.** *The event model $\mathcal{E}' = (\mathsf{E}', (R_a^{\mathcal{E}'})_{a \in Ag}, \text{pre}', \text{post}')$ is defined as follows:*

- $\mathsf{E}' = \{e', e' \in \mathsf{E}\}$*;*
- $R_a^{\mathcal{E}'} = \{(e', f'), (e, f) \in R_a^{\mathcal{E}}\}$ *for all $a \in Ag$;*
- $\text{pre}'(e') = \text{pre}(e)$ *for all $e' \in \mathsf{E}'$;*
- $\text{post}'(e', p) = p_e$ *for all $e' \in \mathsf{E}'$ and $p \in AP$;* $\text{post}'(e', q) = \bot$ *for all $e' \in \mathsf{E}'$ and $q \in AP' \setminus AP$;*
- $\mathsf{E}_0' = \{e', e \in \mathsf{E}_0\}$.

Theorem 2. *For any pointed Kripke model $(\mathcal{M}, w)$ on AP, for every $e \in \mathsf{E}_0$, $(\mathcal{M} \otimes \mathcal{E}, (w, e))$ and $(\mathcal{M} \otimes \left(\bigotimes_{e \in \mathsf{E}} \bigotimes_{p \in AP} \mathcal{E}^{e,p}\right) \otimes \mathcal{E}', (w, \ldots, e'))$ are AP-bisimilar where $(w, \ldots, e')$ is the only possible world corresponding to (w, e).*

Proof. The proof is similar to the one of Theorem 1. Here B associates (w, e) to the only $(w, \ldots, e')$ that exists, with $\ldots$ the $c_\bot^{f,q}/c_\top^{f,q}$ that are the only events that can be inserted in the tuple.

- AP-conservation: for any $p \in AP$, p is in the valuation of $(w, \ldots, e')$ if p_e was true, meaning that $\text{post}(e', p)$ was true in a certain $(\mathcal{M}', w)$ isomorphic to $(\mathcal{M}, w)$. Therefore, p has the same value in (w, e) and $(w, \ldots, e')$.
- Zig and Zag: as remarked before, $\mathcal{M} \otimes \left(\bigotimes_{e \in \mathsf{E}} \bigotimes_{p \in AP} \mathcal{E}^{e,p}\right)$ is isomorphic to $\mathcal{M}$, and $\mathcal{E}'$ is constructed similarly to $\mathcal{E}$, therefore the Zig and Zag properties are direct.

□

As before, we can then define a translation from a formula with event models in $\mathcal{C}_i^j$ to a formula with event models in $\mathcal{C}^0_{\max(i,j)}$. Here the new formula will have $\sum_{\mathcal{E}\in\varphi} |AP| \times |\mathcal{E}|$ new atomic propositions and its size will be in $O(|\varphi| \times \sum_{\mathcal{E}\in\varphi} |AP| \times |\mathcal{E}|)$ which is polynomial in $|\varphi|$.

1.2.3 Removing the knowledge nesting from preconditions and the knowledge operators from postconditions.

This subsection is dedicated to the $\mathcal{C}_0^1$ case.

General idea

To simulate the effect of an event model $(\mathcal{E}, \mathsf{E}_0)$ in $\mathcal{C}_i^j$ with event models with no nesting of knowledge operators, we compute with event models $(\mathcal{E}^1, \mathsf{E}_0^1), \ldots, (\mathcal{E}^{\max(i,j)-1}, \mathsf{E}_0^{\max(i,j)-1})$ the values of subformulas of formulas in $\mathcal{E}$ that are of modal depth $1, \ldots, \max(i,j) - 1$. For each formula φ in $\mathcal{E}$, we introduce one new atomic proposition p_ψ for each subformula ψ of φ. Then at the end we execute an event model $(\mathcal{E}', \mathsf{E}'_0)$ which is $(\mathcal{E}, \mathsf{E}_0)$ where each formula φ is now replaced by p_φ.

Figure 1.10 gives an example of a $(\mathcal{E}^1, \mathsf{E}_0^1)$ event model and of a $(\mathcal{E}', \mathsf{E}'_0)$ event model.

Formal definition

We first need to define the set of subformulas of φ.

Definition 19 (Set of subformulas)**.** *Let $\varphi \in \mathcal{L}_{\mathbf{EL}}$ be a formula. We define the set of subformulas of φ, noted $SF(\varphi)$ in the following way.*

- $SF(p) = \{p\}$;
- $SF(\top) = \{\top\}$
- $SF(\neg\varphi) = \{\neg\varphi\} \cup SF(\varphi)$;
- $SF(\varphi_1 \vee \varphi_2) = \{\varphi_1 \vee \varphi_2\} \cup SF(\varphi_1) \cup SF(\varphi_2)$;
- $SF(K_a\varphi) = \{K_a\varphi\} \cup SF(\varphi)$

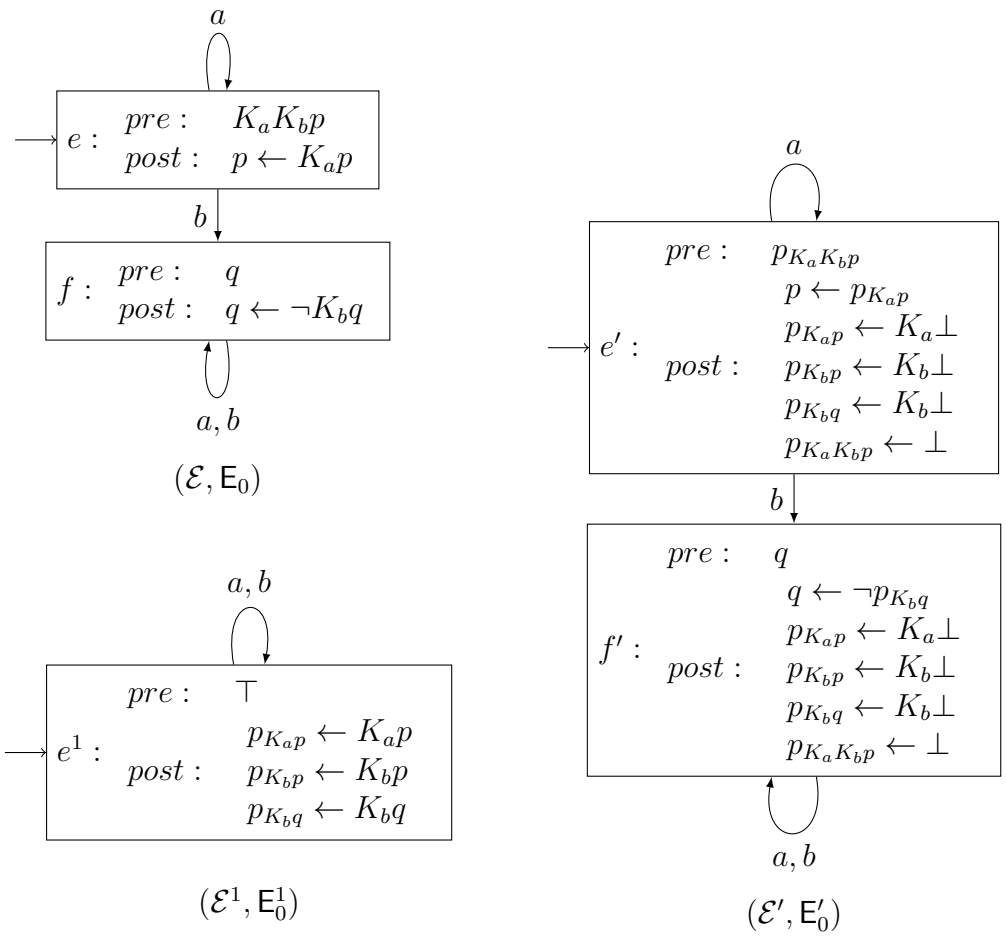


Figure 1.10: Example of an event model $(\mathcal{E}^1, \mathsf{E}^1_0)$ and $(\mathcal{E}', \mathsf{E}'_0)$ for the $\mathcal{C}^1_0$ construction.

Notice that $SF(\varphi)$ has polynomial size compared to $|\varphi|$.

We now define $AP' = AP \cup \{p_\psi, \psi \in SF(\varphi), \varphi \in \mathcal{E}, d(\psi) \geq 1\}$ where $\varphi \in \mathcal{E}$ means that φ appears in a precondition or a postcondition in $\mathcal{E}$. For any formula φ, we define a new formula $tr(\varphi)$ of modal depth one where subformulas ψ have been replaced by p_ψ.

Definition 20. *Let φ be a formula of modal depth at least 2. Then we define $\varphi' = tr(\varphi)$ in the following way.*

- $tr(p) = p$;
- $tr(\top) = \top$;
- $tr(\neg\varphi) = \neg tr(\varphi)$;

- $tr(\varphi_1 \vee \varphi_2) = tr(\varphi_1) \vee tr(\varphi_2)$;
- $tr(K_a\psi) = K_a p_\psi$.

Indeed, $tr(\varphi)$ is a now a formula of modal depth at most 1, and any p_ψ appearing in $tr(\varphi)$ is such that $d(\psi) = d(\varphi) - 1$. The definition is for any formula φ of modal depth at least 2 because we need to be sure that p_ψ exists. For any formula of modal depth at most 1, it is not needed to transform it to a formula of modal depth at most 1 because it already is. In that case we define $tr(\varphi) = \varphi$.

Definition 21 (Event model $(\mathcal{E}^k, \mathsf{E}_0^k)$)**.** *The event model* $\mathcal{E}^k = (\mathsf{E}^k, (R_a^k)_{a\in Ag}, \mathrm{pre}^k, \mathrm{post}^k)$ *is defined in the following way.*

- $\mathsf{E}^k = \{e^k\}$;
- $R_a^k = \{(e^k, e^k)\}$ *for all agents* $a \in Ag$;
- $\mathrm{pre}^k(e^k) = \top$;
- $\mathrm{post}^k(e, p_\varphi) = tr(\varphi)$ *for any formula* φ *of modal depth* k *such that* $p_\varphi \in AP'$;
- post^k *is trivial for all other atomic propositions;*
- $\mathsf{E}_0^k = \{e^k\}$.

Definition 22 (Event model $(\mathcal{E}', \mathsf{E}_0')$)**.** *The event model* $\mathcal{E}' = (\mathsf{E}', (R_a')_{a\in Ag}, \mathrm{pre}', \mathrm{post}')$ *is defined in the following way.*

- $\mathsf{E}' = \{e', e' \in \mathsf{E}\}$;
- $R_a^{\mathcal{E}'} = \{(e', f'), (e, f) \in R_a^{\mathcal{E}}\}$ *for all* $a \in Ag$;
- $\mathrm{pre}'(e') = \begin{cases} p_{\mathrm{pre}(e)} \text{ if } d(\mathrm{pre}(e)) \geq 1 \\ \mathrm{pre}(e) \text{ otherwise.} \end{cases}$ *for all* $e' \in \mathsf{E}'$;
- $\mathrm{post}'(e', p) = \begin{cases} p_{\mathrm{post}(e,p)} \text{ if } d(\mathrm{post}(e,p)) \geq 1 \\ \mathrm{post}(e,p) \text{ otherwise.} \end{cases}$ *for all* $e' \in \mathsf{E}'$ *and* $p \in AP$;
- $\mathrm{post}'(e', p_\psi) = \bot$ *for all* $e' \in \mathsf{E}'$ *and* $q_\psi \in AP' \setminus AP$;
- $\mathsf{E}_0' = \{e', e \in \mathsf{E}_0\}$.

Theorem 3. *For any pointed Kripke model $(\mathcal{M}, w)$ and $e \in \mathsf{E}_0$, the Kripke models $(\mathcal{M} \otimes \mathcal{E}, (w, e))$ and $(\mathcal{M} \otimes \bigotimes_{k=1}^{\max(i,j)} \mathcal{E}^k \otimes \mathcal{E}', (w, e^1, \ldots, e^{\max(i,j)}, e'))$ are AP-bisimilar.*

Proof. The bisimulation is $B = \{((w, e), (w, e^1, \ldots, e^{\max(i,j)}, e'), w \in W, e \in \mathsf{E}\}$. As before, the Zig and Zag properties are easy to prove, the difficult part of the proof is to prove that $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M} \otimes \bigotimes_{k=1}^{\max(i,j)}, (w, e^1, \ldots, e^{\max(i,j)}) \models p_\varphi$ with φ any formula appearing in a precondition or a postcondition in E. Indeed, if such a property is proven, then:

- B is well defined because $(w, e^1, \ldots, e^{\max(i,j)}, e')$ passed all the preconditions if and only if w passed the precondition of e (because the only precondition which is not $\top$ is $\mathrm{pre}'(e') = p_{\mathrm{pre}(e)}$).
- The AP-conservation is direct because:

  $p \in V(w, e)$ if and only if $\mathcal{M}, w \models \mathrm{post}(e, p)$
  if and only if $\mathcal{M} \otimes \bigotimes_{k=1}^{\max(i,j)}, (w, e^1, \ldots, e^{\max(i,j)}) \models tr(\mathrm{post}(e, p))$
  if and only $\mathcal{M} \otimes \bigotimes_{k=1}^{\max(i,j)}, (w, e^1, \ldots, e^{\max(i,j)}) \models \mathrm{post}'(e', p)$
  if and only if $p \in V(w, e^1, \ldots, e^{\max(i,j)}, e')$.

  The above property is a consequence of the following lemma.

Lemma 1. *For any $i \in \{0, \ldots, \max(i, j) - 1\}$, for any formula φ of modal depth at most $i + 1$, $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M} \otimes \bigotimes_{k=1}^{i} \mathcal{E}_k, (w, e^1, \ldots, e^i) \models tr(\varphi)$*

It is proven by recurrence on i:

- $i = 0$: for this case, $tr(\varphi) = \varphi$ and $\mathcal{M} \otimes \bigotimes_{k=1}^{i} \mathcal{E}_k, (w, e^1, \ldots, e^i) = \mathcal{M}$ so the case is direct.
- $i > 0$: if the property is true for $k - 1$ then for i, in $\mathcal{M} \otimes \bigotimes_{k=1}^{i} \mathcal{E}_k, (w, e^1, \ldots, e^i)$, any atomic proposition p_ψ with ψ of modal depth i has now the correct value of ψ in each world w. Therefore for any formula φ of modal depth $i + 1$, we indeed have $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M} \otimes \bigotimes_{k=1}^{i} \mathcal{E}_k, (w, e^1, \ldots, e^i) \models tr(\varphi)$.

□

Replacing $(\mathcal{E}, \mathsf{E}_0)$ by $(\mathcal{E}^1, \mathsf{E}_0^1) \ldots (\mathcal{E}^{\max(i,j)-1}, \mathsf{E}_0^{\max(i,j)-1})(\mathcal{E}', \mathsf{E}_0)$ introduces $\sum_{\varphi \in \mathcal{E}} |SF(\varphi)|$ new atomic propositions, and there are $(\max(i, j) + 1)$ times event models compared to before. Therefore, translating a formula from $\mathcal{C}_i^j$ to $\mathcal{C}_0^1$ is polynomial.

Notice that we have also proven that it is polynomial to go from $\mathcal{C}_i^j$ to $\mathcal{C}_1^0$ by combining the $\mathcal{C}_{\max(i,j)}^0$ result with the $\mathcal{C}_0^1$ result.

Theorem 4. *For any formula φ containing event models without common knowledge, it is possible to polynomially construct a formula φ' with events models either all in $\mathcal{C}_0^1$ or either all in $\mathcal{C}_1^0$.*

1.2.4 Adding common knowledge

Unfortunately, such results do not hold when formulas in event models contain common knowledge operators. The reason is that common knowledge introduces an unbounded modal depth, so it is in no $\mathcal{C}_i^j$ with $i, j \in \mathbb{N}$.

Nevertheless, when the Kripke model $\mathcal{M}$ is fixed from the start and is finite, it is possible to transform the $C_G\varphi$ modality into $\bigcup_{k=0}^{|W|} \bigcup_{a_1,\dots,a_k \in Ag} K_{a_1} \dots K_{a_k}\varphi$, which has exponential size compared to $|C_G\varphi|$. But this formula depends on the initial Kripke model, so the expressivity results do not hold anymore.

1.3 Conclusion

In this chapter, we have first defined *epistemic logic*, seen here as a specification language for multi-agent systems. Its semantics uses *Kripke models*, that express multi-agent systems by a graph of possible configurations, called *worlds*. We considered classical axioms for epistemic logic to restrict the Kripke models considered, as S5, for models about knowledge, and KD45, for models about beliefs. Then, we have defined dynamic epistemic logic (**DEL**) and its extension with common knowledge (**DELCK**), that extends epistemic logic with dynamic operators featuring *event models*. Such models give a precise way of defining events and the perception agents have about the events.

We have shown that in fact, a relevant fragment of **DELCK** is sufficient to express every formula of **DELCK** where event models do not feature common knowledge. We defined the class $\mathcal{C}_i^j$ for events models with preconditions of modal depth at most i and postconditions of modal depth at most j. We have shown that for any formula where its event models are in $\mathcal{C}_i^j$, we could find an equivalent formula where its event models fall into one of the following classes: $\mathcal{C}_0^{\max(i,j)}, \mathcal{C}_{\max(i,j)}^0, \mathcal{C}_0^1$, and deduced that $\mathcal{C}_1^0$ is also suitable. The proof techniques involves dividing a dynamic modality $\langle \mathcal{E}, \mathsf{E}_0 \rangle$ into a sequence of modalities $\langle \mathcal{E}^i, \mathsf{E}_0^i \rangle$ where each event model $\mathcal{E}^i$ is in the relevant class considered.

The main limitation of the expressivity results is common knowledge in event models. Indeed, common knowledge introduces an unbounded number of modal depth, so an

event model containing common knowledge is in no $\mathcal{C}_i^j$ with $i, j \in \mathbb{N}$. We suspect that no polynomial translation exist in this case.

Chapter 2

Model Checking and Satisfiability of Dynamic Epistemic Logic

In this chapter, we study decision problems of **DELCK**. The chapter is constructed as follows.

- First, we show that the model checking problem against **DELCK** is Pspace-complete (same result than **DEL**).
- Second, we recall that the satisfiability problem of **DEL** is NExptime-complete, and show that it is 2-Exptime-complete for **DELCK**.
- Third, we model a game with incomplete information, blind tic tac toe, to illustrate the notions presented in the chapter. We show in particular that finding a uniform strategy in blind tic tac toe can be reduced to the model checking problem against **DELCK**, meaning that finding a uniform strategy in such games is not harder than Pspace.
- Finally, we conclude.

Everything in this chapter is a contribution. The model checking and the satisfiability section have been published in [CS18].

2.1 Complexity of model checking against DELCK

In this section, we explain the model checking problem against **DELCK**, recall that it was already known against **DEL** to be PSPACE-complete, and detail the contribution: the model checking against **DELCK** remains PSPACE-complete even when common knowledge is added (the PSPACE-hardness being easy, we only prove here the PSPACE-membership).

Duality of algorithms. In the sequel, alternating algorithms are paired: a yes-algorithm $algo_{\mathsf{yes}}$ and a no-algorithm $algo_{\mathsf{no}}$.

- Algorithm $algo_{\mathsf{no}}$ rejects exactly when Algorithm $algo_{\mathsf{yes}}$ does not reject, and vice versa;
- Existential choices and universal choices are permuted in the two algorithms.

In the sequel, we will write and explain yes algorithms, and write the no algorithms in Appendix D.

The model checking for **DELCK** is defined as follows.

Definition 23 (Model checking problem)**.**

- ***Input:*** *a pointed Kripke model $\mathcal{M}, w$, a formula φ of $\mathcal{L}_{\mathbf{DELCK}}$;*
- ***Output:*** *yes if $\mathcal{M}, w \models \varphi$, no otherwise.*

In this section, we prove the following theorem.

Theorem 5. *The model checking problem for* **DELCK** *is* PSPACE*-complete.*

Hardness comes directly from the PSPACE-hardness of the model checking of **DEL** without common knowledge [AS13] (actually, it is already PSPACE-hard for single-pointed event models [BJS15], but actually even when the Kripke model is **S5** and event models are **S5** and single-pointed [PRS15]).

For the PSPACE-membership, Figures 2.1 and 2.2 provides the pseudo-code of an alternating Turing machine that decides the model checking problem for **DELCK** in polynomial time. The upper bound is proven since PSPACE = APTIME. The machine starts by calling $mc_{\mathsf{yes}}(\mathcal{M}, w, \varphi)$. The specifications of the procedures mc_{yes}, $inval_{\mathsf{yes}}$, in_{yes}, $access_{\mathsf{yes}}$ and $access^{*}_{\mathsf{yes}}$ (see Figures 2.1 and 2.2) are given in the following proposition:

Proposition 2. *For all formulas φ, for all Kripke models $\mathcal{M}$, for all sequences of event models $\overrightarrow{\mathcal{E}}$, for all worlds $w\overrightarrow{e}, u\overrightarrow{f}$ of $\mathcal{M}\overrightarrow{\mathcal{E}}$, for all agents a, for all groups of agents G, for integers i that are powers of two,*

proc $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$ $\quad |\mathcal{M}\overrightarrow{\mathcal{E}}| + |\varphi|$
▷ accepts whenever $\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e} \models \varphi$
- **case** $\varphi = p$: $inval_{\mathsf{yes}}(p, \mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e})$
- **case** $\varphi = (\varphi_1 \vee \varphi_2)$: (∃) choose $i \in \{1, 2\}$; $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi_i)$
- **case** $\varphi = \neg\psi$: $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \psi)$.
- **case** $\varphi = K_a\psi$:
  - (∀) choose $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$
  - (∃) $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $in_{\mathsf{no}}(u\overrightarrow{f}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$
- **case** $\varphi = \langle\mathcal{E}, \mathsf{E}_0\rangle\psi$:
  - (∃) choose $e \in \mathsf{E}_0$;
  - (∀) $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \mathrm{pre}(e))$ **and** $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}} :: \mathcal{E}, w\overrightarrow{e} :: e, \psi)$.
- **case** $\varphi = C_G\psi$:
  - (∀) choose $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$
  - (∃) $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, B_{\mathcal{M},\varphi}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $in_{\mathsf{no}}(u\overrightarrow{f}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$

proc $inval_{\mathsf{yes}}(p, w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ $\quad |\mathcal{M}\overrightarrow{\mathcal{E}}|$
▷ accepts whenever $p \in V(w\overrightarrow{e})$
- **case** $\overrightarrow{\mathcal{E}} = \epsilon$: **if** $p \in V(w)$ **then accept else reject**
- **case** $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}'::\mathcal{E}$ and $w\overrightarrow{e} = w\overrightarrow{e}'::e$: $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{post}(e, p))$

proc $in_{\mathsf{yes}}(w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ $\quad |\mathcal{M}\overrightarrow{\mathcal{E}}|$
▷ accepts whenever $w\overrightarrow{e} \in \mathcal{M}\overrightarrow{\mathcal{E}}$
- **case** $\overrightarrow{\mathcal{E}} = \epsilon$: **accept**
- **case** $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}'::\mathcal{E}$, $w\overrightarrow{e} = w\overrightarrow{e}'::e$: (∀) $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{pre}(e))$ **and** $in_{\mathsf{yes}}(w\overrightarrow{e}', \mathcal{M}\overrightarrow{\mathcal{E}}')$

proc $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ $\quad |\mathcal{M}\overrightarrow{\mathcal{E}}|$
▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \in R_a$
- **case** $\overrightarrow{\mathcal{E}} = \epsilon$: **if** $(w, u) \in R_a^{\mathcal{M}}$ **then accept else reject**
- **case** $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}' :: \mathcal{E}$, $\overrightarrow{e} = \overrightarrow{e}' :: e$, $\overrightarrow{f} = \overrightarrow{f}' :: f$:
  - (∀) $access_{\mathsf{yes}}(w\overrightarrow{e}', u\overrightarrow{f}', a, \mathcal{M}\overrightarrow{\mathcal{E}}')$ **and if** $(e, f) \in R_a^{\mathcal{E}}$ **then accept else reject**

Figure 2.1: Model checking procedures for **DELCK** (in gray: quantities associated to each procedure call).

proc $access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathcal{M}\overrightarrow{\mathcal{E}})$ $\quad |\mathcal{M}\overrightarrow{\mathcal{E}}| + \log i$

$\triangleright$ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \in (\bigcup_{a\in G} R_a)^j$ for some $j \leq i$

case $i = 1$:

if $u\overrightarrow{f} = w\overrightarrow{e}$ **then accept else** (∃) choose $a \in G$; $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$

case $i \geq 2$:

(∃) choose $v\overrightarrow{g} \in \mathcal{M}\overrightarrow{\mathcal{E}}$

(∀) $in_{\mathsf{yes}}(v\overrightarrow{g}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **and** $access^*_{\mathsf{yes}}(w\overrightarrow{e}, v\overrightarrow{g}, G, i/2, \mathcal{M}\overrightarrow{\mathcal{E}})$

and $access^*_{\mathsf{yes}}(v\overrightarrow{g}, u\overrightarrow{f}, G, i/2, \mathcal{M}\overrightarrow{\mathcal{E}})$

Figure 2.2: Sub-procedures for **DELCK** (in gray: quantities associated to each procedure call).

$mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$ *is accepting*	*iff* $\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e} \models \varphi$,
$mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$ *is accepting*	*iff* $\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e} \not\models \varphi$,
$inval_{\mathsf{yes}}(p, w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $p \in V(w\overrightarrow{e})$,
$inval_{\mathsf{no}}(p, w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $p \notin V(w\overrightarrow{e})$,
$in_{\mathsf{yes}}(w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $w\overrightarrow{e} \in \mathcal{M}\overrightarrow{\mathcal{E}}$,
$in_{\mathsf{no}}(w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $w\overrightarrow{e} \notin \mathcal{M}\overrightarrow{\mathcal{E}}$,
$access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \in R_a$,
$access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \notin R_a$,
$access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \in \bigcup_{j\leq i} (\bigcup_{a\in G} R_a)^j$
and $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathcal{M}\overrightarrow{\mathcal{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \notin \bigcup_{j\leq i} (\bigcup_{a\in G} R_a)^j$.

We do not prove the proposition here, since it mimics the semantics of **DELCK**. Instead, we first detail the meaning of the procedures, then give an example, and finally prove that the quantities given in each procedure in grey are strictly decreasing:

Propositional constructions in mc_{yes}. For $\varphi = p$, the procedure calls $inval_{\mathsf{yes}}(p, \mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e})$ that checks whether the valuation of $w\overrightarrow{e}$ contains p. This procedure has two cases: if $\overrightarrow{\mathcal{E}} = \epsilon$, meaning that there is no event model, we just check that $p \in V(w)$. Else, if $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}' :: \mathcal{E}$, we call $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{post}(e, p))$ to check whether p is in the valuation of $w\overrightarrow{e}$.

As expected, for $\varphi = \neg\psi$, the call switches from mc_{yes} to mc_{no}.

For $\varphi = \varphi_1 \vee \varphi_2$, $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$ consists in existentially choosing $i \in \{1, 2\}$ and then in calling $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi_i)$.

Knowledge operators in mc_{yes}. The call $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, K_a\psi)$ consists in universally choosing a valuation $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$, then checking two conditions such that $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$ must be true if they are verified. The first test is $(w\overrightarrow{e}, u\overrightarrow{f}) \in R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}$, thus calling $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$. This procedure just checks that $(w,u) \in R_a$ and $(e_i, f_i) \in R_a^{\mathcal{E}_i}$ for each $\mathcal{E}_i \in \overrightarrow{\mathcal{E}}$.

The second test is $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$, by calling Algorithm $in_{\mathsf{yes}}(u\overrightarrow{f}, \mathcal{M}\overrightarrow{\mathcal{E}})$. This procedure checks all preconditions.

Both tests are negated in the code of mc_{yes} because the condition is an implication.

Event model operators in mc_{yes} The call $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \langle\mathcal{E}, \mathsf{E}_0\rangle\psi)$ just chooses an event $e \in \mathsf{E}_0$, checks that its precondition is true with the call $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \mathrm{pre}(e))$ and goes on with the model checking by calling $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}} :: \mathcal{E}, w\overrightarrow{e} :: e, \psi)$.

Common knowledge operators in mc_{yes}. The call $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, C_G\psi)$ triggers $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, K_{a_1}...K_{a_i}\psi)$ for all possible choices of a multiset $\{a_1, \ldots, a_i\}$ of G. The semantics of operator C_G yields checking that $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$ does not reject, for all $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$ reachable from $w\overrightarrow{e}$ by iterating $\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}$. This is the purpose of Algorithm $access^*_{\mathsf{yes}}$. Regarding the size of the multiset $\{a_1, \ldots, a_i\}$, it is sufficient to consider at most $B_{\mathcal{M},\varphi}$, that is the smallest power of 2 greater than all $|\mathcal{M}\overrightarrow{\mathcal{E}}|$ that may appear during the evaluation of the formula: indeed, each iteration of $\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}$ adds at least one world into the set of reachable worlds from $w\overrightarrow{e}$, so it will necessarily stabilize within at most $B_{\mathcal{M},\varphi}$ steps. Additionally, it is important to require that after each iteration, the obtained world is in $\mathcal{M}\overrightarrow{\mathcal{E}}$, ensured by the call in_{yes} in $access^*_{\mathsf{yes}}$. To remain in the class APTIME, Algorithm $access^*_{\mathsf{yes}}$ relies on a divide and conquer method, alike the one shown in [Bal+14]; because in case $i \geq 2$, only one of the three recursive calls is universally chosen, it ensures that in an execution branch, the number of recursive calls is logarithmic in $B_{\mathcal{M},\varphi}$. It computes $\left(\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}\right)^i$ for all $i \leq B_{\mathcal{M},\varphi}$ and not just $\left(\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}\right)^{B_{\mathcal{M},\varphi}}$ because in the case $i = 1$, we either compute $\left(\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}\right)^0$ or $\left(\bigcup_{a\in G} R_a^{\mathcal{M}\overrightarrow{\mathcal{E}}}\right)^1$. Therefore, all $i \leq B_{\mathcal{M},\varphi}$ can be computed.

Example 11. *We take the example of $mc_{\mathsf{yes}}(\mathcal{M}, w, \langle\mathcal{E}, \mathsf{E}_0\rangle\neg C_G p)$ depicted in Figure 2.3. The procedure starts by choosing e in E_0. Then, we check that both* $\mathrm{pre}(e)$ *holds in w and that $\neg C_G p$ holds in we. Checking that $\neg C_G p$ holds in we leads to a negated configuration: we negate the fact that $C_G p$ holds in we. It is followed by a universal choice of $uf \in \mathcal{M}\mathcal{E}$. For each choice of $uf \in \mathcal{M}\mathcal{E}$, we progress in an existential configuration that checks that either uf is not a world of $\mathcal{M}\mathcal{E}$, either wf is not reachable from we by at most $B_{\mathcal{M},\varphi}$*

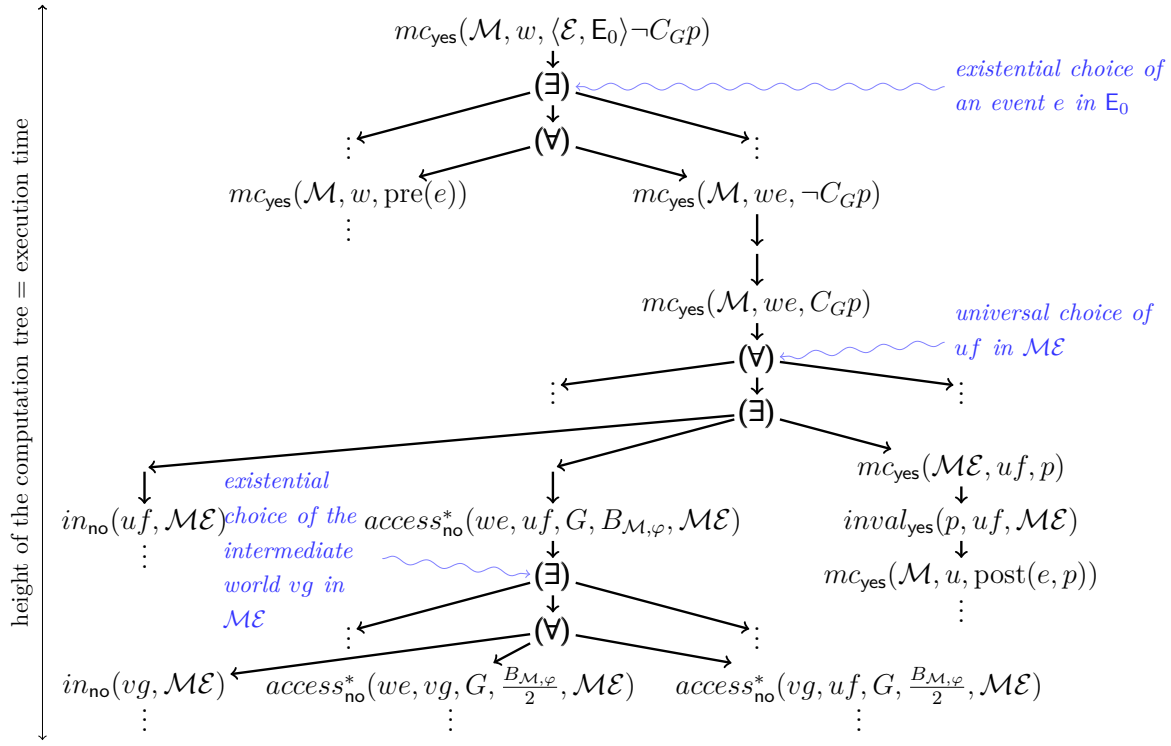


Figure 2.3: Computation tree rooted at $mc_{\mathsf{yes}}(\mathcal{M}, w, \langle \mathcal{E}, \mathsf{E}_0 \rangle \neg C_G p)$.

$\bigcup_{a \in G} R_a^{\mathcal{ME}}$*-steps or that* p *in* uf*. Checking that* p *holds in* uf *is performed by the call of* $inval_{\mathsf{yes}}(p, uf, \mathcal{ME})$*, which itself check that the postcondition* $\mathrm{post}(e, p)$ *holds in* u*.*

The quantities associated to each procedure call used for the proofs change a little bit compared to Definition 9 page 45. The first difference is $|C_G\varphi| := \log_2 B_{\mathcal{M},\varphi} + 1 + |\varphi|$. Since $B_{\mathcal{M},\varphi}$ is polynomial in the size of the biggest $\mathcal{M}\overrightarrow{\mathcal{E}}$, which itself is single-exponential in the size of $\mathcal{M}$ in the worst case, it means that $B_{\mathcal{M},\varphi}$ is single-exponential in $\mathcal{M}$, therefore adding $\log_2 B_{\mathcal{M},\varphi}$ in the new size of a formula $|\varphi|$ is polynomial in $|\mathcal{M}| + |\varphi|$ as given in Definition 9. Furthermore, in the algorithms, when $|\mathcal{M}\overrightarrow{\mathcal{E}}|$ is written, it is not the number of worlds of $|\mathcal{M}\overrightarrow{\mathcal{E}}|$ here but $|\mathcal{M}| + \sum_{\mathcal{E} \in \overrightarrow{\mathcal{E}}} |\mathcal{E}|$, which is clearly linear in $|\mathcal{M}| + |\varphi|$ at all times.

Lemma 2. *The quantities given in gray in Figures 2.1 and 2.2 are strictly decreasing along a branch of the computation tree (i.e. when a recursive call is made, the quantity associated is strictly lower).*

Proof. Let us discuss the following cases (the other ones are straightforward):

- The quantity for $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, C_G\varphi)$ is $|\mathcal{M}\overrightarrow{\mathcal{E}}|+|C_G\varphi|+1 = |\mathcal{M}\overrightarrow{\mathcal{E}}|+\log_2 B_{\mathcal{M},\varphi}+|\varphi|+1$ and is strictly greater than the quantity for $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, B_{\mathcal{M},\varphi}, \mathcal{M}\overrightarrow{\mathcal{E}})$, which is $|\mathcal{M}\overrightarrow{\mathcal{E}}| + \log_2 B_{\mathcal{M},\varphi}$.
- The quantity for $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \langle\mathcal{E}, \mathsf{E}_0\rangle\varphi)$ is $|\mathcal{M}\overrightarrow{\mathcal{E}}| + |\mathcal{E}| + |\varphi| + 1$ and is strictly greater than the quantity for $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \mathrm{pre}(e))$, which is $|\mathcal{M}\overrightarrow{\mathcal{E}}| + |\mathrm{pre}(e)| < |\mathcal{M}\overrightarrow{\mathcal{E}}| + |\mathcal{E}|$.
- The quantity for $inval_{\mathsf{yes}}(p, w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ is $|\mathcal{M}\overrightarrow{\mathcal{E}}'\mathcal{E}| = |\mathcal{M}\overrightarrow{\mathcal{E}}'| + |\mathcal{E}|$ and is strictly greater than the quantity for $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{post}(e, p))$ which $|\mathcal{M}\overrightarrow{\mathcal{E}}'|+\mathrm{post}(e, p)$.

□

Proposition 3. *$mc_{\mathsf{yes}}(\mathcal{M}, w, \Phi)$ is executed in polynomial time in the size of the input $(\mathcal{M}, w, \Phi)$.*

Proof. The time is bounded the height of the computation tree rooted in $mc_{\mathsf{yes}}(\mathcal{M}, w, \Phi)$. Thanks to Lemma 2, the height of the computation tree is bounded by the quantity associated to $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$, that is $|\mathcal{M}| + |\varphi|$. Recall that this quantity is *not* the size of the input $(\mathcal{M}, w, \varphi)$: for instance the weight of C_G-modalities is $\log_2 B_{\mathcal{M},\varphi}$. However this quantity is polynomial in the the size of the input $(\mathcal{M}, w, \varphi)$.

At each node of the computation tree, the computation performed in a single node is polynomial. For instance, the instruction "**(∀)** choose $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$" consists in choosing each bit of $u\overrightarrow{f}$, thus is polynomial in the size of the input.

To conclude, the execution time on each branch in the computation tree is polynomial.

□

2.2 Complexity of the satisfiability problem of DELCK

The satisfiability problem for **DELCK** is defined as follows.

Definition 24 (Satisfiability problem)**.**

- ***Input:*** *a* **DELCK***-formula φ.*
- ***Output:*** *yes if there exists a Kripke model $\mathcal{M}$ and a world $w \in \mathcal{M}$ such that $\mathcal{M}, w \models \varphi$, no otherwise.*

In this section, we prove the following upper bound result:

Theorem 6. *The satisfiability problem of* **DELCK** *is in* 2-Exptime.

We first prove the upper bound result (the membership) and then the lower bound result (the hardness).

2.2.1 Upper Bound of Satisfiability

In order to prove Theorem 6, we will proceed as for proving that Propositional Dynamic Logic is in Exptime and use the method of Pratt [Pra80], but we will simulate tableau method rules of the same kind that in [AS13]. Intuitively, we define a canonical Kripke model, where the worlds are called *Hintikka sets* and the model itself is called the *Hintikka structure*. The algorithm will then construct the Hintikka structure and remove any inconsistent world, and the resulting model will be a model of the input formula if a world satisfying this formula exists.

To ease the reading, we will w.l.o.g consider that formulas are in negative normal form, defined as follows.

Definition 25 (Negative normal form of **DELCK**)**.** *The syntax of negative normal form of* **DELCK** *is defined as follows.*

$$\varphi \ ::= \ \top \mid \bot \mid p \mid \neg p \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid \hat{K}_a\varphi \mid C_G\varphi \mid \hat{C}_G\varphi \mid \langle \mathcal{E}, \mathsf{E}_0\rangle\varphi \mid [\mathcal{E}, \mathsf{E}_0]\varphi$$

Intuitively, it means that negations are pushed in front of atomic propositions, and we will use all connectives $\vee$, $\wedge$, K_a, $\hat{K}_a$, C_a, $\hat{C}_a$, $\langle \mathcal{E}, \mathsf{E}_0\rangle$, $[\mathcal{E}, \mathsf{E}_0]$. The following definition gives an inductive function nnf that transforms a formula φ into an equivalent formula $nnf(\varphi)$ in negative normal form.

Definition 26 (Function nnf)**.** *The function nnf is defined as follows:*

- $nnf(\top) = nnf(\neg\bot) = \top$;
- $nnf(\bot) = nnf(\neg\top) = \bot$;
- $nnf(p) = p$;
- $nnf(\neg p) = \neg p$;
- $nnf(\varphi_1 \vee \varphi_2) = nnf(\varphi_1) \vee nnf(\varphi_2)$;
- $nnf(\neg(\varphi_1 \vee \varphi_2)) = nnf(\neg\varphi_1) \wedge nnf(\neg\varphi_2)$;

- $nnf(\varphi_1 \wedge \varphi_2) = nnf(\varphi_1) \wedge nnf(\varphi_2)$;
- $nnf(\neg(\varphi_1 \wedge \varphi_2)) = nnf(\neg\varphi_1) \vee nnf(\neg\varphi_2)$;
- $nnf(K_a\varphi) = K_a nnf(\varphi)$;
- $nnf(\neg K_a\varphi) = \hat{K}_a nnf(\neg\varphi)$;
- $nnf(\hat{K}_a\varphi) = \hat{K}_a nnf(\varphi)$;
- $nnf(\neg\hat{K}_a\varphi) = K_a nnf(\neg\varphi)$;
- $nnf(C_G\varphi) = C_G nnf(\varphi)$;
- $nnf(\neg C_G\varphi) = \hat{C}_G nnf(\neg\varphi)$;
- $nnf(\hat{C}_G\varphi) = \hat{C}_G nnf(\varphi)$;
- $nnf(\neg\hat{C}_G\varphi) = C_G nnf(\neg\varphi)$;
- $nnf(\langle\mathcal{E}, \mathsf{E}_0\rangle\varphi) = \langle\mathcal{E}, \mathsf{E}_0\rangle nnf(\varphi)$;
- $nnf(\neg\langle\mathcal{E}, \mathsf{E}_0\rangle\varphi) = [\mathcal{E}, \mathsf{E}_0] nnf(\neg\varphi)$;
- $nnf([\mathcal{E}, \mathsf{E}_0]\varphi) = [\mathcal{E}, \mathsf{E}_0] nnf(\varphi)$;
- $nnf(\neg[\mathcal{E}, \mathsf{E}_0]\varphi) = \langle\mathcal{E}, \mathsf{E}_0\rangle nnf(\neg\varphi)$;

The *negation* of a formula φ in negative normal form corresponds to $nnf(\neg\varphi)$, the formula in negative normal form obtained by negating all connectives. The dynamic modal depth of a formula φ, noted $dmd(\varphi)$, is the modal depth by only counting the dynamic operators. E.g. the dynamic modal depth of $K_a[\mathcal{E}, \mathsf{E}_0][\mathcal{E}', \mathsf{E}_0']p \wedge [\mathcal{E}', \mathsf{E}_0']C_G q$ is 2. From now on, in this section, we also suppose that there is a unique event model $\mathcal{E}$, that is the union of all event model appearing in φ. The set E_0 will then give the relevant part of the event model $\mathcal{E}$ we consider, and so the modality $\langle\mathcal{E}, \mathsf{E}_0\rangle$ becomes $\langle\mathsf{E}_0\rangle$. We now define the notion of closure of a formula.

Definition 27. *The* closure[1] *of formula φ is the set $Cl(\varphi)$ that contains elements $in_{\overrightarrow{e}}$ and $(\overrightarrow{e}, \psi)$ where $\overrightarrow{e}$ is a sequence of events in $\mathcal{E}$ of length at most $dmd(\varphi)$, and ψ is a subformula (or negation) of φ or a subformula (or negation) of a precondition or postcondition in $\mathcal{E}$, under the condition that $dmd(\varphi) + |\overrightarrow{e}| \leq dmd(\varphi)$.*

The intended meaning of $in_{\overrightarrow{e}}$ is that the current world is compatible with the sequence of events $\overrightarrow{e}$. The intended meaning of $(\overrightarrow{e}, \varphi)$ is that formula φ is true in the current world after having executed the sequence of events $\overrightarrow{e}$.

Example 12. *Let us take the event model $\mathcal{E}$ of Figure 1.5 page 43 and formula $\varphi := [e]K_a[f]q$. The closure $Cl(\Phi)$ is the set $\{in_\epsilon, in_e, in_{ef}, in_f, in_{fe}, (\epsilon, \varphi), (\epsilon, K_a[f]q), (e, K_a[f]q), \dots\}$.*

Proposition 4. *The size of the closure of φ is exponential in $|\varphi|$.*

1. The definition given here contains 'too many' formulas. We could have given a much more thorough definition, but the definition would have been more complicated to understand and the closure would have had the same asymptotic size.

Proof. There is a direct correspondence between a subformula of φ and a node in the syntactic tree of φ. Therefore, the number of subformulas of φ is in $O(|\varphi|)$. The number of possible ψ is then bounded by $O(|\varphi|)$ (the size of φ is the number of memory cells needed to write down φ, all the information of the event model $\mathcal{E}$ included). The number of possible sequences $\overrightarrow{e}$ is $|\mathcal{E}|^{dmd(\varphi)}$, thus exponential in $|\varphi|$. □

A Hintikka set (see Definition 28) is a maximal subset of $Cl(\Phi)$ that is consistent with respect to propositional logic (points 2-4), common knowledge reflexivity (point 5), dynamic operators (point 6-7), executability of events (point 8-9) and postconditions (point 10).

Definition 28. *A* Hintikka set h *over* $Cl(\Phi)$ *is a subset of* $Cl(\Phi)$ *that satisfies:*

(1)	*If* $(\overrightarrow{e}, \varphi) \in h$	*then*	$in_{\overrightarrow{e}} \in h$;
(2)	$(\overrightarrow{e}, \varphi \wedge \psi) \in h$	*iff*	$(\overrightarrow{e}, \varphi) \in h$ *and* $(\overrightarrow{e}, \psi) \in h$;
(3)	$(\overrightarrow{e}, \varphi \vee \psi) \in h$	*iff*	$(\overrightarrow{e}, \varphi) \in h$ *or* $(\overrightarrow{e}, \psi) \in h$;
(4)	*If* $in_{\overrightarrow{e}} \in h$	*then*	$(\overrightarrow{e}, \varphi) \in h$ *xor* $(\overrightarrow{e}, nnf(\neg\varphi)) \in h$;
(5)	*If* $(\overrightarrow{e}, C_G\varphi) \in h$	*then*	$(\overrightarrow{e}, \varphi) \in h$;
(6)	$(\overrightarrow{e}, \langle \mathsf{E}_0 \rangle \varphi) \in h$	*iff*	*there exists* $e \in \mathsf{E}_0$ *s.t.* $in_{\overrightarrow{e}::e} \in h$ *and* $(\overrightarrow{e}::e, \varphi) \in h$;[2]
(7)	$(\overrightarrow{e}, [\mathsf{E}_0]\varphi) \in h$	*iff*	*for all* $e \in \mathsf{E}_0$, *we have* $in_{\overrightarrow{e}::e} \in h$ *implies* $(\overrightarrow{e}::e, \varphi) \in h$;
(8)	$in_\epsilon \in h$;		
(9)	$in_{\overrightarrow{e}::e} \in h$	*iff*	$in_{\overrightarrow{e}} \in h$ *and* $(\overrightarrow{e}, \mathrm{pre}(e)) \in h$;
(10)	$(\overrightarrow{e}::e, p) \in h$	*iff*	$(\overrightarrow{e}, \mathrm{post}(e)(p)) \in h$.

Point (1) means that if a Hintikka set contains $(\overrightarrow{e}, \varphi)$, then it means that $\overrightarrow{e}$ should be executable (in the intuitive world represented by the Hintikka set). Point (4) means that Hintikka sets are consistent. Point (5) says that if φ is common knowledge then φ is true. Points (6) and (7) mimics the truth condition given in Definition 8. Point (8) means the empty sequence of events ϵ is always executable. Point (9) means that $\overrightarrow{e}::e$ is executable iff $\overrightarrow{e}$ is executable and the precondition of e holds after having executed $\overrightarrow{e}$. Point (10) means that the truth of atomic proposition p after a non-empty sequence $\overrightarrow{e}::e$ of events is given by the truth of its postcondition before the last event e.

Now, we define the following structure that takes care of the consistency of the box modalities K_a, C_G.

Definition 29. *The* Hintikka structure *for* φ *is* $\mathcal{H} := (H, (R_a)_{a \in Ag})$ *where:*

2. We explicitly mentioned $in_{\overrightarrow{e}::e} \in h$ for uniformity with the semantics. However, note that it is implied by point (1).

function is**DELCK**-sat?(φ)
  Compute the Hintikka structure $\mathcal{H} := (H, (R_a)_{a \in Ag})$ for φ
  repeat
    Remove any Hintikka set h from $\mathcal{H}$ if
    ($\hat{K}_a$) either there is $(\overrightarrow{e}, \hat{K}_a\psi) \in h$ but no $h' \in R_a(h)$ with $(\overrightarrow{e}', \psi) \in h$ with $\overrightarrow{e} \rightarrow^a \overrightarrow{e}'$ and $in_{\overrightarrow{e}'} \in h'$;

    ($\hat{C}_G$) or there is $(\overrightarrow{e}, \hat{C}_G\psi) \in h$ but no path $h = h_0 \rightarrow^{a_1} h_1 \dots h_k$ and no path $\overrightarrow{e} = \overrightarrow{e}^{(0)} \rightarrow^{a_1} \overrightarrow{e}^{(1)} \dots \rightarrow^{a_k} \overrightarrow{e}^{(k)}$ such that $(\overrightarrow{e}^{(k)}, \psi) \in h_k$ and $a_1, \dots, a_k \in G$ and $in_{\overrightarrow{e}^{(i)}} \in h_i$.
  until no more Hintikka sets are removed
  if there is still a Hintikka set in $\mathcal{H}$ containing (ϵ, φ) **then accept else reject**
endFunction

Figure 2.4: Algorithm for the satisfiability problem of a **DELCK**-formula φ.

- *H is the set of all possible Hintikka sets over $Cl(\Phi)$;*
- *hR_ah' if the two following conditions holds:*

  (K_a) for all $(\overrightarrow{e}, K_a\varphi) \in h$ we have $(\overrightarrow{e}', \varphi) \in h'$ for all $\overrightarrow{e}'$ such that $\overrightarrow{e} \rightarrow^a \overrightarrow{e}'$ and $in_{\overrightarrow{e}'} \in h'$,

  (C_G) for all $(\overrightarrow{e}, C_G\varphi) \in h$ we have $(\overrightarrow{e}', C_G\varphi) \in h'$ for all $\overrightarrow{e}'$ such that $\overrightarrow{e} \rightarrow^a \overrightarrow{e}'$ with $a \in G$ and $in_{\overrightarrow{e}'} \in h'$.

The size of the Hintikka structure is double-exponential in $|\varphi|$, since there are a double-exponential number of different Hintikka sets. We finish by giving the algorithm is**DELCK**-sat?(see Figure 2.4) whose **repeat...until** loop takes care of the consistency of diamond modalities, $\hat{K}_a, \hat{C}_a$. The algorithm starts with the full Hintikka structure. Points $(\hat{K}_a), (\hat{C}_a)$ remove worlds where $\hat{K}_a\psi$ and $\hat{C}_a\psi$ have no appropriate ψ-successor. We write $\overrightarrow{e} \rightarrow^a \overrightarrow{e}'$ if for all $(\overrightarrow{e}_i, \overrightarrow{e}'_i) \in R_a^{\mathcal{E}}$. Actually, the algorithm decides in double-exponential time whether a **DELCK**-formula is satisfiable (Propositions 5 and 6).

Proposition 5. *Algorithm is***DELCK***-sat? of Figure 2.4 runs in double-exponential time in $|\varphi|$.*

Proof. The computation of $\mathcal{H}$ can be performed by brute-force: enumerate all subsets of $Cl(\Phi)$ and discard those which do not satisfy all conditions (1)-(10) of Definition 28. Compute R_a according to Definition 29. The loop is repeated at most the number of

Hintikka sets in $\mathcal{H}$, that is $O(2^{2^{|\varphi|}})$ times, since at least one Hintikka set is removed or we exit the loop. Both tests $(\hat{K}_a)$ and $(\hat{C}_G)$ can be performed by depth-first search algorithm running in polynomial time in the size of the graph, that is of size double-exponential in $|\varphi|$. □

Proposition 6. *φ is* **DELCK***-satisfiable iff is***DELCK***-sat? accepts φ.*

Proof. ($\Rightarrow$) Let $\mathcal{M}, w$ such that $\mathcal{M}, w \models \varphi$. Given a world u, we note $h(u)$ the Hintikka set obtained by taking $in_{\overrightarrow{e}}$ if $\overrightarrow{e}$ is executable in u and $(\overrightarrow{e}, \psi)$ if ψ holds in $u, \overrightarrow{e}$. We show that no Hintikka set $h(u)$ is removed from $\mathcal{H}$. In particular, $h(w)$ is not removed, and contains (ϵ, φ) so the algorithm is**DELCK**-sat? accepts φ.

($\Leftarrow$) Suppose is**DELCK**-sat? accepts φ. We construct a model $\mathcal{M} = (W, (R_a)_{a \in Ag}, V)$ as follows:

- W is the set of Hintikka sets that remain in the structure at the end of the algorithm;
- R_a is the relation for agent a at the end of the algorithm;
- $V(h) = \{p \in AP \mid (\epsilon, p) \in h\}$.

The proof finishes by proving the following lemma:

Lemma 3. *(truth lemma) The properties $\mathcal{P}(in_{\overrightarrow{e}})$ and $\mathcal{P}((\overrightarrow{e}, \varphi))$ defined below hold:*

- *$\mathcal{P}(in_{\overrightarrow{e}})$: for all $h \in W$, $in_{\overrightarrow{e}} \in h$ iff $\overrightarrow{e}$ is executable in $\mathcal{M}, h$;*
- *$\mathcal{P}((\overrightarrow{e}, \varphi))$: for all $h \in W$, $(\overrightarrow{e}, \varphi) \in h$ iff $\mathcal{M} \otimes \mathcal{E}^{|\overrightarrow{e}|}, (h, \overrightarrow{e}) \models \varphi$.*

Proof. The proof is performed by induction. The quantity for $in_{\overrightarrow{e}}$ is $n|\mathcal{E}|$. The quantity for $(\overrightarrow{e}, \varphi)$ is $n|\mathcal{E}| + |\varphi|$ where n is the length of $\overrightarrow{e}$.

□

We conclude by applying the truth lemma (Lemma 3 to the Hintikka set h that contains (ϵ, φ)) and we obtain that $\mathcal{M}, h \models \varphi$. □

Remark 1. *The 2EXPTIME upper bound also holds for the satisfiability problem of* **DELCK** *in* **S5** *Kripke models. We proceed as in [HM92] (p. 358). We add the following clauses to Definition 28:*

$$\textit{(5')} \quad \textit{If } (\overrightarrow{e}, K_a\varphi) \in h \quad \textit{then} \quad (\overrightarrow{e}, \varphi) \in h;$$

We add the following clause in the definition of R_a in Definition 29:

$$\textit{for all } \overrightarrow{e} \to^a \overrightarrow{e}', \textit{ if } in_{\overrightarrow{e}} \in h \textit{ and } in_{\overrightarrow{e}'} \in h' \textit{ then } (\overrightarrow{e}, K_a\varphi) \in h \textit{ iff } (\overrightarrow{e}', K_a\varphi) \in h'.$$

2.2.2 Lower Bound of Satisfiability

The aim of this section is to prove the following theorem.

Theorem 7. *The satisfiability problem of* **DELCK** *is* 2-Exptime*-hard.*

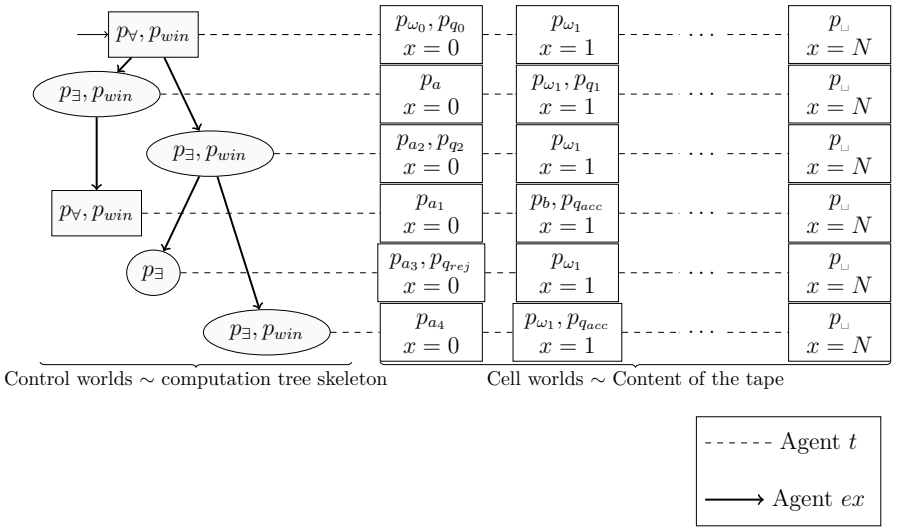


Figure 2.5: (Expected) Kripke model that represents the computation tree of M on the input instance ω.

Let us consider any 2-Exptime decision problem L. As Aexpspace = 2-Exptime [CS76], it is decided by an alternating Turing machine M that runs in exponential space. W.l.o.g we suppose that all executions halt[3] and no state is a negated state. We will define a polynomial reduction tr from L to the satisfiability problem of **DELCK**, that is tr will be computable in polynomial time, and ω is a positive instance of L if and only if $tr(\omega)$ is a satisfiable **DELCK**-formula.

The idea of $tr(\omega)$ is to enforce an expected form of a Kripke model as shown in Figure 2.5 that represents the computation tree of M starting with ω on the tape. The cursor of the machine remains in the N-first cell portion of the tape, where N is exponential in $|\omega|$. We define $N_0 = \log_2(N)$ for the rest of the section. N_0 is polynomial in $|\omega|$.

We introduce two agents: agent ex for the transitions in the computation tree and agent t for the linear structure of tapes. A configuration of the Turing machine is represented by a sequence of worlds linked by agent t: one so-called *control world* followed by *cell worlds*.

3. If not, we add a double exponential counter to the machine and we abort the execution after a double exponential number of steps.

- The control world contains the type of the configuration: existential (resp. universal) if $p_\exists$ (resp. $p_\forall$) is true. A special atomic proposition p_{win} tags control worlds that correspond to winning configurations for player $\exists$.
- Cell worlds represent the cells of the tape and form a linear structure. They are indexed by x from $x = 0$ (left-most cell) to $x = N$ (right-most cell). In each cell world, p_a is true means that the corresponding cell contains letter $a \in \Gamma$. A proposition of the form p_q being true means that the cursor is at that cell and the current state is $q \in Q$.

Besides atomic propositions $p_\exists, p_\forall$, $p_a, a \in \Gamma$ and $p_q, q \in Q$, we also consider the list of atomic propositions for the bits of the cell index x: $x_1, \ldots, x_{N_0}$. We also consider another such list for another cell index v: $v_1, \ldots, v_{N_0}$. The index v will be used to compare cell worlds of tapes of a configuration and a successor configuration different tapes during transitions.

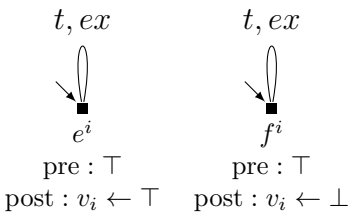


Figure 2.6: Multi-pointed event models $(\mathcal{E}^i, \mathsf{E}_0^i)$.

The definition of $tr(\omega)$ needs multi-pointed event models $(\mathcal{E}^i, \mathsf{E}_0^i)$ given in Figure 2.6 non-deterministically and publicly choose the i^{th} bit of value v. We also consider Boolean formulas $x \leq v$, $x = v$, $x = v - 1$ and finally $K_t x = x + 1$ (the value of $K_t x_0 \ldots K_t x_{N_0}$ is equal to $x+1$). We define the abbreviation $[\texttt{choosev}] = [\mathcal{E}^0, \mathsf{E}_0^0] \ldots [\mathcal{E}^{N_0}, \mathsf{E}_0^{N_0}]$. Technically, it corresponds to non-deterministically choosing and publicly announcing a value for v.

Definition 30. *Formula $tr(\omega)$ is the conjunction of the formulas shown in Table 2.1.*

In formulas of Table 2.1, common knowledge operators C_{ex} and $\hat{C}_{ex}$ are used to talk about any control world, while C_t and $\hat{C}_t$ talk about any cell world.

Importantly, notice that with **DELCK**, it is impossible to force the each world to have exactly one successor. Thus in general, the expected Kripke model is not as depicted in Figure 2.5. Instead, we ensure that cell worlds of depth k have the value $x = k$. We use

Valuations for control worlds

1.	$C_{ex}(x = 0)$	$x = 0$ holds in all control worlds.
2.	$C_{ex} \quad (p_\exists \leftrightarrow \bigvee_{q\|g(q)=\exists} \hat{C}_t p_q)$ $\wedge \quad (p_\forall \leftrightarrow \bigvee_{q\|g(q)=\forall} \hat{C}_t p_q)$	$p_\forall$ and $p_\exists$ match the type of the state on the tape.
3.	$C_{ex}\left(\bigwedge_{a\in\Gamma} \neg p_a \wedge \bigwedge_{q\in Q} \neg p_q\right)$	Every p_a or p_q is false.

Winning condition

4.	$C_{ex}((\hat{C}_t p_{q_{acc}}) \rightarrow p_{win}) \wedge (\hat{C}_t p_{q_{rej}}) \rightarrow \neg p_{win}))$	If the current state is q_{acc} the world is marked as winning, if the current state is q_{rej} it is marked as losing.
5.	$C_{ex}(\quad (p_\forall \wedge (C_t(\neg p_{q_{acc}} \wedge \neg p_{q_{rej}})))$ $\rightarrow (p_{win} \leftrightarrow K_{ex} p_{win}))$	If the current state is not q_{acc} nor q_{rej} and is universal, the world is marked as winning if all successor worlds are marked as winning.
6.	$C_{ex}(\quad (p_\exists \wedge (C_t \neg p_{q_{acc}} \wedge \neg p_{q_{rej}})))$ $\rightarrow (p_{win} \leftrightarrow \hat{K}_{ex} p_{win}))$	If the current state is not q_{acc} nor q_{rej} and is existential, the world is marked as winning if one successor world is marked as winning.

Tape

7.	$C_{ex} K_t(x = 0)$	The cell index of the left-most cell is 0.
8.	$C_{ex} K_t C_t \left(\bigwedge_{i=0}^{N_0} (K_t x_i \vee K_t \neg x_i)\right)$	On any tape world, the value of x is the same in all successors
9.	$C_{ex} K_t C_t (K_t x = x + 1)$	On any tape world, the value of x is incremented by 1 on all successors.
10.	$C_{ex} K_t C_t \left(\oplus_{a\in\Gamma} p_a\right)$	On any tape world, only one p_a is true and represent the current letter on the cell.
11.	$C_{ex} \hat{C}_t \left(\oplus_{q\in Q} p_q\right)$	On any tape, somewhere only one p_q is true
12.	$C_{ex} C_t \bigwedge_{q\in Q} \left(p_q \rightarrow C_t \bigwedge_{q'\in Q} \neg p_{q'}\right)$	Anywhere, if p_q is true then no $p_{q'}$ is true anywhere on the rest of the tape.

Transitions

We define here $\varphi_{(q,a,q',b,d)} = C_t((x = v \rightarrow p_b \wedge \neg p_q) \wedge (x = v + d \rightarrow p_{q'}))$

13.	$[\texttt{choosev}] C_{ex} \bigwedge_{a\in\Gamma} \hat{C}_t (p_a \wedge \bigwedge_{q\in Q} \neg p_q \wedge$ $x = v) \rightarrow K_{ex} C_t (x = v \rightarrow p_a)$	On the tape, if no p_q is true and p_a is true, then at the same position on the successors' tapes, p_a is true.
14.	$\bigwedge_{(q,a,q',b,d)\in\delta} [\texttt{choosev}] \left(C_{ex} (\hat{C}_t (p_q \wedge p_a \wedge$ $x = v) \rightarrow \hat{K}_{ex} \varphi_{(q,a,q',b,d)})\right)$	If there is a transition it must be present on the model.
15.	$[\texttt{choosev}] \bigwedge_{a\in\Gamma} \bigwedge_{q\in Q} C_{ex} \left(\hat{C}_t (p_q \wedge p_a$ $\wedge x = v) \rightarrow K_{ex} \bigvee_{(q,a,q',b,d)\in\delta} \varphi_{(q,a,q',b,d)}\right)$	In every world, any ex-successor must correspond to a transition.

Initial configuration

16.	$\bigwedge_{i=0}^{\|\omega\|-1} C_t((x = i) \rightarrow p_{\omega(i)})$	The letters of the initial word are on the initial tape.
17.	$C_t((x \geq \|\omega\|) \rightarrow p_\sqcup)$	Cells of index $\|\omega\|$ are blank.
18.	$K_t p_{q_0}$	Head in the left-most cell. Initially in the initial state.
19.	p_{win}	The initial control world is winning.

Table 2.1: Clauses of **DELCK**-formula $tr(\omega)$.

formula (8) that imposes the value of x to be the same in all successor cell worlds, and formula (9) saying that everywhere, $K_t x = x + 1$. Formula (10) states that only one p_a is true in each cell world, formula (11) states that only one p_q is true in some cell world, and formula (12) states that if p_q is true in a cell world, then no p'_q is true in all the t-successors. Formulas (16), (17) and (18) define the initial tape. Transitions are ensured by formulas (13) to (15). These formulas automatically ensure that several cell worlds with the same index x have the same valuation over $x, p_a, a \in \Gamma, p_q, q \in Q$.

Formulas that handle transitions use integer v to pinpoint a cell index in the tape. It is used in formula (13) to tell that when the cursor is not in a cell world, then the letter remains the same during any transition. It is also used in formula (14) to check the existence of all compatible transitions and in formula (15) to check that all successor control worlds and their tapes correspond to a transition.

Proposition 7. *$tr(\omega)$ is satisfiable if and only if ω is a positive instance of L.*

The lower bound given in Theorem 7 still holds for the variant of the satisfiability problem where we require the model to be **S5**, that is, epistemic relations, to be equivalence relations.

2.3 Example: blind tic tac toe

In this section, we express in **DEL** the existence of a uniform winning strategy in the blind tic tac toe game. We begin by giving the rules of the game, then define some useful macros for defining the event models, give the models and finally the formula.

For a recall on uniform winning strategies, the reader may refer to Appendix B. We do not model the game here, just explain the rules.

2.3.1 Rules of the game

Blind tic tac toe is a two player game where the goal is to fill a row, a column or a diagonal with only $\circ$ (for Player $\circ$) or only $\times$ (for player $\times$). The difference with basic tic tac toe is that players do not see the actions of the other Player.

The game begins with Player $\circ$. At each turn, the current player attempts to play his token on a position in the grid where he did not put a token before. Then one of the two following cases occur.

- If there is nothing on the cell the token is now played on the cell.
- Otherwise, an external referee informs the player that the move is invalid. The player can then attempt to play somewhere else.

The adversary does not see where the token is played, but hears the referee. Therefore he knows whether the Player played a token or was given an information.

The game stops when a row, a column or a diagonal is filled by tokens of the same player.

Figure 2.7 gives an example of play. Each player maintains his grid according to his previous moves and the information he gained. Notice that until turn 5, each move is played on an empty cell, therefore the players do not gain information on their opponent's previous moves. At turn 5, Player ∘ attempts a move that is invalid, he thus gains a information that a × is present in the cell. Therefore, he has the right to play another move. The game stops at turn 6 because Player × completes a diagonal, and is thus the winner.

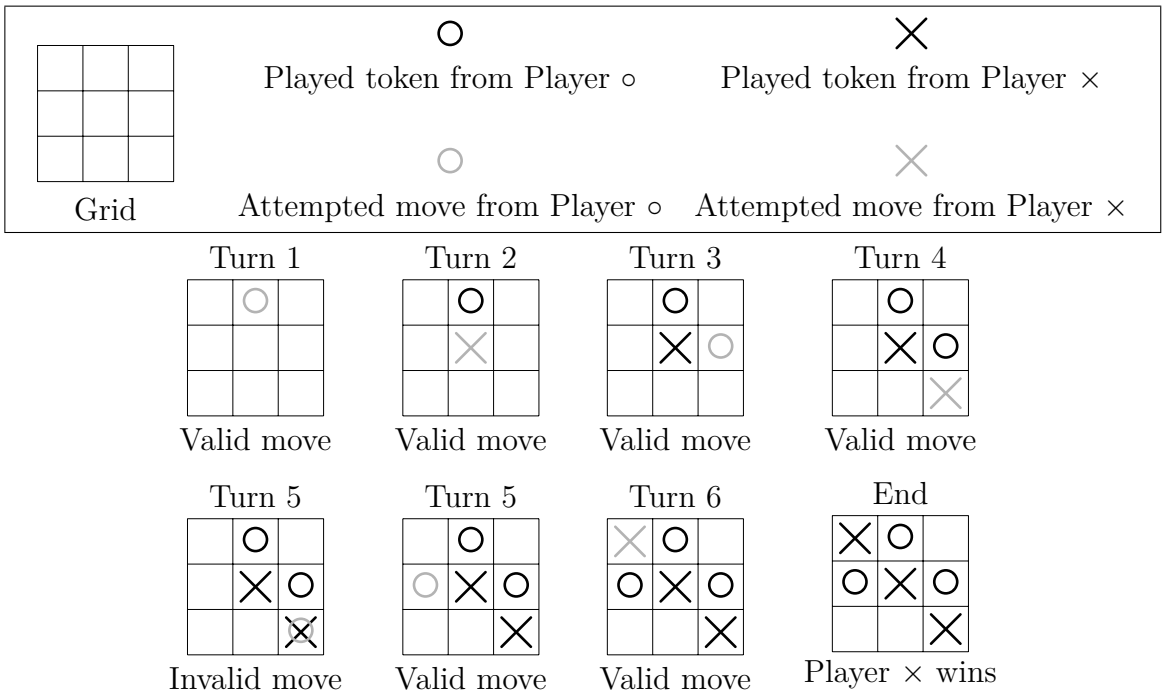


Figure 2.7: Example of a play of blind tic tac toe .

We now define the problem we want to tackle.

Definition 31 (Blind tic tac toe problem)**.** *It is defined as follows.*

- *Input: an integer n (the size of the grid);*
- *Output: does there exist a uniform winning strategy for player $\circ$ in blind tic tac toe in a $n \times n$ grid.*

We now describe the reduction of the tic tac toe problem to the model checking problem of DEL.

2.3.2 Atomic propositions and macros

In this subsection, we consider a fixed integer n. The agents are $Ag = \{\circ, \times\}$. We assign coordinates to each cell. For instance in Figure 2.8, the $\circ$ token is in cell (1,2) and the $\times$ token in cell (3,1).

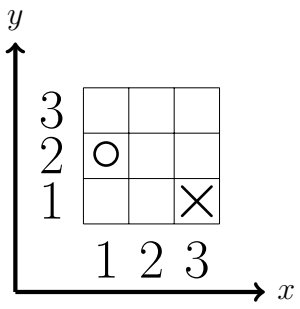


Figure 2.8: Example of coordinates for blind tic tac toe

We begin by defining the set of atomic propositions AP needed for defining the problem:

Definition 32 (Atomic propositions for blind tic tac toe)**.** *We define* $AP = \{p^z_\circ, z \in \{1,\dots,n\}^2\} \cup \{p^z_\times, z \in \{1,\dots,n\}^2\} \cup \{c^z_\circ, z \in \{1,\dots,n\}^2\} \cup \{c^z_\times, z \in \{1,\dots,n\}^2\} \cup \{t_\circ, t_\times, p_{end}\}$.

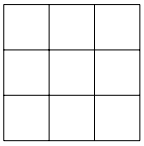
The meaning of the atomic proposition is the following.

- z is a pair (x, y) of integers in $\{1,\dots,n\}$ (position in the grid);
- $p^z_\circ$ (resp. $p^z_\times$) is true iff there is a $\circ$ token (resp. a $\times$ token) in z;

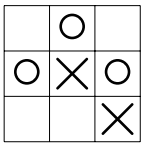
- $c^z_\circ$ (resp. $c^z_\times$) is true iff Player $\circ$ (resp. Player $\times$) has chosen to do a move in position z (such atomic propositions will be used later in event models);
- $t_\circ$ (resp. $t_\times$) is true iff it is Player $\circ$'s turn (resp. Player $\times$'s turn) to play;
- p_{end} is true iff the game has ended.

A state of the game is then represented by a world uniquely characterized by a valuation on AP.

Example 13. *Consider for instance the empty grid:*



The empty grid is represented by the world with valuation $\{t_\circ\}$*, since no* $p^z_\circ$ *and no* $p^z_\times$ *is true, it is Player* $\circ$*'s turn to play and since the game has not ended yet. Consider now the following grid:*



It is represented by the world with valuation $\{p^{(1,2)}_\circ, p^{(2,3)}_\circ, p^{(3,2)}_\circ, p^{(2,2)}_\times, p^{(3,1)}_\times, t_\times\}$.

We now define a formula $\varphi^{win}_\circ$ (resp. $\varphi^{win}_\times$) to express that $\circ$ (resp. $\times$)wins the game.

Definition 33 (Winning condition for blind tic tac toe)**.** *The formula* $\varphi^{win}_\circ$ *is defined as follows.*

$$\varphi^{win}_\circ = \underbrace{\left(\bigvee_{x=1}^{n}\bigwedge_{y=1}^{n} p^{(x,y)}_\circ\right)}_{column} \vee \underbrace{\left(\bigvee_{y=1}^{n}\bigwedge_{x=1}^{n} p^{(x,y)}_\circ\right)}_{row} \vee \underbrace{\left(\bigwedge_{x=1}^{n} p^{(x,x)}_\circ\right) \vee \left(\bigwedge_{x=1}^{n} p^{(x,n+1-x)}_\circ\right)}_{diagonal}$$

Formula $\varphi^{win}_\times$ *is defined similarly by replacing* $\circ$ *by* $\times$.

Notice that formula $\varphi^{win}_\circ$ has quadratic size compared to n.

2.3.3 Reduction into DEL

We now describe the initial Kripke model, the event models and the formula characterizing the existence of a uniform winning strategy in blind tic tac toe.

Initial Kripke model

The initial Kripke model consists in a unique world, the empty grid (Figure 2.9).

Definition 34 (Initial Kripke model for blind Tic Tac Toe)**.** *Let* $\mathcal{M}_0 = (W, (R_a)_{a\in Ag}, V)$ *with:*

- $W = \{w_0\}$*;*
- $R_\circ = R_\times = \{(w_0, w_0)\}$*;*
- $V(w_0) = \{t_\circ\}$.

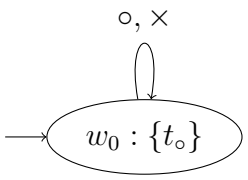


Figure 2.9: The initial Kripke model for blind tic tac toe.

Event models for choosing a position

A Player's turn is divided in two event models: one choosing a position to play a token on and one for the effect of the move. Figure 2.10 shows an example of the event model $(\mathcal{E}_\circ^{(1,1)}, \mathsf{E}_\circ^{(1,1)})$ for Player $\circ$ to choose the cell $(1, 1)$ on a 2×2 grid.

In $(\mathcal{E}_\circ^z, \mathsf{E}_\circ^z)$, there is one event $e^{z'}_{\circ,choose}$ for each possible position z', for "Player $\circ$ chooses to play in position z'". Player $\times$ does not know the position, so he imagines as possible any event $e^{z'}_{\circ,choose}$.

Definition 35 (Event model $(\mathcal{E}_\circ^z, \mathsf{E}_\circ^z)$)**.** *The event model* $\mathcal{E}_\circ^z = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *is defined as follows:*

- $\mathsf{E} = \{e^{z'}_{\circ,choose}, z' \in \{1, \ldots, n\}^2\}$*;*

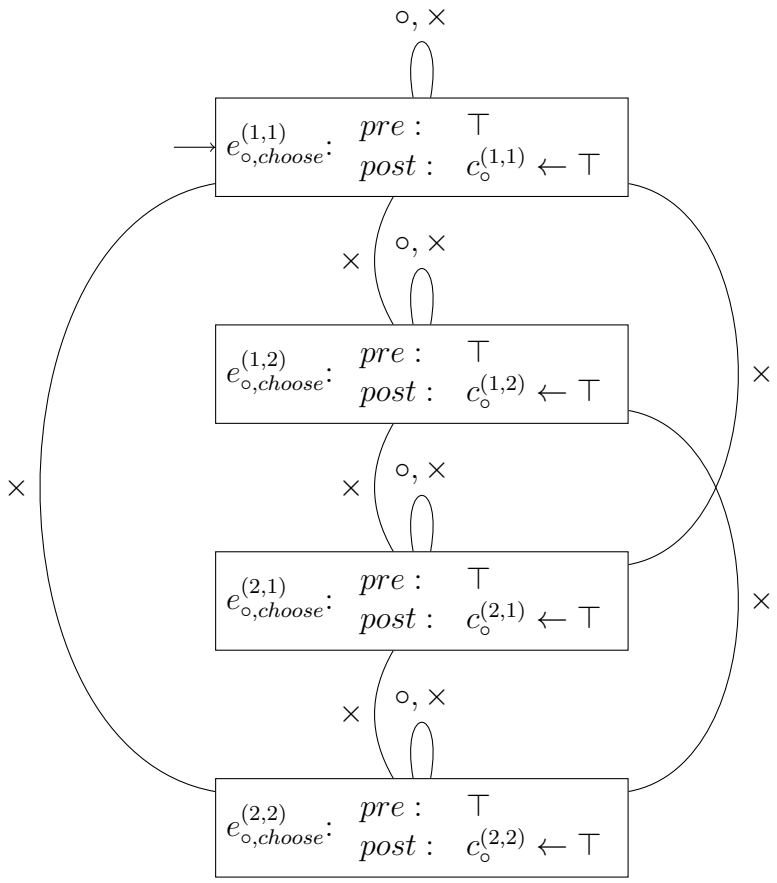


Figure 2.10: Event model $(\mathcal{E}^{(1,1)}_\circ, \mathsf{E}^{(1,1)}_\circ)$ for a 2×2 grid.

.

- $R^{\mathcal{E}}_\circ = \{(e^{z'}_{\circ,choose}, e^{z'}_{\circ,choose}), z' \in \{1,\dots,n\}^2\}$;
- $R^{\mathcal{E}}_\times = \mathsf{E} \times \mathsf{E}$;
- $\mathrm{pre}(e^{z'}_{\circ,choose}) = \top$;
- $\mathrm{post}(e^{z'}_{\circ,choose}, c^{z'}_\circ) = \top$
- post *is trivial otherwise.*

Additionally, $\mathsf{E}^z_\circ = \{e^z_{\circ,choose}\}$.

The event model for $\times$ is defined similarly by switching $\circ$ and $\times$'s roles.

Event model for playing a token

We now describe the event model of playing a token in the position chosen by $\circ$. It is composed of two types of events: $e^z_{\circ,play}$ and $e^z_{\circ,info}$ that model the fact that the move resulted in a token played at position z (for event $e^z_{\circ,play}$) or in an information for Player $\circ$ (for event $e^z_{\circ,info}$). Such events are drawn in Figures 2.11 and 2.12.

Notice first that both events are executable only if there is no $\circ$-token in z ($\neg p^z_\circ$), Player $\circ$ does not know that there is a $\times$ token in z ($\neg K_\circ p^z_\times$), it is Player $\circ$'s turn to play ($t_\circ$), Player $\circ$ has chosen to play in z ($c^z_\circ$) and the game has not ended yet p_{end}. If such conditions are not fulfilled, the previous choice of a position was invalid, thus removing any invalid move from possible strategies. Furthermore:

- Event $e^z_{\circ,play}$ is executable in the case where there is no $\times$-token in z ($\neg p^z_\times$). The effect is that there is now a $\circ$-token in z ($p^z_\circ \leftarrow \top$), and it is now the other Player's turn ($t_\circ \leftarrow \bot; t_\times \leftarrow \top$).
- Event $e^z_{\circ,info}$ is executable in the case where there is a $\times$-token in z ($\neg p^z_\times$). In this case, there is no effect (trivial postcondition).

$$e^z_{\circ,play}: \quad \begin{array}{ll} pre: & \neg p^z_\circ \wedge \neg K_\circ p^z_\times \wedge c^z_\circ \wedge \neg p^z_\times \wedge t_\circ \wedge \neg p_{end} \\ post: & p^z_\circ \leftarrow \top; \quad t_\circ \leftarrow \bot; \quad t_\times \leftarrow \top; \quad c^z_\circ \leftarrow \bot \end{array}$$

Figure 2.11: Event $e^z_{\circ,play}$.

$$e^z_{\circ,info}: \quad \begin{array}{ll} pre: & \neg p^z_\circ \wedge \neg K_\circ p^z_\times \wedge c^z_\circ \wedge p^z_\times \wedge t_\circ \wedge \neg p_{end} \\ post: & c^z_\circ \leftarrow \bot \end{array}$$

Figure 2.12: Event $e^z_{\circ,info}$.

The pointed event model $(\mathcal{E}^{pl}_\circ, \mathsf{E}^{pl}_\circ)$ for Player $\circ$ playing is now the following: it contains one event $e^{z'}_{\circ,play}$ and one event $e^{z'}_{\circ,info}$ for each possible position z' in the grid, and all are pointed. Figure 2.13 gives an example with a 2×2 grid.

Let us know define formally $(\mathcal{E}^{pl}_\circ, \mathsf{E}^{pl}_\circ)$ ($(\mathcal{E}^{pl}_\times, \mathsf{E}^{pl}_\times)$ is defined similarly by switching $\circ$ and $\times$'s roles).

Definition 36 (Event model $(\mathcal{E}^{pl}_\circ, \mathsf{E}^{pl}_\circ)$)**.** *The event model* $(\mathcal{E}^{pl}_\circ, \mathsf{E}^{pl}_\circ) = (\mathsf{E}, (R^{\mathcal{E}}_a)_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *is defined as follows:*

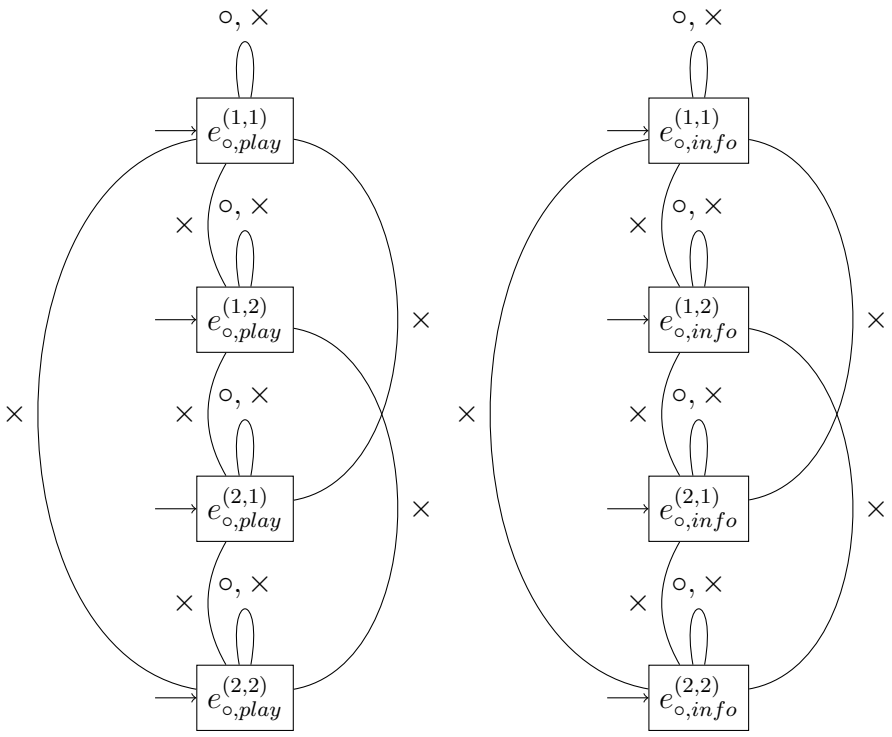


Figure 2.13: Event model $(\mathcal{E}^{pl}_{\circ}, \mathsf{E}^{pl}_{\circ})$ for a 2×2 grid.

.

- $\mathsf{E} = \{e^{z'}_{\circ,play}, z' \in \{1,\dots,n\}^2\} \cup \{e^{z'}_{\circ,info}, z' \in \{1,\dots,n\}^2\}$;
- $R^{\mathcal{E}}_{\circ} = \{(e^{z'}_{\circ,play}, e^{z'}_{\circ,play}), z' \in \{1,\dots,n\}^2\} \cup \{(e^{z'}_{\circ,info}, e^{z'}_{\circ,info})z' \in \{1,\dots,n\}^2\}$;
- $R^{\mathcal{E}}_{\times} = \{(e^{z'}_{\circ,play}, e^{z''}_{\circ,play}), z', z'' \in \{1,\dots,n\}^2\} \cup \{(e^{z'}_{\circ,info}, e^{z''}_{\circ,info})z', z'' \in \{1,\dots,n\}^2\}$;
- $\mathrm{pre}(e^{z'}_{\circ,play}) = (\neg p^{z'}_{\circ} \wedge \neg K_{\circ} p^{z'}_{\times} \wedge c^{z'}_{\circ} \wedge \neg p^{z'}_{\times} \wedge t_{\circ} \wedge \neg p_{end})$ *for all* z';
- $\mathrm{pre}(e^{z'}_{\circ,info}) = (\neg p^{z'}_{\circ} \wedge \neg K_{\circ} p^{z'}_{\times} \wedge c^{z'}_{\circ} \wedge p^{z'}_{\times} \wedge t_{\circ} \wedge \neg p_{end})$ *for all* z';
- $\mathrm{post}(e^{z'}_{\circ,play}, p^{z'}_{\circ}) = \mathrm{post}(e^{z'}_{\circ,play}, t_{\times}) = \top$ *for all* z' ;
- $\mathrm{post}(e^{z'}_{\circ,play}, t_{\circ}) = \mathrm{post}(e^{z'}_{\circ,play}, c^{z'}_{\circ}) = \mathrm{post}(e^{z'}_{\circ,info}, c^{z'}_{\circ}) = \bot$ *for all* z';
- post *is trivial otherwise.*

Additionally, $\mathsf{E}^{z}_{\circ} = \{e^{z}_{\circ,play}, e^{z}_{\circ,info}\}$.

Dummy event model to fill the turns

We do not know in advance the length of a turn, it may need several applications of event models $(\mathcal{E}_\circ^{pl}, \mathsf{E}_\circ^{pl})$. That is why in the formula will we say that event models $(\mathcal{E}_\circ^{pl}, \mathsf{E}_\circ^{pl})$ are played n^2 times, and introduce a dummy event model $(\mathcal{E}_\circ^{du}, \mathsf{E}_\circ^{du})$ that is executable iff the turn has ended (Figure 2.14).

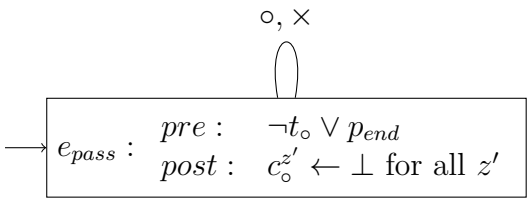


Figure 2.14: Event model $(\mathcal{E}_\circ^{du}, \mathsf{E}_\circ^{du})$

Definition 37 (Event model $(\mathcal{E}_\circ^{du}, \mathsf{E}_\circ^{du})$)**.** *The event model* $(\mathcal{E}_\circ^{du}, \mathsf{E}_\circ^{du}) = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$ *is defined as follows:*

- $\mathsf{E} = \{e_{pass}\}$*;*
- $R_\circ^{\mathcal{E}} = R_\times^{\mathcal{E}} = \{(e_{pass}, e_{pass})\}$*;*
- $\text{pre}(e_{pass}) = \neg t_\circ \vee p_{end}$*;*
- $\text{post}(e_{pass}, c_\circ^{z'}) = \bot$ *for all* z'*;*
- post *is trivial for other propositions.*

Additionally $\mathsf{E}^{pass} = \{e_{pass}\}$.

Event model to declare the end of the game

At the end of each turn, the event model $(\mathcal{E}^{end}, \mathsf{E}^{end})$ (Figure 2.15) will be executed to declare the end of the game (i.e. assign p_{end} to $\top$). It is executable if any of the players wins, meaning that $\varphi_\circ^{win} \vee \varphi_\times^{win}$ is true.

Definition 38 (Event model $(\mathcal{E}^{end}, \mathsf{E}^{end})$)**.** *The event model* $(\mathcal{E}^{end}, \mathsf{E}^{end}) = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$ *is defined as follows:*

- $\mathsf{E} = \{e_{end}\}$*;*

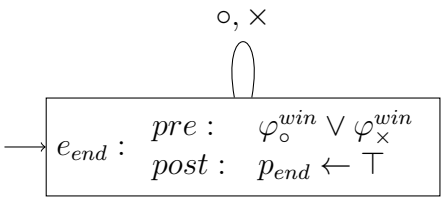


Figure 2.15: Event model $(\mathcal{E}^{end}, \mathsf{E}^{end})$

- $R_{\circ}^{\mathcal{E}} = R_{\times}^{\mathcal{E}} = \{(e_{end}, e_{end})\}$;
- $\mathrm{pre}(e_{end}) = \varphi_{\circ}^{win} \vee \varphi_{\times}^{win}$;
- $\mathrm{post}(e_{end}, p_{end}) = \top$;
- post *is trivial for other propositions.*

Additionally $\mathsf{E}^{end} = \{e_{end}\}$.

Formula

Let $(\mathcal{E}_{\circ}^{ch}, \mathsf{E}_{\circ}^{ch}) = \bigcup_{z' \in \{1,\ldots,n\}^2} (\mathcal{E}_{\circ}^{z'}, \mathsf{E}_{\circ}^{z'})$. Now the formula fragment describing a turn for agent $\circ$ is the following:

$t_{\circ} = (\langle (\mathcal{E}_{\circ}^{ch}, \mathsf{E}_{\circ}^{ch}) \rangle K_{\circ} \langle (\mathcal{E}_{\circ}^{pl}, \mathsf{E}_{\circ}^{pl}) \cup (\mathcal{E}_{\circ}^{du}, \mathsf{E}_{\circ}^{du}) \rangle)^{n^2} \langle (\mathcal{E}^{end}, \mathsf{E}^{end}) \rangle$

The turn is divided in two steps: first the choice of a position $((\mathcal{E}_{\circ}^{ch}, \mathsf{E}_{\circ}^{ch}))$ and then the move, with a possibility of a dummy event if the turn was already ended $((\mathcal{E}_{\circ}^{pl}, \mathsf{E}_{\circ}^{pl}) \cup (\mathcal{E}_{\circ}^{du}, \mathsf{E}_{\circ}^{du}))$. The modality $K_{\circ}$ in the formula $t_{\circ}$ ensures the uniformity of the position choice: it must be winning in any possible world for Player $\circ$. Then, after n^2 moves (which is an obvious and improvable upper bound of the length of the turn), we verify whether the game has ended with $(\mathcal{E}^{end}, \mathsf{E}^{end})$.

For Player $\times$ the formula fragment is similar except that it does not contain any $K_{\times}$ operator (since the winning strategy must work against any action of Player $\times$):

$t_{\times} = ([(\mathcal{E}_{\times}^{ch}, \mathsf{E}_{\times}^{ch})][(\mathcal{E}_{\times}^{pl}, \mathsf{E}_{\times}^{pl}) \cup (\mathcal{E}_{\times}^{du}, \mathsf{E}_{\times}^{du})])^{n^2} [(\mathcal{E}^{end}, \mathsf{E}^{end})]$

Finally the whole formula is: $\varphi = (t_{\circ} t_{\times})^{n^2/2} \varphi_{\circ}^{win}$

Each event model is at most of size $O(n^2)$. Overall, it means that φ is of size $O(n^6)$, which is polynomial in n. Therefore, the reduction is polynomial, leading to the following theorem.

Theorem 8. *The blind tic tac toe problem is in* PSPACE.

2.4 Conclusion

In this chapter, we have defined the model checking and satisfiability problems, and have proven that the complexity of the model checking problem is PSPACE-complete for **DELCK**, meaning that adding common knowledge does not change the complexity of the model checking problem. However, for the satisfiability problem, we go from NEXPTIME (for **DEL**) to 2-EXPTIME (for **DELCK**).

Finally, we have applied the model checking result to the blind tic tac toe game, thus proving that searching for the existence a uniform winning strategy in blind tic tac toe yields a PSPACE problem.

Chapter 3

Symbolic Dynamic Epistemic Logic

In Chapter 1, we have introduced **DELCK** and shown in Chapter 2 that the complexity of the model checking problem is Pspace-complete. Yet, this complexity may seem a bit artificial, because the polynomial is defined with respect to the size of the input, namely $|\mathcal{M}| + |w| + |\varphi|$. Take for instance the muddy children example defined in Example 3 page 37. The associated Kripke model drawn in Figure 1.2 page 38 has 2^n worlds in general where n is the number of children. Indeed, the Kripke model may have an exponential number of worlds when described explicitly. In practice, we may want to have an implicit representation of the model that is more succinct. This is what we propose in this chapter.

Usually, if the description language is (exponentially) more succinct, algorithmic problems become (exponentially) harder. For instance, deciding the existence of an Hamiltonian cycle is NP-complete but it becomes NExptime-complete [Pap03] when the input graph is described succinctly. A succinct representation of a graph with 2^b nodes is a Boolean circuit C such that there is an edge $(i, j) \in \left\{0, \ldots, 2^b - 1\right\}^2$ iff C accepts the binary representations of the b-bit integers i, j as inputs.

In **DELCK**, the results are surprising. Here, we would expect that the succinct version of the model checking is Expspace-complete. Actually, we provide a framework where the representation of events such as attention-based announcements [Bol+16] is exponentially more succinct whereas the model checking remains in Pspace.

Here, we do not use Boolean circuits traditionally used for representing instances for succinct decision problems (see [Pap03], Chapter 20.1) but *accessibility programs* (also called mental programs in the literature) based on Dynamic Logic of Propositional Assignments (DL-PA) ([BHT13], [Bal+14]) as developed in [CS15]. The reason of that choice is that our model checking algorithm directly relies on DL-PA.

The chapter is divided as follows:

- First, we define the notion of symbolic Kripke models and symbolic event models, and prove that we do not lose in expressivity and that some usual Kripke/event models have an exponentially more succinct representation.

- Second, we comment on the complexity of model checking and satisfiability problems that remain the same for **DELCK** with or without symbolic models in input.
- Third, we model the bridge card game with succinct **DELCK** and establish that searching for a uniform winning strategy is a PSPACE-problem even when the initial situation is described symbolically.
- Fourth, we discuss related work on symbolic models for epistemic logic.
- Finally, we conclude.

Our first definition of symbolic models and the expressivity/succinctness results have been published in [CS17]. The current definition of symbolic models with the model checking/satisfiability results have been published in [CS18].

3.1 Programs and Symbolic Models

We describe a symbolic way of describing Kripke models: the set of worlds is the set of all valuations over the set AP of propositional variables and accessibility relations $\xrightarrow{a}$ are given in a symbolic manner by means of *programs*, based on the programs of the Dynamic Logic of Propositional Assignments (DLPA) [BHT13]. In the rest of the manuscript, we let $\mathscr{U} = 2^{AP}$ be the set of all valuations over AP. When we discuss succinct Kripke models, a typical world w is associated with its valuation on $\mathscr{U}$ and we may confuse the notion of world and valuation.

3.1.1 Programs for accessibility relations

We first introduce the syntax of *programs*.

Definition 39 (Syntax of programs)**.** *The syntax of* programs π *is defined as follows.*

$$\pi \quad ::= \quad p{\leftarrow}\beta \mid \beta? \mid \pi;\pi \mid \pi \cup \pi$$

where $p \in AP$, β is a propositional formula.

The intuitive meaning of programs is the following. The program $p{\leftarrow}\beta$ sets variable p to the truth value of β. $\beta?$ tests whether β holds or not. The program $\pi;\pi'$ executes π then execute π'. The program $\pi \cup \pi'$ non-deterministically executes π or π'.

Originally in DL-PA programs also feature the Kleene star π^* for "execute π a non-deterministic finite number of times", yet here we choose to not include it in the syntax since it is not very relevant afterwards. Instead, iteration over programs will be featured with the common knowledge operator. Furthermore, initially the only allowed assignments in DL-PA are $p \leftarrow \bot$ and $p \leftarrow \top$ but here we choose to have a more convenient syntax. Our syntax is equivalent since $p \leftarrow \beta$ is equivalent to $(\beta?; p \leftarrow \top) \cup (\neg\beta?; p \leftarrow \bot)$.

The semantics of a program π is an accessibility relation over $\mathscr{U}$, written $\xrightarrow{\pi}$, and defined by induction over π as follows.

Definition 40 (Semantics of programs)**.** *The semantics of a program π is a subset of $\mathscr{U}^2$ defined as follows.*

- $w \xrightarrow{p\leftarrow\beta} u$ *if* $(u = w\backslash\{p\}$ *and* $w \not\models \beta)$ *or* $(u = w \cup \{p\}$ *and* $w \models \beta)$*;*
- $w \xrightarrow{\beta?} u$ *if* $w \models \beta$ *and* $w = u$*;*
- $w \xrightarrow{\pi_1;\pi_2} u$ *if there exists* $v \in \mathscr{U}$ *such that* $w \xrightarrow{\pi_1} v$ *and* $v \xrightarrow{\pi_2} u$*;*
- $w \xrightarrow{\pi_1\cup\pi_2} u$ *if* $w \xrightarrow{\pi_1} u$ *or* $w \xrightarrow{\pi_2} u$*.*

We distinguish a particular program $set(p_1, ..., p_n)$ that allows to reach all worlds which may differ from the current one only on the valuation of propositions $p_1, ..., p_n$. This program non-deterministically assigns values to $p_1, ..., p_n$, namely:

$$set(p_1, ..., p_n) = (p_1{\leftarrow}\bot \cup p_1{\leftarrow}\top); \ldots; (p_n{\leftarrow}\bot \cup p_n{\leftarrow}\top)$$

We now give examples of programs.

Example 14 (Programs for muddy children)**.** *Since child a sees the forehead of b but not her own, the program for a amounts to modifying the truth value of m_a. That is, $\pi_a = set(m_a)$, and symmetrically $\pi_b = set(m_b)$.*

Example 15 (Program for an omniscient agent)**.** *The program $\top?$ represents an omniscient agent since the only reachable valuation with $\top?$ is the actual valuation, the only possible world is the current one. For instance in the muddy children, the program of the father is $\pi_f = \top?$.*

Example 16 (Program for an agent believing p)**.** *Let $AP = \{p, q\}$. The program $\pi = p{\leftarrow}\top; set(q)$ represents an Agent a that believes p but is uncertain of q.*

The size of a program is defined as follows.

Definition 41 (Size of a program)**.** *It is noted $|\pi|$ and is defined by induction:*

- $|p \leftarrow \beta| = 1 + |\beta|$;
- $|\beta?| = 1 + |\beta|$;
- $|\pi_1; \pi_2| = 1 + |\pi_1| + |\pi_2|$;
- $|\pi_1 \cup \pi_2| = 1 + |\pi_1| + |\pi_2|$.

Now that programs are defined, we turn to symbolic (Kripke) models.

3.1.2 Symbolic Kripke models

Symbolic models are fully defined by means of the programs that denote accessibility relations between the set of valuations.

Definition 42 (Symbolic Kripke models)**.** *A* symbolic Kripke model *is a tuple $\mathfrak{M} = \langle AP_{\mathcal{M}}, \beta_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$ where:*

- *$AP_{\mathcal{M}}$ is a finite set of atomic propositions;*
- *$\beta_{\mathcal{M}}$ is a Boolean formula over $AP_{\mathcal{M}}$;*
- *and π_a is a program over $AP_{\mathcal{M}}$ for each agent a.*

A symbolic model naturally denote a Kripke model as follows.

Definition 43 (Semantics of symbolic Kripke models)**.** *Given a symbolic Kripke model $\mathfrak{M} = \langle AP_{\mathcal{M}}, \beta_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$ the* Kripke model represented by $\mathfrak{M}$*, written $\mathcal{M}_{\mathfrak{M}}$ is the model $(W, \{\xrightarrow{a}\}_{a \in Ag}, V)$ where:*

- $W = \{w \in \mathscr{U} \mid w \models \beta_{\mathcal{M}}\}$;
- $\xrightarrow{a} = \{(w, u) \mid w \xrightarrow{\pi_a} u\}$;
- $V(w) = w$.

Notice that the translation from symbolic Kripke models to Kripke models can be also done conversely, meaning that we do not lose expressivity by considering symbolic Kripke models.

Definition 44. *Let* $\mathcal{M} = (W, (R_a)_{a\in Ag}, V)$ *be a Kripke model. We define the* succinct Kripke model $\mathfrak{M}_\mathcal{M} = \langle AP_\mathcal{M}, \beta_\mathcal{M}, (\pi_a)_{a\in Ag}\rangle$ *where:*

- $AP_\mathcal{M} = AP \cup \{p_w \mid w \in W\}$;
- $\beta_\mathcal{M} = \exists!(\{p_w \mid w \in W\}) \wedge \bigwedge_{w\in W} p_w \rightarrow desc(V(w))$
- $\pi_a = \bigcup_{wR_au} p_w?; set(AP_\mathcal{M}); p_u?$.

The intended meaning of the fresh atomic propositions p_w is to designate the world w (as *nominals* in hybrid logic [Bla00]). Formula $\beta_\mathcal{M}$ describes the set W and the valuation V. Program π_a performs a non-deterministic choice over edges wR_au and then simulate the transition wR_au. The following proposition states that $\mathfrak{M}_\mathcal{M}$ indeed represents $\mathcal{M}$.

Proposition 8. $(\hat{\mathcal{M}}(\mathfrak{M}_\mathcal{M}), \{p_w\} \cup V(w))$ *and* $(\mathcal{M}, w)$ *are AP-bisimilar.*

Proof. We note $\mathcal{M} = (W, (R_a)_{a\in Ag}, V)$ and $\hat{\mathcal{M}}(\mathfrak{M}_\mathcal{M}) = (W', (R'_a)_{a\in Ag}, V')$. We define $B := \{(u, p_u \cup V(u)) \mid u \in W\}$. Let us prove that B is a AP-bisimulation.

- AP-conservation: for all $w \in W$, the valuation of w on AP is $V(w)$ by definition. The valuation of $\{p_w\} \cup V(w)$ is also $V(w)$ since $p_w \notin AP$. The AP-conservation is thus proven.
- Zig: consider $w \in W$. Then we have $p_w \cup V(w) \models \beta_\mathcal{M}$ by Definition 44, therefore $\{p_w\} \cup V(w) \in \hat{\mathcal{M}}(\mathfrak{M}_\mathcal{M})$. For all $u \in W$, we then also have $\{p_u\} \cup V(u) \in \hat{\mathcal{M}}(\mathfrak{M}_\mathcal{M})$ and if wR_au then we have $p_w \cup V(w) \xrightarrow{\pi_a} p_u \cup V(u)$ so we have $wR_a^{\hat{\mathcal{M}}(\mathfrak{M}_\mathcal{M})}u$.
- Zig: the reasoning is symmetrical, since w and $p_w \cup V(w)$ have a one on one correspondence.

□

Therefore, with the potential cost of more atomic propositions, it is always possible to go from Kripke model to symbolic Kripke models and vice versa. Notice that the transformation from Kripke models to symbolic Kripke models introduces $|W|$ new atomic propositions, and that the symbolic Kripke model $\mathfrak{M}_\mathcal{M}$ is of polynomial size compared to the size of the Kripke model $\mathcal{M}$. Yet, from one Kripke model $\mathcal{M}$, there is not an unique succinct Kripke model representing $\mathcal{M}$. For instance, instead of introducing one atomic proposition per world, we could have introduced atomic propositions $p_1, \ldots, p_{\log_2(|W|)}$ and

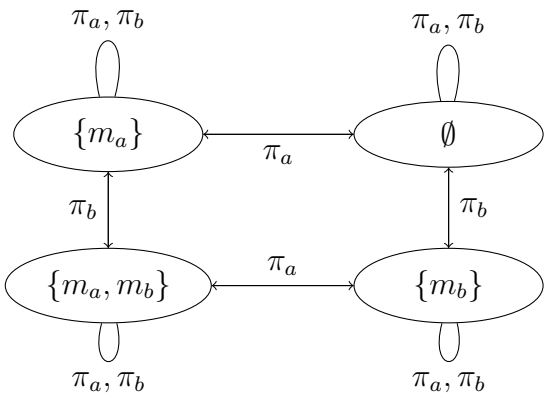


Figure 3.1: $\pi_a = set(p_a)$, $\pi_b = set(p_b)$, and $\pi_f = \top?$

then instead of p_w we would have assigned a valuation on $p_1, \ldots, p_{\log_2(|W|)}$ to each worlds instead, which introduces a logarithmic number of atomic propositions.

We now model some examples and prove at the same time that a family of Kripke models, there exist equivalent symbolic Kripke models that are exponentially smaller.

Example 17. *The Kripke model $\mathcal{M}$ from Figure 1.1 page 37 is modeled by the succinct Kripke model $\mathfrak{M}_{\mathcal{M}} = \langle AP_{\mathcal{M}}, \beta_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$ with $AP_{\mathcal{M}} = \{p, p_w, p_u\}$, $\beta_{\mathcal{M}} = \exists!(\{p_u, p_w\}) \wedge (p_w \to p) \wedge (p_u \to \neg p)$ and $\pi_a = \bigcup_{w_1, w_2 \in W} p_{w_1}?; set(AP_{\mathcal{M}}); p_{w_2}?$.*

Example 18. *The symbolic Kripke model corresponding to the initial situation of the muddy children puzzle is $\mathfrak{M} = \langle AP_{\mathcal{M}}, \beta_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$ with $AP_{\mathcal{M}} = AP$, $\beta_{\mathcal{M}} = \top$ and for each child a, $\pi_a = set(m_a)$. Figure 3.1 shows $\mathcal{M}_{\mathfrak{M}}$ for two children a and b.*

Example 19 (A Kripke model with multi-valuations)**.** *We consider the Kripke model $\mathcal{M}$ of Figure 3.2a with $AP = \{p\}$, in which valuations $\{p\}$ and $\emptyset$ appear twice. In this figure, Agent b believes that Agent a believes that p is true (i.e. $K_b K_a p$ holds with K interpreted as a belief operator). For succinctly representing $\mathcal{M}$, we introduce a new atomic proposition p_{bel} that marks possible worlds for b. We now have $AP = \{p, p_{bel}\}$ and obtain the Kripke model in Figure 3.2b, with the following programs:*

$\pi_a = (\neg p_{bel}?) \cup (p_{bel}?; (p \leftarrow \top))$: Agent a believes p in the p_{bel}-worlds, and is omniscient in the $\neg p_{bel}$-worlds.

$\pi_b = (p_{bel} \leftarrow \top)$: Agent b knows the value of p but believes that p_{bel} is true.

Notice that in the symbolic Kripke model for muddy children, each program π_a has linear size compared to $|Ag|$, $|AP_{\mathcal{M}}| = |AP|$ and $|\beta_{\mathcal{M}}| = 1$. Therefore, the size of the

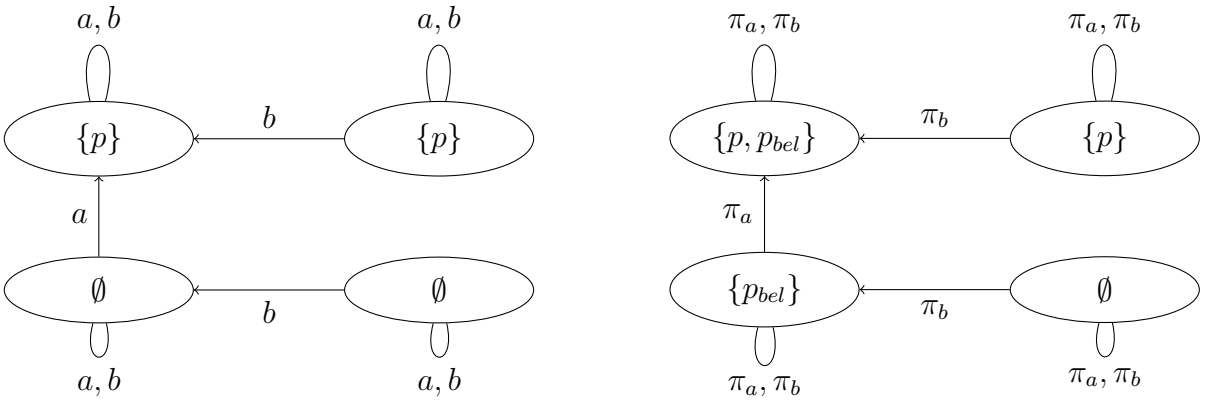


Figure 3.2: Kripke model with multi-valuations and its symbolic representation.

symbolic Kripke model is quadratic in the number of children, whereas the Kripke model has exponential size. Notice that we cannot find another Kripke model exponentially smaller AP-bisimilar since each valuation must be present in an explicit Kripke model representing the muddy children. Therefore, the example of muddy children shows that there exist Kripke models such that there exist a symbolic counterpart exponentially smaller, and that this exponential is significant because the Kripke model could not be reduced written explicitly. Therefore, the following theorem is true.

Theorem 9. *There exists a family of Kripke model $(\mathcal{M}_n)_{n\in\mathbb{N}}$ of exponential size in n such that there exists a family of equivalent succinct Kripke models $(\mathfrak{M}_n)_{n\in\mathbb{N}}$ which are of polynomial size in n.*

Proof. Take n as the number of children in the muddy children puzzle. □

3.1.3 Symbolic Event Models

We adopt a similar method for symbolic event models.

Definition 45. *A* symbolic event model *is a tuple $\mathfrak{E} = \langle AP_{\mathcal{E}}, \chi_{\mathcal{E}}, (\pi_{a,\mathcal{E}})_{a\in Ag}, \mathsf{pre}, \mathsf{post}\rangle$ where:*

- *$AP_{\mathcal{E}}$ is a set of atomic propositions disjoint from AP;*
- *$\chi_{\mathcal{E}}$ is a propositional formula over $AP_{\mathcal{E}}$ characterizing the set of events;*
- *$\pi_{a,\mathcal{E}}$ is a program over $AP_{\mathcal{E}}$ for all $a \in Ag$;*

- pre *is a propositional formula over* $AP_{\mathcal{E}} \cup \mathcal{L}_{\mathbf{EL}}(AP)$ *(meaning that any atom from* $AP_{\mathcal{E}}$ *cannot be under the scope of a* K *or a* C *operator);*
- *for all* $p \in AP$, $\mathsf{post}(p)$ *is a propositional formula over* $AP_{\mathcal{E}} \cup \mathcal{L}_{\mathbf{EL}}(AP)$.

As for symbolic Kripke models, the set of events in symbolic events models is described by a set of valuations. Here, we choose to distinguish the atomic propositions that describe the set of possible events ($AP_{\mathcal{E}}$) from the atomic propositions that describe the preconditions and the postconditions (AP).

Similarly to symbolic Kripke models, it is possible to express any event model as a symbolic event model and vice versa.

Definition 46. *Given a symbolic event model* $\mathfrak{E} = \langle AP_{\mathcal{E}}, \chi_{\mathcal{E}}, (\pi_{a,\mathcal{E}})_{a\in Ag}, \mathsf{pre}, \mathsf{post}\rangle$, *the event model represented by* $\mathfrak{E}$, *noted* $\hat{\mathcal{E}}(\mathfrak{E})$ *is the model* $(\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *on* AP *where:*

- $\mathsf{E} = \{\mathtt{v}_e \in \mathcal{V}(AP_{\mathcal{E}}) \mid \mathtt{v}_e \models \chi_{\mathcal{E}}\}$;
- $R_a^{\mathcal{E}} = \{(\mathtt{v}_e, \mathtt{v}_e{}') \mid \mathtt{v}_e \xrightarrow{\pi_{a,\mathcal{E}}} \mathtt{v}_e{}'\}$;
- $\mathrm{pre}(\mathtt{v}_e) = \mathsf{pre}|_{\mathtt{v}_e}$ *which is* pre *where every atomic proposition of* $AP_{\mathcal{E}}$ *is replaced by its value in* $\mathtt{v}_e$;
- $\mathrm{post}(\mathtt{v}_e, p) = \mathsf{post}(p)|_{\mathtt{v}_e}$.

The definition is quite straightforward. The set of events is the set of valuations satisfying $\chi_{\mathcal{E}}$ We define a symbolic event model $\mathfrak{E}_{\mathcal{E}}$ representing the event model $\mathcal{E}$.

Definition 47. *Let* $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *be an event model on* AP. *We define the symbolic event model* $\mathfrak{E}_{\mathcal{E}} = \langle AP_{\mathcal{E}}, \chi_{\mathcal{E}}, (\pi_{a,\mathcal{E}})_{a\in Ag}, \mathsf{pre}, \mathsf{post}\rangle$ *where:*

- $AP_{\mathcal{E}} = \{p_e \mid e \in \mathsf{E}\}$;
- $\chi_{\mathcal{E}} = \exists!(AP_{\mathcal{E}})$;
- $\pi_{a,\mathcal{E}} = \bigcup_{eR_a^{\mathcal{E}}f} p_e?; p_e \leftarrow \bot; p_f \leftarrow \top$;
- $\mathsf{pre} = \bigwedge_{e\in\mathsf{E}}(p_e \to \mathrm{pre}(e))$;
- $\mathsf{post}(p) = \bigwedge_{e\in\mathsf{E}}(p_e \to \mathrm{post}(e, p))$.

Fresh atomic proposition p_e is used to designate the event e. Formula $\chi_{\mathcal{E}}$ describes the set E. Program $\pi_{a,\mathcal{E}}$ performs a non-deterministic choice in the same spirit than π_a in Definition 44. The formulas pre and $\mathsf{post}(p)$ mimic the formulas pre and post(p).

It is possible to define symbolic multi-pointed event models $(\mathfrak{E}, \beta_0)$: β_0 is just a boolean formula on $AP_{\mathcal{E}}$ defining the set of pointed events.

Example 20. *The event model $\mathcal{E}$ of Figure 1.3 page 41 is modeled by the symbolic event model $\mathfrak{E}_{\mathcal{E}} = \langle AP_{\mathcal{M}}, AP_{\mathcal{E}}, \chi_{\mathcal{E}}, (\pi_{a,\mathcal{E}})_{a\in Ag}, \mathsf{post}\rangle$ with:*

- $AP_{\mathcal{E}} = \{p_e, p_f\}$;
- $\chi_{\mathcal{E}} = \exists!(AP_{\mathcal{E}})$;
- $\pi_{a,\mathcal{E}} = p_e? \cup p_f?$
- $\pi_{b,\mathcal{E}} = (p_e?; p_e \leftarrow \bot; p_f \leftarrow \top) \cup (p_f?)$;
- $\mathsf{pre} = (p_e \to p) \wedge (p_f \to \top)$;
- $\mathsf{post}(p) = (p_e \to \bot) \wedge (p_f \leftarrow p)$.

Proposition 9. *$\hat{\mathcal{E}}(\mathfrak{E}_{\mathcal{E}})$ and $\mathcal{E}$ are equivalent.*

Proof. Let $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \text{pre}, \text{post})$ and $\hat{\mathcal{E}}(\mathfrak{E}_{\mathcal{E}}) = (\mathsf{E}', (R_a^{\mathcal{E}'})_{a\in Ag}, \text{pre}', \text{post}')$. We prove that for any Kripke model $\mathcal{M} = (W, (R_a^{\mathcal{M}})_{a\in Ag}, V)$, $\mathcal{M} \otimes \mathcal{E}$ and $\mathcal{M} \otimes \hat{\mathcal{E}}(\mathfrak{E}_{\mathcal{E}})$ are AP-bisimilar. Let B is the set of tuples $((w, e), (w, e'))$ with $w \in W$, $e \in \mathsf{E}$ and $e' = \{p_e\}$. Such a bisimulation is well defined because $\mathcal{M}, w \models \text{pre}(e)$ if and only if $\mathcal{M}, w \models \mathsf{pre}|_{\{p_e\}}$, because $\mathsf{pre}|_{\{p_e\}} = \text{pre}(e)$. We prove that B is a bisimulation:

- AP-conservation: for all $p \in AP$, we have $\mathcal{M}, w \models \text{post}(p)$ if and only if $\mathcal{M}, w \models \mathsf{post}(p)|_{\{p_e\}}$ so $p \in V^{\mathcal{M}\otimes\mathcal{E}}((w, e))$ if and only if $p \in V^{\hat{\mathcal{E}}(\mathfrak{E}_{\mathcal{E}})}((w, e))$. The AP-conservation is thus proven.
- Zig and Zag: we have $(w, e) \xrightarrow{a} (u, f)$ if and only if $w \xrightarrow{a} u, f$ and $e \xrightarrow{a} f$. By definition, $e \xrightarrow{a} f$ if and only if $\{p_e\} \xrightarrow{\pi_{a,\mathcal{E}}} \{p_f\}$ therefore $(w, e) \xrightarrow{a} (u, f)$ if and only if $(w, e') \xrightarrow{a} (u, f')$. The Zig and Zag properties are directly deduced from this fact.

□

Succinctness of symbolic event models

As for succinct Kripke models, we provide a family of event models having exponentially more compact representations with symbolic event models. First let us define this family of event models.

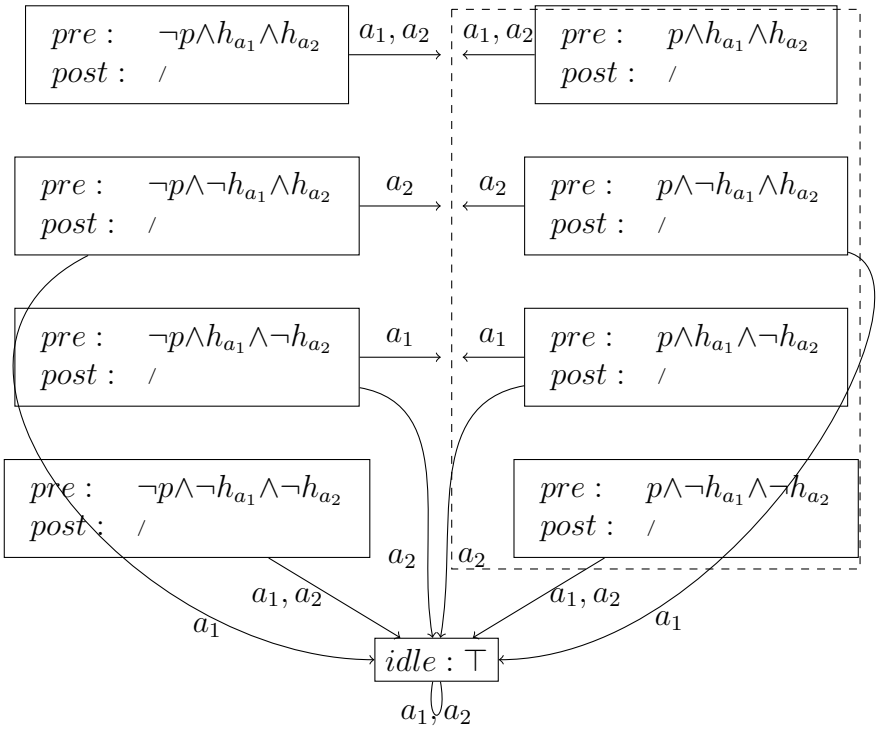


Figure 3.3: The event model $\mathcal{A}_p$ corresponding to an attention-based announcement p for two agents a_1 and a_2. The model consists of nine events. The six edges pointing to the dashed box point to all four events in the box.

Example 21. *We focus on the notion of attention-based announcement of p as shown in [Bol+16]. In addition of classic atomic propositions, we add propositions h_a for "agent a is listening to the announcement". The attention-based announcement of p can be then represented by the event model $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a \in Ag}, \mathrm{pre})$ where:*

- $\mathsf{E} = \mathcal{V}(\{p\} \cup \{h_a \mid a \in Ag\}) \cup \{e_{idle}\}$;
- *for all a, $(e, f) \in R_a^{\mathcal{E}}$ if $e \neq e_{idle}$, $f \neq e_{idle}$, $e \models h_a$ and $f \models p$;*
- *for all a, $(e, e_{idle}) \in R_a^{\mathcal{E}}$ if $e \neq e_{idle}$ and $e \not\models h_a$;*

- *for all* a, $(e_{idle}, e_{idle}) \in R_a^{\mathcal{E}}$;
- $\mathrm{pre}(e) = \begin{cases} e & \text{if } e \neq e_{idle} \\ \top & \text{otherwise.} \end{cases}$;
- post *is trivial.*

Figure 3.3 shows the event model of the attention-based announcement of p *for two agents* a_1 *and* a_2*. Event* e_{idle} *is the event where nothing happens. The relation* $R_a^{\mathcal{E}}$ *is defined as follows: if* a *is listening* ($e \models h_a$) *then* a *believes that* p *has been announced* ($f \models p$)*. If* a *is not listening* ($e \not\models h_a$) *then* a *believes nothing happens. The precondition is defined to match the fact that attentive agents listen to the announcement of* p *(thus leading to events where* p *holds) and that others agents believe that nothing that happened (thus the* $\top$ *precondition on* e_{idle}*). The announcement is purely epistemic so the postcondition function is trivial.*

Let us consider the family of models $(\mathcal{E}_n)_{n\in\mathbb{N}}$ given in Example 21 for all number of agents n. The event model $\mathcal{E}$ is succinctly represented by the symbolic event model $\mathfrak{E} = \langle AP_{\mathcal{M}}, AP_{\mathcal{E}}, \chi_{\mathcal{E}}, (\pi_{a,\mathcal{E}})_{a\in Ag}, \mathsf{post}\rangle$ defined by

- $\chi_{\mathcal{E}} = \top$;
- $\pi_{a,\mathcal{E}} = (\neg p_{idle}?; (h_a?; p{\leftarrow}\top; set(\{h_b, b \in Ag\}))\cup$
  $(\neg h_a?; set(AP_{\mathcal{M}}); p_{idle}{\leftarrow}\top)) \cup (p_{idle}?; set(AP_{\mathcal{M}}))$;
- $\mathsf{post} = \top?$.

Atomic proposition p_{idle} intuitively means that the event e_{idle} is occurring (at the bottom in Figure 3.3). Formula $\chi_{\mathcal{E}} = \top$ means that the set of possible events in unconstrained. Program $\pi_{a,\mathcal{E}}$ works as follows: if p_{idle} is false, if h_a is true, assign $\top$ to p and arbitrarily change h_b for all $b \neq a$; if h_a is false, change valuations of propositions in $AP_{\mathcal{M}}$ and set p_{idle} to true; otherwise if p_{idle} is true, change all truth values of propositions in $AP_{\mathcal{M}}$. The number of worlds in $\mathcal{E}$ is $2^{n+1}+1$ while the size of $\mathfrak{E}$ is $O(n^2)$ (each program $\pi_{a,\mathcal{E}}$ is of size $O(n)$).

Notice that this is not sufficient to say that we have found a family of event models whose symbolic representation is exponentially more succinct. Indeed, the event model itself could be too big, because the event model may be reducible, which was not the case in the muddy children Kripke model before. For instance take the event model with one

event whose precondition is $\top$ and with trivial postcondition, and take a second one with two events with preconditions p and $\neg p$ and trivial postconditions, linked by all agents for epistemic relations. Then both event models are equivalent, although the preconditions in the second one are disjoint.

To show that the event model is not reducible, we need here a notion of equivalence between event model called *action emulation*.

Definition 48. *Let $\mathcal{E} = (E, (\rightarrow_a^{\mathcal{E}})_{a\in Ag}, pre)$ and $\mathcal{E}' = (E', (\rightarrow_a^{\mathcal{E}'})_{a\in Ag}, pre')$ be two event models without postconditions Let Σ be the set of preconditions appearing in $\mathcal{E}$ and $\mathcal{E}'$. Let $\hat{\Sigma}$ be the set of formulas containing Σ and closed under sub-formulas and negation (see [ERS12] for more details). Let $CS(\hat{\Sigma})$ be the set of maximal consistent subsets of $\hat{\Sigma}$. An* action emulation *AE is a set of relations $\{AE_\Gamma\}_{\Gamma\in CS(\hat{\Sigma})} \subseteq E \times E'$ such that whenever $eAE_\Gamma e'$:*

- *Invariance: $pre(e) \in \Gamma$ and $pre(e') \in \Gamma$;*
- *Zig: For all $f \in E$ and $\Gamma' \in CS(\hat{\Sigma})$, if $e \rightarrow_a^{\mathcal{E}} f$, $pre(f) \in \Gamma'$ and the formula $\left(\bigwedge_{\psi\in\Gamma} \psi \wedge \hat{K}_a \bigwedge_{\psi'\in\Gamma'} \psi'\right)$ is consistent then there exists $f' \in E'$ such that $e' \rightarrow_a^{\mathcal{E}'} f'$ and $fAE_{\Gamma'}f'$;*
- *Zag: symmetric of Zig for E'.*

Action emulation is similar to bisimulation in the sense that the types of rules are the same, except that instead of imposing equivalence for preconditions, we just ask here that they are in a same maximal consistent subset of $\hat{\Sigma}$.

Action emulation characterizes equivalence for event models without postconditions: $\mathcal{E}$ and $\mathcal{E}'$ are equivalent if there is an action emulation AE between $\mathcal{E}$ and $\mathcal{E}'$ such that for all $e \in \mathsf{E}$ and $\Gamma \in CS(\hat{\Sigma})$ there exists $e' \in \mathsf{E}'$ such that $eAE_\Gamma e'$ and *vice versa*.

Now we prove that standard event models cannot represent $\mathcal{E}_n$ as succinctly as symbolic event models.

Theorem 10. *There is no propositional event model $\mathcal{E}'_n$ equivalent to $\mathcal{E}_n$ with less that 2^n events.*

Proof. We use the characterization of equivalent models with action emulations (Definition 48). We suppose that there is an action emulation AE between $\mathcal{E}_n$ and $\mathcal{E}'_n$. Let Σ be the set of preconditions of $\mathcal{E}_n$ and $\mathcal{E}'_n$. Note that $\hat{\Sigma}$ (defined as in Definition 48) is a set of propositional formulas. Let e_1 and e_2 be events of $\mathcal{E}$ such that $\text{pre}(e_1) \models p$ and

$\text{pre}(e_2) \models p$ and $e_1 \neq e_2$.

Suppose that there is e' of $\mathcal{E}'_n$, Γ_1, Γ_2 such that $e_1 A E_{\Gamma_1} e'$ and $e_2 A E_{\Gamma_2} e'$. As $e_1 \neq e_2$, there is an agent a such that $\text{pre}(e_1) \models \neg h_a$ and $\text{pre}(e_2) \models h_a$ (we swap e_1 and e_2 if $\text{pre}(e_1) \models h_a$ and $\text{pre}(e_2) \models \neg h_a$).

Then $e_1 R_a^{\mathcal{E}_n} e_{idle}$. We consider the maximal consistent subset $\Gamma' = \{\varphi \in \hat{\Sigma} | \{h_a, a \in Ag\} \models \varphi\}$. We have $\text{pre}(e_{idle}) \in \Gamma'$ and the formula $\left(\bigwedge_{\psi \in \Gamma_1} \psi \wedge \hat{K}_a \bigwedge_{\psi' \in \Gamma'} \psi'\right)$ is consistent (because Γ_1 and Γ' are propositional). By Zig, there exists $f' \in \mathsf{E}'$ such that $e' R_a^{\mathcal{E}'_n} f'$ with $e_{idle} A E_{\Gamma'} f'$. By Zag, as $e_2 A E_{\Gamma_2} e'$ and the formula $\left(\bigwedge_{\psi \in \Gamma_2} \psi \wedge \hat{K}_a \bigwedge_{\psi' \in \Gamma'} \psi'\right)$ is consistent, there exists $f \in \mathsf{E}$ such that $e R_a^{\mathcal{E}_n} f$ and $f A E_{\Gamma'} e'$. By invariance we obtain $\text{pre}(f) \in \Gamma'$. However we necessarily have here $\text{pre}(f) \models p$ so $\{h_a, a \in Ag\} \not\models \text{pre}(f)$. Therefore we derive a contradiction.

This proves that there at least 2^n events. □

3.2 Model checking and satisfiability

3.2.1 Model checking

Let $\mathfrak{M}$ be a symbolic Kripke model and $\varphi \in \mathcal{L}_{\mathbf{DELCK}}$ a formula containing only symbolic event models. We define the notation $\mathfrak{M}, w \models \varphi$ for $\hat{\mathcal{M}}(\mathfrak{M}), w' \models \varphi'$ where w' is the counterpart of w in the Kripke model $\hat{\mathcal{M}}(\mathfrak{M})$ and φ' is φ where each event model $(\mathfrak{E}, \beta_0)$ is replaced by $(\hat{\mathcal{E}}(\mathfrak{E}), \mathsf{E}_0)$ with $\mathsf{E}_0 = \{e \mid e \models \beta_0\}$.

The symbolic model checking problem is now defined as follows.

Definition 49 (Model checking problem)**.** *The succinct model checking problem of* **DELCK** *is defined as follows:*

- ***Input:*** *a symbolic Kripke model $\mathfrak{M}$, a world w of $\mathfrak{M}$, a formula φ of* **DELCK** *whose event models are symbolic.*
- ***Output:*** *yes if $\mathfrak{M}, w \models \varphi$, no otherwise.*

Here we prove that the symbolic model checking problem against **DELCK** is also Pspace-complete. In fact, we do not go into much detail, because the proof is almost identical to the Pspace-completeness of the model checking problem against **DELCK**. The algorithms for the symbolic checking are given in Figures 3.4, 3.5 and 3.6.

The main differences are the following.

proc $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi)$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}| + |\varphi|$
▷ accepts whenever $\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e} \models \varphi$
case $\varphi = p$: $inval_{\mathsf{yes}}(p, \mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e})$
case $\varphi = (\varphi_1 \vee \varphi_2)$: **(∃)** choose $i \in \{1, 2\}$; $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi_i)$
case $\varphi = \neg\psi$: $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \psi)$.
case $\varphi = K_a\psi$:
  (∀) choose $u\overrightarrow{f} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$
  (∃) $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **or** $in_{\mathsf{no}}(u\overrightarrow{f}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **or** $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, u\overrightarrow{f}, \psi)$
case $\varphi = \langle\mathfrak{E}, \beta_0\rangle\psi$:
  (∃) choose e such that $e \models \beta_0$;
  (∀) $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \mathsf{pre}|_{\mathrm{v_e}})$ **and** $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}} :: \mathfrak{E}, w\overrightarrow{e} :: e, \psi)$.
case $\varphi = C_G\psi$:
  (∀) choose $u\overrightarrow{f} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$
  (∃) $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, B_{\mathcal{M},\varphi}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **or** $in_{\mathsf{no}}(u\overrightarrow{f}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **or** $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, u\overrightarrow{f}, \psi)$

Figure 3.4: Model checking procedures for **DELCK** (in gray: quantities associated to each procedure call).

proc $relation_{\mathsf{yes}}(w, u, \pi)$ $|\pi|$
▷ accepts whenever $w \xrightarrow{\pi} u$
case $\pi = p{\leftarrow}\beta$:
  if $(w \models \beta \wedge (u = w \cup \{p\})) \vee (w \not\models \beta \wedge (u = w \setminus \{p\}))$ **then accept else reject**
case $\pi = \pi_1; \pi_2$: **(∃)** $v \in \mathscr{U}$; **(∀)** $relation_{\mathsf{yes}}(w, v, \pi_1)$ **and** $relation_{\mathsf{yes}}(v, u, \pi_2)$
case $\pi = \pi_1 \cup \pi_2$: **(∃)** $relation_{\mathsf{yes}}(w, u, \pi_1)$ **or** $relation_{\mathsf{yes}}(w, u, \pi_2)$
case $\pi = \beta?$: **if** $w = u \wedge w \models_{PL} \beta$ **then accept else reject**

Figure 3.5: Procedures $relation_{\mathsf{yes}}$ and $relation_{\mathsf{no}}$ to check whether $w \xrightarrow{\pi} u$.

- Instead of checking whether $(w, u) \in R_a$ we have $w \xrightarrow{\pi_a} u$, checked by the procedure $relation_{\mathsf{yes}}(w, u, \pi_a)$. This procedure takes $|\pi_a|$ in time so it does not change the overall complexity.

- Instead of pre and post we now have pre and post, their symbolic counterparts. Therefore, instead of pre(e), we now have $\mathsf{pre}|_{\mathrm{v_e}}$. Constructing $\mathsf{pre}|_{\mathrm{v_e}}$ takes linear time in the size of pre. Here we can consider it to simplify that it is constant since the a computation branch is linear in respect of the size, so the constructions of formulas $\mathsf{pre}|_{\mathrm{v_e}}$ and $\mathsf{post}(p)|_{\mathrm{v_e}}$ will add a quadratic factor to the time taken in the worst case.

proc $inval_{\mathsf{yes}}(p, w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$
▷ accepts whenever $p \in V(w\overrightarrow{e})$
case $\overrightarrow{\mathfrak{E}} = \epsilon$: **if** $p \in w$ **then accept else reject**
case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}'::\mathfrak{E}$, $w\overrightarrow{e} = w\overrightarrow{e}'::e$: $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}', w\overrightarrow{e}', \mathsf{post}(p)|_{\mathtt{v_e}})$

proc $in_{\mathsf{yes}}(w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$
▷ accepts whenever $w\overrightarrow{e} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$
case $\overrightarrow{\mathfrak{E}} = \epsilon$: **accept**
case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}'::\mathfrak{E}$, $w\overrightarrow{e} = w\overrightarrow{e}'::e$:
(∀) $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}', w\overrightarrow{e}', \mathsf{pre}|_{\mathtt{v_e}})$ **and** $in_{\mathsf{yes}}(w\overrightarrow{e}', \mathfrak{M}\overrightarrow{\mathfrak{E}}')$

proc $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$
▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \in R_a^{\mathfrak{M}\overrightarrow{\mathfrak{E}}}$
case $\overrightarrow{\mathfrak{E}} = \epsilon$: $relation_{\mathsf{yes}}(w, u, \pi_a)$
case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}' :: \mathfrak{E}$, $\overrightarrow{e} = \overrightarrow{e}' :: e$, $\overrightarrow{f} = \overrightarrow{f}' :: f$:
(∀) $access_{\mathsf{yes}}(w\overrightarrow{e}', u\overrightarrow{f}', a, \mathfrak{M}\overrightarrow{\mathfrak{E}}')$ **and** $relation_{\mathsf{yes}}(e, f, \pi_a^{\mathcal{E}})$

proc $access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}| + \log i$
▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \in (\bigcup_{a\in G} R_a)^j$ with $j \leq i$
case $i = 1$:
if $u\overrightarrow{f} = w\overrightarrow{e}$ **then accept else** (∃) choose $a \in G$; $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$
case $i \geq 2$:
(∃) choose $v\overrightarrow{g} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$
(∀) $in_{\mathsf{yes}}(v\overrightarrow{g}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **and** $access^*_{\mathsf{yes}}(w\overrightarrow{e}, v\overrightarrow{g}, G, i/2, \mathfrak{M}\overrightarrow{\mathfrak{E}})$
and $access^*_{\mathsf{yes}}(v\overrightarrow{g}, u\overrightarrow{f}, G, i/2, \mathfrak{M}\overrightarrow{\mathfrak{E}})$

Figure 3.6: Sub-procedures for **DELCK** (in gray: quantities associated to each procedure call).

Therefore, the complexity remains unchanged, the symbolic model checking problem is in PSPACE. The PSPACE-hardness comes from the hardness of non-symbolic model checking. The specification of the procedures are as follows.

Proposition 10. *For all formulas φ with symbolic event models, for all symbolic Kripke models $\mathfrak{M}$, for all sequences of event models $\overrightarrow{\mathfrak{E}}$, for all worlds $w\overrightarrow{e}, u\overrightarrow{f}$ of $\mathfrak{M}\overrightarrow{\mathfrak{E}}$, for all agents a, for all groups of agents G, for all programs π, for integers i that are powers of two,*

$mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi)$ *is accepting*	*iff* $\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e} \models \varphi$,
$mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi)$ *is accepting*	*iff* $\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e} \not\models \varphi$,
$inval_{\mathsf{yes}}(p, w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $p \in V(w\overrightarrow{e})$,
$inval_{\mathsf{no}}(p, w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $p \notin V(w\overrightarrow{e})$,
$in_{\mathsf{yes}}(w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $w\overrightarrow{e} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$,
$in_{\mathsf{no}}(w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $w\overrightarrow{e} \notin \mathfrak{M}\overrightarrow{\mathfrak{E}}$,
$access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \in R_a$,
$access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \notin R_a$,
$access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \in \bigcup_{j\leq i} (\bigcup_{a\in G} R_a)^j$,
$access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ *is accepting*	*iff* $(w\overrightarrow{e}, u\overrightarrow{f}) \notin \bigcup_{j\leq i} (\bigcup_{a\in G} R_a)^j$,
$relation_{\mathsf{yes}}(w, u, \pi)$ *is accepting*	*iff* $w \xrightarrow{\pi} u$,
and $relation_{\mathsf{no}}(w, u, \pi)$ *is accepting*	*iff* $w \not\xrightarrow{\pi} u$.

The result on the complexity may seem surprising, since usually considering symbolic models increases the complexity. Here, we advocate that the non-symbolic model checking already has some kind of symbolic representation, since the model $\mathcal{M} \otimes \overrightarrow{\mathfrak{E}}$ is not represented explicitly in the input. We add another layer of symbolic here, and fortunately having both symbolic aspects at the same time does not cause any issue here.

3.2.2 Satisfiability

Succinct satisfiability problem

The symbolic satisfiability problem for **DELCK** is defined as follows:

Definition 50 (Symbolic satisfiability problem)**.**

- ***Input:*** $\varphi \in \mathcal{L}^{suc}_{\mathbf{DELCK}}$;

- ***Output:*** *yes if φ is satisfiable (that is, there exists a pointed epistemic model*[1] *$\mathcal{M}, w$ such that $\mathcal{M}, w \models \varphi$); no otherwise.*

Theorem 11. *The symbolic satisfiability problem of* **DELCK** *is in 2EXPTIME.*

Proof. We adapt the algorithm of Figure 2.4 for the non-succinct satisfiability problem. First, we translate in exponential time our formula $\varphi \in \mathcal{L}^{suc}_{\mathbf{DELCK}}$ into an exponential size formula $\tau(\varphi) \in \mathcal{L}_{\mathbf{DELCK}}$. Then we call is**DELCK**-sat?$(\tau(\varphi))$. As the skeleton of formula φ and $\tau(\varphi)$ are the same (only dynamic modalities have been replaced), the cardinality of each Hintikka set h is still exponential in the size of φ. Therefore, the new algorithm for symbolic satisfiability problem is still double-exponential in the size of φ. □

3.3 Example: bridge card game

In this section, we show how to use symbolic model checking of **DELCK** to polynomially express the existence of a uniform winning strategy in the bridge card game [FTF01].

Here, we occult completely the bidding aspect of the bridge, we just want to know if it is possible to win at least k deals once the bidding is done. Notice that even though there are more than two players, we consider the bridge as a two player game, where the adversaries are considered as a coalition.

After recalling the rules of the game, we define a symbolic Kripke model and symbolic event models to express the game, and finally write a formula to express the existence of a winning strategy. With the model checking result for symbolic **DELCK**, we will thus prove that checking for the existence of a winning strategy in bridge is in Pspace. We discuss briefly how to use this result in the bidding phase afterwards.

3.3.1 Rules

In bridge, we have the data of a set of card values $V = \{v_1, \ldots, v_n\}$. In a classical card game, we would have $V = \{1, \ldots, 13\}$ with Jack the card 11, Queen the card 12 and King the card 13. We say that card i is *stronger* than card j if and only if $i > j$. We also have a set of card colors $C = \{c_1, \ldots, c_m\}$. A card is then a couple (v, c) with $v \in V$ the value of the card and $c \in C$ its color. Figure 3.7 shows a example of card set corresponding to a classical 52 cards deck.

1. Succinctness is needed for the *input* of decision problems. Therefore, we do not require the Kripke model in a symbolic version. It would not change the decision problem.

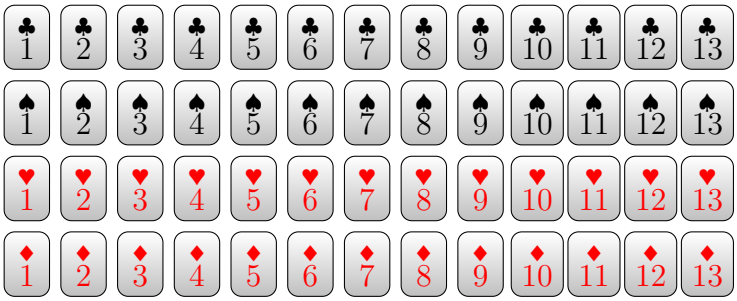


Figure 3.7: Example of a set of cards for $V = \{1, \ldots, 13\}$ (here $J = 11$, $Q = 12$, $K = 13$) and $C = \{\spadesuit, \clubsuit, \heartsuit, \diamondsuit\}$.

Goal of the game As said before, we completely omit the bidding part. Each player initially has the same number of cards and all the cards are given to the players (therefore the number of players n must divide $|V| \times |C|$). The player we want to check the winning strategy for is called the leader. His aim is to win at least k deals where k corresponds to his bidding (in the classical bridge it is 6 + the bidding but it is equivalent).

Dummy players Some players do not play the game. They are called *dummy* players. The dummy players reveal their hands, and the leader is now in control of its own hand and the hands of the dummy players. In classical bridge with 4 players, there is only one dummy player, so the leader controls 2 hands and 2 other players control their own hand.

Execution of a deal In a deal, the first player plays a card. The color of the card is the color players must play during the deal. Each player then plays a card of the same color, and the player with the highest card wins the deal. He becomes the first player for the next deal.

Notable exceptions Initially, there may be a special color called the *trump* color determined by the bidding. If a player cannot play the color of the deal, then he can play a card of the trump color instead. When played, trump colors are stronger than the color of the deal. If the player cannot play the color of the deal nor a trump card, he plays another card but he will necessarily lose the deal.

The problem we address is then the following:

- **Input:** n players, a set of values V, a set of colors C such that n divides $|V| \times |C|$, an integer $k \leq \frac{|V| \times |C|}{n}$, a set of dummy players $D \subseteq \{2, \ldots, n\}$, an optional trump

$h_{i,v,c}$	$i \in \{1,\dots,n\}, v \in V, c \in C$	Player i has card (v,c) in his hand
tu_i	$i \in \{1,\dots,n\}$	It is Player's i turn to play
$p_{i,v,c}$	$i \in \{1,\dots,n\}, v \in V, c \in C$	Player i has played card (v,c) in the current deal.
w_l	$l \in \{0,\dots,\frac{\lvert V\rvert\times\lvert C\rvert}{n}\}$	Player 1 had currently won l deals.
tr_c	$c \in C$	Color $c = c_t$ is the trump color.
du_i	$i \in \{1,\dots,n\}$	The leader plays the hand of Player i (meaning that Player i is dummy or the leader)
de_c	$c \in C$	c is the current color of the deal.

Table 3.1: Set AP for bridge.

color $c_t \in C$, the data of the hand H_1 of the leader and of the dummy players H_d for $d \in D$.

- **Output:** yes if Player 1 has a uniform winning strategy to win at least k deals in the game where he controls his hand and the hands of players in D.

3.3.2 Atomic propositions and useful formulas

Compared to blind tic tac toe (see Section 2.3), the difficulty of the modeling is not in the events since all events are combinations of public announcements. Instead, the heart of matter lies in the atomic propositions and the Kripke model. In Table 3.1 we define the set of atomic proposition AP needed. Notice that there is a polynomial number of atomic propositions in the input size.

We also define the following macros:

$winsdeal(i) = \bigvee_{i=1}^{n}(p_{i,v,c} \wedge$
$((de_c \wedge \neg tr_c \wedge \bigwedge_{1\leq j\leq n, j\neq i}\Big(\bigwedge_{c_2\in C, v_2\in V} p_{j,v_2,c_2} \rightarrow (\neg tr_{c_2} \wedge (de_{c_2} \rightarrow (v_2 < v)))\Big))$
$\vee(tr_c \wedge \bigwedge_{1\leq j\leq n, j\neq i}\Big(\bigwedge_{c_2\in C, v_2\in V} p_{j,v_2,c_2} \wedge tr_{c_2} \rightarrow (v_2 < v)\Big)))$

The formula $winsdeal(i)$ states that Player i wins the current deal. Indeed:

- The first line states that there is a $c \in C$ and $v \in V$ such that Player i has played card (v,c).
- The second line states that if c is the color of the deal then for every Player, the card he played has not the trump color and if it has the color of the deal, its value is lower than v.
- The third line states that if c is the trump color, then for every Player, if he played a card with the trump color then its value is lower than v.

The case where Player i has played a card which has not the color of the deal nor the trump color is not considered because this card would not make Player i win the deal.

$$\begin{aligned} canplay(i,(v,c)) = h_{i,v,c} \wedge (de_c \quad & \vee (tr_c \wedge \textstyle\bigwedge_{c' \in C} (de_{c'} \rightarrow \bigwedge_{v' \in V} \neg h_{i,v',c'})) \\ & \vee (\textstyle\bigwedge_{c' \in C} (de_{c'} \vee tr_{c'} \rightarrow \bigwedge_{v' \in V} \neg h_{i,v',c'}))) \ ) \\ & \vee \textstyle\bigwedge_{c' \in C} \neg de_{c'} \end{aligned}$$

Formula $canplay(i,(v,c))$ states that Player i can play card (v,c). The formula is the conjunction of Player i has card (v,c) in hand ($h_{i,v,c}$) and of a disjunction of four cases:

- Color c is the color of the deal (de_c).
- Color c is the trump color and Player i does not have any card of the color of the deal.
- Color c is nor the color of the deal or the trump color but Player i does not have any of the two colors in hand.
- No color is currently the color of the deal (then Player i can play any card).

3.3.3 Initial Kripke model

Compared to blind tic tac toe, the Kripke model initially has a exponential number of words. Thus, for practical reasons, we consider a symbolic Kripke model instead. It is $\mathfrak{M} = \langle AP_{\mathcal{M}}, \beta_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$. Here $AP_{\mathcal{M}} = AP$. The formula $\beta_{\mathcal{M}}$ is the conjunction of formulas given in Table 3.2.

Remark 2. *Although Formula 1 is not boolean, it is possible to express the equality with a boolean formula in the following way. Let S be a list of atomic propositions. We introduce the atomic proposition $eq_k(S')$ for any $k \leq \frac{|V| \times |C|}{n}$ and S' sublist of S. It is defined as the following conjunction of formulas:*

- $eq_k([]) \leftrightarrow \begin{cases} \top \text{ if } k = 0 \\ \bot \text{ otherwise} \end{cases}$
- $eq_k(S' :: p) \leftrightarrow \begin{cases} (p \wedge eq_{k-1}(S')) \vee (\neg p \wedge eq_k(S')) \text{ if } k > 0 \\ (\neg p \wedge eq_k(S')) \text{ otherwise} \end{cases}$

Then the formula $\bigwedge_{i=1}^{n} \left(|\{h_{i,v,c} = \top, v \in V, c \in C\}| = \frac{|V| \times |C|}{n} \right)$ is replaced by the following formula:

	Content of the hands for players	
1.	$\bigwedge_{i=1}^{n} \left(\lvert\{h_{i,v,c} = \top, v \in V, c \in C\}\rvert = \frac{\lvert V\rvert\times\lvert C\rvert}{n}\right)$	Each player has exactly $\frac{\lvert V\rvert\times\lvert C\rvert}{n}$ cards in hand
2.	$\bigwedge_{v\in V,c\in C} \bigoplus_{i=1}^{n} h_{i,v,c}$	Each card is exactly in one player's hand.
3.	$\bigwedge_{v\in V,c\in C} \bigwedge_{i=1}^{n} \neg p_{i,v,c}$	No card is currently played.
	Initial situation in the game	
4.	$tu_1 \wedge \bigwedge_{i=2}^{n} \neg tu_i$	It is Player's 1 turn to play.
5.	$w_0 \wedge \bigwedge_{l=1}^{\frac{\lvert V\rvert\times\lvert C\rvert}{n}} \neg w_l$	Player 1 has not won any deal yet.
6.	$tr_{c_t} \wedge \bigwedge_{c\in C\setminus\{c_t\}} \neg tr_c$	Color c_t is the only trump color (replace by $\bigwedge_{c\in C} \neg tr_c$ if there is no trump color).
7.	$du_1 \wedge \bigwedge_{i\in D} du_i \wedge \bigwedge_{j\in C\setminus(D\cup\{1\})} \neg du_j$	The dummy players' hands are controlled by Player 1.
8.	$\bigwedge_{c\in C} \neg de_c$	No color is the color of the deal.

Table 3.2: Clauses of $\beta_{\mathcal{M}}$ formula for bridge.

$$\bigwedge_{i=1}^{n} eq_{\frac{\lvert V\rvert\times\lvert C\rvert}{n}}(\{h_{i,v,c}, v \in V, c \in C\}) \text{ where } \{h_{i,v,c}, v \in V, c \in C\} \text{ is viewed as a list.}$$

The number of new atomic propositions introduced is $\lvert V\rvert \times \lvert C\rvert \times \left(\frac{\lvert V\rvert\times\lvert C\rvert}{n} + 1\right)$ *for each Player* i*, so* $\lvert V\rvert \times \lvert C\rvert \times (\lvert V\rvert \times \lvert C\rvert + n)$ *in total, so we remain polynomial. In the rest of the construction we omit the values of* $eq_k(S')$ *because they are irrelevant in event models, they are only useful to define the initial Kripke model.*

The players only have uncertainty about the hands of other non dummy players so π_a is defined as follows.

$\pi_a = set(\{h_{i,v,c}, v \in V, c \in C, i \notin D, i \neq a\})$

The initial symbolic Kripke model $\mathfrak{M}$ is then indeed of polynomial size in the size of the input.

3.3.4 Event models

To be consistent with the definition of symbolic model checking, we should define event models symbolically. Yet, they are of linear size when described explicitly, we use Definition 47 page 92 to obtain symbolic event models and describe them explicitly instead.

Event model of playing a card for Player i $\mathcal{E}_i^{v,c}$

The event model is drawn in Figure 3.8 for $V = \{1\}$ and $C = \{\spadesuit, \clubsuit\}$.

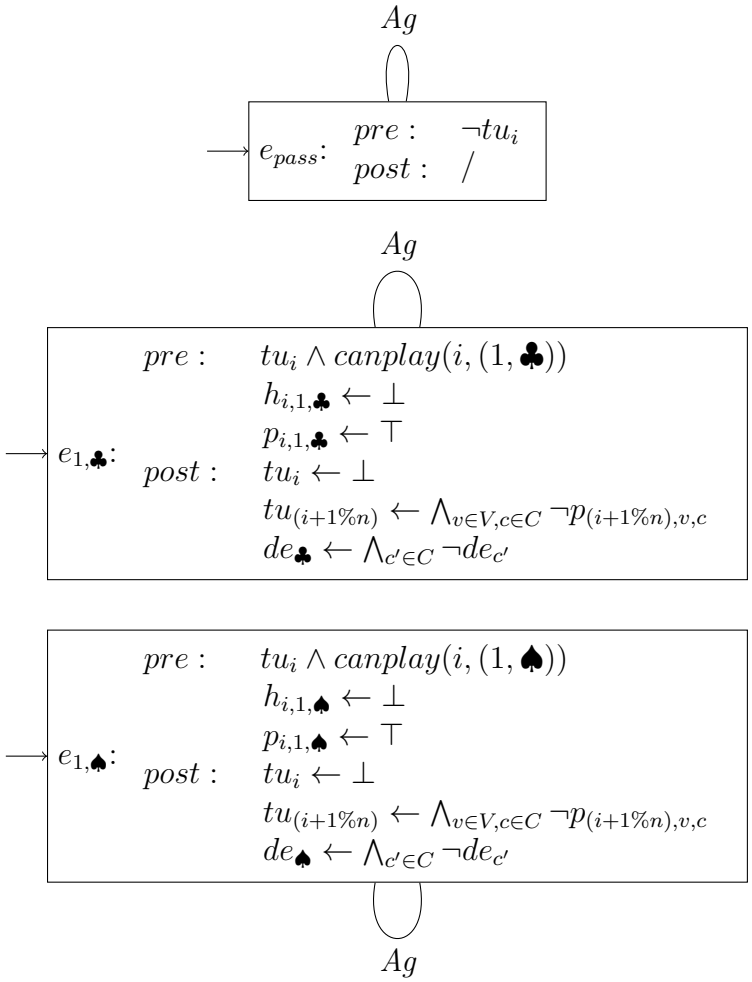


Figure 3.8: Example of event model $(\mathcal{E}^i, \mathsf{E}_0^i)$ for $V = \{1\}$ and $C = \{\spadesuit, \clubsuit\}$.

It contains one event $e_{v,c}$ per card (v, c) which is executable when it is the turn of Player i (tu_i) and that he is allowed to play this card ($canplay(i, (v, c))$). The postcondition is that Player i does not have the card in hand anymore ($h_{i,v,c} \leftarrow \bot$), the card is played ($p_{i,v,c} \leftarrow \top$), it is not Player i's turn anymore ($tu_i \leftarrow \bot$), it is the turn of the next player if he has not played yet in the deal ($tu_{(i+1\%n)} \leftarrow \bigwedge_{v\in V,c\in C} \neg p_{(i+1\%n),v,c}$), and the color the deal becomes c if it was not assigned beforehand in the deal ($de_c \leftarrow \bigwedge_{c\in C} \neg de_c$).

We add one event e_{pass} because afterwards this event model may be used in a case where Player i was not allowed to play, it is a trivial event whose precondition is $\neg tu_i$.

The formal definition in the general case is the following.

Definition 51 (Event model $(\mathcal{E}^i, \mathsf{E}^i_0)$)**.** *The event model* $\mathcal{E}^i = (\mathsf{E}^i, (R^i_a, \mathrm{pre}^i, \mathrm{post}^i))$ *is defined as follows:*

- $\mathsf{E}^i = \{e_{v,c}, v \in V, c \in C\} \cup \{e_{pass}\}$;
- $R_a = \{(e,e), e \in \mathsf{E}^i\}$ *for all agents* $a \in Ag$;
- $\mathrm{pre}^i(e_{v,c}) = tu_i \wedge canplay(i,(v,c))$;
- $\mathrm{pre}^i(e_{pass}) = \neg tu_i$;
- $\mathrm{post}^i(e_{v,c}, h_{i,v,c}) = \bot$;
- $\mathrm{post}^i(e_{v,c}, p_{i,v,c}) = \top$;
- $\mathrm{post}^i(e_{v,c}, tu_i) = \bot$;
- $\mathrm{post}^i(e_{v,c}, tu_{(i+1\%n)}) = \bigwedge_{v' \in V, c' \in C} \neg p_{(i+1\%n),v',c'}$;
- $\mathrm{post}^i(e_{v,c}, de_c) = \bigwedge_{c' \in C} \neg de_{c'}$;
- post^i *is trivial for all other cases;*
- $\mathsf{E}^i_0 = \mathsf{E}^i$.

Event model to end the deal $\mathcal{E}^{end}$

The event model contains one event e_i per Player, where an example is drawn in Figure 3.9.

Intuitively, e_i is fired if Player i has won the deal. The precondition is thus $winsdeal(i)$. The postcondition is do not change the score of Player 1 is Player i is not a dummy player ($w_l \leftarrow w_l \wedge \neg du_i$), increase the score of Player 1 is Player i is a dummy player ($w_{l+1} \leftarrow w_l \wedge du_i$), for the next deal Player i begins ($tu_i \leftarrow \top$), all other players do not begin ($tu_j \leftarrow \bot$), all cards played are now considered not played ($p_{j,v,c} \leftarrow \bot$), and no color is the color of the deal ($de_c \leftarrow \bot$).

The formal definition of $\mathcal{E}^{end}$ is the following:

Definition 52 (Event model $(\mathcal{E}^{end}, \mathsf{E}^{end}_0)$)**.** *The event model* $\mathcal{E}^{end} = (\mathsf{E}^{end}, R^{end}_a, \mathrm{pre}^{end}, \mathrm{post}^{end})$ *is defined as follows:*

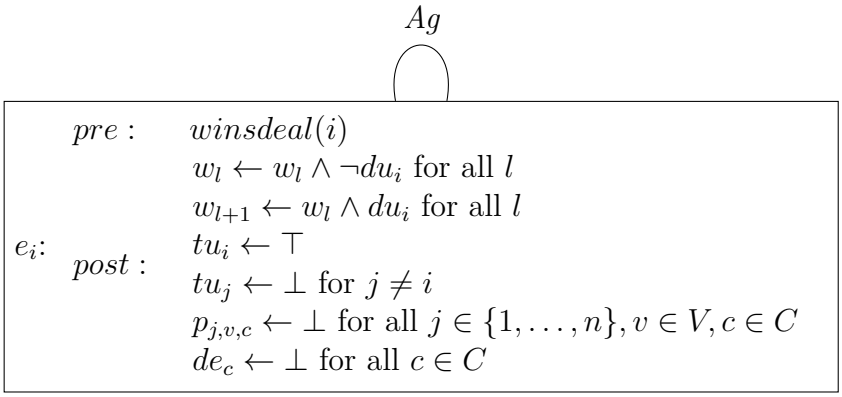


Figure 3.9: Example of event e_i for Player i in the event model $(\mathcal{E}^{end}, \mathsf{E}_0^{end})$.

- $\mathsf{E}^{end} = \{e^i, i \in \{1,\ldots,n\}\}$;
- $R_a = \{(e,e), e \in \mathsf{E}^{end}\}$ *for all agents* $a \in Ag$;
- $\mathrm{pre}^{end}(e^i) = winsdeal(i)$ *for all events* e^i;
- $\mathrm{post}^{end}(e^i, w_l) = w_l \wedge \neg du_i$ *for all events* e^i *and* l;
- $\mathrm{post}^{end}(e^i, w_{l+1}) = w_l \wedge du_i$ *for all events* e^i *and* l;
- $\mathrm{post}^{end}(e^i, tu_i) = \top$ *for all events* e^i
- $\mathrm{post}^{end}(e^i, tu_j) = \bot$ *for all events* e^i *and* $j \neq i$;
- $\mathrm{post}^{end}(e^i, p_{j,v,c}) = \bot$ *for all events* e^i, $j \in \{1,\ldots,n\}, v \in V, c \in C$;
- $\mathrm{post}^{end}(e^i, de_c) = \bot$ *for all events* e^i *and* $c \in C$;
- post^{end} *is trivial for all other cases;*
- $\mathsf{E}_0^{end} = \mathsf{E}^{end}$.

3.3.5 Formula

We consider the succinct versions of $\mathcal{E}^i$ and $\mathcal{E}^{end}$, namely $\mathfrak{E}^i$ and $\mathfrak{E}^{end}$. We define $\{\mathcal{E}^i, \mathsf{E}_0^i\}$ as $\langle \mathcal{E}^i, \mathsf{E}_0^i \rangle K_1$ when i is a dummy player, $[\mathcal{E}^i, \mathsf{E}_0^i]$ otherwise. Then:

$\varphi_g = ((\{\mathfrak{E}^1, \mathsf{E}_0^1\} \ldots \{\mathfrak{E}^n, \mathsf{E}_0^n\})^2 [\mathfrak{E} end])^{\frac{|V|\times|C|}{n}} \bigvee_{l \geq k} w_l$

In each deal, each player plays 2 times in the formula because we do not know in advance which player begins the deal, so we apply the event model 2 times to be sure each player plays once (if we execute $\mathcal{E}^i$ in a case where Player i has already played then necessarily tu_i will be false). We translate the bridge problem into the symbolic model checking instance $\mathfrak{M}, w \models \varphi_g$ where w is the world corresponding to the initial situation. Therefore the following Theorem is true.

Theorem 12. *The bridge problem is in* PSPACE.

3.3.6 Concluding remarks

In the above modeling, we have supposed that the hand of the dummy players were already known in advance. If we want to apply such techniques in the bidding phase, it is possible to change the model such that players do not know the hand of dummy players. It only amounts to change the programs π_a.

The new model can then be used in the bidding phase. If a player announces for instance "I can do 6 deals with ♠ as the trump color", then if all agents only do safe biddings, the agents can deduce that this player has a winning strategy for 6 deals with ♠ as the trump color. It then becomes common knowledge between agents and they can update the Kripke model accordingly. Therefore, it is possible to apply the technique above to make agents that can reason about the biddings.

3.4 Related work

We now briefly survey other works in the literature that display important features of our contribution.

3.4.1 Dynamic Logic of Propositional Assignments.

Dynamic Logic of Propositional Assignments [BHT13] is a instantiated version of Propositional Dynamic Logic (PDL) [FL79] where atomic programs are propositional assignments. The authors of DL-PA have shown that DL-PA behaves better than PDL in many respects, having e.g. compactness and eliminability of the iteration construct in programs. However, the model checking problem against DL-PA is PSPACE-complete, as opposed to the model checking problem against PDL which is P-complete [Lan06].

The iteration-free fragment of DL-PA has been extended with epistemic operators in [DHL12] where the programs are viewed as ontic actions, rather than accessibility relations as done in our contribution. In their setting, the authors showed that the validity problem of the iteration-free fragment of DL-PA is coNP-complete in the single-agent case and PSPACE-complete in the multi-agent case.

3.4.2 Ad-hoc accessibility relations

We highlight frameworks in which Kripke models have ad-hoc accessibility relations, and demonstrate that they can be captured by DL-PA programs. Consequently, all these frameworks may take advantage of the ability to use the full expressive power the full PAPL language.

Geometrical models. Gasquet et al. [GGS15] defined a family of Kripke models for reasoning about the knowledge of agents that are video cameras located in the 2-dimensional space (a position and a direction), according to elementary facts of the kind "Agent a sees b". They also designed a suitable language whose atomic propositions are the form $a \rhd b$, so as to mean "Agent a sees Agent b". In this setting, the worlds of the Kripke models denote configurations of agents (i.e. video cameras); each agent has a fixed position and its fixed range is some cone. It is assumed that the agents' position, as opposed to their direction, is a common knowledge to all agents. An agent cannot distinguish two worlds characterized by different dispositions of agents out of its reach. For instance, in Figure 3.10, the two depicted worlds are indistinguishable for Agent a, since they only differ on the directions of Agents b, d and e that are out of its range.

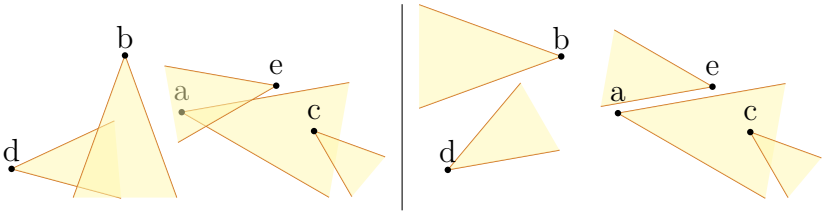


Figure 3.10: Two indistinguishable worlds for Agent a

Such Kripke models can be embedded in our framework. First of all, notice that not all valuations of atomic propositions are meaningful: for instance, the positions of agents in (any world of) Figure 3.10, prevents Agent a from seeing both Agent c and Agent

b at the same time, whichever its direction might be. As a consequence, it is common knowledge that Agent a does not see both Agent b and Agent c simultaneously. This can be captured in our setting by prefixing any property of interest by an announcement modality with an appropriate (Boolean) formula that restricts the set of valuations to meaningful ones. Now, the accessibility relation for Agent a can be described the program that non-deterministically chooses a direction of all agents that are not seen by Agent a, which can formally be written as

$$\dot{;}_{b\in Ag, b\neq a}(a \rhd b?) \cup \Big(\neg(a \rhd b)?; \big(\dot{;}_{c\in Ag, c\neq b} set(b \rhd c)\big)\Big)$$

It is important to observe that our representation of the video camera Kripke models relies on a subtle argument, noticed [GGS15]: any quantitative model (that is where worlds are defined by the real coordinates of video camera's position/direction in the plane) is bisimilar to a Kripke model where the only information about worlds is the truth values of all atomic propositions $a \rhd b$. Incidentally, it is an open question whether this property still holds, when the assumption that the fixed video cameras' position is common knowledge is relaxed, as done in [BGS13].

Observation/Visibility There is a long tradition in inferring epistemic aspects from the ability of agents to observe the value of atomic propositions, e.g Epistemic Coalition Logic of Propositional Control with Partial Observability [HTW11], Logic of Revelation and Concealment [HIW12], Boolean games with epistemic goals [Ågo+13], Alternating-time Temporal Logic under imperfect information [AHK02]. In these approaches, each agent a observe a subset of propositions Obs_a, entailing a natural accessibility relation: two worlds are indistinguishable to Agent a if they agree on Obs_a. Therefore, the program for Agent a is $\pi_a = set(AP \setminus Obs(a))$ and consists in reassigning values to unobservable propositions.

Differently, and somehow connected to a geometrical setting, the more recent work of [HLM15] describes a multi-agent epistemic logic that relies on *visibility atomic propositions* of the form, e.g. S_aS_bp to mean that 'Agent a sees that Agent b sees that p holds'. Such visibility atomic propositions (v-a-p) may contain an arbitrary finite (possibly zero) number of nested observability abilities S_a. Worlds of the Kripke models are valuations over the set of v-a-p's, and the accessibility relations are given by: Agent a cannot distinguish between valuations w and u whenever every v-a-p that a sees in w has the same truth value in u. Such ad-hoc accessibility relations can easily be described by DL-PA-

programs, but in a way that depends on the formula to be evaluated (actually, it would only depend on the knowledge modal depth of the formula) – indeed, it is necessary to select a finite set of v-a-p's the program will act on, as there are infinitely many. To do so, let us first define the program $\pi_{a,p} = (S_a p?) \cup (\neg(S_a p)?; set(p))$, where a is an agent and p is a v-a-p. Intuitively, program $\pi_{a,p}$ non-deterministically reassigns the value of v-a-p p if not seen by Agent a. Now consider the formula $K_a K_b \beta$ where β is a Boolean formula of v-a-p's. For the sake of simplicity, assume additionally that the only v-a-p's occurring in β are p and q. It can be established that the relevant v-a-p's in $K_a K_b \beta$ are p, q, $S_b p$, and $S_b q$ for Agent a, and p and q for Agent b – we do not justify this claim as it is out of the scope of the paper. We would consequently define programs $\pi_a = (\pi_{a,p}; \pi_{a,S_b p}; \pi_{a,q}; \pi_{a,S_b q})$ and $\pi_b = (\pi_{b,p}; \pi_{b,q})$ for the accessibility relations of Agent a and b, respectively.

This framework has subsequently been enlarged to encompass public announcements, yielding *Dynamic Epistemic Logic of Propositional Assignment and Observation* (DEL-PAO) [Cha+16].

3.5 Conclusion

In this chapter, we have defined the notion of symbolic Kripke and event models. We have shown that for a family of Kripke/event models, their symbolic counterparts become exponentially smaller. In fact, this result is true in most applications. In fact, all the related work shown before have symbolic representations which are exponentially smaller than their explicit representation.

We have shown that the complexity results for model checking and satisfiability do not change, namely model checking remains Pspace-complete and satisfiability 2-Exptime-complete. As we highlighted before, this result may seem surprising. In our opinion, the reason this symbolic representation does not make the complexity grow up is because **DELCK** initially has a king of symbolic representation, because the model $\mathcal{M}\overrightarrow{\mathcal{E}}$ is not represented explicitly in the input. Therefore, adding a symbolic representation for the input models does not impact the complexity.

We have applied the model checking result to show that the existence of a uniform strategy in bridge is a Pspace-problem. In fact, the examples of blind tic tac toe and bridge are similar, since in both it is possible to describe the input Kripke model with a representation of polynomial size, and the length of the game is always polynomial. For all games of this kind, we suspect that the existence of a uniform strategy is in Pspace. It

is a strong result, since it proves that introducing imperfect information in a lot of games does not increase their complexity. It refines in particular the result of [DH08] stating that bounded games with imperfect information are NEXPTIME-complete for players in teams. Indeed, we can show that bounded two-player games with imperfection information yield PSPACE problems when the bound is given in unary in input. With this result, we can show for instance that Othello which is PSPACE-complete [IK94] remains PSPACE-complete with imperfect information.

We consider implementing our approach to effectively find these winning strategies. A symbolic model checker of **DEL** already exists: SMCDEL [Ben+15]. Their symbolic representation relies on observation sets, and so they can be expressed easily with DL-PA. We envision introducing DL-PA-programs into their model checker, which may allow simpler and more efficient representations for problems. With this model checker, we then wish to implement the existence of uniform strategies in games to see experimental results on the matter.

Chapter 4

Epistemic Planning

In this chapter, we extend on **DELCK** by considering the so-called *epistemic planning* problem. It is defined as follows:

Definition 53 (Epistemic planning problem)**.**

- ***Input****: a pointed epistemic model* $(\mathcal{M}, w)$*, a finite set of pointed event models* $\mathbb{E}$*, and a goal formula* $\varphi_G \in$ **ELCK***;*
- ***Output****: yes if there exists a sequence of pointed event models* $(\mathcal{E}^1, \mathsf{E}_0^1), \ldots, (\mathcal{E}^p, \mathsf{E}_0^p) \in \mathbb{E}$ *(a* plan*) such that* $\mathcal{M}, w \models \langle \mathcal{E}^1, \mathsf{E}_0^1 \rangle \ldots \langle \mathcal{E}^p, \mathsf{E}_0^p \rangle \varphi_G$*; no otherwise.*

Epistemic planning can be seen as an extension of classical planning [FN71] where the models are the ones considered in **DELCK**, namely Kripke and event models. In classical planning, the model is a valuation and the actions are propositional, which correspond to propositional public actions in **DELCK**.

Here, we study the impact on complexity when all input event models are in $\mathcal{C}_i^j$, the class of event models where preconditions are of modal depth at most i and postconditions are of modal depth at most j.

On the one hand, when only propositional preconditions are used it is known that the problem is decidable if postconditions are also propositional [YWL13]. In this case it is in k-Exptime, where k is the modal depth of goal formulas [AMP14b]; if there are no postconditions, the problem is in Expspace [BJS15]. On the other hand, epistemic preconditions yield undecidability: if propositional postconditions are allowed, then the problem is already undecidable with preconditions of modal depth one [BA11]. It is also known to be undecidable without postconditions, if we allow for preconditions of unbounded modal depth [AB13]. Table 4.1 summarizes the results about epistemic planning.

Here, we refine this table to the one of Figure 4.2 where new results are highlighted in red.

	trivial postconditions ($j = -1$)	with postconditions ($j = \infty$)
$i = 0$	In EXPSPACE [BJS15]	Decidable [YWL13]
$i = 1$	?	Undecidable [BA11]
$i = \infty$	Undecidable [AB13]	Undecidable

Table 4.1: Overview of the complexity of epistemic planning where event models are all in $\mathcal{C}_i^j$.

	trivial postconditions ($j = -1$)	with postconditions ($j = \infty$)
$i = 0$	PSPACE-complete	Decidable [YWL13]
$i = 1$	?	Undecidable [BA11]
$i = 2$	Undecidable	Undecidable
$i = \infty$	Undecidable [AB13]	Undecidable

Table 4.2: Overview of the new complexity results of epistemic planning where event models are all in $\mathcal{C}_i^j$.

The chapter is divided as follows:

- First, we prove an easy result on epistemic planning with only "separable" event models.
- Second, we prove that with propositional preconditions and trivial postconditions, epistemic planning is PSPACE-complete (Theorem 15). The key point is that in this case events commute [LPW11]. This allows for a succinct representation of tuples of events, and we build upon a model checking procedure from [AS13] to devise a polynomial space decision procedure.
- Third, we prove that epistemic planning is already undecidable for the subproblem with only event models in $\mathcal{C}_2^{-1}$, that is with preconditions of modal depth at most two and trivial postconditions (Theorem 16). The proof, by reduction from the halting problem for two counter machines, refines the one given in [AB13], which requires preconditions with unbounded modal depth. By designing more involved gadgets to code the configurations and instructions of the machines, we manage to bound the modal depth of preconditions.
- Finally, we conclude.

The chapter, except the separable events section, corresponds to the published paper [CMS16].

4.1 Separable event models

We prove that when event models are separable (all events have disjoint preconditions) then the following two cases are true.

- If all events also have trivial postconditions, then the epistemic planning problem is NP-complete.
- Else, the epistemic planning is PSPACE-complete.

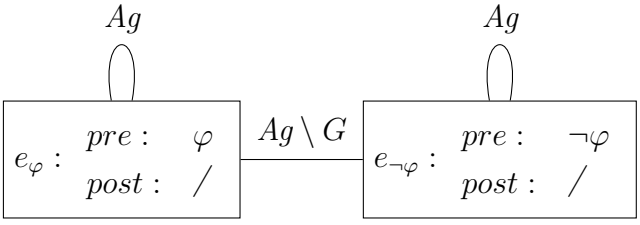
To our knowledge, these results were never published. Yet, they are interested since they legitimate for instance the approach of [KG15], that considered public actions and semi-private announcements, which fall into the separable events setting. The hardness results for both cases are direct because they extend [BJS15] where it is proven that for public announcements, the epistemic planning is NP-complete and with public actions, the epistemic planning problem is PSPACE-complete.

The intuition between separable event models is simple: preconditions must be *disjoint*. For instance, the event model from Figure 1.3 page 41 is not separable since the precondition $\top$ of event f is compatible with the precondition of event e (p). We now define formally separable event models.

Definition 54 (Separable event models)**.** *An event model* $\mathcal{E} = (\mathsf{E}, (R_a^{\mathcal{E}})_{a\in Ag}, \mathrm{pre}, \mathrm{post})$ *is separable if and only if for all events* $e, f \in \mathsf{E}$ *such that* $e \neq f$*, the formula* $\mathrm{pre}(e) \wedge \mathrm{pre}(f)$ *is unsatisfiable.*

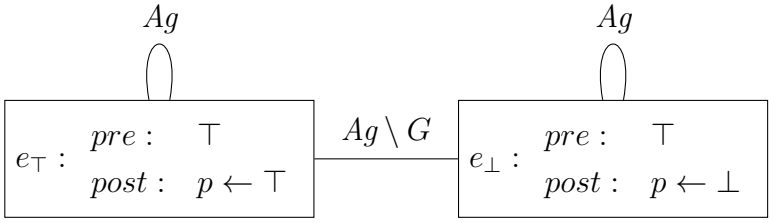
Example 22. *Here are some examples of separable event models:*

- *Public announcements and public actions are separable events models (i.e. event models with only one event).*
- *Semi-private announcements to a group* $G \subseteq Ag$ *are separable, since they are of the following form:*



And some example of event models that are not separable:

- *Semi-private assignments:*



- *The event model* $(\mathcal{E}_\circ^{(1,1)}, \mathsf{E}_\circ^{(1,1)})$ *from Figure 2.10 page 79.*

We now detail the results we obtain for epistemic planning when the event models in input are separable.

Theorem 13. *We consider the epistemic planning problem with the additional restriction that* $\mathbb{E}$ *only contains separable event models with trivial postconditions. Then this new epistemic planning problem is* NP*-complete.*

Proof. **Hardness:** direct from [BJS15].

Membership: We define the following algorithm:

proc $existplan(\mathcal{M}, w, \mathbb{E}, \varphi_G)$
  (∃) $k \in \{0, \ldots, |\mathcal{M}|\}$
  (∃) $(\mathcal{E}^1, \mathsf{E}_0^1), \ldots, (\mathcal{E}^k, \mathsf{E}_0^k) \in \mathbb{E}^k$
  Let $\mathcal{M}_{cur}, w_{cur} = \mathcal{M}, w$
  for $i = 1$ to k:
    (∃) $e \in \mathsf{E}_0^i$
    if $\mathcal{M}_{cur}, w_{cur} \not\models \mathrm{pre}(e)$ **then reject**
    else $\mathcal{M}_{cur}, w_{cur} = \mathcal{M}_{cur} \otimes \mathcal{E}^i, (w_{cur}, e)$
if $\mathcal{M}_{cur}, w_{cur} \models \varphi_G$ **then accept else reject**

The algorithm tries to find the plan by choosing non determiniscally the sequence of event models executed. Here it is important to notice that since the event models are separable and non-ontic, it is sufficient to consider a plan of at most $|\mathcal{M}|$ event models, since each application of an event model will either remove a world, an epistemic edge or do nothing. We then update the Kripke model in $\mathcal{M}_{cur}$. Since the event models are separable, then the product update is performed in polynomial time. Finally, we check that φ_G, which is a formula of **ELCK** is satisfied in $\mathcal{M}_{cur}$. Since the model checking against **ELCK** is in P (see Appendix C) this concludes the proof.

□

We now establish a similar theorem for all separable event models.

Theorem 14. *We consider the epistemic planning problem with the additional restriction that* $\mathbb{E}$ *only contains separable event models. Then this new epistemic planning problem is* PSPACE*-complete.*

Proof. **Hardness:** direct from [BJS15].

Membership: we write a trivial algorithm searching for a plan:

proc $existplan(\mathcal{M}, w, \mathbb{E}, \varphi_G)$
if $\mathcal{M}, w \models \varphi_G$: **accept**
else :
  (∃) choose $(\mathcal{E}, \mathsf{E}_0) \in \mathbb{E}$
  (∃) choose $e \in \mathsf{E}_0$
  if $\mathcal{M}, w \not\models \mathrm{pre}(e)$: **reject else** $existplan(\mathcal{M} \otimes \mathcal{E}, (w, e), \mathbb{E}, \varphi_G)$

It chooses an event model and calls back to itself with the product update in input. It stops whenever the goal has been reached. Since the event models are separable, then $|\mathcal{M} \otimes \mathcal{E}| \leq |\mathcal{M}|$, so the space needed is polynomial, which proves the PSPACE-membership. □

Remark 3. *In practice, if we want to apply Theorems 13 and 14, we need an algorithm to check that an event model is separable. If the event model has a single event, it is trivially separable. Otherwise, checking that* $\mathrm{pre}(e) \wedge \mathrm{pre}(e')$ *is unsatisfiable calls a* PSPACE *procedure since the satisfiability problem for epistemic logic is* PSPACE*-complete. Therefore for the trivial postcondition Theorem 13 is unusable for any event model if we do not know it is separable beforehand. However, for the* PSPACE *variant, having a* PSPACE *procedure beforehand is not a problem. Notice that the same problem arises even if all preconditions are propositional.*

However, for most cases in practice, for most event models we construct, we know beforehand if they are separable or not, so calling a PSPACE procedure to verify separability is a sledgehammer.

Remark 4. *In theory, the separability constraint is too strong. It is sufficient to say that for the input Kripke models* $(\mathcal{M}, w)$*, for any event models* $\mathcal{E}_1, \ldots, \mathcal{E}_n \in \mathbb{E}$ *and event model* $\mathcal{E} \in \mathbb{E}$*, in each world of* $\mathcal{M} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n$*, at most one of the events of* $\mathcal{E}$ *is applicable.*

Indeed if this constraint is true, then the Kripke model will never increase in size, and so the algorithms presented before still work. Yet, this constraint seems very hard to check. That is why separability is an easier criteria.

Similarly, if $|\mathcal{M} \otimes \mathcal{E}_1 \otimes \dots \mathcal{E}_n|$ is never bigger than $P(|\mathcal{M}|)$ for a fixed polynomial P then the Theorems 13 and 14 would still work (just replace $i \leq |\mathcal{M}|$ by $i \leq P(|\mathcal{M}|)$).

4.2 Propositional preconditions

It is already known that the epistemic planning problem restricted to $\mathcal{C}_0^{-1}$ is PSPACE-hard [BJS15]. We establish that it is actually PSPACE-complete.

From now on, we suppose that the event models are single-pointed in the input of the epistemic planning problem. If not, we just need to transform an event model $(\mathcal{E}, \mathsf{E}_0)$ into a set of event models $(\mathcal{E}, e)$ for each $e \in \mathsf{E}_0$.

As pointed out in [LPW11], epistemic event models with propositional preconditions commute. Formally:

Lemma 4. *For all pointed epistemic models $(\mathcal{M}, w)$, for all pointed event models $(\mathcal{E}_1, e_1)$ and $(\mathcal{E}_2, e_2)$ in $\mathcal{C}_0^{-1}$, $\mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2, (w, e_1, e_2)$ exists iff $\mathcal{M} \otimes \mathcal{E}_2 \otimes \mathcal{E}_1, (w, e_2, e_1)$ exists, and in that case they are bisimilar.*

As a consequence, in the rest of the section, the order in which events are applied in an initial world is indifferent. Only the number of times each event occurs is relevant, and the proof of our result heavily relies on this property.

We first establish a preliminary result on the model checking problem for a dedicated language: we extend the dynamic epistemic language with iterations of event models in $\mathcal{C}_0^{-1}$, that is, constructions of the form $\langle (\mathcal{E}, e)^\ell \rangle \psi$ where $(\mathcal{E}, e)$ is a pointed event model in $\mathcal{C}_0^{-1}$ and ℓ is a positive integer. We suppose here that ℓ is written *in binary* so that this language, called $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$, is exponentially more succinct than DEL. The *size* $|\varphi|$ of a formula φ is defined as usual, with the following additional inductive case: $|\langle (\mathcal{E}, e)^\ell \rangle \psi| = 1 + |\mathcal{E}| + \lceil \log_2 \ell \rceil + |\psi|$ (by convention $\lceil \log_2 0 \rceil = 0$). Classically, the model checking problem for $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$ is, given a pointed epistemic model $(\mathcal{M}, w)$ and a formula $\varphi \in \mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$, to decide whether $\mathcal{M}, w \models \varphi$.

Proposition 11. *Model checking $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$ is in* PSPACE.

Proof. We design a deterministic algorithm that takes as an input a pointed epistemic model $(\mathcal{M}, w_0)$ and a formula $\varphi \in \mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$, and decides whether $\mathcal{M}, w_0 \models \varphi$. Without loss of generality, we suppose that all event models appearing in the formula are the same, noted $\mathcal{E} = (\mathsf{E}, \to, \mathrm{pre})$ (if not, we replace each one by their disjoint union). Let $e_1, \dots, e_{|\mathcal{E}|}$

function $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, \vec{n}, \varphi)$
  match φ **do**
    case p: **return** (p is true in w)
    case $\neg\psi$: **return** not $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, \vec{n}, \psi)$
    case $(\psi_1 \vee \psi_2)$: **return** $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, \vec{n}, \psi_1)$ or $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, \vec{n}, \psi_2)$
    case $K_a\psi$:
      for $u \in R_a(w), \vec{\ell} \in \mathbb{N}^{|\mathcal{E}|}$ s.t. $\sum_{i=1}^{|\mathcal{E}|} \ell_i = \sum_{i=1}^{|\mathcal{E}|} n_i$ **do**
        if $\texttt{preok}(\mathcal{M}, u, \mathcal{E}, \vec{\ell}) \wedge \texttt{succ}(\mathcal{E}, a, \vec{n}, \vec{\ell}) \wedge \neg\texttt{mc}(\mathcal{M}, u, \mathcal{E}, \vec{\ell}, \psi)$ **then return** false
      endFor
      return true
    case $\langle(\mathcal{E}, e_i)^{\ell}\rangle\psi$:
      if $\mathrm{pre}(e_i)$ is false in w **then return** false
      else return $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, (n_1, .., n_{i-1}, n_i + \ell, n_{i+1}, .., n_k), \psi)$
  endMatch
endFunction

function $\texttt{preok}(\mathcal{M}, u, \mathcal{E}, \vec{\ell})$
  for $l_i \in \vec{\ell}$:
    if $(l_i > 0) \wedge \neg\texttt{mc}(\mathcal{M}, u, \mathcal{E}, \vec{\ell}, \mathrm{pre}(e_i))$ **then return** false
  return true
endFunction

function $\texttt{succ}(\mathcal{E}, a, \vec{n}, \vec{\ell})$
  for $l_i \in \vec{\ell}$:
    if $(l_i > 0) \wedge \neg\texttt{mc}(\mathcal{M}, u, \mathcal{E}, \vec{\ell}, \mathrm{pre}(e_i))$ **then return** false
  return true
endFunction

Figure 4.1: Algorithm mc for model checking $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$.

be an enumeration of the possible events in $\mathcal{E}$. By Lemma 4, all permutations of events in a tuple $(w, e_{i_1}, \ldots, e_{i_p})$ are equivalent in the sense that either they all are worlds in $\mathcal{M} \otimes \mathcal{E}^p$ and they all are bisimilar, or none of them exists: only the number of times each event occurs is relevant. For a world w and a vector $\vec{n} = (n_1, \ldots, n_{|\mathcal{E}|})$, we thus let $w{\bullet}\vec{n}$ denote the representative permutation $(w, \underbrace{e_1, \ldots, e_1}_{n_1 \text{ times}}, \ \ldots\ , \underbrace{e_{|\mathcal{E}|}, \ldots, e_{|\mathcal{E}|}}_{n_{|\mathcal{E}|} \text{ times}})$.

Let mc be the algorithm given in Figure 4.1, and let $0^{|\mathcal{E}|}$ denote the null $|\mathcal{E}|$-vector. We claim that $\texttt{mc}(\mathcal{M}, w_0, \mathcal{E}, 0^{|\mathcal{E}|}, \varphi)$ returns true iff $\mathcal{M}, w_0 \models \varphi$. To prove this claim we establish that for all $w \in \mathcal{M}$, all integers $n_1, \ldots, n_{|\mathcal{E}|}$ and all subformula ψ of φ, the following property $\mathcal{P}$ holds:

If $\mathcal{M} \otimes \mathcal{E}^{\sum_{i=1}^{|\mathcal{E}|} n_i}, w{\bullet}\vec{n}$ exists then $\texttt{mc}(\mathcal{M}, w, \mathcal{E}, (n_1, \ldots, n_{|\mathcal{E}|}), \psi)$ returns true iff

$$\mathcal{M} \otimes \mathcal{E}^{\sum_{i=1}^{|\mathcal{E}|} n_i}, w\bullet\vec{n} \models \psi.$$

Property $\mathcal{P}$ is proven by induction on φ. We omit the boolean cases and case $\langle(\mathcal{E}, e_i)^\ell\rangle\psi$ which are trivial.

Case $K_a\psi$: the algorithm has to check that ψ holds in all a-successors of $w\bullet\vec{n}$ in $\mathcal{M} \otimes \mathcal{E}^{\sum_{i=1}^{|\mathcal{E}|} n_i}$. Every a-successor of $w\bullet\vec{n}$ is a permutation of some $u\bullet\vec{\ell}$ and is bisimilar to it. We thus need to enumerate all worlds u and vectors $\vec{\ell}$ that represent some a-successor, and verify that ψ holds in $u\bullet\vec{\ell}$. Given a tuple $u\bullet\vec{\ell}$, to check whether it is a permutation of some a-successor of $w\bullet\vec{n}$, we first check that it is an existing world in $\mathcal{M} \otimes \mathcal{E}^{\sum_{i=1}^{|\mathcal{E}|} n_i}$. Since events are purely epistemic and propositional, preconditions of successive events can all be checked in the initial world u. This is done by calling function $\texttt{preok}(\mathcal{M}, u, \mathcal{E}, \vec{\ell})$, which checks that for all $i \in \{1, \ldots, |\mathcal{E}|\}$, if $\ell_i > 0$ then $\mathrm{pre}(e_i)$ is true in u. Next, we check that some permutation of $u\bullet\vec{\ell}$ is indeed a-related to $w\bullet\vec{n}$: we should first have $u \in R_a(w)$; then, it should be possible to map each occurrence of an event e_i in $w\bullet\vec{n}$ to some occurrence of some a-related event e_j in $u\bullet\vec{\ell}$ so as to form a bijection. Deciding whether such a bijection exists amounts to solving the following integer linear program: checking whether there exist positive integers $(x_{i,j})_{(i,j)\in\{1,\ldots,|\mathcal{E}|\}^2 | e_j \in R_a(e_i)}$, where $x_{i,j}$ is the number of times e_j is chosen as a-successor for e_i, such that:

$$(S) \begin{cases} n_i = \sum_{j | e_j \in R_a^{\mathcal{E}}(e_i)} x_{i,j} & \text{for all } i \in \{1, \ldots, |\mathcal{E}|\}, \text{ and} \\ \ell_j = \sum_{i | e_i \in R_a^{\mathcal{E}}(e_j)} x_{i,j} & \text{for all } j \in \{1, \ldots, |\mathcal{E}|\}. \end{cases}$$

This is done by calling $\texttt{succ}(\mathcal{E}, a, \vec{n}, \vec{\ell})$.

Spatial complexity. We justify that `mc` can be implemented in polynomial space in the size of the input (which is $|\mathcal{M}| + |\varphi|$). The maximal number of nested calls is bounded by $|\varphi|$, so that the number of local variables to be stored is polynomial in $|\varphi|$. We now bound the space needed to store vector $\vec{n}$ in each call, which is in $O(\sum_{i=1}^{|\mathcal{E}|} \lceil \log_2 n_i \rceil)$. Letting $\ell_1, \ldots, \ell_m$ be an enumeration of the numbers appearing in φ, it is clear that $\sum_{i=1}^{k} n_i \leq \sum_{i=1}^{m} \ell_i$, and thus for all $i \in \{1, \ldots, |\mathcal{E}|\}$, $n_i \leq \sum_{j=1}^{m} \ell_j$. We obtain

$$\sum_{i=1}^{|\mathcal{E}|} \lceil \log_2 n_i \rceil \leq |\mathcal{E}| \lceil \log_2 (\sum_{i=1}^{m} \ell_i) \rceil.$$

Now, it can be proven by studying the variation of function $f : (x_1, \ldots, x_m) \mapsto \log_2(\sum_{i=1}^{m} x_i) -$

$\sum_{i=1}^{m} \log_2 x_i - \log_2 m$, that since $\ell_i \geq 1$ for all $i \in \{1, \ldots, m\}$, we have

$$\log_2(\sum_{i=1}^{m} \ell_i) \leq \sum_{i=1}^{m} \log_2 \ell_i + \log_2 m,$$

and because the ceiling of a sum is less than the sum of the ceilings, we get

$$k\lceil \log_2(\sum_{i=1}^{m} \ell_i)\rceil \leq |\mathcal{E}|(\sum_{i=1}^{m} \lceil \log_2 \ell_i\rceil + \lceil \log_2 m\rceil).$$

By definition of the size of $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$ formulas, $k = |\mathsf{E}| \leq |\varphi|$, $\sum_{i=1}^{m} \lceil \log_2 \ell_i\rceil \leq |\varphi|$ and $\lceil \log_2 m\rceil \leq m \leq |\varphi|$, so that the space used to store $\vec{n}$ is in $O(|\varphi|^2)$.

It only remains to note that checking consistency of a system (S) can be done in non-deterministic time polynomial in the number of bits needed to encode $\vec{n}$ and $\vec{\ell}$ [Pap81], and therefore `succ` is in deterministic space polynomial in $|\varphi|$. □

We now describe a (non-deterministic) algorithm for the epistemic planning problem, which consists in guessing a plan and then model-check an $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$-formula to check that this plan realizes the goal. The crucial points here are, first, that we can restrict to plans of exponential length, second, that thanks to commutation of events they can be represented in polynomial space, and third, that verifying whether a plan works can be done in polynomial space (Proposition 11).

Theorem 15. *The epistemic planning problem restricted to $\mathcal{C}_0^{-1}$ is in* PSPACE.

Proof. We adapt the algorithm given in [BJS15, Theorem 5.8]. First it is proved in [Sad06] that, noting $\simeq_d$ the d-bisimulation[1] for event models (see [BJS15; Sad06; DHK07]), for every $d \geq 0$, every pointed event model $(\mathcal{E}, e)$ is $\simeq_d$-stabilizing at iteration $|\mathcal{E}|^d$; formally, $(\mathcal{E}, e_i)^k \simeq_d (\mathcal{E}, e_i)^{k+1}$ for all $k \geq |\mathcal{E}|^d$.[2] Secondly, by Lemma 4, event models with propositional preconditions commute. Therefore, the following algorithm correctly solves the epistemic planning problem for event models with propositional preconditions:

Given input $\langle (\mathcal{M}, w), \{(\mathcal{E}_1, e_1), \ldots, (\mathcal{E}_m, e_m)\}, \varphi_G\rangle$:

1. Compute d, the modal depth of the goal formula φ_G;
2. For each $i \in \{1, \ldots, m\}$, non-deterministically guess $n_i \in \left\{0, \ldots, |\mathcal{E}_i|^d\right\}$;

1. Bisimulation up to modal depth d.
2. Actually a better bound is proved in [BJS15].

3. Accept if $\mathcal{M}, w \models \langle(\mathcal{E}_1, e_1)^{n_1}\rangle \dots \langle(\mathcal{E}_m, e_m)^{n_m}\rangle\varphi_G$.

This algorithm is non-deterministic. The first step is clearly performed in space polynomial in the size of the input. Concerning the second point, each n_i can be exponential in d and thus in $|\varphi_G|$, but its binary representation uses polynomial space. Since $\langle(\mathcal{E}_1, e_1)^{n_1}\rangle \dots \langle(\mathcal{E}_m, e_m)^{n_m}\rangle\varphi_G$ is an $\mathcal{L}^{it}_{\mathcal{C}_0^{-1}}$ formula, it follows from Proposition 11 that the last step can also be performed in polynomial space. The epistemic planning problem restricted to $\mathcal{C}_0^{-1}$ is therefore in NPSPACE and thus in PSPACE by Savitch's theorem [Sav70]. □

4.3 Preconditions of modal depth 2

We now prove the following theorem by refining the reduction given in [AB13].

Theorem 16. *The epistemic planning problem restricted to $\mathcal{C}_2^{-1}$ is undecidable.*

We first recall the halting problem for two-counter machines, known to be undecidable [Min67], and then we reduce it to the epistemic planning problem restricted to $\mathcal{C}_2^{-1}$.

4.3.1 Two-counter machines

We present two-counter machines as introduced in [Min67].

Definition 55. *A two-counter machine M is a sequence of instructions $(I_0, \dots, I_N)$ where*

- *For $\ell < N$, I_ℓ is either inc(i), goto(ℓ') or gotocond(i, ℓ'), with $i \in \{1, 2\}$, $\ell' \leq N$ and $\ell \neq \ell'$;*
- *$I_N = halt$.*

We call program line *a pair k:I_k.*

Example 23. *The following four programs lines define a two-counter machine M_{ex}:*

0:inc(1)
1:gotocond(1, 3)
2:goto(0)
3:halt

A configuration of a two-counter machine M is a triple (ℓ, c_1, c_2) where $\ell \in \{0, \ldots, N\}$ is the program counter and $c_1, c_2 \in \mathbb{N}$ are the two data counters.
Let $C_M = \{0, \ldots, N\} \times \mathbb{N} \times \mathbb{N}$ be the set of all possible configurations.

The transition function $\rightarrow_M$ on C_M is defined as follows. For all $(\ell, c_1, c_2) \in C_M$:

- If $I_\ell = inc(1)$, $(\ell, c_1, c_2) \rightarrow_M (\ell + 1, c_1 + 1, c_2)$;
- If $I_\ell = inc(2)$, $(\ell, c_1, c_2) \rightarrow_M (\ell + 1, c_1, c_2 + 1)$;
- If $I_\ell = goto(\ell')$, $(\ell, c_1, c_2) \rightarrow_M (\ell', c_1, c_2)$;
- If $I_\ell = gotocond(1, \ell')$, $(\ell, c_1, c_2) \rightarrow_M \begin{cases} (\ell', 0, c_2) & \text{if } c_1 = 0; \\ (\ell + 1, c_1 - 1, c_2) & \text{otherwise;} \end{cases}$
- If $I_\ell = gotocond(2, \ell')$, $(\ell, c_1, c_2) \rightarrow_M \begin{cases} (\ell', c_1, 0) & \text{if } c_2 = 0; \\ (\ell + 1, c_1, c_2 - 1) & \text{otherwise.} \end{cases}$

A two-counter machine M *halts* if there exist c_1, c_2 such that $(0, 0, 0) \rightarrow_M^* (N, c_1, c_2)$, where $\rightarrow_M^*$ denotes the reflexive transitive closure of $\rightarrow_M$. For instance, the machine M_{ex} given in Example 23 above does not halt. The halting problem for two-counter machines consists in deciding, given a two-counter machine, whether it halts or not. This problem is well known to be undecidable [Min67].

4.3.2 The reduction

From now on in the figures, since there are a lot of worlds/events, we keep the notation circles for worlds and squares for events, but write the valuation/precondition beside the world/event.

We define an effective reduction tr that, given a two-counter machine M, computes an instance $tr(M)$ of the epistemic planning problem restricted to $\mathcal{C}_2^{-1}$. We fix M and the rest of the section is devoted to defining $tr(M) = \langle (\mathcal{M}_0, w_0); \mathbb{E}; \varphi_G \rangle$ and justifying its correctness (Proposition 12). As in [AB13], we only use one agent a ($\Box$ stands for K_a and $\Diamond$ stands for $\hat{K}_a$), configurations of M are represented by pointed epistemic models, and the initial pointed epistemic model represents the initial configuration $(0, 0, 0)$. Each program line ℓ:I_ℓ is represented by one or two pointed event model(s), such that a plan corresponds to a sequence of program lines. The goal formula expresses that the final pointed epistemic model represents a halting configuration.

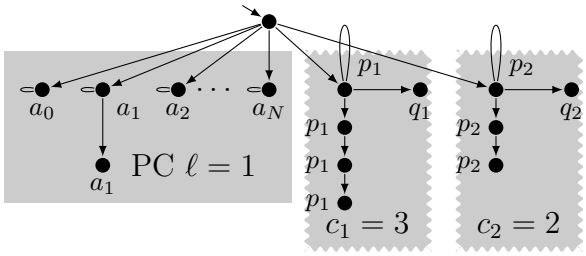


Figure 4.2: Pointed epistemic model $(\mathcal{M}, w)_{(1,3,2)}$.

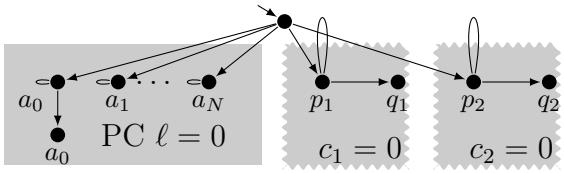


Figure 4.3: Pointed epistemic model representing the initial configuration $(0, 0, 0)$.

Pointed epistemic models

Let (ℓ, c_1, c_2) be a configuration of M. We describe the model $(\mathcal{M}_{(\ell,c_1,c_2)}, w_{(\ell,c_1,c_2)})$ (shortened as $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$) that represents (ℓ, c_1, c_2). For instance, Figure 4.2 shows $(\mathcal{M}, w)_{(1,3,2)}$. It is a tree-like structure rooted at $w_{(\ell,c_1,c_2)}$. In each world except the root, there is exactly one true atomic proposition, and we call p-world any world where p holds. The root $w_{(\ell,c_1,c_2)}$ verifies no atomic proposition, and it has three groups of children, one for each counter:

Program counter. For each program line $\ell' : I_{\ell'}$, $w_{(\ell,c_1,c_2)}$ has one reflexive child labeled by proposition $a_{\ell'}$. The a_ℓ-child of $w_{(\ell,c_1,c_2)}$ has a child also labeled by a_ℓ, without any outgoing edge: we say that there is an *a_ℓ-strip*. Intuitively, the a_ℓ-strip represents the fact that I_ℓ is the next instruction to be executed.

Data counter c_i**.** For each $i \in \{1, 2\}$, $w_{(\ell,c_1,c_2)}$ has a reflexive p_i-child that has an irreflexive q_i-child, and is followed by a chain of irreflexive p_i-worlds of length c_i. Intuitively, the number of irreflexive p_i-children represents the value of c_i.

We define the first component of $tr(M)$: $(\mathcal{M}_0, w_0) := (\mathcal{M}, w)_{(0,0,0)}$ as depicted in Figure 4.3. We call *configuration model* a pointed epistemic model of the form $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$.

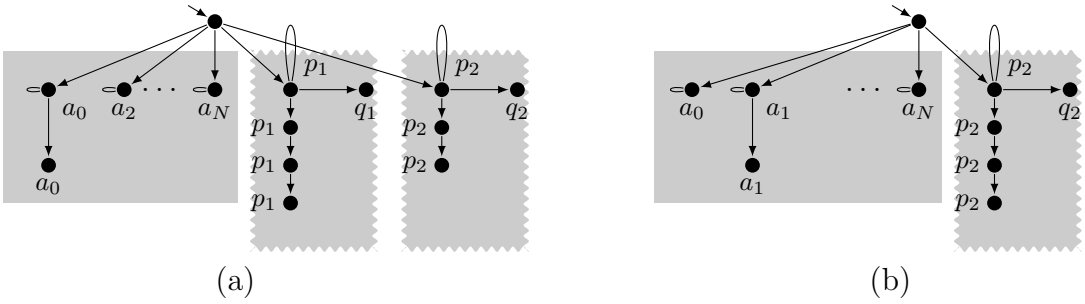


Figure 4.4: Examples of non valid epistemic models.

Pointed event models

For each program line $\ell{:}I_\ell$ of M where I_ℓ is of the form $goto(\ell')$ or $inc(i)$, we define a pointed event model $(\mathcal{E}_{\ell:I_\ell}, e_{\ell:I_\ell})$ (shortened as $(\mathcal{E}, e)_{\ell:I_\ell}$) that mimics the semantics of $\ell{:}I_\ell$ (Figures 4.6 and 4.8). For each program line $\ell{:}gotocond(i, \ell')$ of M, we define *two* pointed event models $(\mathcal{E}, e)_{\ell:gotocond(i,\ell')}$ and $(\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(i,\ell')}$, respectively for the case $c_i = 0$ and $c_i > 0$ (Figure 4.9). These pointed event models form the second component of $tr(M)$:

$$\mathbb{E} := \{(\mathcal{E}, e)_{\ell:I_\ell} \mid \ell < N\} \cup \{(\mathcal{E}^{>0}, e^{>0})_{\ell:I_\ell} \mid \ell < N \text{ and } I_\ell = gotocond(i, \ell')\}.$$

In the model $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$ where $\ell < N$, the only pointed event model of $\mathbb{E}$ that should be applied is the one representing the behavior of program line $\ell{:}I_\ell$ in configuration (ℓ, c_1, c_2). This event model is defined as follows:

$$\mathbb{E}(\ell, c_1, c_2) := \begin{cases} (\mathcal{E}^{>0}, e^{>0})_{\ell:I_\ell} & \text{if } I_\ell = gotocond(i, \ell') \\ & \text{and } c_i > 0, \\ (\mathcal{E}, e)_{\ell:I_\ell} & \text{otherwise.} \end{cases}$$

The product with any other event model from $\mathbb{E}$ results in a model that is not *valid* according to the following definition:

Definition 56. *A pointed epistemic model $(\mathcal{M}, w)$ is* valid *if w has an a_ℓ-child for each $\ell \in \{0, \ldots, N\}$ and a p_i-child for each $i \in \{1, 2\}$.*

Example 24. *The model shown in Figure 4.2, corresponding to $(\mathcal{M}, w)_{(1,3,2)}$, is valid. Notice that by definition, every configuration model is valid.*

The two models shown in Figure 4.4 are not valid. Indeed:

- *In the model shown in Figure 4.4(a), the root does not have a a_1-child.*
- *In the model shown in Figure 4.4(b), the root does not have a p_1-child.*

Further down, we will define event models of $\mathbb{E}$ such that:

Lemma 5. *For every configuration (ℓ, c_1, c_2), it holds that*

1. *$(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}(\ell, c_1, c_2)$ is isomorphic[3] to $(\mathcal{M}, w)_{(\ell',c_1',c_2')}$, where $(\ell, c_1, c_2) \rightarrow_M (\ell', c_1', c_2')$.*
2. *The product of $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$ with any other event model from $\mathbb{E}$ is defined but not valid.*
3. *For any non-empty sequence of event models $(\mathbb{E}_1, \ldots, \mathbb{E}_n, \mathbb{E}_{n+1})$ in $\mathbb{E}$, if the model $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$ is not valid, then $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n \otimes \mathbb{E}_{n+1}$ is also not valid.*

We now describe the event models in $\mathbb{E}$ and at the same time we prove Lemma 5. Each of these models has three groups (from left to right on Figures 4.6, 4.8, 4.9), that update respectively the program counter group, the data counter c_1 group and the data counter c_2 group of configuration models.

Event model for $\ell : goto(\ell')$**.** The pointed event model $(\mathcal{E}, e)_{\ell:goto(\ell')}$, that mimics the effect of ℓ:$goto(\ell')$, is depicted in Figure 4.6. Portion $repl(\ell, \ell')$ concerns the program counter group and is described in Figure 4.5. The two other groups leave the data counter groups c_1 and c_2 unchanged.

- The product $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:goto(\ell')}$ is isomorphic to $(\mathcal{M}, w)_{(\ell',c_1,c_2)}$: indeed, portion $repl(\ell, \ell')$ removes the a_ℓ-strip and adds an $a_{\ell'}$-strip in the program counter group (recall that $\ell \neq \ell'$).
- The product $(\mathcal{M}, w)_{(\ell'',c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:goto(\ell')}$ with $\ell'' \neq \ell$ is not valid. Indeed, as $(\mathcal{M}, w)_{(\ell'',c_1,c_2)}$ does not have an a_ℓ-strip in its program counter group, its a_ℓ-world violates precondition $a_\ell \wedge \Diamond\Box\bot$ in portion $repl(\ell, \ell')$. As a consequence, $(\mathcal{M}, w)_{(\ell'',c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:goto(\ell')}$ has no a_ℓ-child at its root and is thus not valid.

3. More precisely, the reachable parts of the pointed epistemic models are isomorphic.

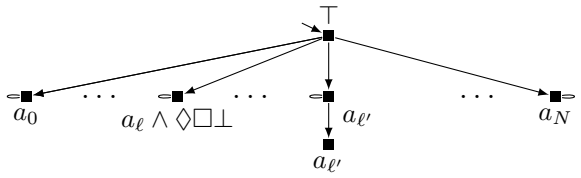


Figure 4.5: Event model portion $repl(\ell, \ell')$ for $\ell \neq \ell'$.

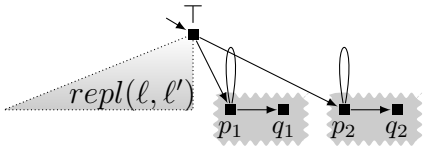


Figure 4.6: Event model $(\mathcal{E}, e)_{\ell:goto(\ell')}$ for $\ell{:}goto(\ell')$.

Event model for $\ell : inc(i)$**.** Figure 4.8 shows $(\mathcal{E}, e)_{\ell:inc(1)}$ that mimics the effect of $\ell : inc(1)$ (for $inc(2)$, the construction is symmetric). Portion $repl(\ell, \ell + 1)$ is meant to increment the program counter. Portion $lengthen(1)$ (described in Figure 4.7) is meant to increment the data counter c_1. The intermediate event of precondition $p_1 \wedge \Diamond q_1$ duplicates once the p_1-child of the root: it adds one p_1-world at the start of the p_1-chain. The last group leaves data counter c_2 unchanged.

- The product $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:inc(1)}$ is isomorphic to $(\mathcal{M}, w)_{(\ell+1,c_1+1,c_2)}$.
- For the same reason as for $(\mathcal{E}, e)_{\ell:goto(\ell')}$, the product $(\mathcal{M}, w)_{(\ell'',c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:inc(1)}$ with $\ell'' \neq \ell$ is not valid.

Event models for $\ell : gotocond(i, \ell')$**.** Figure 4.9 describes models $(\mathcal{E}, e)_{\ell:gotocond(1,\ell')}$ and $(\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(1,\ell')}$. They mimic the effect of $\ell : gotocond(1, \ell')$ in case $c_1 = 0$ and case $c_1 > 0$, respectively (for $\ell : gotocond(2, \ell')$, constructions are symmetric).

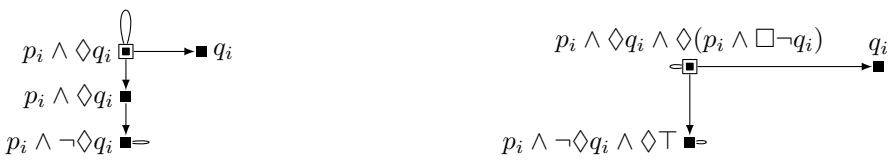


Figure 4.7: Event model portions $lengthen(i)$ and $shorten(i)$.

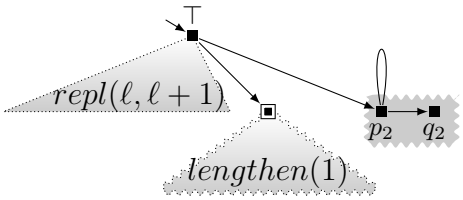


Figure 4.8: Event model $(\mathcal{E}, e)_{\ell:inc(1)}$ for $\ell{:}inc(1)$.

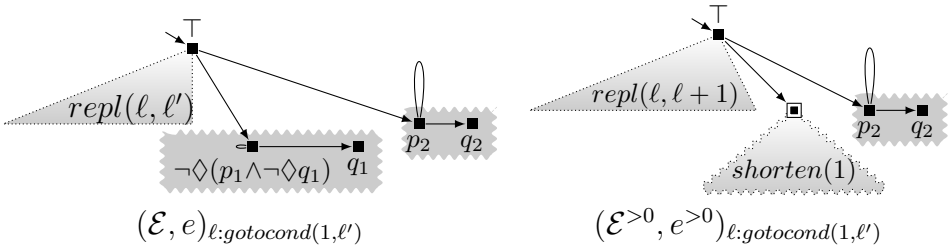


Figure 4.9: Event models for $\ell{:}gotocond(1, \ell')$.

- $(\mathcal{M}, w)_{(\ell,0,c_2)} \otimes (\mathcal{E}, e)_{\ell:gotocond(1,\ell')}$ is isomorphic to $(\mathcal{M}, w)_{(\ell',0,c_2)}$: indeed, the precondition $\neg\Diamond(p_1 \wedge \neg\Diamond q_1)$ checks that the p_1-chain in the data counter c_1 group is of length 0. Here it is the case, so that the data counter group c_1 remains unchanged. However, when $c_1 > 0$, the p_1-child of the root of $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$ violates this precondition. It is thus removed, so that the product $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:gotocond(1,\ell')}$ is not valid.

- $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes (\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(1,\ell')}$ with $c_1 > 0$ is isomorphic to $(\mathcal{M}, w)_{(\ell+1,c_1-1,c_2)}$. Indeed, portion $shorten(1)$ (Figure 4.7) is meant to decrement data counter c_1 by one: precondition $p_1 \wedge \neg\Diamond q_1 \wedge \Diamond\top$ checks that we are in the p_1-chain (p_1), but not at the start ($\neg\Diamond q_1$) nor the end ($\Diamond\top$) of the chain. The last world of the p_1-chain is thus removed when $c_1 > 0$. When $c_1 = 0$, precondition $\Diamond(p_i \wedge \Box\neg q_i)$ is violated by the p_1-child of the root of $(\mathcal{M}, w)_{(\ell,0,c_2)}$: indeed, this precondition checks that the length of the p_1-chain is at least 1. The product $(\mathcal{M}, w)_{(\ell,0,c_2)} \otimes (\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(1,\ell')}$ is thus not valid.

- For $\ell'' \neq \ell$, $(\mathcal{M}, w)_{(\ell'',c_1,c_2)} \otimes (\mathcal{E}, e)_{\ell:gotocond(1,\ell')}$ and $(\mathcal{M}, w)_{(\ell'',c_1,c_2)} \otimes (\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(1,\ell')}$ are not valid.

We now prove point 3 of Lemma 5:

We first prove the following assertion A_n for all $n \geq 1$: for all (ℓ, c_1, c_2), for all $\mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$, if $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$ is not valid then there exists $p \in \{a_0, \ldots, a_N, p_1, p_2\}$

such that there is no p-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$. It is proven by recurrence on $n \geq 1$.

- A_1: if $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1$ is not valid, then:
  - Either $\mathbb{E}_1$ contains a $repl(\ell'', \ell')$ part such that $\ell'' \neq \ell$, then there is no a_ℓ-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1$, so we take $p = a_\ell$;
  - Or $I_\ell = gotocond(i, \ell)$ (with $i \in \{1, 2\}$) and $\mathbb{E}_1$ is $(\mathcal{E}, e)_{\ell:gotocond(i,\ell')}$ with $c_i > 0$. In this case, there is no p_i-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1$, so we take $p = p_i$.
  - Or $I_\ell = gotocond(i, \ell)$ (with $i \in \{1, 2\}$) and $\mathbb{E}_1$ is $(\mathcal{E}^{>0}, e^{>0})_{\ell:gotocond(i,\ell')}$ with $c_i = 0$. In this case, there is no p_i-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1$, so we take $p = p_i$.

  In other cases, $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1$ is valid by point 1 of Lemma 5.
- $A_n \Rightarrow A_{n+1}$: We suppose A_n holds for a given $n \geq 1$. For any $\mathbb{E}_{n+1} \in \mathbb{E}$, if $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_{n+1}$ is not valid then:
  - Either $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$ is valid and by points 1 and 2 of Lemma 5, it is isomorphic to some $(\mathcal{M}, w)_{(\ell',c_1',c_2')}$. We apply A_1 to conclude.
  - Either $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$ is not valid, so by A_n there exists p such that there is no p-world. Because there is no postcondition, there is no p-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_{n+1}$ either.

  To conclude, we have proven that if $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_n$ is not valid, by A_n there is $p \in \{a_0, \ldots, a_N, p_1, p_2\}$ that is false in every world. Therefore, for any $\mathbb{E}_{n+1}$, there is no p-world in $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_{n+1}$, so $(\mathcal{M}, w)_{(\ell,c_1,c_2)} \otimes \mathbb{E}_1 \otimes \cdots \otimes \mathbb{E}_{n+1}$ is not valid either. This concludes the proof of point 3 of Lemma 5.

Goal formula

The goal formula φ_G in $tr(M)$ is $\varphi_{valid} \wedge \varphi_{halt}$, where:

- $\varphi_{valid} := \bigwedge_{\ell=0}^{N} \Diamond a_\ell \wedge \Diamond p_1 \wedge \Diamond p_2$, and
- $\varphi_{halt} := \Diamond(a_N \wedge \Diamond\Box\bot)$.

Intuitively, formula φ_{valid} forces the final pointed epistemic model of a plan to be valid (and therefore, by Point 3 of Lemma 5, all intermediate pointed epistemic models of the plan also are valid). Formula φ_{halt} enforces the final pointed epistemic model to represent a halting configuration $(N, *, *)$.

Proposition 12. *M halts iff there is a plan for $tr(M)$.*

Proof. $\boxed{\Rightarrow}$ If M halts, let $(\ell^t, c_1^t, c_2^t)_{t=0,\dots,T}$ be the sequence of configurations of the halting execution. We build a plan for $tr(M)$ by taking the sequence of pointed event models $(\mathbb{E}(\ell^t, c_1^t, c_2^t))_{t=0,\dots,T-1}$. One can prove by recurrence, using point 1 of Lemma 5, that each intermediate product $(\mathcal{M}, w)_{(0,0,0)} \otimes \mathbb{E}(0,0,0) \otimes \dots \otimes \mathbb{E}(\ell^t, c_1^t, c_2^t)$ is isomorphic to $(\mathcal{M}, w)_{(\ell^t, c_1^t, c_2^t)}$. The final product is thus isomorphic to $(\mathcal{M}, w)_{(N,c_1,c_2)}$ for some c_1, c_2, so that $(\mathcal{M}, w)_{(N,c_1,c_2)} \models \varphi_{halt}$. In addition, $(\mathcal{M}, w)_{(N,c_1,c_2)}$ is valid, so that $(\mathcal{M}, w)_{(N,c_1,c_2)} \models \varphi_{valid}$.

$\boxed{\Leftarrow}$ Suppose that there is a plan $(\mathcal{E}_t, e_t)_{t=0,\dots,T-1}$ for $tr(M)$. As the final product satisfies φ_{valid}, it is valid. Using point 3 of Lemma 5 we can prove by backward recurrence that all intermediate products are valid. By forward recurrence, using points 1 and 2 of Lemma 5, we can prove that each intermediate model is isomorphic to a model of the form $(\mathcal{M}, w)_{(\ell,c_1,c_2)}$, and that the event model applied to it in the plan is $\mathbb{E}(\ell, c_1, c_2)$. We extract a sequence of configurations $(\ell, c_1, c_2)_{t=0,\dots,T}$ that starts with $(0,0,0)$ and that, by point 1 of Lemma 5, follows the transition function of M. As the final product satisfies φ_{halt}, it is isomorphic to $(\mathcal{M}, w)_{(N,c_1,c_2)}$ for some c_1, c_2, so that the final configuration is (N, c_1, c_2). Therefore, M halts. □

4.3.3 Comparison

In [AB13] the program counter as well as the data counters are represented with chains of worlds, and incrementation, decrementation and replacement of a value by another one are implemented on such chains. While the first two operations can be performed with preconditions of modal depth two, $repl(\ell, \ell')$ requires unbounded nesting in general to be implemented on chains. We observed that unlike data counters, the program counter is *bounded* so that we can avoid chains for its representation, and provide an alternative gadget for $repl(\ell, \ell')$ that only uses preconditions of modal depth two.

4.4 Conclusion

In this chapter, we have introduced the epistemic planning problem, which takes in input a Kripke model, event models and a goal formula, and searches for a sequence of event models to apply to reach the goal formulas.

We proved that when event models are separable, meaning that any pair of events in an event models have disjoint preconditions, the epistemic planning problem is NP-complete when all events are non-ontic, and PSPACE-complete otherwise. We also proved that in the general case, when all event models are in $\mathcal{C}_0^{-1}$, the epistemic planning problem is PSPACE-complete. Finally, we showed that if the epistemic planning problem is restricted to the $\mathcal{C}_2^{-1}$, it is already undecidable. This negative result shows that it is impossible to deal with epistemic planning in all its generality, but there is still hope in practice. Indeed, for separable events, epistemic planning has the same complexity than classical planning, so maybe for non separable event model with insightful constraints, we might find other decidable fragments of epistemic planning. For instance, semi-private assignments, i.e. assign a variable to a value but some agents are oblivious about the value, may be a non separable event model such that the epistemic planning problem is still decidable. It corresponds to playing cards secretly in card games for instance. We wish to continue studying epistemic planning in this respect.

Chapter 5

Symbolic Epistemic Logic with Arbitrary Announcements

5.1 Introduction

In this chapter, we do not consider the full expressivity of **DELCK**, but instead focus on a special type of event models: public announcements, that is event models with only one event with a trivial postcondition. The family of announcement logics has been the subject of much work as they open the way to formal reasoning in many practical applications. We here mention a few, at the intuitive level only. For example, such logics enable one to reason about human/robot interaction via a public channel of communication: message exchanges between robots can modeled by public announcements when there is common knowledge of the reliability of the network and when it is assumed that messages are received instantaneously [Lem+14]. Announcement logics, as well as dynamic epistemic logic, are also relevant in games [Löw+08]: in the Battleships game, players publicly announce that there is a ship in a given cell. In card games, players often publicly show some cards to other players or announce something. Some issues in security may also be approached with announcement logics: for example, one may wish to verify that no announcement leads the system to a critical/bad state, say, where Intruder knows some secret [CD17]. Finally, gossip-based algorithms in distributed systems, where agents privately share their secrets in order to achieve shared knowledge of all secrets, may be analyzed with announcement logics [HM15; Dit+16; Dit+17].

Since epistemic planning is undecidable in general but NP-complete when restricted to announcements [BJS15] [1], studying in depth the the announcement operator is reasonable.

Here, we go further by synthesizing an announcement from scratch. Several logics were created to express the existence of an announcement, as arbitrary public announcement

1. Section 4.1 is a more general proof for separable event models

logic (APAL) [Bal+07] or group announcement logic (GAL) [Ågo+10].

In this chapter, we study the symbolic model checking against the Public Announcement Protocol Logic (PAPL), that embeds both APAL and GAL, where symbolic models are defined as in Chapter 3. The content of this chapter extends a previous paper [CS15] that only focused on symbolic model checking of APAL. The content of the chapter is the following.

	No arbitrary announcements	Arbitrary/group announcements
star-free protocols	PSPACE-complete	$\mathrm{A_{pol}}$EXPTIME-complete
protocols with stars	$\mathrm{A_{pol}}$EXPTIME-complete	$\mathrm{A_{pol}}$EXPTIME-complete

Table 5.1: Symbolic model checking of PAPL

- First, we define the logic PAPL as an extension of epistemic logic with announcement protocols.
- Second, we prove that even if a formula contains arbitrary announcements, it is sufficient to consider the set of atomic propositions in this formula in symbolic models to capture the whole expressivity.
- Third, we design algorithms that establish upper bound complexities.
- Fourth, we establish tight lower bounds.
- Fifth, we discuss the non-symbolic case.
- Finally, we conclude.

See Table 5.1 for a summary of the results. The chapter was initially published in [CS15] but only for APAL and GAL. The extensions presented in this chapter are submitted and accepted to the Journal of Logic in Computation.

5.2 Public announcement protocol logic

We define a unifying logic, called *PAPL* (Public Announcement Protocol Logic), that extends *Public Announcement Logic* (PAL) [Pla07], *Arbitrary Public Announcement Logic* (APAL) [Bal+07] and *Group Arbitrary Public Announcement Logic* (GAL) [Ågo+10]. It also contains constructions to express *announcement protocols* in the dynamic logic style.

5.2.1 Syntax of the full language $PAPL$

The formulas of $PAPL$, that may rely on announcement protocols, obey to the following syntax.

Definition 57 (Syntax of $PAPL$)**.**

$$\begin{array}{rrl} \textit{(Formulas)} & \varphi & ::= p \mid (\varphi \wedge \varphi) \mid \neg\varphi \mid K_a\varphi \mid C_G\varphi \mid \langle\gamma\rangle\varphi \\ \textit{(Announcement protocols)} & \gamma & ::= \varphi! \mid \bullet! \mid \bullet!_G \mid \varphi? \mid (\gamma;\gamma) \mid (\gamma \cup \gamma) \mid \gamma^* \end{array}$$

where $p \in AP$ is a proposition, $a \in Ag$ is an agent, and $G \subseteq Ag$ is a group of agents.

The intuitive meanings of the modal construction $\langle\gamma\rangle\varphi$ is "there is a successful execution of the protocol γ after which φ holds".

Protocols can be understood as knowledge-based programs [Fag+97], except that we do not impose tests $\varphi?$ to be formulas of the form $K_a\psi$, as protocols could be executed by the environment. Their intuitive behavior is given by th following table.

$\varphi!$	announce the true formula φ; fail if φ is false
$\bullet!$	announce an arbitrary true formula
$\bullet!_G$	announce an arbitrary true formula of the form $\bigwedge_{a\in G} K_a\psi_a$
$\varphi?$	succeed if φ is true, fail otherwise
$\gamma_1;\gamma_2$	sequentially execute γ_1 then γ_2
$\gamma_1 \cup \gamma_2$	non-deterministically choose between γ_1 and γ_2
γ^*	execute protocol γ a finite number of times (possibly 0 times) [2]

Noticeable syntactic fragments of $PAPL$ can be exhibited. *Arbitrary public announcement logic (APAL)* is the fragment where only operator K_a, $\langle\psi\rangle$ and $\langle\bullet!\rangle$ are permitted. *Group announcement logic (GAL)* is the fragment where only K_a, $\langle\psi\rangle$ and $\langle\bullet!_G\rangle$ can be used.

We now give examples of $PAPL$-formulas.

Example 25 (Muddy children with two children)**.** *Formula $\langle\bullet!\rangle\langle(\neg K_a p_a \wedge \neg K_b p_b)!\rangle C_{\{a,b\}} p_a$ states that "there exists an arbitrary public announcement such that after having announced it, the announcement that Agent a does not know that he is muddy and Agent b does not know he is muddy makes it common knowledge that Agent a is muddy". It is true for instance in the case with two children where $p_a = p_b = \top$ by instantiating the arbitrary announcement by the announcement of $p_a \vee p_b$.*

2. This protocol is non-deterministic as the number of times can be *any* finite number.

Example 26 (Muddy children in the general case)**.** *The announcement protocol*

$$\gamma = \left(\bigvee_{a\in Ag} p_a!\right); \left(\bigwedge_{a\in Ag} \neg K_a p_a!\right)^*$$

consists in first announcing that one of the children is muddy, and second, non-deterministically announcing a finite number of times that no child knows that he is muddy. Formula $\langle\gamma\rangle \bigvee_{a\in Ag} K_a p_a$ *is true in any situation where at least one child is muddy.*

Example 27 (Protocol of a dialogue)**.** *Consider four agents* a, b, c, d *that discuss together. The "dialogue" protocol where* a *speaks first and leading to the situation where* c *does not know* p*, followed by a sentence expressed by* b *and leading to the situation where* d *does not know* p *is modeled by the following announcement protocol.*

$$\bullet!_a; \neg K_c p?; \bullet!_b; \neg K_d p?$$

In the following, a formula φ or a protocol γ is *arbitrary-free* if it does not contain any occurrence of operators $\bullet!$ or $\bullet!_G$. Typically the protocol of Example 26 is arbitrary-free, but the protocol of Example 27 is not. Similarly, a protocol is *star-free* if it does not contain any occurrence of operators $*$. Thus the protocol of Example 25 is not star-free but the protocol of Example 26 is. Arbitrary-free and star-free protocols are noted τ instead of γ.

Remark 5. *We can express epistemic planning with announcements by a PAPL formulas. Let* $\{a_1, \ldots, a_n\}$ *be a series of announcements that can public or arbitrary. Then the epistemic planning problem with this set announcements as the set of possible actions is expressed by the PAPL-formula* $\langle(a_1 \cup \cdots \cup a_n)^*\rangle\varphi_G$ *where* φ_G *is the goal formula. In this setting, the action* $\bullet!$ *is interpreted as "find a formula to announce" and* $\bullet!_G$ *is interpreted as "find any formula announced by group* G*".*

5.2.2 Well-founded order over protocols and formulas

We can equip the set $PAPL$ with well-founded orders over protocols and formulas, that will be useful in subsequent proofs. We first define the order $\prec_\gamma$ over protocols (Definition 58) and then define the order $\prec$ over formulas (Definition 59).

Definition 58 (Well-founded order over protocols)**.** *We let* $\prec_\gamma$ *be the least relation on protocols that contains the standard "subprotocols"-based ordering and that satisfies:*

- $\varphi! \prec_\gamma \bullet!$ *for any arbitrary-free formula* φ*;*
- $\varphi! \prec_\gamma \bullet!_G$ *for any arbitrary-free formula* φ *and any set* G*;*
- $\gamma^i \prec_\gamma \gamma^*$ *for any protocol* γ *and any* $i \in \mathbb{N}$*;*

Proposition 13. $\prec_\gamma$ *is well-founded.*

Proof. For any protocol γ we define $q(\gamma) = (\bullet!_G(\gamma), \bullet!(\gamma), *(\gamma), symb(\gamma)) \in \mathbb{N}^4$ with $\bullet!_G(\gamma)$ the number of $\bullet!_G$ operators in γ, $\bullet!(\gamma)$ the number of $\bullet!$ operators in γ, $*(\gamma)$ the number of Kleene stars in γ and $symb(\gamma)$ the total number of symbols in γ. We define $Q_\gamma = \{q(\gamma)\} \subseteq \mathbb{N}^4$ equipped with the lexicographical order $<_{lex}$ on $\mathbb{N}^4$. We obtain directly that $\gamma \prec_\gamma \gamma'$ implies $q(\gamma) <_{lex} q(\gamma')$, which shows that $\prec_\gamma$ is well-founded. □

We now define the well-founded operator over formulas.

Definition 59 (Well-founded order over formulas)**.** *We let* $\prec \subseteq PAPL \times PAPL$ *be the least relation that contains the standard "subformula"-based ordering and that satisfies:*

- $\langle\gamma\rangle\varphi \prec \langle\gamma'\rangle\varphi$ *for any formula* φ *and any protocols* γ *and* γ' *such that* $\gamma \prec_\gamma \gamma'$*;*
- $K_{a_1}...K_{a_n}\varphi \prec C_G\varphi$ *for any set* G*, any sequence* $a_1, \ldots, a_n$ *of agents in* G *and any formula* φ*.*

We prove the following.

Proposition 14. $\prec$ *is well-founded.*

Proof. Similarly to Proposition 13, we define $q(\varphi) = (\bullet!_G(\varphi), \bullet!(\varphi), *(\varphi), C_G(\varphi), symb(\varphi))$ with $\bullet!_G(\varphi)$ being the number of $\bullet!_G$ operators in φ, $\bullet!(\varphi)$ being the number of $\bullet!$ operators in φ etc. We conclude with the same argument. □

In the rest of the chapter, the notion of induction for protocols and formulas is relative respectively to $\prec_\gamma$ and $\prec$.

5.2.3 Semantics

The semantics of $PAPL$-formulas is defined as follows.

Definition 60 (Semantics of PAPL)**.** *Let* $\mathcal{M} = (W, \{\xrightarrow{a}\}_{a \in Ag}, V), w \in W$ *and* φ *a formula. We extend the definition of* $\mathcal{M}, w \models \varphi$ *given in Definition 3 page 38 for* **ELCK** *with the following inductive case.*

- $\mathcal{M}, w \models \langle\gamma\rangle\varphi$ *if there exists* $\mathcal{M}', w$ *such that* $(\mathcal{M}, w) \rightsquigarrow_{\gamma} (\mathcal{M}', w)$ *and* $\mathcal{M}', w \models \varphi$.

We define $\rightsquigarrow_{\gamma}$ *a binary relation between pointed models by induction over protocols* γ.

- $(\mathcal{M}, w) \rightsquigarrow_{\varphi!} (\mathcal{M}^{\{u \in W \mid \mathcal{M}, u \models \varphi\}}, w)$ *whenever* $\mathcal{M}, w \models \varphi$;
- $(\mathcal{M}, w) \rightsquigarrow_{\bullet!} (\mathcal{M}', w)$ *whenever* $(\mathcal{M}, w) \rightsquigarrow_{\varphi!} (\mathcal{M}', w)$ *for some arbitrary-free formula* φ;
- $(\mathcal{M}, w) \rightsquigarrow_{\bullet!_G} (\mathcal{M}', w)$ *whenever* $(\mathcal{M}, w) \rightsquigarrow_{\bigwedge_{a \in G} K_a \psi_a!} (\mathcal{M}', w)$ *for some arbitrary-free formulas* $(\psi_a)_{a \in G}$;
- $(\mathcal{M}, w) \rightsquigarrow_{\varphi?} (\mathcal{M}, w)$ *whenever* $\mathcal{M}, w \models \varphi$;
- $\rightsquigarrow_{(\gamma \cup \gamma')}$ *is* $\rightsquigarrow_{\gamma} \cup \rightsquigarrow_{\gamma'}$; $\rightsquigarrow_{\gamma;\gamma'}$ *is the composition* $\rightsquigarrow_{\gamma} \circ \rightsquigarrow_{\gamma'}$; *and* $\rightsquigarrow_{\gamma^*}$ *is the reflexive transitive closure of* $\rightsquigarrow_{\gamma}$.

5.3 Relevant atomic propositions in symbolic Kripke models

5.3.1 Symbolic Kripke models

We consider a similar definition to Definition 42 page 88. The main difference here is that the symbolic Kripke model is only described by the programs, the set of worlds being $\mathscr{U}$ by default. If we want to begin with a set W of worlds instead of $\mathscr{U}$, it is always possible to express W with a formula φ_W and replace for any formula φ the model checking of φ by the model checking of $\langle\varphi_W!\rangle\varphi$.

Definition 61 (Symbolic Kripke models)**.** *A* symbolic (Kripke) model *is a collection* $\mathfrak{M} = (\pi_a)_{a \in Ag}$ *of programs over the set* AP *of propositional variables.*

Before proving the main theorem of this section, let us detail an example to highlight why arbitrary announcements and symbolic models are relevant together.

Example 28 (Russian cards, ([Dit03; DK15]))**.** *We consider a card game where the cards range from 1 to 7, with three players/agents Agent a, Agent b and Agent c, respectively, each of them having in hand 3 cards, 3 cards and 1 card. Such a typical situation is depicted in Figure 5.1.*

Figure 5.1: Example of hands for the Russian cards puzzle

We address the question of whether Agents a and b can or cannot publicly announce a course of truthful facts so that they end-up commonly knowing each player's hand, while Agent c not being able to learn any card from Agent a's or Agent b's hands from any of these announcements.

From [Dit03], we know that in any possible initial situation, Agents a and b can indeed share full information of their hand by performing each a single public announcement. For example, from the situation of Figure 5.1, agents can do the following. Notice first that a cannot announce the hand, because c would then get a forbidden information. However a may announce a set of possible hands so that Agent b can infer Agent a's hand but Agent c cannot. Let Agent a announce the sentence "My hand is either 134, 126, 367, 465 or 275". Whichever hand Agent b and Agent c have, Agent b will always be able to deduce Agents a's hand while Agent c will never find out any card of Agent a. After Agent a's announcement, Agent b actually knows all hands of the players and achieves the goal by announcing Agent c's hand so that Agent a knows all hands.

We formalize the Russian cards puzzle with a symbolic Kripke model and state a formula expressing that both agents a and b commonly know the card configurations while Agent c does not know any others' card.

To do so, we first let proposition $p_{i,a}$ denote the fact "Agent a has card i", where $i \in \{1, \ldots, 7\}$ ranges over the set of cards, and we let AP_σ be the set of propositions $p_{i,a}, p_{i,b}, p_{i,c}$, where $i \in \{1, \ldots, 7\}$.

Legal hands. *Let S_7 be the set of all permutations of $(1, 2, 3, 4, 5, 6, 7)$ whose typical element is $\sigma = (\sigma(1), ..., \sigma(7))$. The valuations that correspond to legal hands are the models of the formula $\varphi_{\ell h}$ defined by*

$$\varphi_{\ell h} = \bigvee_{\sigma \in S_7} \varphi_\sigma$$

where φ_σ describes the hands given by the permutation $\sigma = (\sigma(1), ..., \sigma(7)) \in S_7$. Namely,

$$\varphi_\sigma = p_{\sigma(1),a} \wedge p_{\sigma(2),a} \wedge p_{\sigma(3),a} \wedge p_{\sigma(4),b} \wedge p_{\sigma(5),b} \wedge p_{\sigma(6),b} \wedge p_{\sigma(7),c} \wedge \bigwedge_{p \in AP_\sigma \setminus \{p_{\sigma(1),a},\dots,p_{\sigma(7),c}\}} \neg p.$$

The public announcement $\varphi_{\ell h}!$ restricts the set of valuations to those denoting legal hands.

Symbolic Kripke model. *The programs for the three players are the following.*

$$\begin{aligned} \pi_a &= set(p_{1,b}, ..., p_{7,b}, p_{1,c}, ..., p_{7,c}) \\ \pi_b &= set(p_{1,a}, ..., p_{7,a}, p_{1,c}, ..., p_{7,c}) \\ \pi_c &= set(p_{1,a}, ..., p_{7,a}, p_{1,b}, ..., p_{7,b}) \end{aligned}$$

Each program expresses that its agent considers possible all worlds obtained by changing other players' cards, and in a legal manner since formula $\varphi_{\ell h}$ has previously been announced.

Formula. *We express formula φ stating that both agents a and b commonly know the card configurations while Agent c does not know any others' card.*

$$\varphi = \bigvee_{\sigma \in S_7} (C_{\{a,b\}} \varphi_\sigma) \wedge \bigwedge_{p \in \{p_{1,a},\dots,p_{7,a},p_{1,b},\dots,p_{7,b}\}} \neg(K_c p \vee K_c \neg p)$$

We can show that for all $w \in \mathscr{U}$, $\mathscr{U}, w \not\models \langle \varphi_{\ell h}! \rangle \langle \bullet!_a \rangle \varphi$ but for all $w \in \mathscr{U}$, $\mathscr{U}, w \models [\varphi_{\ell h}!] \langle \bullet!_a \rangle \langle \bullet!_b \rangle \varphi$.

Notation 1. *In the rest of the chapter, we suppose that the symbolic Kripke model $\mathfrak{M}$ is clear from the context. Thus, for every set of valuations $U \subseteq \mathscr{U}$, for every valuation $u \in U$, we simply write (U, u) instead of $(\mathcal{M}^U_{\mathfrak{M}}, u)$.*

5.3.2 Relevant set of propositions

We establish non-surprising but important results regarding the relevant set of propositions that should be considered in a symbolic model. Given a symbolic model $\mathfrak{M}$ and a formula $\varphi \in PAPL$, the set of atomic propositions that appear in both is sufficient to evaluate the truth value of φ. In particular, it is not necessary to consider arbitrary

announcements that involve formulas referring to propositions not occurring in any of the program π among $\mathfrak{M}$ and in φ.

This property (stated in Proposition 15) plays an important role in the design of our algorithms in Section 5.4. Before we prove this property, we need to introduce notations and to define the notion of relevant propositions.

In the rest of this proof, given a program π, two sets of valuations $W, U \subseteq \mathscr{U}$, and a valuation $w \in W \cap U$, we consider the two following sets.

- $\widetilde{pre}_{\pi,W}(U) = \left\{w \in W \mid \forall v \in W, w \xrightarrow{\pi} v \text{ implies } v \in U\right\}$ is the set of worlds whose all π-successors are in U;
- $post_{\pi,W}(U) = \left\{v \in W \mid \exists w \in U, w \xrightarrow{\pi} v\right\}$ is the set of worlds that are π-successors of worlds in U.

We now define relevant propositions.

Definition 62. *Given a symbolic model $\mathfrak{M}$, we define the* set of relevant propositions *$AP(\varphi)$ in a formula φ, $AP(\gamma)$ in a protocol γ, and $AP(\pi)$ in a program π as follows, in a crossed-inductive manner.*

- *For formulas:*

– $AP(\top) = \emptyset$; $AP(p) = \{p\}$;
– $AP(\neg\varphi) = AP(\varphi)$;
– $AP(\varphi_1 \vee \varphi_2) = AP(\varphi_1) \cup AP(\varphi_2)$;
– $AP(K_a\varphi) = AP(\pi_a) \cup AP(\varphi)$;
– $AP(C_G\varphi) = AP(\varphi) \cup \bigcup_{a\in G} AP(\pi_a)$;
– $AP(\langle\gamma\rangle\varphi) = AP(\gamma) \cup AP(\varphi)$;

- *For programs:*

– $AP(p \leftarrow \bot) = AP(p \leftarrow \top) = \{p\}$;
– $AP(\beta?) = AP(\beta)$;
– $AP(\pi_1;\pi_2) = AP(\pi_1 \cup \pi_2) = AP(\pi_1) \cup AP(\pi_2)$;

- *For protocols:*

– $AP(\varphi!) = AP(\varphi)$;
– $AP(\bullet!) = \emptyset$;
– $AP(\bullet!_G) = \bigcup_{a\in G} AP(\pi_a)$;
– $AP(\varphi?) = AP(\varphi)$;
– $AP(\gamma_1;\gamma_2) = AP(\gamma_1 \cup \gamma_2) = AP(\gamma_1) \cup AP(\gamma_2)$;
– $AP(\gamma^*) = AP(\gamma)$;

Example 29. *If $\pi_a = set(q, r)$, then $AP(K_a p) = \{p, q, r\}$.*

Given a symbolic model $\mathfrak{M}$, sole the set $AP(\varphi)$ is relevant to evaluate a formula φ, as stated by the following proposition.

Proposition 15. *Let $\varphi \in PAPL$ and $\mathfrak{M}$ be a symbolic model. For every $U \subseteq \mathscr{U}$ and $u \in T$,*

$$U, u \models \varphi \text{ iff } U_u^\varphi, u^\varphi \models \varphi$$

where $U_u^\varphi = \{v^\varphi \mid v \in U \text{ and for all } p \in AP \setminus AP(\varphi), p \in v \text{ iff } p \in u\}$ with $v^\varphi = v \cap AP(\varphi)$.

Proof. In Appendix E.1. □

Thanks to Proposition 15, we can alternatively characterize the semantics of $\langle \bullet !_G \rangle \varphi$ by the existence of subsets of valuations U_a instead of the existence of formulas ψ_a as stated in Proposition 16. This characterization will be used for the design of algorithms in Section 5.4.

Proposition 16. *Let AP be a finite set. For all set $W, w \in W, \varphi$, $(W, u) \models \langle \bullet !_G \rangle \varphi$ iff there exist $(U_a)_{a \in G} \subseteq W$, such that $U_a = \widetilde{pre}_{\pi_a, W}(post_{\pi_a, W}(U_a))$, and $\bigcap_{a \in G} U_a, w \models \varphi$*

Proof. In Appendix E.2. □

We end this section by redefining the notion of size to take in account the symbolic model:

Definition 63 (Size of programs, formulas, protocols and symbolic Kripke models)**.** *Given a symbolic Kripke model $\mathfrak{M}$, the sizes of a program π, a formula φ and of a protocol γ, written $|\pi|$, $|\varphi|, |\gamma|$ are mutually inductively defined as follows.*

- *Programs π*
  - $|p \leftarrow \beta| = |\beta?| = 1 + |\beta|$;
  - $|\pi_1; \pi_2| = |\pi_1 \cup \pi_2| = 1 + |\pi_1| + |\pi_2|$.
- *Formulas φ:*
  - $|\top| = |p| = 1$
  - $|\neg\psi| = 1 + |\psi|$
  - $|\psi_1 \vee \psi_2| = 1 + |\psi_1| + |\psi_2|$
  - $|\hat{K}_a\varphi| = 1 + |\pi_a| + |\varphi|$

proc $relation_{\mathsf{yes}}(w, u, \pi)$ $|\pi|$
  $\triangleright$ accepts whenever $w \xrightarrow{\pi} u$
  case $\pi = p{\leftarrow}\beta$:
    if $(w \models \beta \wedge (u = w \cup \{p\})) \vee (w \not\models \beta \wedge (u = w \setminus \{p\}))$ **then accept else reject**
  case $\pi = \pi_1; \pi_2$: $(\exists)$ $v \in \mathscr{U}$; $(\forall)$ $relation_{\mathsf{yes}}(w, v, \pi_1)$ **and** $relation_{\mathsf{yes}}(v, u, \pi_2)$
  case $\pi = \pi_1 \cup \pi_2$: $(\exists)$ $relation_{\mathsf{yes}}(w, u, \pi_1)$ **or** $relation_{\mathsf{yes}}(w, u, \pi_2)$
  case $\pi = \beta?$: **if** $w = u \wedge w \models_{PL} \beta$ **then accept else reject**

Figure 5.2: Algorithm for program relation verification.

- $|C_G\psi| = 1 + |AP| + \sum_{a\in G} |\pi_a| + |\psi|$
- $|\langle\gamma\rangle\varphi| = 1 + |\gamma| + |\varphi|$

- *Announcement protocols γ:*
  - $|\varphi!| = |\varphi?| = 1 + |\varphi|$;
  - $|\bullet!| = 1$
  - $|\bullet!_G| = 1 + \sum_{a\in G} |\pi_a|$;
  - $|\gamma_1 \cup \gamma_2| = |\gamma_1; \gamma_2| = 1 + |\gamma_1| + |\gamma_2|$;
  - $|\gamma^*| = |AP| + |\gamma|$.

5.4 Upper bounds

We design algorithms and establish upper bound complexity results. The alternating algorithm to decide whether $w \xrightarrow{\pi} u$, $relation_{\mathsf{yes}}(w, u, \pi)$ is in Figure 5.2 and is the same than the one of Chapter 3. In Subsection 5.4.1, we present an alternating polynomial-time algorithm to model check against $PAPL$-formulas with star-free and alternation-free protocols only; since APTIME = PSPACE [CS76], we obtain a PSPACE upper bound (Theorem 17). In Subsection 5.4.2, we transform the previous algorithm into an $\mathrm{A_{pol}}$EXPTIME algorithm for model-checking against formulas that may contain protocols with stars and arbitrary announcement operators, yielding Theorem 18.

By Proposition 15, $\mathscr{U}, w \models \varphi$ is equivalent to $\mathscr{U}^{\varphi}_{w}, w^{\varphi} \models \varphi$ so we abuse notations by taking $\mathscr{U}$ for $\mathscr{U}^{\varphi}_{w}$, w for w^{φ} for w and AP for $AP(\varphi)$, thus considering only finite valuation sets.

proc $mc_{\mathsf{yes}}(L, w, \varphi)$ $|L| + |AP| + |\varphi|$
  ▷ accepts whenever $W_L, w \models \varphi$
  match φ **with**
    case $\varphi = p$: **if** $p \in w$ **then accept else reject**
    case $\varphi = \neg\psi$: $mc_{\mathsf{no}}(L, w, \psi)$
    case $\varphi = (\psi_1 \vee \psi_2)$: **(∃)** $mc_{\mathsf{yes}}(L, w, \psi_1)$ **or** $mc_{\mathsf{yes}}(L, w, \psi_2)$
    case $\varphi = K_a\psi$: **(∀)** $u \in \mathscr{U}$; **(∃)** $access_{\mathsf{no}}(w, u, \pi_a)$ **or** $in_{\mathsf{no}}(L, u)$ **or** $mc_{\mathsf{yes}}(L, u, \psi)$
    case $\varphi = C_G\psi$:
      (∀) $u \in \mathscr{U}$
      (∃) $access^*_{\mathsf{no}}(L, w, u, \bigcup_{a\in G} \pi_a, 2^{\#AP})$ **or** $in_{\mathsf{no}}(L, u)$ **or** $mc_{\mathsf{yes}}(L, u, \psi)$
    case $\varphi = \langle\psi!\rangle\chi$: **(∀)** $mc_{\mathsf{yes}}(L, w, \psi)$ **and** $mc_{\mathsf{yes}}(L :: \psi, w, \chi)$
    case $\varphi = \langle\psi?\rangle\chi$: **(∀)** $mc_{\mathsf{yes}}(L, w, \psi)$ **and** $mc_{\mathsf{yes}}(L, w, \chi)$
    case $\varphi = \langle\tau_1 \cup \tau_2\rangle\chi$: **(∃)** $mc_{\mathsf{yes}}(L, w, \langle\tau_1\rangle\chi)$ **or** $mc_{\mathsf{yes}}(L, w, \langle\tau_2\rangle\chi)$
    case $\varphi = \langle\tau_1; \tau_2\rangle\chi$: $mc_{\mathsf{yes}}(L, w, \langle\tau_1\rangle\langle\tau_2\rangle\chi)$

Figure 5.3: Algorithm mc_{yes} for model-checking against star-free and arbitrary-free formulas.

5.4.1 Model checking against formulas with star-free and arbitrary-free protocols

The aim of this subsection is to prove the following theorem.

Theorem 17. *The symbolic model checking problem against PAPL-formulas with star-free and arbitrary-free protocols is in* PSPACE.

The algorithm for the symbolic model checking problem against formulas with a star-free and arbitrary-free protocols is the following, where [ ] is (initially) the empty list of announcements, w is the current valuation and φ is the formula to model check. The algorithm for mc_{yes} is described in Figure 5.3.

proc $mainmc(w, \varphi)$
  ▷ accepts whenever $\mathscr{U}, w \models \varphi$
  $mc_{\mathsf{yes}}([\,], w, \varphi)$

We now detail algorithm mc_{yes} of Figure 5.3. In a call $mc_{\mathsf{yes}}(L, w, \varphi)$, L is the list of all previous announcements, $w \in \mathscr{U}$ is a valuation, and φ is a formula. We write $L :: \varphi$ for the list L followed by φ. The call $mc_{\mathsf{yes}}(L, w, \varphi)$ accepts whenever $W_L, w \models \varphi$, where the set of valuations W_L is the restriction of $\mathscr{U}$ to worlds satisfying all the announcements in list L. Notice that the overall complexity is confined to APTIME by avoiding an explicit representation of the announcement context W_L: instead, we use the list L of previous announcements, that implicitly denotes this context.

Propositional constructions in mc_{yes} (Figure 5.3). For $\varphi = p$, the call $mc_{\mathsf{yes}}(L, w, \varphi)$ accepts if and only if $p \in w$. As expected, for $\varphi = \neg\psi$, the call switches from mc_{yes} to mc_{no}. For $\varphi = \varphi_1 \vee \varphi_2$, $mc_{\mathsf{yes}}(L, w, \varphi)$ consists in existentially choosing $i \in \{1, 2\}$ and then calling $mc_{\mathsf{yes}}(L, w, \varphi_i)$.

Knowledge operators in mc_{yes} (Figure 5.3). The call $mc_{\mathsf{yes}}(L, w, K_a\varphi)$ consists in universally choosing a valuation $u \in \mathscr{U}$, then in performing two tests. The first test is $w \xrightarrow{\pi_a} u$, thus calling $relation_{\mathsf{yes}}(w, u, \pi_a)$. Notice that intermediate valuations along the execution of π_a are unconstrained. The second test is $u \in W_L$, by calling Algorithm $in_{\mathsf{yes}}(L, u)$ shown in Figure 5.4. If both tests succeed then $mc_{\mathsf{yes}}(L, u, \varphi)$ terminates without rejecting.

proc $in_{\mathsf{yes}}(L, w)$ $|L| + |AP| + 1$
  ▷ accepts whenever $w \in W_L$
  match L **with**
    case $L = [\,]$: **accept**
    case $L = L'::\varphi$: **(∀)** $mc_{\mathsf{yes}}(L', w, \varphi)$ **and** $in_{\mathsf{yes}}(L', w)$

Figure 5.4: Algorithm in_{yes} to check membership in the current context.

Common knowledge operators in mc_{yes} (Figure 5.3). The call $mc_{\mathsf{yes}}(L, w, C_G\psi)$ triggers $mc_{\mathsf{yes}}(L, w, K_{a_1}...K_{a_i}\psi)$ for all possible choices of a multiset $\{a_1, \ldots, a_i\}$ of G. The semantics of operator C_G yields checking that $mc_{\mathsf{yes}}(L, u, \varphi)$ does not reject, for all $u \in \mathscr{U}$ reachable from w by iterating $\bigcup_{a\in G} \pi_a$. This is the purpose of Algorithm $access^*_{\mathsf{yes}}$ (Figure 5.5). Regarding the size of the multiset $\{a_1, \ldots, a_i\}$, it is sufficient to consider at most $i = 2^{\#AP}$: indeed, each iteration of $\bigcup_{a\in G} \pi_a$ adds at least one valuation into the set of reachable valuations from w, so it will necessarily stabilize within at most $2^{\#AP}$ steps, which is the size of $\mathscr{U}$. Additionally, it is important to require that after each iteration, the obtained valuation is in W_L, ensured by the call in_{yes} in $access^*_{\mathsf{yes}}$. To remain in the class APTIME, Algorithm $access^*_{\mathsf{yes}}$ relies on a divide and conquer method, similarly to $access^*_{\mathsf{yes}}$ in Chapters 2 and 3.

Announcement protocol operators in mc_{yes} (Figure 5.3). We separate the case of a formula of the form $\langle\tau\rangle\chi$ into four cases depending on the form of τ: namely, either $\psi!$, or $\psi?$, or $\tau_1 \cup \tau_2$, or $\tau_1; \tau_2$.

proc $access^*_{\mathsf{yes}}(L, w, u, \pi, i)$ $\quad |L| + |AP| + \log_2(i) + |\pi|$

$\triangleright$ accepts whenever $w = u$ or there exist $v_1, \ldots, v_k \in W_L$ with $k < i$ such that $w \xrightarrow{\pi} v_1 \ldots \xrightarrow{\pi} v_{i-1} \xrightarrow{\pi} u$ (for $i > 1$)

case $i = 1$: **if** $u \neq w$ **then** $access_{\mathsf{yes}}(w, u, \pi)$ **else accept**

case $i \geq 2$:

- $(\exists)$ $v \in \mathscr{U}$
- $(\forall)$ $in_{\mathsf{yes}}(L, u)$ **and** $access^*_{\mathsf{yes}}(L, w, v, \pi, i/2)$ **and** $access^*_{\mathsf{yes}}(L, v, u, \pi, i/2)$

Figure 5.5: Iteration algorithm $access^*_{\mathsf{yes}}$ for symbolic accessibility relations (arbitrary-free and star-free case).

When considering an announcement (some ψ!), it is necessary to check that this announcement is consistent with w (by calling $mc_{\mathsf{yes}}(L, w, \psi)$) and that χ holds after the announcement (by calling $mc_{\mathsf{yes}}(L :: \psi, w, \chi)$). The case for $\psi?$ is similar: it is needed to check ψ in w (by calling $mc_{\mathsf{yes}}(L, w, \psi)$), but in this case the list L needs not being changed (hence the call to $mc_{\mathsf{yes}}(L, w, \chi)$). Regarding a non-deterministic choice $\tau_1 \cup \tau_2$, the algorithm existentially chooses which protocol to consider. Finally, for a sequence $\tau_1; \tau_2$, the algorithm uses the fact that formula $\langle \tau_1; \tau_2 \rangle \chi$ is equivalent to formula $\langle \tau_1 \rangle \langle \tau_2 \rangle \chi$.

We now state the correctness of Algorithms mc_{yes}, $access^*_{\mathsf{yes}}$, in_{yes}, and $mainmc$ (Proposition 17).

Proposition 17. *Given a list $L = \varphi_1; \ldots; \varphi_n$ of announcements, a valuation w, a formula φ, and $i \leq 2^{\#AP}$ we have:*

$mc_{\mathsf{yes}}(L, w, \varphi)$ *accepts*	*iff* $W_L, w \models \varphi$;
$mc_{\mathsf{no}}(L, w, \varphi)$ *accepts*	*iff* $W_L, w \not\models \varphi$;
$access^*_{\mathsf{yes}}(L, w, u, \pi, i)$ *accepts*	*iff there are* $w_1, \ldots, w_{j-1} \in W_L$ *such that* $w \xrightarrow{\pi} w_1 \xrightarrow{\pi} \ldots \xrightarrow{\pi} w_{j-1} \xrightarrow{\pi} u$ *and* $j \leq i$;
$access^*_{\mathsf{no}}(L, w, u, \pi, i)$ *accepts*	*iff for all* $w_1, \ldots, w_{j-1} \in W_L$ *such that* $j \leq i$, *it is false that* $w \xrightarrow{\pi} w_1 \xrightarrow{\pi} \ldots \xrightarrow{\pi} w_{j-1} \xrightarrow{\pi} u$;
$in_{\mathsf{yes}}(L, w)$ *accepts*	*iff* $w \in W_L$;
$in_{\mathsf{no}}(L, w)$ *accepts*	*iff* $w \notin W_L$;
$mainmc(w, \varphi)$ *accepts*	*iff* $\mathscr{U}, w \models \varphi$.

Proof. Let $H(n)$ be the Proposition 17 but only for all L, w, φ, i such that $|L| + |\varphi| \leq n$. We prove $H(n)$ by induction on n. The case $access^*_{\mathsf{yes}}$ is proven by another induction on i. Details are left to the reader. □

Algorithm *mainmc* is achieved by an alternating Turing machine running in polynomial time. Indeed in the algorithms, the quantities in gray decrease strictly with the recursive calls, and are polynomial in the size of the input. Furthermore in each recursive call, only polynomial time instructions are performed. Since PSPACE = APTIME, it concludes the proof of Theorem 17.

5.4.2 Model checking against $PAPL$-formulas

In this section, we prove the following theorem.

Theorem 18. *The symbolic model checking problem against formulas (even with protocols with stars and arbitrary announcements) is in* A_{pol}EXPTIME.

In order to prove Theorem 18, we provide the following model checking algorithm *mainmc* that resorts on Algorithm mc_{yes} subsequently defined (Figure 5.6).

proc $mainmc(w, \varphi)$
| $\triangleright$ accepts whenever $\mathscr{U}, w \models \varphi$
| $mc_{\mathsf{yes}}(\mathscr{U}, w, \varphi)$

The algorithm mc_{yes}, shown in Figure 5.6 is similar to the algorithm of Figure 5.3. However, instead of using the history of announcements L, we directly consider the context W_L, thus written W in the algorithms. Namely, the call to $mc_{\mathsf{yes}}(L, w, \varphi)$ in Figure 5.3 is now replaced by the call $mc_{\mathsf{yes}}(W, w, \varphi)$, and similarly for all other auxiliary algorithms. Typically, for the case of a formula of the form $K_a\varphi$, and in order to check that a given valuation u is in W, we replace the call $in_{\mathsf{yes}}(L, u)$ (in Figure 5.3) by choosing directly u in W (in Figure 5.6).

We now detail each case of Algorithm mc_{yes} of Figure 5.6.

Common knowledge. We tune the algorithms $access^*_{\mathsf{yes}}$ and $access^*_{\mathsf{no}}$ of Figure 5.5 into those in Figure 5.7.

Announcement protocols. We define algorithm $exec_{\mathsf{yes}}$ (Figure 5.8) such that the call $exec_{\mathsf{yes}}(W, w, W', \gamma)$ checks that $(W, w) \overset{\gamma}{\rightsquigarrow} (W', w)$. For announcements $\varphi!$, we check that W' is the restriction of W with respect to the announcement φ: namely every element u in W' should satisfy φ ($mc_{\mathsf{yes}}(W, u, \varphi)$) and every element v outside W' should not satisfy φ

proc $mc_{\mathsf{yes}}(W, w, \varphi)$ $|AP| + |\varphi|$
  ▷ accepts whenever $W, w \models \varphi$
  match φ **with**
    case $\varphi = p$: **if** $p \in w$ **then accept else reject**
    case $\varphi = \neg\psi$: $mc_{\mathsf{no}}(W, w, \psi)$
    case $\varphi = (\psi_1 \vee \psi_2)$: **(∃)** $mc_{\mathsf{yes}}(W, w, \psi_1)$ **or** $mc_{\mathsf{yes}}(W, w, \psi_2)$
    case $\varphi = K_a\psi$: **(∀)** $u \in W$; **(∃)** $access_{\mathsf{no}}(w, u, \pi_a)$ **or** $mc_{\mathsf{yes}}(W, u, \psi)$
    case $\varphi = C_G\psi$:
      (∀) $u \in W$
      (∃) $access^*_{\mathsf{no}}(W, w, u, \bigcup_{a\in G} \pi_a, 2^{\#AP})$ **or** $mc_{\mathsf{yes}}(W, u, \psi)$
    case $\varphi = \langle\gamma\rangle\chi$:
      (∃) $W' \subseteq W \mid w \in W'$
      (∀) $exec_{\mathsf{yes}}(W, w, W', \gamma)$ **and** $mc_{\mathsf{yes}}(W', w, \chi)$

Figure 5.6: Model checking sub-procedure mc_{yes} that handles protocols with arbitrary announcements and Kleene star.

$(mc_{\mathsf{no}}(W, v, \varphi))$. For an arbitrary announcement $\bullet!$, the call $exec_{\mathsf{yes}}$ simply succeeds since $W' \subseteq W$ is granted.

For group announcements $\bullet!_G$, we guess subsets $(W_a)_{a\in G}$ of W containing w. We then check whether there exist $(\psi_a)_{a\in G}$ such that for all $a \in G$, $(W, w) \xrightarrow{K_a\psi_a!} (W_a, w)$. According to Proposition 16, it amounts to checking that $\widetilde{pre}_{\pi,W}(post_{\pi,W}(W_a)) = W_a$, as achieved by algorithm $stable_{\mathsf{yes}}$ shown in Figure 5.10. The idea for checking that $\widetilde{pre}_{\pi,W}(post_{\pi,W}(W_a)) = W_a$ is that for all $w \in W_a$, check that for all $v \in W$ such that $w \xrightarrow{\pi_a} v$ (which is done by filtering the other ones by $access_{\mathsf{no}}(w, v, \pi_a)$), all π_a-predecessors of v are in W_a. Therefore, the algorithm $stable_{\mathsf{yes}}$ checks that $\widetilde{pre}_{\pi,W}(post_{\pi,W}(W_a)) \subseteq W_a$ [3]. After having chosen the W_a, the model-checking goes on with the new current set of valuations $\bigcap_{a\in G} W_a$.

For γ'^*, we introduce the algorithm $exec^*_{\mathsf{yes}}$ (Figure 5.9), based on divide-and-conquer as $access^*_{\mathsf{yes}}$.

3. The other implication is always true.

proc $access^*_{\mathsf{yes}}(W, w, u, \pi, i)$ $|AP| + \log_2(i) + |\pi|$
- $\triangleright$ accepts whenever $w = u$ or there exist $v_1, \ldots, v_k \in W$ with $k < i$ such that $w \xrightarrow{\pi} v_1 \ldots \xrightarrow{\pi} v_{i-1} \xrightarrow{\pi} u$ (for $i > 1$)
- **case** $i = 1$: **if** $u \neq w$ **then** $access_{\mathsf{yes}}(w, u, \pi)$ **else accept**
- **case** $i \geq 2$: **(∃)** $v \in W$; **(∀)** $access^*_{\mathsf{yes}}(W, w, v, \pi, i/2)$ **and** $access^*_{\mathsf{yes}}(W, v, u, \pi, i/2)$

Figure 5.7: Iteration algorithms $access^*_{\mathsf{yes}}$ for programs.

proc $exec_{\mathsf{yes}}(W, w, W', \gamma)$ $|AP| + |\gamma|$
- $\triangleright$ accepts whenever $(W, w) \overset{\gamma}{\rightsquigarrow} (W', w)$
- **match** γ **with**
  - **case** $\gamma = \varphi!$:
    - **(∀)** $(u, v) \in W' \times (W \backslash W')$
    - **(∀)** $mc_{\mathsf{yes}}(W, u, \varphi)$ **and** $mc_{\mathsf{no}}(W, v, \varphi)$
  - **case** $\gamma = \bullet!$: **accept**
  - **case** $\gamma = \bullet!_G$:
    - **(∃)** $(W_a)_{a \in G} \mid$ for all $a \in G, W_a \subseteq W, w \in W_a$
    - **if** $\bigcap_{a \in G} W_a \neq W'$ **then reject**
    - **else (∀)** $a \in G$; $stable_{\mathsf{yes}}(W, W_a, \pi_a)$
  - **case** $\gamma = \gamma_1; \gamma_2$:
    - **(∃)** $W'' \subseteq W$ such that $W' \subseteq W''$
    - **(∀)** $exec_{\mathsf{yes}}(W, w, W'', \gamma_1)$ **and** $exec_{\mathsf{yes}}(W'', w, W', \gamma_2)$
  - **case** $\gamma = \gamma_1 \cup \gamma_2$: **(∃)** $k \in \{1, 2\}$; $exec_{\mathsf{yes}}(W, w, W', \gamma_k)$
  - **case** $\gamma = \varphi?$: **(∀) if** $W' \neq W$ **then reject and** $mc_{\mathsf{yes}}(W, w, \varphi)$
  - **case** $\gamma = \gamma'^*$: **(∃)** $i \in \{0, ..., 2^{\#AP} - 1\}$ $exec^*_{\mathsf{yes}}(W, w, W', \gamma', i)$

Figure 5.8: Dual path searching algorithm $exec_{\mathsf{yes}}$ for announcements.

proc $exec^*_{\mathsf{yes}}(W, w, W', \gamma, i)$ $|AP| + |\gamma| + \log_2(i)$
- $\triangleright$ accepts whenever $(W, w) \overset{\gamma^i}{\rightsquigarrow} (W', w)$
- **case** $i = 1$: **if** $W = W'$ **then accept else** $exec_{\mathsf{yes}}(W, w, W', \gamma)$
- **case** $i \geq 2$:
  - **(∃)** W'' s.th. $W' \subseteq W'' \subseteq W$
  - **(∀)** $exec^*_{\mathsf{yes}}(W, w, W'', \gamma, i/2)$ **and** $exec^*_{\mathsf{yes}}(W'', w, W', \gamma, i/2)$

Figure 5.9: Iteration algorithm $exec^*_{\mathsf{yes}}$.

proc $stable_{\mathsf{yes}}(W, W', \pi)$ $|\pi| + 1$
  ▷ accepts whenever $\widetilde{pre}_{\pi,W}\ (post_{\pi,W}(W')) \subseteq W'$
  (∀) $(w, v, u) \in W' \times W \times W$
  if $u \notin W'$ **then (∃)** $access_{\mathsf{no}}(w, v, \pi)$ **or** $access_{\mathsf{no}}(u, v, \pi)$

Figure 5.10: Algorithm that checks that $\widetilde{pre}_{\pi,W}\ (post_{\pi,W}(W')) \subseteq W'$.

We now state the correction (Proposition 18).

Proposition 18. *We have:*

$mc_{\mathsf{yes}}(L, w, \varphi)$ *accepts*	*iff* $W_L, w \models \varphi$;
$mc_{\mathsf{no}}(L, w, \varphi)$ *accepts*	*iff* $W_L, w \not\models \varphi$;
$access^*_{\mathsf{yes}}(L, w, u, \pi, i)$ *accepts*	*iff there are* $w_1, \ldots, w_{j-1} \in W_L$ *such that* $w \xrightarrow{\pi} w_1 \xrightarrow{\pi} \ldots \xrightarrow{\pi} w_{j-1} \xrightarrow{\pi} u$ *and* $j \leq i$;
$access^*_{\mathsf{no}}(L, w, u, \pi, i)$ *accepts*	*iff for all* $w_1, \ldots, w_{j-1} \in W_L$ *such that* $j \leq i$, *it is false that* $w \xrightarrow{\pi} w_1 \xrightarrow{\pi} \ldots \xrightarrow{\pi} w_{j-1} \xrightarrow{\pi} u$;
$in_{\mathsf{yes}}(L, w)$ *accepts*	*iff* $w \in W_L$;
$in_{\mathsf{no}}(L, w)$ *accepts*	*iff* $w \notin W_L$;
$exec_{\mathsf{yes}}(W, w, W', \gamma)$ *accepts*	*iff* $(W, w) \overset{\gamma}{\rightsquigarrow} (W', w)$
$exec_{\mathsf{no}}(W, w, W', \gamma)$ *accepts*	*iff* $(W, w) \not\overset{\gamma}{\rightsquigarrow} (W', w)$
$exec^*_{\mathsf{yes}}(W, w, W', \gamma, i)$ *accepts*	*iff* $(W, w) \overset{\gamma^j}{\rightsquigarrow} (W', w)$ *for some* $j \leq i$
$exec_{\mathsf{no}}(W, w, W', \gamma)$ *accepts*	*iff* $(W, w) \not\overset{\gamma}{\rightsquigarrow} (W', w)$ *for all* $j \leq i$
$stable_{\mathsf{yes}}(W, W', \pi)$ *accepts*	*iff* $\widetilde{pre}_{\pi,W}\ (post_{\pi,W}(W')) \subseteq W'$
$stable_{\mathsf{no}}(W, W', \pi)$ *accepts*	*iff* $\widetilde{pre}_{\pi,W}\ (post_{\pi,W}(W')) \not\subseteq W'$
$mainmc(w, \varphi)$ *accepts*	*iff* $\mathscr{U}, w \models \varphi$.

Proof. By induction on φ, i, γ. □

The algorithm *mainmc* runs on exponential time and introduces a polynomial number of alternations. Indeed, the quantities in gray are polynomial and decrease strictly, and in each recursive call, there is a polynomial number of alternations and the call runs in exponential time.

5.5 Lower bounds

In this section, we establish tight lower bounds for the symbolic model checking problem. First, as DL-PA-model checking is PSPACE-hard [Her+11], we obtain that:

Theorem 19. *The symbolic model checking against formulas with arbitrary-free and star-free protocols is* PSPACE*-hard.*

To prove the other lower bound results of Table 5.1, we first prove in Subsection 5.5.1, as an intermediate step, the NEXPTIME lower bound for the fragment $\exists_1 PAPL$ defined as the restriction to formulas of the form $\langle \bullet! \rangle \varphi$ where φ does not contain any arbitrary public announcement and no protocols. Then we use this NEXPTIME lower bound in Subsection 5.5.2 to show the $\mathrm{A_{pol}}$EXPTIME-hardness for the fragment of $PAPL$ containing only formulas with star-free protocols, by taking alternations of $\exists_1 PAPL$-formulas. The proofs described generalize Cook's theorem [Coo71]. Cook's theorem states that we can express the existence of a polynomially long accepting execution of a given non-deterministic Turing machine with a formula of propositional logic. Analogously, we will prove that:

- the existence of an exponentially long accepting execution of a given non-deterministic Turing machine can be stated by a formula of $\exists_1 PAPL$.
- the existence of an exponentially long accepting execution of a given alternating Turing machine with a polynomial number of alternations can be stated by a with a formula of $PAPL$ with star-free protocols.

In order to ease the notations, we confuse an agent and its program. We write K_π instead of K_a, where a is an agent whose program is π.

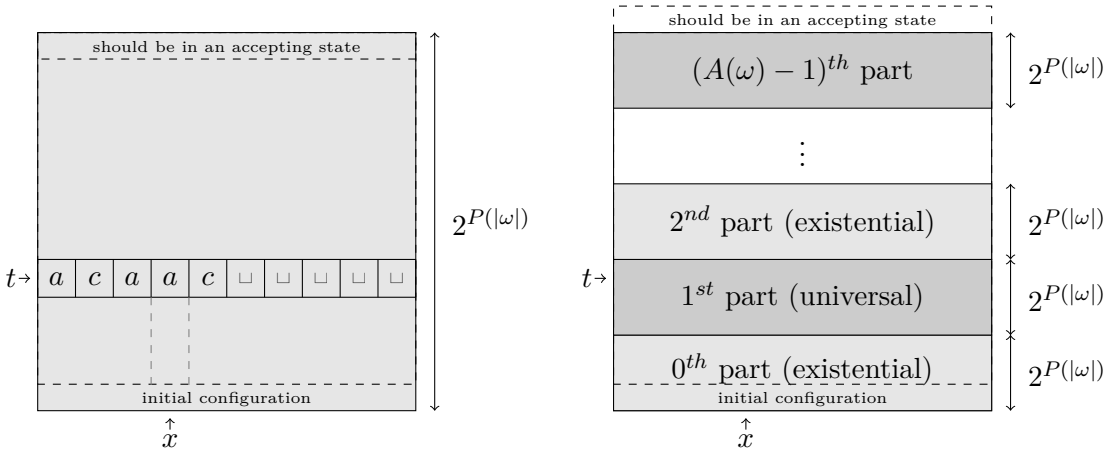
In the Subsection 5.5.3, we tackle the $\mathrm{A_{pol}}$EXPTIME-hardness for the case of protocols without arbitrary announcements.

5.5.1 NEXPTIME-hardness for $\exists_1 PAPL$

The aim of this Subsection is to prove Theorem 20.

Theorem 20. *The $\exists_1 PAPL$-model checking problem is* NEXPTIME*-hard.*

Proof. Let L be a NEXPTIME problem. We define a polynomial reduction tr from L to the $\exists_1 PAPL$-model checking problem. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc})$ be an non-deterministic Turing machine running in exponential time that decides L. Q is a finite set of states, Σ is the finite input alphabet, Γ is the finite tape alphabet where the blank symbol $\sqcup$ is in Γ and $\Sigma \subseteq \Gamma$, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$, q_0 is the start state and q_{acc} is the accept state. We suppose here that all end-states that are not q_{acc} are rejecting. Let P be



a. NEXPTIME case **b.** A_{pol}EXPTIME case

Figure 5.11: Tableaux for executions of M

a polynomial function such that for all inputs $\omega \in \Sigma^*$, the length of any execution of M on ω is bounded by $2^{P(|\omega|)}$. Note that the length of the non-empty part of the tape (that is the smallest portion of the tape such that there are only occurrences of the symbol ␣ on the left and on the right of that portion) is also bounded by $2^{P(|\omega|)}$ at any step of the execution.

Let ω be an instance of L. We encode the existence of an accepting execution of M for ω (that is, which ends in the state q_{acc}) in an $\exists_1 PAPL$-model checking instance $tr(\omega)$. Without loss of generality, we suppose that the machine M is built so that we loop on state q_{acc}: formally $(q_{acc}, a, q_{acc}, a, 0) \in \delta$ for all $a \in \Gamma$.

The atomic propositions to represent an execution. An execution of M is represented by a *tableau* whose rows are the configurations of M, as depicted in Figure 5.11a. The integers x and t in $\left\{0, \ldots, 2^{P(|\omega|)} - 1\right\}$ respectively represent a cell index on the tape and the time index. We identify x and t with their binary representations respectively expressed by means of the atomic propositions $x_1, \ldots, x_n$ and $t_1, \ldots, t_n$ where $n = P(|\omega|)$. For instance, if $n = 4$, $x_1, \neg x_2, x_3, \neg x_4$ represents $x = 5$.

We use the following atomic propositions to represent a given tape cell of index x at a given moment t:

- $x = x_1 \ldots x_n$: the position of the current tape cell;
- $t = t_1 \ldots t_n$: the moment we consider;
- *cur*: true if the cursor of the tape is in position x at time t, false otherwise;

- $symbol_a$: true if the symbol a is stored at position x at time t, false otherwise;
- $state_q$: at time t, M is in state $q \in Q$ (q could be the initial state q_0, the accepting state q_{acc} or any other state of the finite state space of the Turing machine; notice that the truth values of propositions $state_q$ should be independent from x);
- $tr_{q,a,q',b,d}$: the transition starting from state $q \in Q$ and going to state $q' \in Q$, reading the symbol $a \in \Gamma$ on the tape and writing $b \in \Gamma$, moving the cursor on direction $d \in \{-1, 0, +1\}$ is executed at time t (note that the truth values of propositions $tr_{q,a,q',b,d}$ should be independent from x).

The set of the atomic propositions we just described is noted AP'.

Example 30. *Let us suppose that* $n = 4$. *Valuation* $\{x_1, x_3, t_2, state_{q'}, symbol_c, tr_{q',b,q'',a,-1}\}$ *is about time 2 (as* $\neg t_1, t_2, \neg t_3, \neg t_4$ *represents* $0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3$*) and the* 5^{th} *cell of the tape (as* $x_1, \neg x_2, x_3, \neg x_4$ *represents* $2^0 + 0 \times 2^1 + 2^2 + 0 \times 2^3$*):*

- *The cursor is not in the* 5^{th} *cell of the tape at time 2 (because cur is false);*
- *Letter c is written in the* 5^{th} *cell of the tape;*
- *The machine M is in state* q' *at time 2;*
- *From time 2 to time 3, the machine M triggers the transition from state* q' *to state* q'' *by reading b, writing a and moving the cursor left.*

The existence of an accepting execution as an arbitrary public announcement. Remark that there are *non legal* valuations in $2^{AP'}$, such as a valuation in which both $symbol_a$ and $symbol_b$ are true with $a \neq b$. The idea is to perform an arbitrary public announcement yielding to a set of legal valuations representing an accepting execution for ω of M. Each remaining valuation describes the content of one cell on the tape at a given moment along this execution. The rest of the proof consists in defining a formula of the form $\langle \bullet! \rangle \varphi$ such that φ expresses that the set resulting from the announcement represents an accepting execution of M on ω.

Abbreviations. To ease the reading of the formula expressing the accepting execution, we introduce the following abbreviations:

- Exactly one proposition among $\{p_1, \ldots, p_n\}$ is true
  $\exists!(p_1, \ldots, p_n) := \bigvee_{i \in \{1,\ldots,n\}} \left(p_i \wedge \bigwedge_{j \neq i} \neg p_j \right)$;
- $x{=}y$, $x{\geq}y$, etc. are formulas over $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ whose intended meaning is that the values of propositions $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are such that the number x (denoted by $x_1, \ldots, x_n$) is equal to y (denoted by $y_1, \ldots, y_n$), greater or equal than y, etc. We define these formulas by recurrence on the number of digits, with ϵ the empty word (0 digit):
  - $\epsilon = \epsilon$ is $\top$ and $x_1 \ldots x_n = y_1 \ldots y_n$ is $(x_n \leftrightarrow y_n) \wedge (x_1 \ldots x_{n-1} = y_1 \ldots y_{n-1})$
  - $\epsilon \geq \epsilon$ is $\top$ and $x_1 \ldots x_n \geq y_1 \ldots y_n$ is $(x_n \wedge \neg y_n) \vee ((y_n \rightarrow x_n) \wedge (x_1 \ldots x_{n-1} \geq y_1 \ldots y_{n-1}))$

  For instance the formula $x{=}5$ is equivalent to $x_1 \wedge \neg x_2 \wedge x_3 \wedge \bigwedge_{k=4}^{n} \neg x_k$. Formula $t{=}2^{P(|\omega|)}{-}1$ is equivalent to $\bigwedge_{k=1}^{n} t_k$.
- Non-deterministically choose values for $\{p_1, \ldots, p_n\}$ such that at least one p_i's value changes:
  $$\begin{aligned} set_{\neq}(p_1, \ldots, p_n) \quad := \quad & ((p_1?; \overline{p_1} \leftarrow \top) \cup (\neg p_1?; \overline{p_1} \leftarrow \bot)) \; ; \ldots ; \\ & ((p_n?; \overline{p_n} \leftarrow \top) \cup (\neg p_n?; \overline{p_n} \leftarrow \bot)) \; ; \; set(p_1, \ldots, p_n) \; ; \\ & \left(\neg \bigwedge_{i \in \{1,\ldots,n\}} (p_i \leftrightarrow \overline{p_i}) \right)?; (\overline{p_1} \leftarrow \bot); \ldots ; (\overline{p_n} \leftarrow \bot) \end{aligned}$$
  where $\overline{p_1} \ldots \overline{p_n}$ are fresh propositions;
- $x{\leftarrow}x{-}1$, $x{\leftarrow}x{+}1$ and $t{\leftarrow}t{+}1$ are programs of polynomial size in $|\omega|$ that respectively decrement x, increment x and increment t. Notice that $x{\leftarrow}x{+}1$ is *not executable* when $x = 2^{P(|\omega|)} - 1$ and that $x{\leftarrow}x{-}1$ is not executable when $x = 0$. More precisely:

$$x{\leftarrow}x{+}1 := \bigcup_{k=0}^{n-1} \left((\neg x_{k+1} \wedge \bigwedge_{i=1}^{k} x_i)?; x_1 \leftarrow \bot; \ldots x_k \leftarrow \bot; x_{k+1} \leftarrow \top \right)$$

$$x{\leftarrow}x{-}1 := \bigcup_{k=0}^{n-1} \left((x_{k+1} \wedge \bigwedge_{i=1}^{k} \neg x_i)?; x_1 \leftarrow \top; \ldots x_k \leftarrow \top; x_{k+1} \leftarrow \bot \right)$$

Definition of the reduction. The instance $tr(\omega)$ is of the form $(w, \langle \bullet ! \rangle K_{set(AP')}(\varphi_{exe} \wedge \varphi_{init} \wedge \varphi_{accept}))$.

- First, w is the valuation $\left\{ symbol_{\omega_0}, cur, state_{q_0} \right\}$. Valuation w represents the most-left tape cell in initial configuration of M ($\mathscr{U}, w \models (x = 0 \wedge t = 0)$).

Tableau structure	
1. $\hat{K}_{(x\leftarrow x+1 \cup x=2^{P(\lvert\omega\rvert)}-1?);set(state,symbol,cur,tr)} \top$	If $x < 2^{P(\lvert\omega\rvert)}$, the $x+1^{th}$ cell exists
2. $\hat{K}_{(t\leftarrow t+1 \cup t=2^{P(\lvert\omega\rvert)}-1?);set(state,symbol,cur,tr)} \top$	if $t < 2^{P(\lvert\omega\rvert)}$, time $t+1$ exists
States	
3. $\exists!(state_q)_{q\in Q}$	the machine is in a unique state
4. $\Big(state_q \rightarrow K_{set(x,state,symbol,cur,tr)} state_q\Big)$	the state of the machine only depends on time
Cursor	
5. $\hat{K}_{set(x,symbol,cur,tr)} cur$	on each possible tape, there must be a cursor
6. $cur \rightarrow K_{set_{\neq}(x,symbol,cur,tr)} \neg cur$	the cursor has a position
7. $(cur \rightarrow K_{set(cur)} cur) \wedge (\neg cur \rightarrow K_{set(cur)} \neg cur)$	at a given time, the cursor has a unique position
Tape	
8. $\exists!(symbol_a)_{a\in\Gamma}$	a tape cell contains a unique letter
9. $symbol_a \rightarrow K_{set(symbol)} symbol_a$	on each cell, there is no world in which all atomic propositions are the same except the letter written in the cell
Transitions	
10. $\exists!(tr_\delta)_{\delta\in\delta}$	there is a unique transition to be triggered
11. $\bigwedge_{\delta\in\delta} \Big(tr_\delta \rightarrow K_{set(x,state,symbol,cur,tr)} tr_\delta\Big)$	the triggered transition only depends on time
12. $\bigwedge_{a\in\Gamma}(\neg cur \wedge symbol_a) \rightarrow K_{t\leftarrow t+1} symbol_a$	Tape cells that are not under the cursor remain unchanged
13. $\bigwedge_{(q,a,q',b,d)\in\delta} tr_{q,a,q',b,d} \rightarrow state_q$	The triggered transition is compatible with the current state
14. $\bigwedge_{(q,a,q',b,d)\in\delta} tr_{q,a,q',b,d} \wedge cur \rightarrow symbol_a$	The triggered transition is compatible with th read letter
15. $\bigwedge_{(q,a,q',b,d)\in\delta} tr_{q,a,q',b,d} \wedge cur \rightarrow K_{t\leftarrow t+1} symbol_b$	The triggered transition is compatible with the written letter
16. $\bigwedge_{(q,a,q',b,d)\in\delta} tr_{q,a,q',b,d} \rightarrow K_{t\leftarrow t+1} state_{q'}$	The triggered transition is compatible with the next state
17. $\bigwedge_{(q,a,q',b,+1)\in\delta} tr_{q,a,q',b,d} \wedge cur \rightarrow K_{t\leftarrow t+1;x\leftarrow x+1} cur$	Behavior of the cursor that moves to the right
18. $\bigwedge_{(q,a,q',b,-1)\in\delta} tr_{q,a,q',b,d} \wedge cur \rightarrow K_{t\leftarrow t+1;x\leftarrow x-1} cur$	Behavior of the cursor that moves to the left
19. $\bigwedge_{(q,a,q',b,0)\in\delta} tr_{q,a,q',b,d} \wedge cur \rightarrow K_{t\leftarrow t+1} cur$	Behavior of the cursor that stays in the same cell

Table 5.2: Formulas that constraint the set of valuations to represent an execution

- Formula φ_{exe} is the conjunction of the formulas in Table 5.2. Formula φ_{exe} imposes the set of surviving valuations to represent an execution of M. For instance, formula 4 in Table 5.2 says that changing the current valuation via the program $set(x, state, symbol, cur, tr)$ (that only time t is fixed) will not change the truth valuation of proposition $state_q$.

- Formula φ_{init} says that in that execution, the configuration at time $t = 0$ is the initial configuration on ω. Formally: $\varphi_{init} := t{=}0 \rightarrow state_{q_0} \wedge [(x{=}0 \rightarrow (cur \wedge symbol_{\omega_0})) \wedge \bigwedge_{i=1}^{|\omega|-1}(x{=}i \rightarrow (\neg cur \wedge symbol_{\omega_i})) \wedge (x{\geq}|\omega| \rightarrow (\neg cur \wedge symbol_{\sqcup}))]$, where $\sqcup$ is the blank symbol.

- Formula φ_{accept} says that the configuration at time $t = 2^{P(|\omega|)}{-}1$ is an accepting configuration. Formally: $\varphi_{accept} := \left(t{=}2^{P(|\omega|)}{-}1\right) \rightarrow state_{q_{acc}}$.

By construction, $tr(\omega)$ is computable from ω in polynomial time in $|\omega|$ and $tr(\omega)$ is a positive instance of $\exists_1 PAPL$-model checking problem if and only if ω is a positive instance of L. □

5.5.2 $\mathrm{A_{pol}}$EXPTIME-hardness for $PAPL$ with star-free protocols

The aim of this subsection is to prove Theorem 21 given below:

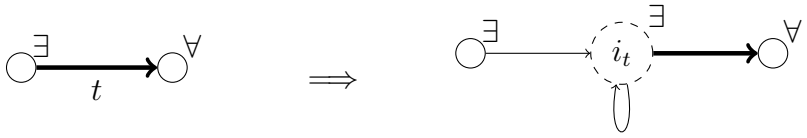
Theorem 21. *The symbolic model checking of any formula is* A_{pol}EXPTIME*-hard.*

To prove the theorem, we use Theorem 20 as a basic step for the part of an execution where all states are of the same type (existential or universal). Now we will alternate the use of $\langle\bullet!\rangle$ and $[\bullet!]$ operators to simulate alternations in executions of alternating Turing machines.

Proof. (of Theorem 21) Let L be a problem in $\mathrm{A_{pol}}$EXPTIME. We are going to define a polynomial reduction tr from L to our symbolic model checking problem. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, type)$ be an alternating Turing machine running in exponential time that decides L with a polynomial number of alternations, where $Q, \Sigma, \Gamma, \delta, q_0, q_{acc}$ are defined as in the proof of Proposition 20 and $type : Q \times \{\exists, \forall\}$ says whether a given state is existential ($\exists$) or universal ($\forall$). Let A be a polynomial function such that for all inputs ω, the number of alternations of any execution starting from ω is bounded by $A(|\omega|)$. Without loss of generality, we suppose that the number of alternations is exactly $A(|\omega|)$ and that the initial state is existential. The execution is segmented into maximal

parts in which each configuration is of the same type: in a given part, either all states are existential states or all are universal states. Let P be a polynomial function such that for all inputs ω, the length of any such part is bounded by $2^{P(|\omega|)}$.

Transformation of M so that all execution parts are of length $2^{P(|\omega|)}$. We transform the machine M so that we can suppose that the length of any part of an execution is exactly $2^{P(|\omega|)}$ in the following way. For all transitions t from an existential state to a universal state, we add an intermediate existential state i_t as pictured below:



Let us explain the transformation formally. For all transitions $\delta = (q, a, q', b, d) \in \delta$, such that $type(q) = \exists$ and $type(q') = \forall$, we add a new state i_t in δ, we remove the transition t from δ, we add the transitions $(q, a, i_t, a, 0), (i_t, a, q', b, d)$ to δ and also the transitions $(i_t, \alpha, i_t, \alpha, 0)$ for all tape symbols $\alpha \in \Gamma$.

We do the same transformation for transitions that go from a universal state to an existential state. Figure 5.11b shows the shape of an execution of M.

Abbreviations. Now we use the notation of the proof of Proposition 20 but now numbers x and t ranges over $\left\{0, \ldots, A(|\omega|)\times 2^{P(|\omega|)}\right\}$ so we take the number n of digits of the numbers x and t to be $P(\omega) + \log_2(A(\omega)) + 1$. Now we define formula $\varphi^{\exists}_{exe}$ which is similar to formula φ_{exe} (see proof of theorem 20) except that we impose end states of transitions to be existential. We define formula $\varphi^{\forall}_{exe}$ which is similar to formula φ_{exe} except that we impose end states of transitions to be universal.

We define the following abbreviations:

- We are in a k'^{th} part with $k' \geq k$: $k\times 2^{P(|\omega|)} \leq t$
- We are in a k'^{th} part with $k' \leq k$: $t < (k+1)\times 2^{P(|\omega|)}$
- Go to the k^{th} part : $\mathscr{U}_k := set(AP'); \left(\left(k\times 2^{P(|\omega|)} \leq t\right) \wedge \left(t < (k+1)\times 2^{P(|\omega|)}\right)\right)?$
- Go after the k^{th} part : $\mathscr{U}_k^{\text{after}} := set(AP'); \left(t > k\times 2^{P(|\omega|)}\right)?$
- We are at the end of the execution: $t = A(|\omega|)\times 2^{P(|\omega|)}$

- All valuations concerning the k^{th} part and beyond exist (i.e. the parts from the k^{th} part are unfixed):

$$\varphi^k_{\text{unfixed}} := K_{\mathscr{U}_k^{\text{after}}} \begin{pmatrix} \hat{K}_{x \leftarrow x+1 \cup \left(x=A(|\omega|)\times 2^{P(|\omega|)}\right)?} \top \wedge \hat{K}_{t \leftarrow t+1 \cup t=A(|\omega|)\times 2^{P(|\omega|)}?} \top \wedge \\ \bigwedge_{p \in AP' \setminus \{x_1,\ldots,x_n,t_1,\ldots,t_n\}} \left(\neg(p \rightarrow K_{set(p)} p) \wedge \neg(\neg p \rightarrow K_{set(p)} \neg p) \right) \end{pmatrix}$$

- Transitions at time $(k+1)\times 2^{P(|\omega|)}-1$ only end into universal states:

$$\varphi^{\exists \rightarrow \forall}_{exe,k} := \left(t=(k+1)\times 2^{P(|\omega|)}-1\right) \rightarrow \exists!(tr_\delta)_{\delta \in \delta | \delta \text{ ends in a universal state}}$$

- Transitions at time $(k+1)\times 2^{P(|\omega|)}-1$ only end into existential states:

$$\varphi^{\forall \rightarrow \exists}_{exe,k} := \left(t=(k+1)\times 2^{P(|\omega|)}-1\right) \rightarrow \exists!(tr_\delta)_{\delta \in \delta | \delta \text{ ends in a existential state}}$$

Describing the accepting condition. Now we define a sequence of formulas $(\psi_k)_{k \in \{0,\ldots,A(|\omega|)\}}$ that describe the execution of the Turing machine M one part after another. Assuming the execution is already defined until the time $2k \times 2^{P(|\omega|)}$, the formula ψ_{2k} chooses the next transitions to execute in the $(2k)^{th}$ part so that the last transition executed in the $(2k)^{th}$ part leads to a universal state. Moreover, it leaves all valuations in remaining parts unfixed so that formula ψ_{2k+1} is carrying on the rest of the execution. Assuming the execution is already defined until the time $(2k+1) \times 2^{P(|\omega|)}$, the formula ψ_{2k+1} checks that all possible executions in the $(2k+1)^{th}$ part are accepting. Assuming the execution is defined until the end, the last formula $\psi_{A(|\omega|)}$ checks that the last state is q_{acc}. Formally:

- $\psi_{A(|\omega|)} := K_{\mathscr{U}_{A(|\omega|)}}[(t = A(|\omega|)\times 2^{P(|\omega|)}) \rightarrow state_{q_{acc}}]$;

- for all k such that $2k+1 < A(|\omega|)$, $\psi_{2k+1} := [\bullet!] \left(\left(K_{\mathscr{U}_{2k+1}} (\varphi^{\forall}_{exe} \wedge \varphi^{\forall \rightarrow \exists}_{exe,2k+1}) \wedge \varphi^{2k+2}_{\text{unfixed}} \right) \rightarrow \psi_{2k+2} \right)$;

- for all k such that $0 \leq 2k < A(|\omega|)$, $\psi_{2k} := \langle \bullet! \rangle \left(K_{\mathscr{U}_{2k}} (\varphi^{\exists}_{exe} \wedge \varphi^{\exists \rightarrow \forall}_{exe,2k}) \wedge \varphi^{2k+1}_{\text{unfixed}} \wedge \psi_{2k+1} \right)$;

For all $k \in \{0,\ldots,A(|\omega|)\}$, we define the following property $P(k)$: For all $W \subseteq \mathscr{U}$, such that W defines a unique configuration C at time $t = k \times 2^{P(|\omega|)}$ and contains all valuations for $t > k \times 2^{P(|\omega|)}$, $W, w \models \psi_k$ if and only if C is an accepting configuration.

We can prove by induction on k that $P(k)$ is true for all $k \in \{0,\ldots,A(|\omega|)\}$.

Definition of the reduction. The instance $tr(\omega)$ is of the form (w, φ) where:

- w is the valuation $\left\{symbol_{\omega_0}, cur, state_{q_0}\right\}$;
- $\varphi = \langle \bullet! \rangle \left(K_{set(AP')} \varphi_{init} \wedge \varphi^0_{\text{unfixed}} \wedge \psi_0 \right)$.

Formula φ ensures that the initial configuration ($t = 0$) is fixed, leaves all valuations concerning $t > 0$ unfixed and ψ_0 checks whether the initial configuration is accepting. $tr(\omega)$ is computable from ω in polynomial time in $|\omega|$. We have that ω is a positive instance of L if and only if the initial configuration for ω in M is accepting if and only if (w, φ) is a positive instance of the model checking problem. □

5.5.3 $\mathrm{A_{pol}}$EXPTIME-hardness for $PAPL$ with protocols without arbitrary announcements

In this section, we prove that:

Theorem 22. *The symbolic model checking of formulas with protocols but without arbitrary announcements is* A_{pol}EXPTIME*-hard.*

Proof. We reduce the symbolic model checking of formulas with arbitrary announcements (and without protocols) to it. Suppose that the formula to check is over propositions $AP = \{p_1, \ldots, p_n\}$. The idea is to simulate an arbitrary announcement $\bullet!$ by a protocol $\gamma_{\bullet!}$. The protocol consists in iterating over all valuations, and, each time, either we keep the current valuation or we remove it from the current context.

For representing the current valuation in the iteration, we introduce fresh propositions $\iota_1, \ldots, \iota_n$ for each occurrence of $\bullet!$ in φ. $\iota_1, \ldots, \iota_n$ represents an integer in $\{0, \ldots, 2^n - 1\}$. Thus the new set of atomic propositions is $AP' = AP \cup \{\iota_i\}_{1 \leq i \leq n, \bullet! \in \varphi}$. The idea in each protocol is to iterate $\iota_1, \ldots, \iota_n$ over integers from $2^n - 1$ down to 0. At the beginning, all valuations are in the model hence integer $\iota_1 \ldots \iota_n = 1 \ldots 1 = 2^n - 1$ is the current integer.

We introduce symbolic accessibility relation $succ(\iota)$ to link a valuation over ι, different from $2^n - 1$ to the next integer representing the next integer. Formally, we define $succ(\iota_n \ldots \iota_1)$ (ι_1 is the least significant bit) by induction on n:

- $succ(\iota_1) = (\neg\iota_1?; \iota_1 \leftarrow \top)$;
- $succ(\iota_n \ldots \iota_1) = (\neg\iota_1?; \iota_1 \leftarrow \top) \cup (\iota_1?; \iota_1 \leftarrow \bot; succ(\iota_n \ldots \iota_2))$;

Now we define protocols to keep the current valuation (*keepval*), remove the current valuation (*removeval*), consider the next valuation (*nextval*):

- $keepval = \top!$: the model is unchanged;
- $removeval = K_{set(\iota)}(\neg\hat{K}_{succ(\iota)}\top \rightarrow \bigvee_{i=1}^{n}\neg(\iota_i \leftrightarrow p_i))$: we change the value of ι and if ι has no successor (meaning that ι is the current valuation/integer) then valuation $\vec{p}$ should be different from valuation ι;
- $nextval = \hat{K}_{succ(\iota)}\top!$: we remove valuations without successors (we remove valuation $\iota = 1\ldots1$, then valuation $\iota = 1\ldots10$, etc.).

The protocol is then:

$$\gamma_{\bullet!} = ((keepval \cup removeval); ((\neg end?; nextval) \cup end?))^*; end?$$

where formula *end* states the protocol is over. Formally: $end := K_{set(AP')}(\iota = 0)$.

In the reduction of Subsection 5.5.2 we now replace $\langle\bullet!\rangle$ by $\langle\gamma_{\bullet!}\rangle$ and $[\bullet!]$ by $[\gamma_{\bullet!}]$, thus transforming the formula φ into a new formula φ'. Let $\mathscr{U}' = 2^{AP'}$. The new model checking instance is now $\mathscr{U}', w \models \varphi'$ instead of $\mathscr{U}, w \models \varphi$ (by taking the same valuation in the new model checking instance, thus having all ι to 0). Both are equivalent. □

5.6 Non-symbolic model checking of PAPL

The aim of this section is to prove the following theorem.

Theorem 23. *The non-symbolic model checking against PAPL is* PSPACE*-complete.*

5.6.1 Hardness

To prove the PSPACE-hardness, we reduce the Quantified Boolean Formula (QBF) problem which is known to be PSPACE-complete [Sip96].

Definition 64 (QBF)**.** *A QBF formula is of the form $Q_1p_1Q_2p_2\ldots Q_np_n\beta$ where Q_i is a quantifier, either $\forall$ or $\exists$, and $p_i \in AP$ for all $i \in \{1,\ldots,n\}$, and β is a propositional formula.*

We define the following translation *tr* that takes a QBF formula and return a PAPL formula.

Definition 65 (Translation *tr* from QBF to PAPL)**.**

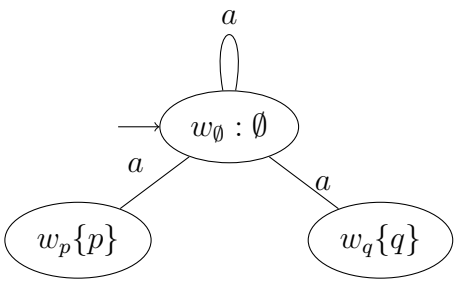


Figure 5.12: Example of a Kripke model for the hardness proof of Theorem 23 with $AP(\beta) = \{p, q\}$.

- $tr(p) = \hat{K}_a p$
- $tr(\neg\varphi) = \neg tr(\varphi)$
- $tr(\varphi_1 \wedge \varphi_2) = tr(\varphi_1) \wedge tr(\varphi_2)$
- $tr(\exists p\varphi) = \langle \neg p! \cup \top! \rangle tr(\varphi)$
- $tr(\forall p\varphi) = [\neg p! \cup \top!] tr(\varphi)$

The input Kripke model $\mathcal{M}$ is represented in Figure 5.12. Intuitively, the formula $tr(\varphi)$ prunes $\mathcal{M}$ with public announcements. Announcing $\neg p!$ means that we assign p to $\bot$ and announcing $\top!$ means we assign p to $\top$. In the resulting Kripke model, we evaluate β where p is replaced by $\hat{K}_a p$. Indeed, p is assigned to $\top$ is equivalent to p is present in the Kripke model after the announcement, thus p is assigned to $\top$ if and only if $\hat{K}_a p$ is true in the Kripke model. It means that when we evaluate β, p is true if and only if in the model $\hat{K}_a p$ is true. To express the quantifier, we consider the modality $\langle\rangle$ for $\exists$ and $[\,]$ for $\forall$, which effectively simulates the quantifier.

Definition 66 (Input model $\mathcal{M}$)**.** *The Kripke model $\mathcal{M} = (W, R_a, V)$ is defined as follows.*

- $W = \{w_\emptyset\} \cup \{w_p, p \in AP\}$
- $R_a = \{(w_\emptyset, w), w \in W\}$
- $V(w_\emptyset) = \emptyset; V(w_p) = \{p\}$

We now formally define the reduction.

Definition 67 (Reduction)**.** *Let φ be a QBF formula. Then the input of the model checking procedure is $(\mathcal{M}, w_\emptyset)$ for the pointed Kripke model and $tr(\varphi)$ for the formula to model check.*

The reduction is indeed polynomial and we can verify easily that φ is true if and only if $\mathcal{M}, w_\emptyset \models tr(\varphi)$, which concludes the PSPACE-hardness proof.

5.6.2 Membership

For the membership, we simply adapt the algorithms of Section 5.4.2 into the non-symbolic case (Figures 5.13, 5.14, 5.15, 5.16 and 5.17). The main difference is that $access_{\mathsf{yes}}(w, u, \pi_a)$ is now replaced by a checking of $(w, u) \in R_a$. In each recursive call, the time taken is now polynomial, because choosing $W' \subseteq W$ takes a polynomial amount of time. To insure that the notations are uniform with the symbolic case, we supposed that $\mathcal{M}$ is fixed and only focus on the current set of worlds W. We leave apart the detail of the algorithms since they are very similar to 5.4.2. As usual, the dual procedures are in Appendix D.

proc $mc_{\mathsf{yes}}(W, w, \varphi)$ $|AP| + |\varphi|$
- ▷ accepts whenever $W, w \models \varphi$
- **match** φ **with**
  - **case** $\varphi = p$: **if** $p \in V(w)$ **then accept else reject**
  - **case** $\varphi = \neg\psi$: $mc_{\mathsf{no}}(W, w, \psi)$
  - **case** $\varphi = (\psi_1 \vee \psi_2)$: **(∃)** $mc_{\mathsf{yes}}(W, w, \psi_1)$ **or** $mc_{\mathsf{yes}}(W, w, \psi_2)$
  - **case** $\varphi = K_a\psi$: **(∀)** $u \in W$; **if** $(w, u) \in R_a$ **then** $mc_{\mathsf{yes}}(W, u, \psi)$
  - **case** $\varphi = C_G\psi$:
    - **(∀)** $u \in W$
    - **(∃)** $access^*_{\mathsf{no}}(W, w, u, \bigcup_{a\in G} R_a, 2^{\log_2(|W|)+1})$ **or** $mc_{\mathsf{yes}}(W, u, \psi)$
  - **case** $\varphi = \langle\gamma\rangle\chi$:
    - **(∃)** $W' \subseteq W \mid w \in W'$
    - **(∀)** $exec_{\mathsf{yes}}(W, w, W', \gamma)$ **and** $mc_{\mathsf{yes}}(W', w, \chi)$

Figure 5.13: Model checking sub-procedure mc_{yes} that handles protocols with arbitrary announcements and Kleene star.

proc $access^*_{\mathsf{yes}}(W, w, u, R, i)$ $|AP| + \log_2(i)$
- ▷ accepts if $w = u$ or there exist $v_1, \ldots, v_k \in W$ with $k < i$ such that $w \xrightarrow{\pi} v_1 \ldots \xrightarrow{R} v_{i-1} \xrightarrow{R} u$ (for $i > 1$)
- **case** $i = 1$: **if** $u = w$ or $(w, u) \in R$ **then accept else reject**
- **case** $i \geq 2$: **(∃)** $v \in W$; **(∀)** $access^*_{\mathsf{yes}}(W, w, v, \pi, i/2)$ **and** $access^*_{\mathsf{yes}}(W, v, u, \pi, i/2)$

Figure 5.14: Iteration algorithms $access^*_{\mathsf{yes}}$ for programs.

proc $exec_{\mathsf{yes}}(W, w, W', \gamma)$ $|AP| + |\gamma|$
- $\triangleright$ accepts whenever $(W, w) \overset{\gamma}{\rightsquigarrow} (W', w)$
- **match** γ **with**
  - **case** $\gamma = \varphi!$:
    - $(\forall)$ $(u, v) \in W' \times (W \backslash W')$; $(\forall)$ $mc_{\mathsf{yes}}(W, u, \varphi)$ **and** $mc_{\mathsf{no}}(W, v, \varphi)$
  - **case** $\gamma = \bullet!$: **accept**
  - **case** $\gamma = \bullet!_G$:
    - $(\exists)$ $(W_a)_{a \in G}$ | for all $a \in G, W_a \subseteq W, w \in W_a$
    - **if** $\bigcap_{a \in G} W_a \neq W'$ **then reject else** $(\forall)$ $a \in G$; $stable_{\mathsf{yes}}(W, W_a, R_a)$
  - **case** $\gamma = \gamma_1; \gamma_2$:
    - $(\exists)$ $W'' \subseteq W$ such that $W' \subseteq W''$
    - $(\forall)$ $exec_{\mathsf{yes}}(W, w, W'', \gamma_1)$ **and** $exec_{\mathsf{yes}}(W'', w, W', \gamma_2)$
  - **case** $\gamma = \gamma_1 \cup \gamma_2$: $(\exists)$ $k \in \{1, 2\}$; $exec_{\mathsf{yes}}(W, w, W', \gamma_k)$
  - **case** $\gamma = \varphi?$: $(\forall)$ **if** $W' \neq W$ **then reject and** $mc_{\mathsf{yes}}(W, w, \varphi)$
  - **case** $\gamma = \gamma'^*$: $(\exists)$ $i \in \{0, ..., 2^{\log_2(|W|)+1}\}$; $exec^*_{\mathsf{yes}}(W, w, W', \gamma', i)$

Figure 5.15: Dual path searching algorithm $exec_{\mathsf{yes}}$ for announcements.

proc $exec^*_{\mathsf{yes}}(W, w, W', \gamma, i)$ $|AP| + |\gamma| + \log_2(i)$
- $\triangleright$ accepts whenever $(W, w) \overset{\gamma^i}{\rightsquigarrow} (W', w)$
- **case** $i = 0$: **if** $W = W'$ **then accept else** $exec_{\mathsf{yes}}(W, w, W', \gamma)$;
- **case** $i \geq 2$:
  - $(\exists)$ W'' s.th. $W' \subseteq W'' \subseteq W$
  - $(\forall)$ $exec^*_{\mathsf{yes}}(W, w, W'', \gamma, i/2)$ **and** $exec^*_{\mathsf{yes}}(W'', w, W', \gamma, i/2)$

Figure 5.16: Iteration algorithm $exec^*_{\mathsf{yes}}$.

proc $stable_{\mathsf{yes}}(W, W', R)$ $|\pi| + 1$
$\triangleright$ accepts whenever $\widetilde{pre}_{R,W}\ (post_{R,W}(W')) \subseteq W'$
- $(\forall)$ $(w, v, u) \in W' \times W \times W$
- **if** $((w, v) \in R$ and $(u, v) \in R$ implies $u \in W')$ **then accept else reject**

Figure 5.17: Algorithm that checks that $\widetilde{pre}_{\pi,W}\ (post_{\pi,W}(W')) \subseteq W'$.

5.7 Conclusion

In this chapter, we have introduced PAPL, a logic extending APAL and GAL. It contains an arbitrary announcement operator that checks for the existence of an announcement. The logic PAPL features announcement protocols, that combine public announcements and arbitrary announcements, and allow to express in particular epistemic planning in this

setting. We have shown that the symbolic model checking against PAPL is $\mathrm{A}_{\mathrm{pol}}$EXPTIME-complete and that the non-symbolic model checking against PAPL is PSPACE-complete. We thus observe a jump of complexity when considering symbolic models, which is a usual phenomenon.

Chapter 6

Implementation of Succinct Epistemic Logic with Arbitrary Announcements

---

In this chapter, we focus on the implementation of symbolic model checking of PAPL in practice. For that, we reduce the model checking of PAPL into the model checking of monadic monadic second-order logic (MMSO), that is, monadic second order logic where all predicates are monadic, not only the quantified ones.

Since the model checking of PAPL is $\mathrm{A}_{\mathrm{pol}}$Exptime-complete, it seems rather unreasonable to have an implementation of the full logic. That is why we consider the existential fragment of PAPL, ∃PAPL , and reduce the model checking of ∃PAPL into the satisfiability of monadic first-order logic (MFO), that is first-order logic where all predicates are monadic.

The chapter is divided as follows:

- First, we briefly recall the definitions of first and second-order logics.
- Second, we reduce the model checking of PAPL into the satisfiability of MMSO.
- Third, we reduce the model checking of ∃PAPL into the satisfiability of MFO.
- Finally we conclude.

The content of the chapter was initially published in [CPS17], but for APAL and GAL instead of PAPL. Therefore, we extend the results to PAPL here.

6.1 First and second-order logics

Monadic monadic second-order logic MMSO and its fragment monadic first-order logic MFO are central in the proposed approach. These monadic fragments of MSO and FO

disallow the use of non-unary predicates and of function symbols. We first define the syntax of these two logics.

Definition 68. *We suppose that we have first-order variables* $\mathsf{x}, \mathsf{y}, \mathsf{z}$ *and second-order variables* $\mathsf{X}, \mathsf{Y}, \ldots$ *The syntax of* MMSO *is defined as follows:*

$$\varphi ::= \mathsf{X}(\mathsf{x}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \forall \mathsf{x}\varphi \mid \forall \mathsf{X}\varphi$$

MFO *is the fragment of* MMSO *where the operator* $\forall \mathsf{X}\varphi$ *is removed.*

MMSO-formulas are thus monadic second-order formulas with first-order and second-order variables but with no occurrence of non-unary predicates. The signature of MMSO mimics the set of atomic propositions AP: to each atomic proposition $p \in AP$, we introduce a corresponding unary predicate symbol $P(.)$. We take the convention that atomic propositions are written in lowercase while the corresponding predicates are written in uppercase.

The semantics of MMSO is defined on models defined as follows.

Definition 69. *A model* $\mathscr{M}$ *of* MMSO *is a structure* $(D, (P^{\mathscr{M}})_{p \in AP})$ *where* D *is a non-empty domain and each predicate* $P^{\mathscr{M}}$ *is such that* $P^{\mathscr{M}} \subseteq D$.

We will use the classical notation of the form $\mathscr{M}[...]$ for the model $\mathscr{M}$ extended with (first-order and second-order) variable assignments: for instance, $\mathscr{M}[\mathsf{x} \leftarrow e, \mathsf{y} \leftarrow e', \mathsf{X} \leftarrow D', \mathsf{Y} \leftarrow D'']$ is the model $\mathscr{M}$ in which first-order variables x and y are interpreted by element $e \in D$ and $e' \in D$ respectively, and second-order variables X and Y are interpreted by element $D' \subseteq D$ and $D'' \subseteq D$ respectively.

The semantics of MMSO is defined as follows.

Definition 70. *Let* $\mathscr{M}$ *be a model with variable assignments. The semantics of* MMSO *is defined by induction on* φ *as follows.*

- $\mathscr{M} \models \mathsf{X}(\mathsf{x})$ *iff* $\mathscr{M}(\mathsf{x}) \in \mathscr{M}(\mathsf{X})$;
- $\mathscr{M} \models \varphi_1 \wedge \varphi_2$ *iff* $\mathscr{M} \models \varphi_1$ *and* $\mathscr{M} \models \varphi_2$;
- $\mathscr{M} \models \neg\varphi$ *iff* $\mathscr{M} \not\models \varphi$;
- $\mathscr{M} \models \forall \mathsf{x}\varphi$ *iff for all* $c \in D$, $\mathscr{M}[\mathsf{x} \leftarrow c] \models \varphi$;

- $\mathscr{M} \models \forall \mathsf{X} \varphi$ *iff for all* $D' \subseteq D$, $\mathscr{M}[\mathsf{X} \leftarrow D'] \models \varphi$.

The model checking problem for MFO and MMSO is defined as follows.

Definition 71 (Model checking problem)**.**

- ***Input:*** *a model* $\mathscr{M}$*, a formula* φ *of* MMSO/MFO *such that all free variables (i.e. variables not bound by a quantifier) are assigned in* $\mathscr{M}$*.*
- ***Output:*** *yes if* $\mathscr{M} \models \varphi$*, no otherwise.*

Definition 72 (Satisfiability problem)**.**

- ***Input:*** *a formula* φ*.*
- ***Output:*** *yes if there exists* $\mathscr{M}$ *with variable assignments such that* $\mathscr{M} \models \varphi$*, no otherwise.*

Notice that for MMSO, the model checking problem and satisfiability problem are equivalent, since φ is satisfiable if and only if $\mathscr{M} \models \exists \mathsf{x}_1 \ldots \exists \mathsf{x}_n \exists \mathsf{X}_1 \ldots \exists \mathsf{X}_m \varphi$ where $\mathsf{x}_1, \ldots, \mathsf{x}_n, \mathsf{X}_1, \ldots, \mathsf{X}_m$ are the free variables of φ and $\mathscr{M}$ is arbitrary.

Regarding the properties of MMSO and MFO, it is known that the satisfiability problem of a MFO-formula is NExptime-complete [BGW93; Lew80]. Also, there are plenty of FO provers: Isabelle, iprover, Z3 [MB08], CVC4 [Bar+11]. In particular, the prover iprover won CASC 2016 in EPR division [Sut16].

6.2 PAPL into monadic monadic second-order logic

We reduce the model checking against PAPL to the satisfiability problem of MMSO. Intuitively, second-order variables denote current sets of valuations, called *contexts*, and first-order variables denote possible worlds/valuations. We present the reduction in four steps:

1. we define an MMSO-theory that enforce the MMSO-model to contain all valuations (Theorem 24);
2. we translate arbitrary accessibility programs into first-order logic (Theorem 25);
3. we translate PAPL formulas into MMSO (Theorem 26);
4. we give the reduction of the PAPL-model checking into the MMSO-satisfiability problem (Theorem 27).

6.2.1 The theory of models of valuations

In this section, we fix a set of atomic propositions AP. Since we evaluate PAPL-formulas on a symbolic model $\mathfrak{M}$ meant to denote the Kripke model with all valuations, we therefore need to enforce that all such valuations are captured.

Definition 73. *The model of valuations $\mathscr{M}_{AP}$ on AP is the structure $\mathscr{M}_{AP} = (D, (P^{\mathscr{M}_{AP}})_{p\in AP})$ with D is the domain of all valuations on AP and the interpretation of P is defined by as $P^{\mathscr{M}_{AP}}(\mathtt{w})$ iff $p \in \mathtt{w}$.*

In what follows, we write P_{AP} for the set of atomic predicates associated to some $p \in AP$.

Definition 74. *Let β be a Boolean formula over AP. We define the first-order formula $tr(\beta)(\mathsf{x})$ to be formula β in which each occurrence of $p \in AP$ is replaced by $P(\mathsf{x})$. Similarly, for a valuation $\mathtt{w}$, we define $tr(\mathtt{w})(\mathsf{x})$ for the formula describing $\mathtt{w}$ where all p are replaced by $P(\mathsf{x})$.*

Example 31. *Let $\beta = (p \vee q) \wedge (\neg p \vee q)$. Then $tr(\beta)(\mathsf{x}) = (P(\mathsf{x}) \vee Q(\mathsf{x})) \wedge (\neg P(\mathsf{x}) \vee Q(\mathsf{x}))$.*

Example 32. *Let $\mathtt{w} = \{p, q\}$ a valuation over $AP = \{p, q, r\}$. $tr(\mathtt{w})(\mathsf{x}) = P(\mathsf{x}) \wedge Q(\mathsf{x}) \wedge \neg R(\mathsf{x})$.*

We define a theory $\mathscr{T}_{AP}$ such that $\mathscr{M}_{AP}$ satisfies $\mathscr{T}_{AP}$ and every model satisfying $\mathscr{T}_{AP}$ is isomorphic to $\mathscr{M}_{AP}$.

Currently, in an arbitrary structure $(D, (P_i^{\mathscr{M}})_{p_i\in AP})$, two distinct elements e, e' in D may be such that $e \in P_i^{\mathscr{M}}$ iff $e' \in P_i^{\mathscr{M}}$ for all $p_i \in AP$. To prevent it, we define $\varphi_{unique} = \forall \mathsf{x} \forall \mathsf{y}(\mathsf{x} = \mathsf{y}) \leftrightarrow \bigwedge_{p\in AP}(P(\mathsf{x}) \leftrightarrow P(\mathsf{y}))$. It says that two elements satisfy the same predicates (i.e. are the same valuation) iff they are equal. We define too φ_{exists} says that for each valuation, for each atomic proposition p, there exists another valuation that differs only on p. In other words, $\varphi_{exists} = \forall \mathsf{x} \bigwedge_{p\in AP} \Big(\exists \mathsf{y} \Big((P(\mathsf{x}) \leftrightarrow \neg P(\mathsf{y})) \wedge \bigwedge_{q\in AP, q\neq p}(Q(\mathsf{x}) \leftrightarrow Q(\mathsf{y}))\Big)\Big)$, imposing all valuations to appear in the model.

By letting $\mathscr{T}_{AP} = \{\varphi_{unique}, \varphi_{exists}\}$, we get the following.

Theorem 24. *For all* MMSO*-models $\mathscr{M}$, we have $\mathscr{M} \models \mathscr{T}_{AP}$ iff $\mathscr{M}$ is isomorphic to $\mathscr{M}_{AP}$.*

Proof.$\Leftarrow$: It is sufficient to prove that $\mathscr{M}_{AP} \models \mathscr{T}_{AP}$:

– $\mathscr{M}_{AP} \models \varphi_{unique}$ because each valuation is represented exactly one time in D and by Definition 73, P mimics the role of the atomic propositions in the valuations.

– $\mathscr{M}_{AP} \models \varphi_{exists}$ because all valuations are represented in D.

Therefore $\mathscr{M}_{AP} \models \mathscr{T}_{AP}$ and thus $\mathscr{M} \models \mathscr{T}_{AP}$.

$\Rightarrow$: Let $\mathscr{M}$ be such that $\mathscr{M} \models \mathscr{T}_{AP}$. Let D' be the domain of $\mathscr{M}$ and P' be the monadic predicates of $\mathscr{M}$. We define the mapping $f : D' \to D$ such that for all $e \in D$, $f(e)$ is the valuation $\{p \mid e \in P'\} \in D$. We conclude by showing that f is an isomorphism.

– f is injective: if $f(e) = f(e')$, it means that for all P, $e \in P^{\mathscr{M}}$ iff $e' \in P^{\mathscr{M}}$. With $\mathscr{M} \models \varphi_{unique}$, we conclude that $e = e'$.

– f is surjective: let $\mathtt{w}$ be an element of D. As D' is non-empty, let e be in D'. As $\mathscr{M} \models \varphi_{exists}$, we can, from e, guarantee the existence of an element e' of D' such that $f(e') = \mathtt{w}$.

□

From Theorem 24, we obtain the following.

Corollary 1. *Let φ be an* MMSO*-formula. Then $\mathscr{M}_{AP} \models \varphi$ if, and only if, $\mathscr{T}_{AP} \wedge \varphi$ is* MMSO*-satisfiable.*

6.2.2 From programs to FO-formulas

Definition 75. *Let π be a program and x, y be two first-order variables. We define the first-order formula $\pi(\mathsf{x}, \mathsf{y})$ by induction π as follows:*

$$
\begin{aligned}
(p \leftarrow \beta)(\mathsf{x}, \mathsf{y}) &= (P(\mathsf{y}) \leftrightarrow tr(\beta)(\mathsf{x})) \wedge \textstyle\bigwedge_{q \in AP, q \neq p} (Q(\mathsf{x}) \leftrightarrow Q(\mathsf{y}));\\
\beta?(\mathsf{x}, \mathsf{y}) &= tr(\beta)(\mathsf{x}) \wedge (\mathsf{x} = \mathsf{y});\\
(\pi_1; \pi_2)(\mathsf{x}, \mathsf{y}) &= \exists \mathsf{z}\ \pi_1(\mathsf{x}, \mathsf{z}) \wedge \pi_2(\mathsf{z}, \mathsf{y}).\\
(\pi_1 \cup \pi_2)(\mathsf{x}, \mathsf{y}) &= \pi_1(\mathsf{x}, \mathsf{y}) \vee \pi_2(\mathsf{x}, \mathsf{y});
\end{aligned}
$$

The formula $\pi(\mathsf{x}, \mathsf{y})$ expresses that y is a π-successor of x. It should be noticed that formulas $\pi(\mathsf{x}, \mathsf{y})$ are in MFO, even though they have two parameters x and y. We consider those formulas as macros to shorten the definitions afterwards, so they are monadic. Formally:

Theorem 25. *For all worlds $\mathtt{w}, \mathtt{u}$ and π, $\mathtt{w} \xrightarrow{\pi} \mathtt{u}$ if, and only if, $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models \pi(\mathsf{x}, \mathsf{y})$.*

Proof. By induction on π.

- $\pi = p \leftarrow \beta$:

  $\mathtt{w} \xrightarrow{p \leftarrow \beta} \mathtt{u}$ iff $(p \in \mathtt{u}$ iff $\mathtt{w} \models \beta)$ and for all $q \neq p, (q \in \mathtt{w}$ iff $q \in \mathtt{u})$.
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models P(\mathsf{y}) \leftrightarrow tr(\beta)(\mathsf{x})$
  and for all $q \neq p, \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models Q(\mathsf{x}) \leftrightarrow Q(\mathsf{y})$.
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models (p \leftarrow \beta)(\mathsf{x}, \mathsf{y})$.

- $\pi = \beta?$:

  $\mathtt{w} \xrightarrow{\beta?} \mathtt{u}$ iff $\mathtt{w} = \mathtt{u}$ and $\mathtt{w} \models \beta$
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models (\mathsf{x} = \mathsf{y})$ and $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models tr(\beta)(\mathsf{x})$.
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models \beta?(\mathsf{x}, \mathsf{y})$.

- $\pi = \pi_1; \pi_2$:

  $\mathtt{w} \xrightarrow{\pi_1;\pi_2} \mathtt{u}$ iff there exists $\mathtt{v}$ such that $\mathtt{w} \xrightarrow{\pi_1} \mathtt{v}$ and $\mathtt{v} \xrightarrow{\pi_2} \mathtt{u}$
  iff there exists $\mathtt{v}$ such that $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{z} \leftarrow \mathtt{v}] \models \pi_1(\mathsf{x}, \mathsf{z}) \wedge \pi_2(\mathsf{z}, \mathsf{y})$.
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models (\pi_1; \pi_2)(\mathsf{x}, \mathsf{y})$.

- $\pi = \pi_1 \cup \pi_2$:

  $\mathtt{w} \xrightarrow{\pi_1 \cup \pi_2} \mathtt{u}$ iff $\mathtt{w} \xrightarrow{\pi_1} \mathtt{u}$ or $\mathtt{w} \xrightarrow{\pi_2} \mathtt{u}$
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models \pi_1(\mathsf{x}, \mathsf{y})$ or $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models \pi_2(\mathsf{x}, \mathsf{y})$
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}] \models (\pi_1 \cup \pi_2)(\mathsf{x}, \mathsf{y})$.

□

6.2.3 From PAPL-formulas to MMSO-formulas

In the following definition, we define $tr_{\mathsf{X}}(\varphi)(\mathsf{x})$ to be the translation of the PAPL-formula φ, where x is a first-order variable representing the valuation in which the formula φ is evaluated and X is a second-order variable representing the context (namely, the set of valuations that survived the previous announcements). Both variables x and X are the sole free variables of $tr_{\mathsf{X}}(\varphi)(\mathsf{x})$. We also define $tr(\gamma)(\mathsf{X}, \mathsf{Y})$ for $\mathsf{X} \xrightarrow{\gamma} \mathsf{Y}$, whose sole free variables are X and Y.

Definition 76. *Let $\mathfrak{M} = \langle AP_{\mathcal{M}}, (\pi_a)_{a \in Ag} \rangle$ be a symbolic model, φ be a PAPL-formula, γ an announcement protocol, X and Y be two second-order variables, and x be a first-order*

variable. We define the MMSO*-formulas* $tr_{\mathsf{X}}(\varphi)(\mathsf{x})$ *and* $tr(\gamma)(\mathsf{X},\mathsf{Y})$ *by mutual induction over* φ *and* γ*, with the notation* $\mathsf{Y} \subseteq \mathsf{X}$ *for* $\forall \mathsf{x}(\mathsf{Y}(\mathsf{x}) \to \mathsf{X}(\mathsf{x}))$.

$$
\begin{array}{ll}
tr_{\mathsf{X}}(p)(\mathsf{x}) & = P(\mathsf{x}); \\
tr_{\mathsf{X}}(\neg\varphi)(\mathsf{x}) & = \neg tr_{\mathsf{X}}(\varphi)(\mathsf{x}); \\
tr_{\mathsf{X}}(\varphi_1 \vee \varphi_2)(\mathsf{x}) & = tr_{\mathsf{X}}(\varphi_1)(\mathsf{x}) \vee tr_{\mathsf{X}}(\varphi_2)(\mathsf{x}); \\
tr_{\mathsf{X}}(K_a\varphi)(\mathsf{x}) & = \forall \mathsf{y}\ [(\mathsf{X}(\mathsf{y}) \wedge \pi_a(\mathsf{x},\mathsf{y})) \to tr_{\mathsf{X}}(\varphi)(\mathsf{y})]; \\
tr_{\mathsf{X}}(C_G\varphi)(\mathsf{x}) & = \exists \mathsf{X}_G\ (group_{\mathsf{X}}(\mathsf{X}_G,\mathsf{x}) \wedge \forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y},\mathsf{x}) \to \mathsf{X}_G \subseteq \mathsf{Y}) \\
 & \qquad \wedge \forall \mathsf{y}(\mathsf{X}_G(\mathsf{y}) \to tr_{\mathsf{X}}(\varphi)(\mathsf{y}))) \\
\multicolumn{2}{l}{\text{with } group_{\mathsf{X}}(\mathsf{Y},\mathsf{x}) = (\mathsf{Y}(\mathsf{x}) \wedge \forall \mathsf{y}(\mathsf{Y}(\mathsf{y}) \to (\forall \mathsf{z}\mathsf{X}(\mathsf{z}) \wedge \bigvee_{a\in G} \pi_a(\mathsf{y},\mathsf{z}) \to \mathsf{Y}(\mathsf{z})))} \\
tr_{\mathsf{X}}(\langle\gamma\rangle\varphi)(\mathsf{x}) & = \exists \mathsf{Y} tr(\gamma)(\mathsf{X},\mathsf{Y}) \wedge \mathsf{Y}(\mathsf{x}) \wedge tr_{\mathsf{Y}}(\varphi)(\mathsf{x}) \\
 & \\
tr(\varphi!)(\mathsf{X},\mathsf{Y}) & = \forall \mathsf{y}\ \mathsf{Y}(\mathsf{y}) \leftrightarrow (\mathsf{X}(\mathsf{y}) \wedge tr_{\mathsf{X}}(\varphi)(\mathsf{y}))); \\
tr(\bullet!)(\mathsf{X},\mathsf{Y}) & = \mathsf{Y} \subseteq \mathsf{X}; \\
tr(\bullet!_G)(\mathsf{X},\mathsf{Y}) & = \mathsf{Y} \subseteq \mathsf{X} \wedge \bigwedge_{a\in G} \forall \mathsf{x}\ (\forall \mathsf{y}\ \pi_a(\mathsf{x},\mathsf{y}) \to (\exists \mathsf{z}\ \pi_a(\mathsf{z},\mathsf{y}) \wedge \mathsf{Y}(\mathsf{z}))) \to \mathsf{Y}(\mathsf{x}). \\
tr(\gamma_1;\gamma_2)(\mathsf{X},\mathsf{Y}) & = \exists \mathsf{Z}\mathsf{Z} \subseteq \mathsf{X} \wedge \mathsf{Y} \subseteq \mathsf{Z} \wedge tr(\gamma_1)(\mathsf{X},\mathsf{Z}) \wedge tr(\gamma_2)(\mathsf{Z},\mathsf{Y}) \\
tr(\gamma_1 \cup \gamma_2)(\mathsf{X},\mathsf{Y}) & = tr(\gamma_1)(\mathsf{X},\mathsf{Y}) \vee tr(\gamma_2)(\mathsf{X},\mathsf{Y}) \\
tr(\gamma^*)(\mathsf{X},\mathsf{Y}) & = \bigvee_{i=0}^{2^{AP_{\mathcal{M}}}} tr(\gamma^i)(\mathsf{X},\mathsf{Y})
\end{array}
$$

We now explain the formulas and protocols that are not direct.

Formulas:

- Formula $tr_{\mathsf{X}}(K_a\varphi)(\mathsf{x})$ mimics the standard translation of modal logic into first-order logic ([BRV01], p. 84), except that we use the MFO-formula $\pi_a(\mathsf{x},\mathsf{y})$ instead of $R_a(\mathsf{x},\mathsf{y})$.
- Formula $tr_{\mathsf{x}}(C_G\varphi)(\mathsf{X})$ first computes in a variable X_G the set of G-successors of x by the macro $group_{\mathsf{X}}(\mathsf{X},\mathsf{x})$. This macro states that all iterated successors of x are present in X_G. Second, the quantification on Y avoid having other worlds in X_G by saying that X_G is the smallest set Y such that $group_{\mathsf{X}}(\mathsf{Y},\mathsf{x})$ is true. Finally, the last part of the formula states that $tr_{\mathsf{y}}(\varphi)(\mathsf{X})$ must be true for all $\mathsf{y} \in \mathsf{X}_G$.
- Formula $tr_{\mathsf{x}}(\langle\gamma\rangle\varphi)(\mathsf{X})$ first existentially finds a context Y such that $tr(\gamma)(\mathsf{X},\mathsf{Y})$ is true, checks that x is in Y (by $\mathsf{Y}(\mathsf{x})$ and goes on with $tr_{\mathsf{Y}}(\varphi)(\mathsf{x})$.

Protocols:

- Formula $tr(\varphi!)(\mathsf{X},\mathsf{Y})$ asks that Y corresponds to the set of valuations in which φ holds $(\forall \mathsf{y}\ \mathsf{Y}(\mathsf{y}) \leftrightarrow (\mathsf{X}(\mathsf{y}) \wedge tr_{\mathsf{X}}(\psi)(\mathsf{y})))$

- Formula $tr(\bullet!)(\mathsf{X},\mathsf{Y})$ imposes that the announcement is a group announcement, which is the same characterization than Proposition 16 page 144.

We now state and prove the correctness of the translation.

Theorem 26. *Let $\mathfrak{M}$ be a symbolic model on AP, φ be an PAPL-formula on AP, γ a protocol, and $\mathtt{w} \in \mathfrak{M}$. Let $D_{\mathfrak{M}}$ be the set of valuations of $\mathfrak{M}$. Let $D, D' \subseteq D_{\mathfrak{M}}$. Then $\mathfrak{M}, \mathtt{w} \models \varphi$ iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\varphi)(\mathsf{x})$, and $D \xrightarrow{\gamma} D'$ iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D']tr(\gamma)(\mathsf{X},\mathsf{Y})$.*

Proof. By mutual induction on φ and γ.

- $\varphi = p$:

$$\begin{aligned}\mathfrak{M}, \mathtt{w} \models \varphi \quad & \text{iff } p \in \mathtt{w} \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models P(\mathsf{x})\end{aligned}$$

- $\varphi = \neg\psi$:

$$\begin{aligned}\mathfrak{M}, \mathtt{w} \models \neg\psi \quad & \text{iff } \mathfrak{M}, \mathtt{w} \not\models \psi \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \not\models tr_{\mathsf{X}}(\varphi)(\mathsf{x}) \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \neg tr_{\mathsf{X}}(\varphi)(\mathsf{x})\end{aligned}$$

- $\varphi = \varphi_1 \vee \varphi_2$:

$$\begin{aligned}\mathfrak{M}, \mathtt{w} \models \varphi_1 \vee \varphi_2 \quad & \text{iff } \mathfrak{M}, \mathtt{w} \models \varphi_1 \\ & \qquad \text{or } \mathfrak{M}, \mathtt{w} \models \varphi_2 \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\varphi_1)(\mathsf{x}) \text{ or } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\varphi_2)(\mathsf{x}) \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\varphi_1)(\mathsf{x}) \vee tr_{\mathsf{X}}(\varphi_2)(\mathsf{x})\end{aligned}$$

- $\varphi = K_a\varphi$:

$$\begin{aligned}\mathfrak{M}, \mathtt{w} \models (K_a\psi) \quad & \text{iff for all } \mathtt{u} \in D_{\mathfrak{M}} \text{ such that } \mathtt{w} \xrightarrow{\pi_a} \mathtt{u},\ \mathfrak{M}, \mathtt{u} \models \psi \\ & \text{iff for all } \mathtt{u} \in D_{\mathfrak{M}} \text{ such that } \mathtt{w} \xrightarrow{\pi_a} \mathtt{u},\ \mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\psi)(\mathsf{y}) \\ & \text{iff for all } \mathtt{u} \in D_{\mathfrak{M}} \text{ such that } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \pi_a(\mathsf{x},\mathsf{y}), \\ & \qquad \mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\psi)(\mathsf{y}) \\ & \text{iff } \mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \forall \mathsf{y}\ (\mathsf{X}(\mathsf{y}) \wedge \pi_a(\mathsf{x},\mathsf{y}) \rightarrow tr_{\mathsf{X}}(\varphi)(\mathsf{y}))\end{aligned}$$

- $\varphi = C_G\psi$: for this case we first need to prove that $\mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{u}$ if and only if $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \exists \mathsf{X}_G\ (group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x}) \wedge \forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x}) \rightarrow \mathsf{X}_G \subseteq \mathsf{Y}) \wedge \mathsf{X}_G(\mathsf{y}))$:

– If $\mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{u}$ then let $D' = \{\mathtt{v} \in D_{\mathfrak{M}} \mid \mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{v}\}$. Obviously we have $\mathtt{w}, \mathtt{u} \in D'$. We now prove that $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}, \mathsf{X}_G \leftarrow D'] \models (group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x}) \wedge \forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x}) \rightarrow \mathsf{X}_G \subseteq \mathsf{Y}) \wedge \mathsf{X}_G(\mathsf{y}))$: by definition of *group*, we obtain directly that $group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x})$ is true and if we take another Y such that $group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x})$ is true, its interpretation will necessarily contain D'.

– If $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \exists \mathsf{X}_G\ (group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x}) \wedge \forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x}) \rightarrow \mathsf{X}_G \subseteq \mathsf{Y}) \wedge \mathsf{X}_G(\mathsf{y}))$ then let D' be the interpretation of X_G. By the deifnition of *group*, D' contains $\{\mathtt{v} \in D_{\mathfrak{M}} \mid \mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{v}\}$, and it is the smallest one, so the interpretation of X_G is necessarily $\{\mathtt{v} \in D_{\mathfrak{M}} \mid \mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{v}\}$, which concludes the proof.

$\mathfrak{M}, \mathtt{w} \models (C_G\psi)$ iff for all $\mathtt{u} \in D_{\mathfrak{M}}$ such that $\mathtt{w} \xrightarrow{(\bigcup_{a\in G}\pi_a)^*} \mathtt{u}$, $\mathfrak{M}, \mathtt{u} \models \psi$
iff for all $\mathtt{u} \in D_{\mathfrak{M}}$ such that
$\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \exists \mathsf{X}_G\ (group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x}) \wedge \forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x}) \rightarrow \mathsf{X}_G \subseteq \mathsf{Y}) \wedge \mathsf{X}_G(\mathsf{y}))$,
$\mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models tr_{\mathsf{X}}(\psi)(\mathsf{y})$
iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D_{\mathfrak{M}}] \models \exists \mathsf{X}_G\ (group_{\mathsf{X}}(\mathsf{X}_G, \mathsf{x}) \wedge$
$\forall \mathsf{Y}\ (group_{\mathsf{X}}(\mathsf{Y}, \mathsf{x}) \rightarrow \mathsf{X}_G \subseteq \mathsf{Y}) \wedge \forall \mathsf{y}(\mathsf{X}_G(\mathsf{y}) \rightarrow tr_{\mathsf{X}}(\varphi)(\mathsf{y})))$

- $\varphi = \langle\gamma\rangle\psi$:

  $\mathfrak{M}, \mathtt{w} \models (\langle\gamma\rangle\psi)$ iff there exists D' such that $D' \subseteq D_{\mathfrak{M}}$, $\mathtt{w} \in D'$ and
  $\mathfrak{M}', \mathtt{w} \models \psi$ (where $\mathfrak{M}'$ is $\mathfrak{M}$ restricted to D').
  iff there exists D' such that
  $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D_{\mathfrak{M}}, \mathsf{Y} \leftarrow D', \mathsf{x} \leftarrow \mathtt{w}] \models tr(\gamma)(\mathsf{X}, \mathsf{Y}) \wedge \mathsf{Y}(\mathsf{x}) \wedge tr_{\mathsf{x}}(\psi)(\mathsf{Y})$
  iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D_{\mathfrak{M}}, \mathsf{x} \leftarrow \mathtt{w}] \models \exists \mathsf{Y}\ tr(\gamma)(\mathsf{X}, \mathsf{Y}) \wedge \mathsf{Y}(\mathsf{x}) \wedge tr_{\mathsf{x}}(\psi)(\mathsf{Y})$

- $\gamma = \chi!$:

  $D \xrightarrow{\chi!} D'$ iff for all $\mathtt{u}, \mathtt{u} \in D'$ iff $\mathtt{u} \in D$ and $\mathfrak{M}, \mathtt{u} \models \chi$.
  iff for all $\mathtt{u}$, $\mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{Y} \leftarrow D'] \models \mathsf{Y}(\mathsf{y})$ iff
  $\mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D] \models \mathsf{X}(\mathsf{y})$ and $\mathscr{M}_{AP}[\mathsf{y} \leftarrow \mathtt{u}, \mathsf{X} \leftarrow D] \models tr_{\mathsf{X}}(\chi)(\mathsf{y})$
  iff $\mathscr{M}_{AP}[\mathsf{x} \leftarrow \mathtt{w}, \mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models \forall \mathsf{y}\ \mathsf{Y}(\mathsf{y}) \leftrightarrow (\mathsf{X}(\mathsf{y}) \wedge tr_{\mathsf{X}}(\psi)(\mathsf{y}))$

- $\gamma = \bullet!$ and $\gamma = \bullet!_G$: both cases use characterizations established in the previous chapter.

- $\gamma = \gamma_1; \gamma_2$:

$D \xrightarrow{\gamma_1;\gamma_2} D'$ iff there exists D'' such that $D' \subseteq D'' \subseteq D$ and $D \xrightarrow{\gamma_1} D''$ and $D'' \xrightarrow{\gamma_2} D'$.
iff there exists D'' such that $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Z} \leftarrow D''] \models \mathsf{Z} \subseteq \mathsf{X} \wedge tr(\gamma_1)(\mathsf{X}, \mathsf{Z})$
and $\mathscr{M}_{AP}[\mathsf{Z} \leftarrow D'', \mathsf{Y} \leftarrow D'] \models \mathsf{Y} \subseteq \mathsf{Z} \wedge tr(\gamma_2)(\mathsf{Z}, \mathsf{Y})$.
iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models \exists \mathsf{Z}\ \mathsf{Z} \subseteq \mathsf{X} \wedge \mathsf{Y} \subseteq \mathsf{Z} \wedge tr(\gamma_1)(\mathsf{X}, \mathsf{Z}) \wedge tr(\gamma_2)(\mathsf{Z}, \mathsf{Y})$.

- $\gamma = \gamma_1 \cup \gamma_2$:

  $D \xrightarrow{\gamma_1 \cup \gamma_2} D'$ iff $D \xrightarrow{\gamma_1} D'$ or $D \xrightarrow{\gamma_2} D'$
  iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models tr(\gamma_1)(\mathsf{X}, \mathsf{Y})$ or $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models tr(\gamma_2)(\mathsf{X}, \mathsf{Y})$.
  iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models tr(\gamma_1)(\mathsf{X}, \mathsf{Y}) \vee tr(\gamma_2)(\mathsf{X}, \mathsf{Y})$

- $\gamma = \gamma'^*$:

  $D \xrightarrow{\gamma'^*} D'$ iff there exists $i \in \{0, 2^{AP_{\mathcal{M}}}\}$ such that $D \xrightarrow{\gamma'^i} D'$
  iff there exists $i \in \{0, 2^{AP_{\mathcal{M}}}\}$ such that $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models tr(\gamma'^i)(\mathsf{X}, \mathsf{Y})$
  iff $\mathscr{M}_{AP}[\mathsf{X} \leftarrow D, \mathsf{Y} \leftarrow D'] \models \bigvee_{i=0}^{2^{AP_{\mathcal{M}}}} tr(\gamma'^i)(\mathsf{X}, \mathsf{Y})$

□

6.2.4 Reduction from PAPL-mc to MMSO-sat

Definition 77 (reduction)**.** *Given a pointed symbolic Kripke model* $(\mathfrak{M}, w)$ *and an PAPL-formula* φ*, we let* $\leq^P (\mathfrak{M}, w, \varphi)$ *be the* MMSO *formula* $\mathscr{T}_{AP} \wedge tr(\mathtt{w})(\mathsf{x}) \wedge \forall \mathsf{y} \mathsf{X}(\mathsf{y}) \wedge tr_{\mathsf{X}}(\varphi)(\mathsf{x})$.

Notice that unfortunately, the formula for γ^* causes an exponential blowup. We could fix this blowup by creating the set of y that we can construct from X with γ^*, but we suspect that it is not possible. Furthermore, the non-deterministic choice in protocol also causes an exponential blowup.

If we want to obtain a polynomial reduction, we thus need to consider PAPL without non-deterministic choice or Kleene star in protocols.

By Corollary 1 and Theorem 26 we get the following.

Theorem 27. $\mathfrak{M}, w, \models \varphi$ *iff* $\leq^P (\mathfrak{M}, w, \varphi)$ *is* MMSO*-satisfiable.*

Because the symbolic model checking of APAL is $\mathrm{A_{pol}}$EXPTIME-hard [CS15], and PAPL without non-deterministic choice nor star contain APAL, we obtain:

Corollary 2. MMSO*-satisfiability problem is* A_{pol}EXPTIME*-hard.*

However, as discussed in the next section, restricting to logic ∃PAPL yields a reduction to the satisfiability problem of monadic first-order logic MFO.

6.3 Existential announcement logic into monadic first-order logic

We first define ∃PAPL.

Definition 78. *The syntax of ∃PAPL is defined as follows.*

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \hat{K}_a\varphi \mid \hat{C}_G\varphi \mid \langle\gamma\rangle\varphi \mid \psi$$
$$\gamma ::= \varphi! \mid \bullet! \mid \bullet!_G \mid \gamma;\gamma \mid \gamma \cup \gamma \mid \gamma^*$$

with $\psi \in \mathcal{L}_{\mathbf{ELCK}}$.

Notice that here, for the reduction to work, we need to remove the second order quantification in the C_G case in the following way.

$$tr_{\mathsf{x}}(C_G\varphi)(\mathsf{x}) = \bigwedge_{i=0}^{2^{AP}\mathcal{M}} \forall \mathsf{y} \left(\bigcup_{a\in G} \pi_a\right)^i (\mathsf{x},\mathsf{y}) \to tr_{\mathsf{x}}(\varphi)(\mathsf{y})$$

It causes an exponential blowup but it is now a first-order formula.

If we restrict inputs $\mathfrak{M}, w, \varphi$ of the PAPL-model checking by letting $\varphi \in$ ∃PAPL, then $\leq^P (\mathfrak{M}, w, \varphi)$ is an MMSO-formula where all second-order quantifiers are existential and are not under the scope of universal quantifiers. Such second-order quantifiers can be removed from the formula $\leq^P (\mathfrak{M}, w, \varphi)$ resulting in a MFO-formula.

In the next section, we make use of this reduction to solve the symbolic model checking problem against ∃PAPL.

6.4 Implementation

We implemented the reduction from ∃PAPL to MFO in OCaml. We also built benchmarks. The code and a readme file can be found at the following link

`https://github.com/tcharrie/agpal-mmso`

6.4.1 Description of the implementation

The input is an ∃PAPL formula of the type `agpal_formula` in the source code. The type `acc_program` represents accessibility programs, the type `bool_formula` boolean formulas, and the type `fo_formula` MFO-formulas (the output of the code). The function

`agpal_formula_to_mfo` defines the translation from ∃PAPL formulas to MFO formulas (as in Definition 77).

In addition to the algorithm for the reduction, we implemented a function from existential formulas to the TPTP format [SSY94] used by the FO-SAT-solvers, called `agpal_formula_to_tptp`. It first calls the function `agpal_formula_to_mfo`, then calls the function `mfo_formula_to_tptp` that transforms a MFO-formula into its TPTP representation.

6.4.2 Benchmarks

We provide benchmarks for FO-provers built from the muddy children and the Russian card puzzles in order to tests the combinatorial ability of FO-provers.

Muddy children. We consider the following true properties:

- $\varphi^{muddy}_{standard} = \langle \bigvee_{a\in Ag} p_a! \rangle \langle \bigwedge_{a\in Ag} \neg(K_a p_a \wedge \neg K_a \neg p_a)! \rangle \dots \langle \bigwedge_{a\in Ag} \neg(K_a p_a \wedge \neg K_a \neg p_a)! \rangle \bigvee_{a\in Ag}(K_a p_a \vee K_a \neg p_a)$: standard formalization of the muddy children.
- $\varphi^{muddy}_{arbitrary} = \langle \bigvee_{a\in Ag} p_a! \rangle \langle \bullet! \rangle \bigwedge_{a\in Ag}(K_a p_a \vee K_a \neg p_a)$: variant with an arbitrary announcement.
- $\varphi^{muddy}_{group} = \langle \bigvee_{a\in Ag} p_a! \rangle \langle \bullet!_{Ag} \rangle \bigwedge_{a\in Ag}(K_a p_a \vee K_a \neg p_a)$: variant with a group announcement.

where $Ag = \{1, \dots, n\}$.

Russian cards. For this example, agents a and b holds the same number of cards n. For instance, the classical Russian cards problem corresponds to $n = 3$. Let $\varphi^{Russian}_{goal} = \bigwedge_{i=1}^{2n+1}(K_a p_{i,b} \vee K_a \neg p_{i,b}) \wedge (K_b p_{i,a} \vee K_b \neg p_{i,a}) \wedge \neg K_c p_{i,a} \wedge \neg K_c \neg p_{i,a} \wedge \neg K_c p_{i,b} \wedge \neg K_c \neg p_{i,b}$. We consider three types of properties:

- $\varphi^{Russian}_{arbitrary} = \langle \varphi_{Rh}! \rangle \langle \bullet! \rangle \varphi^{Russian}_{goal}$: formalization of the Russian cards with a unique arbitrary announcement.
- $\varphi^{Russian}_{group_1} = \langle \varphi_{Rh}! \rangle \langle \bullet!_{\{a\}} \rangle \varphi^{Russian}_{goal}$: formalization with only one announcement from a. This formula is not satisfiable.
- $\varphi^{Russian}_{group_2} = \langle \varphi_{Rh}! \rangle \langle \bullet!_{\{a\}} \rangle \langle \bullet!_{\{b\}} \rangle \varphi^{Russian}_{goal}$: normal formalization of the Russian cards problem.

$n =$	$\varphi^{muddy}_{arbitrary}$		$n =$	$\varphi^{muddy}_{standard}$	φ^{muddy}_{group}		$n =$	$\varphi^{Russian}_{arbitrary}$	$\varphi^{Russian}_{group_1}$	$\varphi^{Russian}_{group_2}$
3	0.03s		3	0.07s	0.04s		2	0.18s	0.32s	0.45s
10	0.20s		4	0.09s	0.08s		3	0.44s	0.85s	0.92s
25	1.32s		5	0.19s	0.22s		4	3.80s	3.51s	3.32s
40	3.23s		6	0.24s	0.25s		5	23.48s	26.80s	24.20s
55	9.405s		7	$>$ 10min	$>$ 10min		6	$>$ 10min	$>$ 10min	$>$ 10min

Figure 6.1: Results for the implementation of the reduction from ∃PAPL to MFO, using the FO-SAT-solver Iprover.

6.4.3 Experiments

To perform the tests, we used the FO-solver Iprover [Kor08] on a HP EliteBook 840 G2. The prover Iprover enabled us to test whether a FO-formula is satisfiable or not. The results are summarized in Figure 6.4.3.

We now briefly comment on the experiments.

Muddy children. For $\varphi^{muddy}_{arbitrary}$, the FO-SAT solver seems to perform well in all cases, as arbitrary announcements only require the new context to be included in the previous one. Hence, in this example, it is sufficient to restrict the model to the current world in order to satisfy the goal of $\varphi^{muddy}_{arbitrary}$. However, for the other tests, namely $\varphi^{muddy}_{standard}$ and φ^{muddy}_{group}, the FO-SAT-solver is able to test up to $n = 6$ agents. This can be explained by the fact public announcements and group announcements add significant combinatorial constraints to the specification.

Russian cards. For the three properties, the tests cannot exceed $n = 6$ cards, the main reason being that the rules of the game are very combinatorial, as for the muddy children.

Notice that the problems we have considered are puzzles, thus highly combinatorial. For the muddy children puzzle, the existential second-order quantification ranges over 2^{2^n} subsets. For $n = 7$, we have $2^{2^7} = 2^{128} \sim 10^{38}$, that is, about the number of positions $1.15868.. \times 10^{42}$ of a chess board.

Still, our implementation is promising and provides some interesting benchmarks for FO-provers.

6.5 Conclusion

In this chapter, we have expressed the model checking against PAPL as a model checking problem against MMSO, the monadic second-order logic where all predicates are monadic. Unfortunately, the reduction is exponential because of the star operator, but is polynomial if we remove this operator. We have also shown that the existential fragment of PAPL, ∃PAPL, can be expressed in the satisfiability problem of MFO, monadic first-order logic, and implemented the model checking problem against existential PAPL using a first-order solver.

Perspectives

In this manuscript, we have introduced epistemic logic, studied dynamic operators, and defined symbolic Kripke and event models. We now discuss missing features of **DEL** and possible perspectives.

In the introduction, we recalled a variety of applications of multi-agent systems, as robots, security, and games. We explained that high-order knowledge was an important notion to consider and claimed that **DEL** was suited for all these applications. In multi-robot systems, it makes perfect sense that the robots have to be able to reason about the other robots' reasoning to enable meaningful collaboration. Actually, **DEL** is especially adapted when robots interact with other robots that are perfectly rational. Unfortunately, the logic **DEL** reaches its limits when human agents are considered, because they are not perfectly rational [De +06]. However, in the case when the humans do not reason about the knowledge of robots but the robots do reason about the knowledge of humans, **DEL** may be suited in the following sense: a robot would have an algorithm to reason about an idealized rational version of the human, and the human would only have to deal with his own knowledge. A potential application would be in cases where the robot must adapt to the human, but not the other way around. If we need humans to collaborate with robots, a solution would be to define AIs that consider all agents to be rational at first, and then lower their expectations if some agent does not act rationally. Introducing *levels of rationality* for agents in **DEL** would be a workaround. Technically, it means that if a world in the model is too hard to consider for an agent, he may consider as possible a false world instead. We could express levels of rationality by bounding modal depth for certain agents or considering limited belief (as in [LLL04]).

In Chapters 2 and 3 we showed that the existence of a uniform strategy can be reduced to the model checking against **DEL**. This result opens up the use of **DEL** for AIs in games. Yet, for now, few applications currently exist for **DEL**. One reason is that it is too technical for a non-expert to design the event models. In the near future, we need to have a clear understanding of which event models are relevant in practice, and define a higher-level language to specify them. The third version of Game Description Language (GDL-III) is a promising specification language for **DEL**, given that they are both equivalent [Eng+18].

GDL-III is well suited for expressing games, but it is not clear yet it if is for expressing properties in multi-agent systems in general. Furthermore we strongly need to develop the implementations of **DEL** to make it widely convincing. The symbolic model checking for **DEL** (SMCDEL [Ben+15]) is a possibility, but we need to make it more accessible to non-experts. Hintikka's world [Sch18] has been developed to help people understand **DEL**, which is a good start to show researchers and engineers outside of our community how **DEL** works.

Also, in some applicative scenarios, for instance involving intelligent domestic robots, the system never stops or if it does, it is unknown when the systems stops. This is modeled classically in temporal logic by considering infinite executions of a finite transition system. Because of results about epistemic planning (see Theorem 16), we know that, in **DEL**, if event models feature knowledge modalities, verifying properties yield an undecidable problem. Nevertheless, we know that it is decidable to reason about infinite executions generated by propositional event models [DPS18]. In the future, the epistemic planning community will continue working on exhibiting restrictions of **DEL** (e.g. separable event models) for which this reasoning is decidable.

Finally in real applications, the notion of *intentions* is crucial. For instance in human-robot interactions, if the humans says "the keys are on the table", he may also implicitly mean that he wants the robot to take the keys. In the security example, if an attacker takes a key, the information is not only that he obtained these keys but also that he may unlock some door with these keys. In the game Hanabi, when a player gives an information, e.g. "this card has number 2", he does not only mean that the card has number 2 but also that the other player should play this card. If we want to design realistic AIs for this kind of games, we need to include intentions in **DEL**. Games are suited to study intentions because they are fairly simple and have the same mechanisms for knowledge than real-life examples in robotics and security. That is why, although games seem somehow pointless to study, they remain very interesting because they are a stepping stone to study more intricated problems.

Index

Bibliography

[AB13] Guillaume Aucher and Thomas Bolander, "Undecidability in Epistemic Planning", *in*: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013, URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6903.

[AG11] Krzysztof R Apt and Erich Grädel, *Lectures in game theory for computer scientists*, Cambridge University Press, 2011.

[Ågo+10] Thomas Ågotnes et al., "Group announcement logic", *in*: *J. Applied Logic* 8.*1* (2010), pp. 62–81, DOI: 10.1016/j.jal.2008.12.002, URL: http://dx.doi.org/10.1016/j.jal.2008.12.002.

[Ågo+13] Thomas Ågotnes et al., "Boolean Games with Epistemic Goals", *in*: *Logic, Rationality, and Interaction - 4th International Workshop, LORI 2013, Hangzhou, China, October 9-12, 2013, Proceedings*, 2013, pp. 1–14, DOI: 10.1007/978-3-642-40948-6_1, URL: http://dx.doi.org/10.1007/978-3-642-40948-6_1.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman, "Alternating-time temporal logic", *in*: *J. ACM* 49.*5* (2002), pp. 672–713, DOI: 10.1145/585265.585270, URL: http://doi.acm.org/10.1145/585265.585270.

[AMP14a] Guillaume Aucher, Bastien Maubert, and Sophie Pinchinat, "Automata Techniques for Epistemic Protocol Synthesis", *in*: (2014), pp. 97–103, DOI: 10.4204/EPTCS.146.13, URL: https://doi.org/10.4204/EPTCS.146.13.

[AMP14b] Guillaume Aucher, Bastien Maubert, and Sophie Pinchinat, "Automata Techniques for Epistemic Protocol Synthesis", *in*: *Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014.* 2014, pp. 97–103, DOI: 10.4204/EPTCS.146.13, URL: https://doi.org/10.4204/EPTCS.146.13.

[Art08] Sergei N. Artëmov, "Justification Logic", *in*: *Logics in Artificial Intelligence, 11th European Conference, JELIA 2008, Dresden, Germany, September 28 - October 1, 2008. Proceedings*, 2008, pp. 1–4, DOI: 10.1007/978-3-540-87803-2_1, URL: https://doi.org/10.1007/978-3-540-87803-2%5C_1.

[AS13] Guillaume Aucher and François Schwarzentruber, "On the Complexity of Dynamic Epistemic Logic", *in*: *TARK 2013* abs/1310.6406 (2013), URL: http://arxiv.org/abs/1310.6406.

[Aum76] Robert J Aumann, "Agreeing to disagree", *in*: *The annals of statistics* (1976), pp. 1236–1239.

[BA11] Thomas Bolander and Mikkel Birkegaard Andersen, "Epistemic planning for single and multi-agent systems", *in*: *Journal of Applied Non-Classical Logics* 21.*1* (2011), pp. 9–34, DOI: 10.3166/jancl.21.9-34, URL: http://dx.doi.org/10.3166/jancl.21.9-34.

[Bal+07] Philippe Balbiani et al., "What can we achieve by arbitrary announcements?: A dynamic take on Fitch's knowability", *in*: *TARK*, 2007, pp. 42–51.

[Bal+08] Philippe Balbiani et al., "'Knowable' as 'known after an announcement'", *in*: *Rew. Symb. Logic* 1.*3* (2008), pp. 305–334, DOI: 10.1017/S1755020308080210, URL: https://doi.org/10.1017/S1755020308080210.

[Bal+14] Philippe Balbiani et al., "DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE", *in*: *CoRR* abs/1411.7825 (2014), URL: http://arxiv.org/abs/1411.7825.

[Bar+11] Clark Barrett et al., "CVC4", *in*: *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, Snowbird, UT: Springer-Verlag, 2011, pp. 171–177, ISBN: 978-3-642-22109-5, URL: http://dl.acm.org/citation.cfm?id=2032305.2032319.

[Bar+17] Chitta Baral et al., "Epistemic Planning (Dagstuhl Seminar 17231)", *in*: *Dagstuhl Reports*, vol. 7, 6, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[BBW06] Patrick Blackburn, Johan FAK van Benthem, and Frank Wolter, *Handbook of modal logic*, vol. 3, Elsevier, 2006.

[Ben+15] Johan van Benthem et al., "Symbolic Model Checking for Dynamic Epistemic Logic", *in*: *Logic, Rationality, and Interaction - 5th International Workshop, LORI 2015 Taipei, Taiwan, October 28-31, 2015, Proceedings*, 2015, pp. 366–378, DOI: 10.1007/978-3-662-48561-3_30, URL: https://doi.org/10.1007/978-3-662-48561-3_30.

[BGS13] Philippe Balbiani, Olivier Gasquet, and François Schwarzentruber, "Agents that look at one another", *in*: *Logic Journal of the IGPL* 21.*3* (2013), pp. 438–467, DOI: 10.1093/jigpal/jzs052, URL: https://doi.org/10.1093/jigpal/jzs052.

[BGW93] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann, "Set Constraints are the Monadic Class", *in*: *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, 1993, pp. 75–83, DOI: 10.1109/LICS.1993.287598, URL: https://doi.org/10.1109/LICS.1993.287598.

[BHT13] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard, "Dynamic Logic of Propositional Assignments: A Well-Behaved Variant of PDL", *in*: *LICS*, 2013, pp. 143–152.

[BJS15] Thomas Bolander, Martin Holm Jensen, and François Schwarzentruber, "Complexity Results in Epistemic Planning", *in*: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 2015, pp. 2791–2797, URL: http://ijcai.org/papers15/Abstracts/IJCAI15-395.html.

[BK08] Christel Baier and Joost-Pieter Katoen, *Principles of model checking*, MIT Press, 2008, ISBN: 978-0-262-02649-9.

[Bla00] Patrick Blackburn, "Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto", *in*: *Logic Journal of the IGPL* 8.*3* (2000), pp. 339–365, DOI: 10.1093/jigpal/8.3.339, URL: http://dx.doi.org/10.1093/jigpal/8.3.339.

[BMP15] Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat, "Uniform strategies, rational relations and jumping automata", *in*: *Inf. Comput.* 242 (2015), pp. 80–107, DOI: 10.1016/j.ic.2015.03.012, URL: https://doi.org/10.1016/j.ic.2015.03.012.

[BMS98] Alexandru Baltag, Lawrence S. Moss, and Slawomir Solecki, "The Logic of Public Announcements and Common Knowledge and Private Suspicions", *in*: *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-98), Evanston, IL, USA, July 22-24, 1998*, 1998, pp. 43–56.

[Bol+16] Thomas Bolander et al., "Announcements to Attentive Agents", *in*: *Journal of Logic, Language and Information* 25.*1* (2016), pp. 1–35, DOI: 10.1007/s10849-015-9234-3, URL: http://dx.doi.org/10.1007/s10849-015-9234-3.

[BRV01] P. Blackburn, M. de Rijke, and Y. Venema, *Modal Logic*, Cambridge University Press, 2001.

[Bur+90] Jerry R. Burch et al., "Symbolic Model Checking: 10ˆ20 States and Beyond", *in*: *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, 1990, pp. 428–439, DOI: 10.1109/LICS.1990.113767, URL: https://doi.org/10.1109/LICS.1990.113767.

[CD17] Christophe Chareton and Hans van Ditmarsch, "Strategic Knowledge of the Past in Quantum Cryptography", *in*: *Logic, Rationality, and Interaction - 6th International Workshop, LORI 2017, Sapporo, Japan, September 11-14, 2017, Proceedings*, 2017, pp. 347–361, DOI: 10.1007/978-3-662-55665-8_24, URL: https://doi.org/10.1007/978-3-662-55665-8%5C_24.

[Cha+16] Tristan Charrier et al., "Building Epistemic Logic from Observations and Public Announcements", *in*: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.* 2016, pp. 268–277, URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12899.

[CMS16] Tristan Charrier, Bastien Maubert, and François Schwarzentruber, "On the Impact of Modal Depth in Epistemic Planning", *in*: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 1030–1036, URL: http://www.ijcai.org/Abstract/16/150.

[Coo+16] Martin C. Cooper et al., “A Simple Account of Multi-Agent Epistemic Planning”, *in*: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, 2016, pp. 193–201, DOI: 10.3233/978-1-61499-672-9-193, URL: https://doi.org/10.3233/978-1-61499-672-9-193.

[Coo71] Stephen A. Cook, “The Complexity of Theorem-Proving Procedures”, *in*: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, 1971, pp. 151–158, DOI: 10.1145/800157.805047, URL: http://doi.acm.org/10.1145/800157.805047.

[CPS17] Tristan Charrier, Sophie Pinchinat, and François Schwarzentruber, “Model Checking Against Arbitrary Public Announcement Logic: A First-Order-Logic Prover Approach for the Existential Fragment”, *in*: *Dynamic Logic. New Trends and Applications - First International Workshop, DALI 2017, Brasilia, Brazil, September 23-24, 2017, Proceedings*, 2017, pp. 133–152, DOI: 10.1007/978-3-319-73579-5_9, URL: https://doi.org/10.1007/978-3-319-73579-5%5C_9.

[CS15] Tristan Charrier and François Schwarzentruber, “Arbitrary Public Announcement Logic with Mental Programs”, *in*: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 2015, pp. 1471–1479, URL: http://dl.acm.org/citation.cfm?id=2773340.

[CS17] Tristan Charrier and François Schwarzentruber, “A Succinct Language for Dynamic Epistemic Logic”, *in*: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, 2017, pp. 123–131, URL: http://dl.acm.org/citation.cfm?id=3091148.

[CS18] Tristan Charrier and François Schwarzentruber, “Complexity of Dynamic Epistemic Logic with Common Knowledge”, *in*: *Proceedings of AiML2018*, 2018.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer, “Alternation”, *in*: *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-*

27 October 1976, 1976, pp. 98–108, DOI: 10.1109/SFCS.1976.4, URL: https://doi.org/10.1109/SFCS.1976.4.

[DB98] Rolf Drechsler and Bernd Becker, *Binary Decision Diagrams - Theory and Implementation*, Springer, 1998, ISBN: 978-0-7923-8193-8, URL: http://www.springer.com/engineering/circuits+&+systems/book/978-0-7923-8193-8.

[De +06] Benedetto De Martino et al., "Frames, biases, and rational decision-making in the human brain", *in*: *Science* 313.*5787* (2006), pp. 684–687.

[DH08] Erik D. Demaine and Robert A. Hearn, "Constraint Logic: A Uniform Framework for Modeling Computation as Games", *in*: *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, 2008, pp. 149–162, DOI: 10.1109/CCC.2008.35, URL: https://doi.org/10.1109/CCC.2008.35.

[DHK07] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Pieter Kooi, *Dynamic epistemic logic*, vol. 337, Springer Science & Business Media, 2007.

[DHL12] Hans P. van Ditmarsch, Andreas Herzig, and Tiago De Lima, "Public announcements, public assignments and the complexity of their logic", *in*: *Journal of Applied Non-Classical Logics* 22.*3* (2012), pp. 249–273.

[Dit+15] Hans Ditmarsch et al., *Handbook of epistemic logic*, College Publications, 2015.

[Dit+16] Hans van Ditmarsch et al., "Parameters for epistemic gossip problems", *in*: *Proc. LOFT 2016* (2016).

[Dit+17] Hans van Ditmarsch et al., "Epistemic protocols for dynamic gossip", *in*: *J. Applied Logic* 20 (2017), pp. 1–31, DOI: 10.1016/j.jal.2016.12.001, URL: https://doi.org/10.1016/j.jal.2016.12.001.

[Dit03] Hans P. van Ditmarsch, "The Russian Cards Problem", *in*: *Studia Logica* 75.*1* (2003), pp. 31–62.

[DK15] Hans van Ditmarsch and Barteld Kooi, *One Hundred Prisoners and a Light Bulb*, Springer, 2015, ISBN: 978-3-319-16693-3, DOI: 10.1007/978-3-319-16694-0, URL: https://doi.org/10.1007/978-3-319-16694-0.

[DPS18] Gaëtan Douéneau-Tabot, Sophie Pinchinat, and François Schwarzentruber, "Chain-Monadic Second Order Logic over Regular Automatic Trees and Epistemic Planningn Synthesis", *in*: *Proceedings of AiML2018*, 2018.

[DT11] Catalin Dima and Ferucio Laurentiu Tiplea, "Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable", *in*: *CoRR* abs/1102.4225 (2011), arXiv: 1102.4225, URL: http://arxiv.org/abs/1102.4225.

[Eng+18] Thorsten Engesser et al., "Game Description Language and Dynamic Epistemic Logic Compared", *in*: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* 2018, pp. 1795–1802, DOI: 10.24963/ijcai.2018/248, URL: https://doi.org/10.24963/ijcai.2018/248.

[ERS12] Jan van Eijck, Ji Ruan, and Tomasz Sadzik, "Action emulation", *in*: *Synthese* 185.*Supplement-1* (2012), pp. 131–151, DOI: 10.1007/s11229-012-0083-1, URL: http://dx.doi.org/10.1007/s11229-012-0083-1.

[Fag+95] R. Fagin et al., *Reasoning about Knowledge*, MIT Press, 1995.

[Fag+97] Ronald Fagin et al., "Knowledge-Based Programs", *in*: *Distributed Computing* 10.*4* (1997), pp. 199–225, DOI: 10.1007/s004460050038, URL: http://dx.doi.org/10.1007/s004460050038.

[FD08] Tim French and Hans P. van Ditmarsch, "Undecidability for arbitrary public announcement logic", *in*: *Advances in Modal Logic*, 2008, pp. 23–42.

[FH87] Ronald Fagin and Joseph Y. Halpern, "Belief, Awareness, and Limited Reasoning.", *in*: *Artif. Intell.* 34.*1* (1987), pp. 39–76, DOI: 10.1016/0004-3702(87)90003-8, URL: https://doi.org/10.1016/0004-3702(87)90003-8.

[FL79] Michael J. Fischer and Richard E. Ladner, "Propositional Dynamic Logic of Regular Programs", *in*: *J. Comput. Syst. Sci.* 18.*2* (1979), pp. 194–211, DOI: 10.1016/0022-0000(79)90046-1, URL: https://doi.org/10.1016/0022-0000(79)90046-1.

[FN71] Richard Fikes and Nils J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *in*: (1971), pp. 608–620, URL: http://ijcai.org/Proceedings/71/Papers/055.pdf.

[FTF01] Henry Francis, A Truscott, and D Francis, *The Official Encyclopedia of Bridge*, American Contract Bridge League, Incorporated, 2001.

[GGS15] Olivier Gasquet, Valentin Goranko, and François Schwarzentruber, "Big Brother Logic: visual-epistemic reasoning in stationary multi-agent systems", English, *in*: *Autonomous Agents and Multi-Agent Systems* (2015), pp. 1–33, ISSN: 1387-2532, DOI: 10.1007/s10458-015-9306-4, URL: http://dx.doi.org/10.1007/s10458-015-9306-4.

[Her+11] Andreas Herzig et al., "A Dynamic Logic of Normative Systems", *in*: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2011, pp. 228–233, DOI: 10.5591/978-1-57735-516-8/IJCAI11-049, URL: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-049.

[Hin65] Jaakko Hintikka, "Knowledge and belief. An introduction to the logic of the two notions", *in*: (1965).

[HIW12] Wiebe van der Hoek, Petar Iliev, and Michael Wooldridge, "A logic of revelation and concealment", *in*: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, 2012, pp. 1115–1122, URL: http://dl.acm.org/citation.cfm?id=2343856.

[HLM15] Andreas Herzig, Emiliano Lorini, and Faustine Maffre, "A Poor Man's Epistemic Logic Based on Propositional Assignment and Higher-Order Observation", *in*: *Logic, Rationality, and Interaction - 5th International Workshop, LORI 2015 Taipei, Taiwan, October 28-31, 2015, Proceedings*, 2015, pp. 156–168, DOI: 10.1007/978-3-662-48561-3_13, URL: https://doi.org/10.1007/978-3-662-48561-3_13.

[HM15] Andreas Herzig and Faustine Maffre, "How to Share Knowledge by Gossiping", *in*: *Multi-Agent Systems and Agreement Technologies - 13th European Conference, EUMAS 2015, and Third International Conference, AT 2015, Athens, Greece, December 17-18, 2015, Revised Selected Papers*, 2015, pp. 249–263, DOI: 10.1007/978-3-319-33509-4_20, URL: https://doi.org/10.1007/978-3-319-33509-4%5C_20.

[HM92] Joseph Y. Halpern and Yoram Moses, "A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief", *in*: *Artif. Intell.* 54.*2* (1992), pp. 319–379, DOI: 10.1016/0004-3702(92)90049-4, URL: https://doi.org/10.1016/0004-3702(92)90049-4.

[HP18] Ronald de Haan and Iris van de Pol, "On the Computational Complexity of Model Checking for Dynamic Epistemic Logic with S5 Models", *in*: *CoRR* abs/1805.09880 (2018), arXiv: 1805.09880, URL: http://arxiv.org/abs/1805.09880.

[HTW11] Wiebe van der Hoek, Nicolas Troquard, and Michael Wooldridge, "Knowledge and control", *in*: *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, 2011, pp. 719–726, URL: http://portal.acm.org/citation.cfm?id=2031720&CFID=54178199&CFTOKEN=61392764.

[HV91] Joseph Y. Halpern and Moshe Y. Vardi, "Model Checking vs. Theorem Proving: A Manifesto", *in*: (1991), pp. 325–334.

[IK94] Shigeki Iwata and Takumi Kasai, "The Othello game on an n*n board is PSPACE-complete", *in*: *Theor. Comput. Sci.* 123.*2* (1994), pp. 329–340, DOI: 10.1016/0304-3975(94)90131-7, URL: https://doi.org/10.1016/0304-3975(94)90131-7.

[KG15] Filippos Kominis and Hector Geffner, "Beliefs In Multiagent Planning: From One Agent to Many", *in*: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.* 2015, pp. 147–155, URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10617.

[Koo03] Barteld P. Kooi, "Probabilistic Dynamic Epistemic Logic", *in*: *Journal of Logic, Language and Information* 12.*4* (2003), pp. 381–408, DOI: 10.1023/A:1025050800836, URL: https://doi.org/10.1023/A:1025050800836.

[Kor+11] Barbara Kordy et al., "Foundations of Attack–Defense Trees", *in*: *Formal Aspects of Security and Trust*, ed. by Pierpaolo Degano, Sandro Etalle, and Joshua Guttman, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 80–95, ISBN: 978-3-642-19751-2.

[Kor08] Konstantin Korovin, “iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description)”, *in*: *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, 2008, pp. 292–298, DOI: 10.1007/978-3-540-71070-7_24, URL: https://doi.org/10.1007/978-3-540-71070-7_24.

[Lan06] Martin Lange, “Model checking propositional dynamic logic with all extras”, *in*: *J. Applied Logic* 4.*1* (2006), pp. 39–49, DOI: 10.1016/j.jal.2005.08.002, URL: https://doi.org/10.1016/j.jal.2005.08.002.

[Lem+10] Séverin Lemaignan et al., “ORO, a knowledge management platform for cognitive architectures in robotics”, *in*: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, 2010, pp. 3548–3553, DOI: 10.1109/IROS.2010.5649547, URL: https://doi.org/10.1109/IROS.2010.5649547.

[Lem+14] Séverin Lemaignan et al., “Human-Robot Interaction: Tackling the AI Challenges”, *in*: *Artificial Intelligence* (2014).

[Lew80] Harry R. Lewis, “Complexity Results for Classes of Quantificational Formulas”, *in*: *J. Comput. Syst. Sci.* 21.*3* (1980), pp. 317–353, DOI: 10.1016/0022-0000(80)90027-6, URL: https://doi.org/10.1016/0022-0000(80)90027-6.

[LLL04] Yongmei Liu, Gerhard Lakemeyer, and Hector J. Levesque, “A Logic of Limited Belief for Reasoning with Disjunctive Information”, *in*: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, 2004, pp. 587–597, URL: http://www.aaai.org/Library/KR/2004/kr04-061.php.

[Löw+08] Benedikt Löwe et al., “Logic and the simulation of interaction and reasoning: Introductory remarks”, *in*: (2008).

[LPW11] Benedikt Löwe, Eric Pacuit, and Andreas Witzel, “DEL Planning and Some Tractable Cases”, *in*: *Logic, Rationality, and Interaction - Third International Workshop, LORI 2011, Guangzhou, China, October 10-13, 2011. Proceedings*, 2011, pp. 179–192, DOI: 10.1007/978-3-642-24130-7_13, URL: http://dx.doi.org/10.1007/978-3-642-24130-7_13.

[LQR17] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi, "MCMAS: an open-source model checker for the verification of multi-agent systems", *in*: *STTT* 19.*1* (2017), pp. 9–30, DOI: 10.1007/s10009-015-0378-x, URL: https://doi.org/10.1007/s10009-015-0378-x.

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner, "Z3: An Efficient SMT Solver", *in*: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, 2008, pp. 337–340, DOI: 10.1007/978-3-540-78800-3_24, URL: https://doi.org/10.1007/978-3-540-78800-3_24.

[McC87] John McCarthy, "Formalization of two puzzles involving knowledge", *in*: (1987).

[McD+98] Drew McDermott et al., "PDDL-the planning domain definition language", *in*: (1998).

[Min67] Marvin L. Minsky, *Computation: Finite and Infinite Machines*, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967, ISBN: 0-13-165563-9.

[MOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996, ISBN: 0-8493-8523-7.

[Mui+15] Christian J. Muise et al., "Planning Over Multi-Agent Epistemic States: A Classical Planning Approach", *in*: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.* 2015, pp. 3327–3334, URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9974.

[MV04] J-J Ch Meyer and Wiebe Van Der Hoek, *Epistemic logic for AI and computer science*, vol. 41, Cambridge University Press, 2004.

[Pap03] Christos H Papadimitriou, *Computational complexity*, John Wiley and Sons Ltd., 2003.

[Pap81] Christos H. Papadimitriou, "On the complexity of integer programming", *in*: *J. ACM* 28.*4* (1981), pp. 765–768, DOI: 10.1145/322276.322287, URL: http://doi.acm.org/10.1145/322276.322287.

[Par08] Lynne E Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems", *in*: *Journal of Physical Agents* 2.*1* (2008), pp. 5–14.

[PKS16] Radia Perlman, Charlie Kaufman, and Mike Speciner, *Network security: private communication in a public world*, Pearson Education India, 2016.

[Pla07] Jan Plaza, "Logics of public communications", *in*: *Synthese* 158.*2* (2007), pp. 165–179, DOI: 10.1007/s11229-007-9168-7, URL: http://dx.doi.org/10.1007/s11229-007-9168-7.

[Pra80] V. R. Pratt, "A Near-Optimal Method for Reasoning about Action", *in*: *J. Comput. Syst. Sci.* 20.*2* (1980), pp. 231–254, DOI: 10.1016/0022-0000(80)90061-6, URL: http://dx.doi.org/10.1016/0022-0000(80)90061-6.

[Pra81] Vaughan R. Pratt, "A Decidable mu-Calculus: Preliminary Report", *in*: *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, 1981, pp. 421–427, DOI: 10.1109/SFCS.1981.4, URL: https://doi.org/10.1109/SFCS.1981.4.

[PRS15] Iris van de Pol, Iris van Rooij, and Jakub Szymanik, "Parameterized Complexity Results for a Model of Theory of Mind Based on Dynamic Epistemic Logic", *in*: *Proceedings Fifteenth Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2015, Carnegie Mellon University, Pittsburgh, USA, June 4-6, 2015.* 2015, pp. 246–263, DOI: 10.4204/EPTCS.215.18, URL: https://doi.org/10.4204/EPTCS.215.18.

[RSY09] Bryan Renne, Joshua Sack, and Audrey Yap, "Dynamic Epistemic Temporal Logic", *in*: *Logic, Rationality, and Interaction, Second International Workshop, LORI 2009, Chongqing, China, October 8-11, 2009. Proceedings*, 2009, pp. 263–277, DOI: 10.1007/978-3-642-04893-7_21, URL: https://doi.org/10.1007/978-3-642-04893-7%5C_21.

[Sad06] Tomasz Sadzik, "Exploring the Iterated Update Universe", ILLC Publications PP-2006-26, 2006, URL: http://www.illc.uva.nl/Research/Publications/Reports/PP-2006-26.text.pdf.

[Sav70] Walter J. Savitch, "Relationships Between Nondeterministic and Deterministic Tape Complexities", *in*: *J. Comput. Syst. Sci.* 4.*2* (1970), pp. 177–192,

DOI: 10.1016/S0022-0000(70)80006-X, URL: http://dx.doi.org/10.1016/S0022-0000(70)80006-X.

[Sch18] François Schwarzentruber, "Hintikka's World: Agents with Higher-order Knowledge", *in*: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.* 2018, pp. 5859–5861, DOI: 10.24963/ijcai.2018/862, URL: https://doi.org/10.24963/ijcai.2018/862.

[Sip96] Michael Sipser, *Introduction to the Theory of Computation*, vol. 27, 1, 1996, pp. 27–29, DOI: 10.1145/230514.571645, URL: http://doi.acm.org/10.1145/230514.571645.

[SSY94] Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis, "The TPTP Problem Library", *in*: *Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994, Proceedings*, 1994, pp. 252–266, DOI: 10.1007/3-540-58156-1_18, URL: https://doi.org/10.1007/3-540-58156-1%5C_18.

[Sut16] Geoff Sutcliffe, "The CADE ATP System Competition—CASC", *in*: *AI Magazine* 37.*2* (2016), pp. 99–101.

[SW18] Jeremy Seligman and Yanjing Wang, "Call Me by Your Name: Epistemic Logic with Assignments and Non-rigid Names", *in*: *CoRR* abs/1805.03852 (2018), arXiv: 1805.03852, URL: http://arxiv.org/abs/1805.03852.

[Thi16] Michael Thielscher, "GDL-III: A Proposal to Extend the Game Description Language to General Epistemic Games", *in*: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, 2016, pp. 1630–1631, DOI: 10.3233/978-1-61499-672-9-1630, URL: https://doi.org/10.3233/978-1-61499-672-9-1630.

[vV02] W. van der Hoek and R. Verbrugge, "Epistemic Logic: A Survey", *in*: *Game Theory and Applications* 8 (2002), ed. by L.A. Petrosjan and V.V. Mazalov, pp. 53–94.

[YWL13] Quan Yu, Ximing Wen, and Yongmei Liu, "Multi-Agent Epistemic Explanatory Diagnosis via Reasoning about Actions.", *in*: *IJCAI*, ed. by Francesca

Rossi, IJCAI/AAAI, 2013, ISBN: 978-1-57735-633-2, URL: http://dblp.uni-trier.de/db/conf/ijcai/ijcai2013.html#YuWL13.

Appendix A

Reminder on Complexity Classes

A.1 Classical classes

Through the manuscript, we analyze the complexity of many decision problems. First we define what a decision problem is.

Definition 79 (Decision problem)**.** *A* decision problem *is a problem that only has two outputs for any input: yes or no. It will be presented in the following way.*

- ***Input:*** *...*
- ***Output:*** *yes if ..., no otherwise.*

Decision problems are naturally associated to the concept of Turing machines. They are automata, thus taking a word (the input) and saying yes/no, but expressive enough to capture all algorithms we can write.

Definition 80 (Turing machine)**.** *A* Turing machine *is a tuple* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ *where:*

- Q *is the finite set of states of* M*;*
- Σ *is the finite input alphabet;*
- Γ *is the finite tape alphabet with* $\Sigma \subseteq \Gamma$*;*
- $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, +1\}$ *is the transition function;*
- $q_0 \in Q$ *is the initial state,* $q_{acc} \in Q$ *the accepting state and* $q_{rej} \in Q$ *is the rejecting state;*

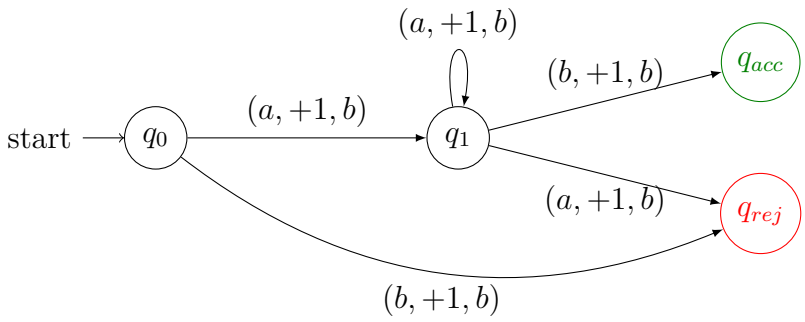


Figure A.1: Example of a Turing machine

A Turing machine is an automaton where a word is written on an input tape. The input tape contains only letters of an alphabet Σ called the input alphabet. The Turing machine then works with a larger alphabet, called the tape alphabet Γ. Anytime on the tape, a unique cell is pointed by a cursor, initially it is the first cell. In each state, when transition $(q_1, a, q_2, b, +1)$ is fired, we go from state q_1 to state q_2 by reading the letter a, write the letter a and then move the cursor to the next cell $(+1)$. If the transition contains -1 instead of +1, we go back to the previous cell. Let us give an example.

Example 33. *We consider the Turing machine of Figure A.1. The accepting state is in green, the rejecting state in red, the initial state is marked by an incoming edge. Here we consider* $\Sigma = \Gamma = \{a, b\}$*. On the tape, initially the word aaaaaaab is written. Here several executions are possible. For instance* $q_0 \to q_1 \to q_{rej}$ *is possible by reading two times a. The execution* $q_0 \to q_1 \to q_1 \to q_{rej}$ *is also possible. Unfortunately, both executions are rejecting, whereas the execution* $q_0 \to q_1 \to q_1 \to q_1 \to q_1 \to q_1 \to q_1 \to q_{acc}$ *is accepting. In this case, because such an execution exists, we say that the Turing machine accepts the input.*

Here, the language accepted by this Turing machine is a^+b *(read at least one a and then end on some b).*

We do not define formally the notion of execution here. It is just the data of a sequence of states visited with the state of the tape given in each step. The set of words such that there exist an execution reaching q_{acc} is called the *language* of M, noted $\mathcal{L}(M)$.

In general, the following decision problem is undecidable:

- **Input:** a Turing machine M, a word ω.

- **Output:** yes if $\omega \in \mathcal{L}(M)$, no otherwise.

It means that it is not possible to find a Turing machine M_0 taking in input any Turing machine M and any word ω and accepting the input if and only if $\omega \in \mathcal{L}(M)$.

However, Turing machines are used to define complexity classes of algorithms. Indeed, when we restrict the set of possible Turing machines in respect of the length of the execution/the memory needed, we obtain classes of problems that we call complexity classes. In the following definition, we refer to *deterministic* and *non-deterministic* Turing machines. Deterministic Turing machines are such that at most one transition is possible in each state for each letter $a \in \Sigma$. Non-deterministic Turing machines do not have this restriction.

Example 34. *The Turing machine of Figure A.1 is non-deterministic because in q_1, there are two transitions with letter a: one leading to q_1 and one leading to q_{rej}.*

We now define complexity classes.

Definition 81 (Complexity classes)**.**

Deterministic time *The classes* P*,* Exptime*,* 2-Exptime*, are the classes of deterministic Turing machines whose executions always end in $O(P(n))$ time, $O(2^{P(n)})$ time, $O(2^{2^{P(n)}})$ time, etc., where P is a polynomial and n is the size of the input word.*

Deterministic space *The classes* Pspace*,* Expspace*,* 2-Expspace*, are the classes of deterministic Turing machines whose executions need only $O(P(n))$, $O(2^{P(n)})$, $O(2^{2^{P(n)}})$ new cells (the memory) to work. Such classes are not defined with the length of the execution but with the memory needed.*

Non-deterministic time and space *The non-deterministic counterparts of these complexity classes are called* NPtime *(also called* NP*),* NExptime*,* NPspace*, etc.*

It is known that P $\subseteq$ NP $\subseteq$ Pspace $\subseteq$ NPspace $\subseteq$ Exptime $\subseteq$ NExptime ...

Interestingly, it is also known that Xspace = NXspace for X $\in$ { P, Exp, 2Exp, ...} (by Savitch's theorem [Sav70]). However, it is not known whether Xtime = NXtime or Xtime $\neq$ NXtime.

We conclude the classical complexity classes by defining the notion of membership, hardness and completeness of a decision problem in a complexity class:

Definition 82 (Membership, Hardness, Completeness)**.** *Let P we a decision problem and C a complexity class (*Pspace*,* NExptime *...).*

We say that P is a member *of C if there exist a Turing machine M of C deciding P.*

We say that P is hard *for C if for all Turing machines of M, there exists a function f running in polynomial time taking in parameter an input M, returning an input of P, and such that M accepts ω if and only if $f(\omega)$ is a positive instance of P.*

P is complete *for C if it is a member of C and it is hard for C.*

Instead of writing Turing machines to prove the membership of problems, we will write algorithms in pseudo-code. Interestingly, the correspondence between algorithms and Turing machines is direct: an algorithm running in polynomial time/space induces a Turing machine running in polynomial time/space.

We now define a different class of Turing machines that will prove very handy to establish the complexity of decision problems later on.

A.2 Alternating Turing machines

Alternating Turing machines [CS76] are an extension of Turing machines where states now have a type: $\exists$ or $\forall$. In an execution, a state labeled with $\exists$ must have at least one transition which makes the execution succeed. For $\forall$ states, every compatible transition must make the execution succeed. If all states are $\exists$, we go back to non-deterministic machines.

Definition 83 (Alternating Turing machine)**.**

An alternating Turing machine *is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}, g)$ where:*

- *Q is the finite set of states of M;*
- *Σ is the finite input alphabet;*
- *Γ is the finite tape alphabet with $\Sigma \subseteq \Gamma$;*
- *$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, +1\}$ is the transition function;*
- *$q_0 \in Q$ is the initial state, $q_{acc} \in Q$ the accepting state and $q_{rej} \in Q$ is the rejecting state;*
- *$g : Q \to \{\exists, \forall\}$ is the quantification function for the states.*

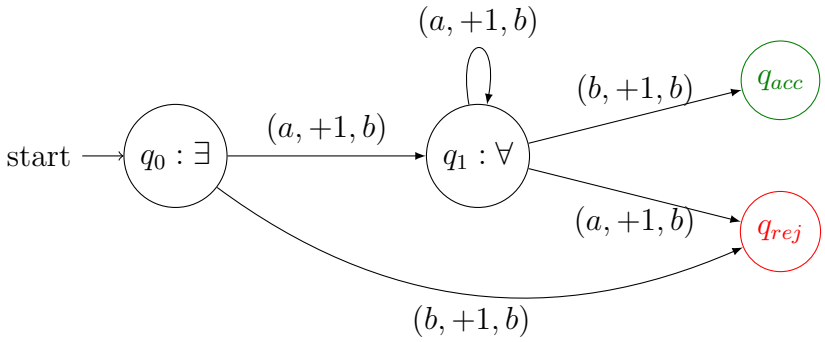


Figure A.2: Example of an alternating Turing machine

Example 35. *Figure A.2 gives an example of an alternating Turing machine. Each state now have a type:* $\exists$ *or* $\forall$*, except the accept and reject state that do not really need a type since the execution always end when these states are reached.*

The word aaaaaaab is now not accepted by the Turing machine. Indeed, when in q_1*, firing the* $q_1 \to q_1$ *transition and the* $q_1 \to q_{rej}$ *must lead to a successful execution. Yet, the transition to* q_{rej} *is obviously not successful.*

In fact, here, the only word recognized is ab.

Alternating Turing machines create new complexity classes, defined as follows.

Definition 84 (Alternating complexity classes)**.**

Alternating time *The classes* Aptime, Aexptime, A2Exptime, *are the classes of alternating Turing machines whose executions always end in* $O(P(n))$ *time,* $O(2^{P(n)})$ *time,* $O(2^{P(n)})$ *time, etc., where* P *is a polynomial and* n *is the size of the input word.*

Alternating space *The classes* Apspace, Aexpspace, A2Expspace, *are the classes of alternating Turing machines whose executions need only* $O(P(n))$*,* $O(2^{P(n)})$*,* $O(2^{P(n)})$ *new cells (the memory) to work.*

We also define the class A_{pol}Exptime *which is the class of alternating Turing machine running in exponential time but there is only a polynomial number of alternation of* $\exists$ *and* $\forall$ *in any execution.*

The main advantage of alternating classes is that they are directly correlated with deterministic class. We have AXtime = Xspace and AXspace = XExptime for every X $\in$ { P, Exp, 2Exp, ...}. For instance, Apspace = Exptime, A2Exptime = 2Expspace. When we wish to analyze the membership of problems, we then choose to

establish the complexity with an alternating class of complexity instead of its deterministic class. For that, we write alternating algorithms that we now define.

A.3 Algorithms for alternating complexity classes

Alternating algorithms are standard algorithms augmented with two types of non-deterministic choices: *existential* (∃) choices and *universal* (∀) choices. From an existential choice, the algorithm accepts if there exists a choice that makes the continuation of algorithm accept. From a universal choice, any choice should make the continuation of the algorithm accept. For example, when we write the instruction "(∃) $w \in W$", the algorithm non-deterministically chooses w in W and continues its execution with this chosen world w. As it is an existential choice, the instruction "(∃) $w \in W$" succeeds if there is a way to choose w, so that the algorithm accepts. Dually, the instruction "(∀) $w \in W$" succeeds whenever any choice of w makes the algorithm accept.

In the following existential and universal choices between several algorithms will be written as follows.

(∃) $algo_1$ **or** $algo_2$	(∀) $algo_1$ **and** $algo_2$

"(∀) $algo_1$ **and** $algo_2$" means that both $algo_1$ and $algo_2$ must accept, "(∃) $algo_1$ **or** $algo_2$" means that either $algo_1$ or $algo_2$ must accept.

Instruction I	Running time $T(I)$
(**Q**) $i \in \{0, 1\}$	1
(**Q**) $i \in \{1, ..., n\}$	$log_2(n)$
(**Q**) $u \in W$	$\log_2(\|W\|)$
Choose $W' \subseteq W$	$\|W\|$
(∃) $algo_1$ **or** $algo_2$	$1 + \max(T(algo_1), T(algo_2))$
(∀) $algo_1$ **and** $algo_2$	$1 + \max(T(algo_1), T(algo_2))$
Sub-algorithm $proc$	$1 + T(proc)$

Table A.1: Execution time of alternating algorithms where $|AP|$ is the cardinal of AP.

Execution time of alternating algorithms. The execution time of an alternating algorithm corresponds to the depth of its computation tree. Table A.1 describes the execution time of some instructions up to a constant factor (we write (**Q**) for either (∃) or

(∀)). Choosing a world $w \in W$ takes $\log_2(|W|)$ steps because it consists in assigning an integer to each world of W and choosing $\log_2(|W|)$. The argument is similar for choosing $i \in \{1, ..., n\}$. Choosing a set of worlds $W \subseteq W$ takes $|W|$ steps because it amounts for each world $w \in W$ to choose whether it is in W'.

Appendix B

Reminder on Uniform Strategies

The definitions from this chapter are inspired from [AG11]. For more information about game theory and uniform strategies, the reader may refer to this book. In particular, although here games are finite, the book considers infinite games.

B.1 Games with imperfect information

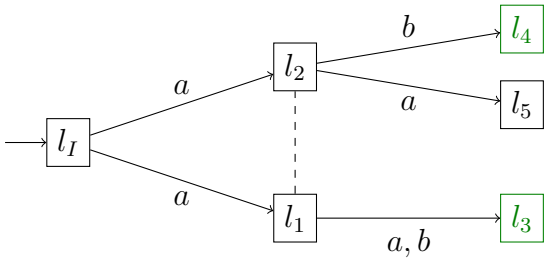
We consider the setting of two-player games such that one player does not have perfect knowledge of the state of the game whereas the other does. The reason is explained afterwards when winning strategies are defined. Such games are defined as follows.

Definition 85 (Game with imperfect information)**.** *A game with imperfect information is a tuple* $G = \langle L, l_I, L_G, \Sigma, \delta, O \rangle$ *where:*

- L *is a finite set of states;*
- l_I *is the initial state;*
- $L_G \subseteq L$ *is the set of winning states;*
- Σ *is a finite alphabet of actions;*
- $\delta \subseteq L \times \Sigma \times L$ *is the transition function;*
- O *is a partition of* L *called the observations.*

Here we suppose that states in L_G *are end states, meaning there are no outgoing transitions from states in* L_G*.*

Intuitively, in such games, one player has not full knowledge of the current state of the game and plays the actions, and the other player chooses which transition is taken. The aim for the player controlling the action is to reach one of the states in L_G, the goal of the other player is to avoid it.



l_3, l_4 are the winning states, l_1, l_2 are indistinguishable for the player.

Figure B.1: Example of a game with imperfect information

Example 36. *We take the game of Figure B.1. Initially, the game starts in l_I. The player only has one possible action, a, that leads either into l_1 or l_2. The other player that chooses whether the next state is l_1 or l_2. Both states l_1 and l_2 are indistinguishable for the player, so he does not know which one is the current state. Then in state l_2, if the player plays b, he wins, otherwise he loses. In state l_1, whatever the player plays, he wins. Formally the game is defined as follows:*

- $L = \{l_I, l_1, l_2, l_3, l_4, l_5\}$;
- $L_G = \{l_3, l_4\}$;
- $\Sigma = \{a, b\}$;
- $\delta = \{(l_I, a, l_1), (l_I, a, l_2), (l_1, a, l_3), (l_1, b, l_3), (l_2, b, l_4), (l_2, a, l_5)\}$;
- $O = \{\{l_I\}, \{l_1, l_2\}, \{l_3\}, \{l_4\}, \{l_5\}\}$.

We now define the notion of history and run in a game.

Definition 86. *History and run* *A* history *is a sequence of states $l_I = l_0, l_1, \ldots, l_n$ such that for all $0 \leq i \leq n-1$, there exists $a \in \Sigma$ such that $(l_i, a, l_{i+1}) \in \delta$.*

A run *is a history such that l_n is an end state. It is* winning *if $l_n \in L_G$, losing otherwise.*

There exist games where infinite runs are possible (like Go for instance). For the manuscript we forbid infinite runs since the games we model only have finite runs.

B.2 Uniform strategies

We now define the notion of uniform strategies. We begin by considering the previous example.

Example 37. *In the game of Figure B.1, the player must first play a in l_I. Then, he has to choose an action but does not know whether he is in state l_1 or state l_2. If he chooses to play a, he wins if the current state is l_1 but loses if the current state is l_2. Here, in a strategy, we cannot allow the player to play differently depending on the state because he does not know which state is the current one. Therefore to have a winning strategy, playing a is forbidden. Yet, playing b is allowed here because for any of the states l_1 or l_2, playing b leads to a winning state.*

The definition of a uniform strategies is thus the following.

Definition 87. *Uniform strategy A* strategy *is a function $\sigma : L^+ \to \Sigma$ that assigns an action to a history $(l_I, \dots, l_n)$ of visited states.*

The strategy σ is said uniform *if for all histories $(l_I, l_1, \dots, l_n)$ and $(l_I, l'_1, \dots, l'_n)$ such that $(l_i, l'_i) \in O$ for all i, $\sigma(l_I, \dots, l_n) = (l_I, \dots, l'_n)$.*

A uniform strategy is said winning if whatever the other player does, by following the strategy the games always end in a winning state.

Definition 88. *Let σ be a uniform strategy. It is said winning if all runs $l_I = l_0, l_1, \dots, l_n$ such that $(l_i, \sigma(l_0, \dots, l_i), l_{i+1}) \in \delta$ for all $0 \leq i \leq n-1$ are winning.*

Example 38. *In the game of Figure B.1, the strategy defined by $\sigma(l_I) = a, \sigma(l_I, l_1) = \sigma(l_I, l_2) = a$ is not winning but the strategy defined by $\sigma'(l_I) = a, \sigma'(l_I, l_1) = \sigma'(l_I, l_2) = b$ is winning.*

Appendix C

Model Checking against ELCK is in P

This Appendix is dedicated to the proof of the following theorem:

Theorem 28. *The model checking problem against* **ELCK** *is in* P*.*

The proof is an extension of the proof for CTL [BK08]. Indeed, CTL corresponds to **ELCK** with one agent and the until operator added. We just add here multi-agent aspects.

Let $\mathcal{M} = (W, (R_a)_{a \in Ag}, V)$ be a Kripke model. The algorithm of Figure C.1 computes the set of worlds W_φ satisfying a formula φ. The correctness proof is direct for propositional constructions. For $\varphi = K_a\psi$, we construct the set of worlds W' satisfying ψ and compute the set of worlds W'' such that all a-successors fall in W', which corresponds to the set of worlds satisfying ψ. For $\varphi = C_G\psi$, we first compute the set of worlds W' satisfying ψ, and then repeat a loop where we remove the worlds from W' with a successor by an agent of G which is not in W'. In the end, as for the $A\Box$ operator for CTL, we obtain exactly the set of worlds satisfying $C_G\psi$.

For the complexity of the algorithm, the only real matter is for case $\varphi = C_G\psi$, since it contains a while loop. Notice that the while loop only removes worlds from W' and stops when no world is removed in an iteration, therefore there is at most $|W'|$ loops, which remains polynomial.

```
function setofworlds(M, φ)
  case φ = p:
    W' = ∅
    for w ∈ W:
      if p ∈ V(w): W' = W' ∪ {w}
    return W'
  case φ = ¬ψ: return (W \ setofworlds(M, ψ))
  case φ = φ1 ∨ φ2: return (setofworlds(M, φ1) ∪ setofworlds(M, φ2))
  case φ = K_a ψ:
    Let W' = setofworlds(M, ψ)
    Let W'' = W
    for w ∈ W:
      for u ∈ W:
        if (w, u) ∈ R_a and u ∉ W':
          W'' = W'' \ {w}
    return W''
  case φ = C_G ψ:
    Let W' = setofworlds(M, ψ)
    Let p_stop = ⊥
    while ¬p_stop:
      p_stop = ⊤
      for w ∈ W:
        for u ∈ W:
          for a ∈ G:
            if (w, u) ∈ R_a and u ∉ W':
              W' = W' \ {w}
              p_stop = ⊥
    return W'
```

Figure C.1: Procedure for **ELCK** computing the set of worlds satisfying a formula

Appendix D

Dual Algorithms

D.1 Algorithms of Chapter 2

proc $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi)$ $|\mathcal{M}\overrightarrow{\mathcal{E}}| + |\varphi|$

▷ accepts whenever $\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e} \not\models \varphi$

- **case** $\varphi = p$: $inval_{\mathsf{no}}(p, \mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e})$
- **case** $\varphi = (\varphi_1 \vee \varphi_2)$: **(∀)** choose $i \in \{1, 2\}$; $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \varphi_i)$
- **case** $\varphi = \neg\psi$: $mc_{\mathsf{yes}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \psi)$.
- **case** $\varphi = K_a\psi$:
  - **(∃)** choose $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$
  - **(∀)** $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ **and** $in_{\mathsf{yes}}(u\overrightarrow{f}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **and** $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$
- **case** $\varphi = \langle\mathcal{E}, \mathsf{E}_0\rangle\psi$:
  - **(∀)** choose $e \in \mathsf{E}_0$;
  - **(∃)** $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, w\overrightarrow{e}, \mathrm{pre}(e))$ **or** $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}} :: \mathcal{E}, w\overrightarrow{e} :: e, \psi)$.
- **case** $\varphi = C_G\psi$:
  - **(∃)** choose $u\overrightarrow{f} \in \mathcal{M}\overrightarrow{\mathcal{E}}$
  - **(∀)** $access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, B_{\mathcal{M},\varphi}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **and** $in_{\mathsf{yes}}(u\overrightarrow{f}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **and** $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}, u\overrightarrow{f}, \psi)$

proc $inval_{\mathsf{no}}(p, w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ $|\mathcal{M}\overrightarrow{\mathcal{E}}|$

▷ accepts whenever $p \notin V(w\overrightarrow{e})$

- **case** $\overrightarrow{\mathcal{E}} = \epsilon$: **if** $p \in V(w)$ **then reject else accept**
- **case** $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}'::\mathcal{E}$ and $w\overrightarrow{e} = w\overrightarrow{e}'::e$: $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{post}(e, p))$

proc $in_{\mathsf{no}}(w\overrightarrow{e}, \mathcal{M}\overrightarrow{\mathcal{E}})$ $|\mathcal{M}\overrightarrow{\mathcal{E}}|$

▷ accepts whenever $w\overrightarrow{e} \notin \mathcal{M}\overrightarrow{\mathcal{E}}$

- **case** $\overrightarrow{\mathcal{E}} = \epsilon$: **reject**
- **case** $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}'::\mathcal{E}$, $w\overrightarrow{e} = w\overrightarrow{e}'::e$: **(∃)** $mc_{\mathsf{no}}(\mathcal{M}\overrightarrow{\mathcal{E}}', w\overrightarrow{e}', \mathrm{pre}(e))$ **or** $in_{\mathsf{no}}(w\overrightarrow{e}', \mathcal{M}\overrightarrow{\mathcal{E}}')$

proc $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$ $|\mathcal{M}\overrightarrow{\mathcal{E}}|$

▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \notin R_a$

case $\overrightarrow{\mathcal{E}} = \epsilon$: **if** $(w,u) \notin R_a^{\mathcal{M}}$ **then accept else reject**

case $\overrightarrow{\mathcal{E}} = \overrightarrow{\mathcal{E}}' :: \mathcal{E}$, $\overrightarrow{e} = \overrightarrow{e}' :: e$, $\overrightarrow{f} = \overrightarrow{f}' :: f$:

if $(e,f) \notin R_a^{\mathcal{E}}$ **then accept else** $access_{\mathsf{no}}(w\overrightarrow{e}', u\overrightarrow{f}', a, \mathcal{M}\overrightarrow{\mathcal{E}}')$

proc $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathcal{M}\overrightarrow{\mathcal{E}})$ $|\mathcal{M}\overrightarrow{\mathcal{E}}| + \log i$

▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \notin (\bigcup_{a\in G} R_a)^j$ for all $j \le i$

case $i = 1$:

if $u\overrightarrow{f} = w\overrightarrow{e}$ **then reject else (∀)** choose $a \in G$; $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathcal{M}\overrightarrow{\mathcal{E}})$

case $i \ge 2$:

(∀) choose $v\overrightarrow{g} \in \mathcal{M}\overrightarrow{\mathcal{E}}$

(∃) $in_{\mathsf{no}}(v\overrightarrow{g}, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $access^*_{\mathsf{no}}(w\overrightarrow{e}, v\overrightarrow{g}, G, i/2, \mathcal{M}\overrightarrow{\mathcal{E}})$ **or** $access^*_{\mathsf{no}}(v\overrightarrow{g}, u\overrightarrow{f}, G, i/2, \mathcal{M}\overrightarrow{\mathcal{E}})$

D.2 Algorithms for Chapter 3

proc $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi)$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}| + |\varphi|$

▷ accepts whenever $\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e} \not\models \varphi$

case $\varphi = p$: $inval_{\mathsf{no}}(p, \mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e})$

case $\varphi = (\varphi_1 \vee \varphi_2)$: **(∀)** choose $i \in \{1,2\}$; $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \varphi_i)$

case $\varphi = \neg\psi$: $mc_{\mathsf{yes}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \psi)$.

case $\varphi = K_a\psi$:

(∃) choose $u\overrightarrow{f} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$

(∀) $access_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **and** $in_{\mathsf{yes}}(u\overrightarrow{f}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **and** $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, u\overrightarrow{f}, \psi)$

case $\varphi = \langle\mathfrak{E}, \beta_0\rangle\psi$:

(∀) choose e such that $e \models \beta_0$;

(∃) $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, w\overrightarrow{e}, \mathsf{pre}|_{\mathrm{v_e}})$ **or** $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}} :: \mathfrak{E}, w\overrightarrow{e} :: e, \psi)$.

case $\varphi = C_G\psi$:

(∃) choose $u\overrightarrow{f} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$

(∀) $access^*_{\mathsf{yes}}(w\overrightarrow{e}, u\overrightarrow{f}, G, B_{\mathcal{M},\varphi}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **and** $in_{\mathsf{yes}}(u\overrightarrow{f}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **and** $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}, u\overrightarrow{f}, \psi)$

proc $inval_{\mathsf{no}}(p, w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$

▷ accepts whenever $p \notin V(w\overrightarrow{e})$

case $\overrightarrow{\mathfrak{E}} = \epsilon$: **if** $p \notin w$ **then accept else reject**

case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}' :: \mathfrak{E}$, $w\overrightarrow{e} = w\overrightarrow{e}' :: e$: $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}', w\overrightarrow{e}', \mathsf{post}(p)|_{\mathrm{v_e}})$

proc $in_{\mathsf{no}}(w\overrightarrow{e}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$

▷ accepts whenever $w\overrightarrow{e} \notin \mathfrak{M}\overrightarrow{\mathfrak{E}}$

case $\overrightarrow{\mathfrak{E}} = \epsilon$: **reject**

case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}'::\mathfrak{E}$, $w\overrightarrow{e} = w\overrightarrow{e}'::e$: (∃) $mc_{\mathsf{no}}(\mathfrak{M}\overrightarrow{\mathfrak{E}}', w\overrightarrow{e}', \mathsf{pre}|_{\mathsf{v_e}})$ **or** $in_{\mathsf{no}}(w\overrightarrow{e}', \mathfrak{M}\overrightarrow{\mathfrak{E}}')$

proc $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}|$

▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \notin R_a$

case $\overrightarrow{\mathfrak{E}} = \epsilon$: $relation_{\mathsf{no}}(w, u, \pi_a)$

case $\overrightarrow{\mathfrak{E}} = \overrightarrow{\mathfrak{E}}' :: \mathfrak{E}$, $\overrightarrow{e} = \overrightarrow{e}' :: e$, $\overrightarrow{f} = \overrightarrow{f}' :: f$:

(∃) $access_{\mathsf{no}}(w\overrightarrow{e}', u\overrightarrow{f}', a, \mathfrak{M}\overrightarrow{\mathfrak{E}}')$ **or** $relation_{\mathsf{no}}(e, f, \pi_a^{\mathcal{E}})$

proc $access^*_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, G, i, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ $|\mathfrak{M}\overrightarrow{\mathfrak{E}}| + \log i$

▷ accepts whenever $(w\overrightarrow{e}, u\overrightarrow{f}) \notin (\bigcup_{a\in G} R_a)^j$ for all $j \leq i$

case $i = 1$:

if $u\overrightarrow{f} = w\overrightarrow{e}$ **then reject else** (∀) choose $a \in G$; $access_{\mathsf{no}}(w\overrightarrow{e}, u\overrightarrow{f}, a, \mathfrak{M}\overrightarrow{\mathfrak{E}})$

case $i \geq 2$:

(∀) choose $v\overrightarrow{g} \in \mathfrak{M}\overrightarrow{\mathfrak{E}}$

(∃) $in_{\mathsf{no}}(v\overrightarrow{g}, \mathfrak{M}\overrightarrow{\mathfrak{E}})$ **or** $access^*_{\mathsf{no}}(w\overrightarrow{e}, v\overrightarrow{g}, G, i/2, \mathfrak{M}\overrightarrow{\mathfrak{E}})$

or $access^*_{\mathsf{no}}(v\overrightarrow{g}, u\overrightarrow{f}, G, i/2, \mathfrak{M}\overrightarrow{\mathfrak{E}})$

proc $relation_{\mathsf{no}}(w, u, \pi)$ $|\pi|$

▷ accepts whenever $w \not\overset{\pi}{\to} u$

case $\pi = p{\leftarrow}\beta$:

if $(w \models \beta \wedge (u \neq w \cup \{p\})) \vee (w \not\models \beta \wedge (u \neq w \setminus \{p\}))$ **then accept else reject**

case $\pi = \pi_1; \pi_2$: (∀) $v \in \mathscr{U}$; (∃) $relation_{\mathsf{no}}(w, v, \pi_1)$ **or** $relation_{\mathsf{no}}(v, u, \pi_2)$

case $\pi = \pi_1 \cup \pi_2$: (∀) $relation_{\mathsf{no}}(w, u, \pi_1)$ **and** $relation_{\mathsf{no}}(w, u, \pi_2)$

case $\pi = \beta?$: **if** $w \neq u \vee w\neg \models_{PL} \beta$ **then accept else reject**

D.3 Algorithms for Chapter 5

D.3.1 Symbolic model checking

proc $relation_{\mathsf{no}}(w, u, \pi)$ $|\pi|$

▷ accepts whenever $w \overset{\pi}{\not\to} u$

case $\pi = p{\leftarrow}\beta$:

if $(w \models \beta \wedge (u \neq w \cup \{p\})) \vee (w \not\models \beta \wedge (u \neq w \setminus \{p\}))$ **then accept else reject**

case $\pi = \pi_1; \pi_2$: (∀) $v \in \mathscr{U}$; (∃) $relation_{\mathsf{no}}(w, v, \pi_1)$ **or** $relation_{\mathsf{no}}(v, u, \pi_2)$

case $\pi = \pi_1 \cup \pi_2$: (∀) $relation_{\mathsf{no}}(w, u, \pi_1)$ **and** $relation_{\mathsf{no}}(w, u, \pi_2)$

case $\pi = \beta?$: **if** $w \neq u \vee w\neg \models_{PL} \beta$ **then accept else reject**

Star-free fragment

proc $in_{\mathsf{no}}(L, w)$ $|L| + |AP| + 1$

▷ accepts whenever $w \notin W_L$

case $L = [\,]$: **reject**

case $L = L'{::}\varphi$: (∃) $mc_{\mathsf{no}}(L', w, \varphi)$ **or** $in_{\mathsf{no}}(L', w)$

proc $access^*_{\mathsf{no}}(L, w, u, \pi, i)$ $|L| + |AP| + \log_2(i) + |\pi|$

▷ accepts whenever for all $k < i$ and $w_1, \ldots, w_{k-1}$, it is false that $w \xrightarrow{\pi} w_1 \ldots \xrightarrow{\pi} w_{k-1} \xrightarrow{\pi} u$

case $i = 1$: **if** $w \neq u$ **then** $access_{\mathsf{no}}(w, u, \pi)$ **else reject**

case $i \geq 2$:

(∀) $v \in \mathscr{U}$

(∃) $in_{\mathsf{no}}(L, u)$ **or** $access^*_{\mathsf{no}}(L, w, v, \pi, i/2)$ **or** $access^*_{\mathsf{no}}(L, v, u, \pi, i/2)$

proc $mc_{\mathsf{no}}(L, w, \varphi)$ $|L| + |AP| + |\varphi|$

▷ accepts whenever $W_L, w \not\models \varphi$

case $\varphi = p$: **if** $p \notin w$ **then accept else reject**

case $\varphi = \neg\psi$: $mc_{\mathsf{yes}}(L, w, \psi)$

case $\varphi = (\psi_1 \vee \psi_2)$: **(∀)** $mc_{\mathsf{no}}(L, w, \psi_1)$ **and** $mc_{\mathsf{no}}(L, w, \psi_2)$

case $\varphi = K_a\psi$: **(∃)** $u \in \mathscr{U}$; **(∀)** $access_{\mathsf{yes}}(w, u, \pi_a)$ **and** $in_{\mathsf{yes}}(L, u)$ **and** $mc_{\mathsf{no}}(L, u, \psi)$

case $\varphi = C_G\psi$:

- **(∃)** $u \in \mathscr{U}$
- **(∀)** $access^*_{\mathsf{yes}}(L, w, u, \bigcup_{a\in G} \pi_a, 2^{\#AP})$ **and** $in_{\mathsf{yes}}(L, u)$ **and** $mc_{\mathsf{no}}(L, u, \psi)$

case $\varphi = \langle\psi!\rangle\chi$: **(∃)** $mc_{\mathsf{no}}(L, w, \psi)$ **or** $mc_{\mathsf{no}}(L :: \psi, w, \chi)$

case $\varphi = \langle\psi?\rangle\chi$: **(∃)** $mc_{\mathsf{no}}(L, w, \psi)$ **or** $mc_{\mathsf{no}}(L, w, \chi)$

case $\varphi = \langle\tau_1 \cup \tau_2\rangle\chi$: **(∀)** $mc_{\mathsf{no}}(L, w, \langle\tau_1\rangle\chi)$ **and** $mc_{\mathsf{no}}(L, w, \langle\tau_2\rangle\chi)$

case $\varphi = \langle\tau_1; \tau_2\rangle\chi$: $mc_{\mathsf{no}}(L, w, \langle\tau_1\rangle\langle\tau_2\rangle\chi)$

General case

proc $access^*_{\mathsf{no}}(W, w, u, \pi, i)$ $|AP| + \log_2(i) + |\pi|$

▷ accepts whenever for all $k < i$ and $w_1, \ldots, w_{k-1}$, it is false that $w \xrightarrow{\pi} w_1 \ldots \xrightarrow{\pi} w_{k-1} \xrightarrow{\pi} u$

case $i = 1$: **if** $w \neq u$ **then** $access_{\mathsf{no}}(w, u, \pi)$ **else reject**

case $i \geq 2$: **(∀)** $v \in W$; **(∃)** $access^*_{\mathsf{no}}(W, w, v, \pi, \lfloor i/2 \rfloor)$ **or** $access^*_{\mathsf{no}}(W, v, u, \pi, \lceil i/2 \rceil)$

proc $mc_{\mathsf{no}}(W, w, \varphi)$ $|AP| + |\varphi|$

▷ accepts whenever $W, w \not\models \varphi$

case $\varphi = p$: **if** $p \notin w$ **then accept**

case $\varphi = \neg\psi$: $mc_{\mathsf{yes}}(W, w, \psi)$

case $\varphi = (\psi_1 \vee \psi_2)$: **(∀)** $mc_{\mathsf{no}}(W, w, \psi_1)$ **and** $mc_{\mathsf{no}}(W, w, \psi_2)$

case $\varphi = K_a\psi$: **(∃)** $u \in W$; **(∀)** $access_{\mathsf{yes}}(w, u, \pi_a)$ **and** $mc_{\mathsf{no}}(W, u, \psi)$

case $\varphi = C_G\psi$:

- **(∃)** $u \in W$
- **(∀)** $access^*_{\mathsf{yes}}(W, w, u, \bigcup_{a\in G} \pi_a, 2^{\#AP})$ **and** $mc_{\mathsf{no}}(W, u, \psi)$

case $\varphi = \langle\gamma\rangle\chi$:

- **(∀)** $W' \subseteq W \mid w \in W'$
- **(∃)** $exec_{\mathsf{no}}(W, w, W', \gamma)$ **or** $mc_{\mathsf{no}}(W', w, \chi)$

proc $stable_{\mathsf{no}}(W, W', \pi)$ $|\pi| + 1$

▷ accepts whenever $\widetilde{pre}_{\pi,W}(post_{\pi,W}(W')) \not\subseteq W'$

(∃) $(w, v, u) \in W' \times W \times W$

if $u \in W'$ **then reject else (∀)** $access_{\mathsf{yes}}(w, v, \pi)$ **and** $access_{\mathsf{yes}}(u, v, \pi)$

proc $exec_{\mathsf{no}}(W, w, W', \gamma)$ $|AP| + |\gamma|$

▷ accepts whenever $(W, w) \not\rightsquigarrow (W', w)$

- **case** $\gamma = \varphi!$:
  - (∃) $(u, v) \in W' \times (W \backslash W')$
  - (∃) $mc_{\mathsf{no}}(W, u, \varphi)$ **or** $mc_{\mathsf{yes}}(W, v, \varphi)$
- **case** $\gamma = \bullet!$: **reject**
- **case** $\gamma = \bullet!_G$:
  - (∀) $(W_a)_{a \in G} \mid$ for all $a \in G, W_a \subseteq W, w \in W_a$
  - **if** $\bigcap_{a \in G} W_a \neq W'$ **then accept else** (∃) $a \in G$; $stable_{\mathsf{no}}(W, W_a, \pi_a)$
- **case** $\gamma = \gamma_1; \gamma_2$:
  - (∀) $W'' \subseteq W$ such that $W' \subseteq W''$
  - (∃) $exec_{\mathsf{no}}(W, w, W'', \gamma_1)$ **or** $exec_{\mathsf{no}}(W'', w, W', \gamma_2)$
- **case** $\gamma = \gamma_1 \cup \gamma_2$: (∀) $k \in \{1, 2\}$; $exec_{\mathsf{no}}(W, w, W', \gamma_k)$
- **case** $\gamma = \varphi?$: **if** $W' \neq W$ **then accept else** $mc_{\mathsf{no}}(W, w, \varphi)$
- **case** $\gamma = \gamma'^*$: $exec^*_{\mathsf{no}}(W, w, W', \gamma', 2^{\#AP} - 1)$

proc $exec^*_{\mathsf{no}}(W, w, W', \gamma, i)$ $|AP| + |\gamma| + \log_2(i)$

▷ accepts whenever $(W, w) \not\rightsquigarrow^j (W', w)$ for all $j \leq i$

- **case** $i = 0$: **if** $W \neq W'$ **then accept else** $exec_{\mathsf{no}}(W, w, W', \gamma)$;
- **case** $i \geq 2$:
  - (∀) W'' s.th. $W' \subseteq W'' \subseteq W$
  - (∃) $exec^*_{\mathsf{no}}(W, w, W'', \gamma, i/2)$ **or** $exec^*_{\mathsf{no}}(W'', w, W', \gamma, i/2)$

D.4 Non symbolic model checking

proc $mc_{\mathsf{no}}(W, w, \varphi)$ $|AP| + |\varphi|$

▷ accepts whenever $W, w \not\models \varphi$

match φ **with**

- **case** $\varphi = p$: **if** $p \in V(w)$ **then reject else accept**
- **case** $\varphi = \neg\psi$: $mc_{\mathsf{yes}}(W, w, \psi)$
- **case** $\varphi = (\psi_1 \vee \psi_2)$: (∀) $mc_{\mathsf{no}}(W, w, \psi_1)$ **and** $mc_{\mathsf{no}}(W, w, \psi_2)$
- **case** $\varphi = K_a\psi$: (∃) $u \in W$; **if** $(w, u) \in R_a$ **then** $mc_{\mathsf{no}}(W, u, \psi)$
- **case** $\varphi = C_G\psi$:
  - (∃) $u \in W$
  - (∀) $access^*_{\mathsf{yes}}(W, w, u, \bigcup_{a \in G} R_a, 2^{\log_2(|W|)+1})$ **and** $mc_{\mathsf{no}}(W, u, \psi)$
- **case** $\varphi = \langle\gamma\rangle\chi$:
  - (∀) $W' \subseteq W \mid w \in W'$
  - (∃) $exec_{\mathsf{no}}(W, w, W', \gamma)$ **or** $mc_{\mathsf{no}}(W', w, \chi)$

proc $access^*_{\mathsf{no}}(W, w, u, R, i)$ $|AP| + \log_2(i)$

$\triangleright$ accepts if $w \neq u$ and there do not exist $v_1, \ldots, v_k \in W_L$ with $k < i$ such that $w \xrightarrow{\pi} v_1 \ldots \xrightarrow{R} v_{i-1} \xrightarrow{R} u$ (for $i > 1$)

case $i = 1$: **if** $u = w$ or $(w, u) \in R$ **then reject else accept**

case $i \geq 2$: **(∀)** $v \in W$; **(∃)** $access^*_{\mathsf{no}}(W, w, v, \pi, i/2)$ **or** $access^*_{\mathsf{no}}(W, v, u, \pi, i/2)$

---

proc $exec_{\mathsf{no}}(W, w, W', \gamma)$ $|AP| + |\gamma|$

$\triangleright$ accepts whenever $(W, w) \not\rightsquigarrow (W', w)$

match γ **with**

case $\gamma = \varphi!$:

(∃) $(u, v) \in W' \times (W \backslash W')$; **(∃)** $mc_{\mathsf{no}}(W, u, \varphi)$ **or** $mc_{\mathsf{yes}}(W, v, \varphi)$

case $\gamma = \bullet!$: **reject**

case $\gamma = \bullet!_G$:

(∀) $(W_a)_{a \in G}$ | for all $a \in G, W_a \subseteq W, w \in W_a$

if $\bigcap_{a \in G} W_a \neq W'$ **then accept else (∃)** $a \in G$; $stable_{\mathsf{no}}(W, W_a, R_a)$

case $\gamma = \gamma_1; \gamma_2$:

(∀) $W'' \subseteq W$ such that $W' \subseteq W''$

(∃) $exec_{\mathsf{no}}(W, w, W'', \gamma_1)$ **or** $exec_{\mathsf{no}}(W'', w, W', \gamma_2)$

case $\gamma = \gamma_1 \cup \gamma_2$: **(∀)** $k \in \{1, 2\}$; $exec_{\mathsf{no}}(W, w, W', \gamma_k)$

case $\gamma = \varphi?$: **(∃) if** $W' \neq W$ **then accept or** $mc_{\mathsf{no}}(W, w, \varphi)$

case $\gamma = \gamma'^*$: **(∀)** $i \in \{0, ..., 2^{\log_2(|W|)+1}\}$; $exec^*_{\mathsf{no}}(W, w, W', \gamma', i)$

---

proc $exec^*_{\mathsf{no}}(W, w, W', \gamma, i)$ $|AP| + |\gamma| + \log_2(i)$

$\triangleright$ accepts whenever $(W, w) \not\rightsquigarrow^i (W', w)$

case $i = 0$: **if** $W = W'$ **then reject else** $exec_{\mathsf{no}}(W, w, W', \gamma)$;

case $i \geq 2$:

(∀) W'' s.th. $W' \subseteq W'' \subseteq W$

(∃) $exec^*_{\mathsf{no}}(W, w, W'', \gamma, i/2)$ **or** $exec^*_{\mathsf{no}}(W'', w, W', \gamma, i/2)$

---

proc $stable_{\mathsf{no}}(W, W', R)$ $|\pi| + 1$

$\triangleright$ accepts whenever $\widetilde{pre}_{R,W}(post_{R,W}(W')) \not\subseteq W'$

(∃) $(w, v, u) \in W' \times W \times W$

if $((w, v) \in R$ and $(u, v) \in R$ implies $u \in W')$ **then reject else accept**

Appendix E

Proofs of Chapter 5

E.1 Proof of Proposition 15

We prove the following more general Lemma:

Lemma 6. *Let $\varphi \in PAPL$, γ be a protocol and $\mathfrak{M}$ be a symbolic model. Let AP' be such that $AP(\varphi) \subseteq AP'$. Then:*

$H(\varphi)$: for every $U \subseteq \mathscr{U}$ and $u \in U, U, u \models \varphi$ iff $U'_u, u' \models \varphi$.

$H(\gamma)$: for every $U \subseteq \mathscr{U}$, $W \subseteq U'_u$, and $u \in U$, $(U'_u, u') \overset{\gamma}{\rightsquigarrow} (W, u')$ iff there exists T such that $(U, u) \overset{\gamma}{\rightsquigarrow} (T, u)$ and $T'_u = W$.

where $U'_u = \left\{v' \mid v \in U \text{ and for all } p \in AP \setminus AP', p \in v \text{ iff } p \in u\right\}$ with $v' = v \cap AP'$.

We show $H(\varphi)$ and $H(\gamma)$ by mutual induction over φ and γ.

- $\underline{\varphi = p}$: clearly $U, u \models p$ iff $U'_u, u' \models p$.
- $\underline{\varphi = \neg\psi}$:

  $U, u \models \varphi$ iff $U, u \not\models \psi$
  iff $U'_u, u' \not\models \psi$ (by $H(\psi)$)
  iff $U'_u, u' \models \varphi$.

- $\underline{\varphi = \varphi_1 \vee \varphi_2}$:

  $U, u \models \varphi$ iff $U, u \models \varphi_1$ or $U, u \models \varphi_2$
  iff $U'_u, u' \models \varphi_1$ or $U'_u, u' \models \varphi_2$ (by using $H(\varphi_1)$ and $H(\varphi_2)$)
  iff $U'_u, u' \models \varphi$.

- $\underline{\varphi = K_a\psi}$:

  $U, u \models \varphi$ iff for all $v \in U$ such that $u \xrightarrow{\pi_a} v, U, v \models \psi$.
  iff for all $v \in U$ such that $u \xrightarrow{\pi_a} v, U'_v, v' \models \psi$ (by $H(\psi)$).
  iff for all $v \in U$ such that $u \xrightarrow{\pi_a} v, U'_u, v' \models \psi$ (because $U'_u = U'_v$).
  iff for all $w \in U'_u$ such that $u' \xrightarrow{\pi_a} w, U'_u, w \models \psi$ (because $AP(\pi) \subseteq AP'$).
  iff $U'_u, u' \models \varphi$.

- $\underline{\varphi = C_G\psi}$:

  $U, u \models \varphi$ iff for all $(a_1, \ldots, a_m) \in G, U, u \models K_{a_1} \ldots K_{a_m}\psi$
  iff for all $(a_1, \ldots, a_m) \in G, U'_u, u' \models K_{a_1} \ldots K_{a_m}\psi$ (by repeating m times the previous reasoning and using $H(\psi)$).
  iff $U'_u, u' \models \varphi$.

- $\underline{\varphi = \langle\gamma\rangle\psi}$:

  $U, u \models \varphi$ iff there exists $T \subseteq U$ such that $(U, u) \overset{\gamma}{\rightsquigarrow} (T, u)$ and $T, u \models \psi$.
  iff there exists $T \subseteq U$ such that $(U, u) \overset{\gamma}{\rightsquigarrow} (T, u)$ and $T'_u, u' \models \psi$ (by $H(\psi)$).
  iff there exists $W \subseteq U'_u$ such that $(U'_u, u) \overset{\gamma}{\rightsquigarrow} (W, u)$ and $W, u' \models \psi$ (by $H(\gamma)$, take $W = T'_u$).
  iff $U'_u, u' \models \varphi$.

- $\underline{\gamma = \varphi?}$:

  $(U'_u, u') \overset{\varphi?}{\rightsquigarrow} (W, u')$ iff $U'_u = W$ and $U'_u, u' \models \varphi$
  iff $U'_u = W$ and $U, u \models \varphi$ (by $H(\varphi)$).
  iff there exists T such that $T = U$ and $T'_u = W$ and $U, u \models \varphi$.
  iff there exists T such that $(U, u) \overset{\varphi?}{\rightsquigarrow} (T, u)$ and $W = T'_u$.

- $\underline{\gamma = \varphi!}$:

  - If $(U'_u, u') \overset{\varphi!}{\rightsquigarrow} (W, u')$ then we have $w \in W$ iff $U'_u, w \models \varphi$. Let $T = \{w \cup (u \setminus u'), w \in W\}$. Intuitively, T is the extension of W by taking values of u outside AP'. We obtain directly that $T'_u = W$. Furthermore, $v \in T$ iff $v' \in T'_u$, iff $U'_u, v' \models \varphi$, iff $U, v \models \varphi$ (by $H(\varphi)$). We conclude that there exists T such that $(U, u) \overset{\varphi!}{\rightsquigarrow} (T, u)$ and $W = T'_u$.
  - If there exists T such that $(U, u) \overset{\varphi!}{\rightsquigarrow} (T, u)$ and $W = T'_u$ then $w \in W$ iff there exists $v \in T$ such that $v' = w$. For such v, we have by definition of T, $U, v \models \varphi$, so $U'_u, v' \models \varphi$. Thus $w \in W$ iff $U'_u, w \models \varphi$. Therefore, $(U'_u, u') \overset{\varphi!}{\rightsquigarrow} (W, u')$.

- $\underline{\gamma = \gamma_1 \cup \gamma_2}$:

  $(U'_u, u') \overset{\gamma_1 \cup \gamma_2}{\rightsquigarrow} (W, u')$ iff $(U'_u, u') \overset{\gamma_1}{\rightsquigarrow} (W, u')$ or $(U'_u, u') \overset{\gamma_2}{\rightsquigarrow} (W, u')$.
  iff there exists T such that $(U, u) \overset{\gamma_1}{\rightsquigarrow} (T, u)$ and $W = T'_u$ or there exists T such that $(U, u) \overset{\gamma_2}{\rightsquigarrow} (T, u)$ and $W = T'_u$ (by $H(\gamma_1)$ and $H(\gamma_2)$).
  iff there exists T such that $(U, u) \overset{\gamma_1}{\rightsquigarrow} (T, u)$ or $(U, u) \overset{\gamma_2}{\rightsquigarrow} (T, u)$, and $W = T'_u$.
  iff there exists T such that $(U, u) \overset{\gamma_1 \cup \gamma_2}{\rightsquigarrow} (T, u)$ and $W = T'_u$.

- $\underline{\gamma = \gamma_1;\gamma_2}$:

  $(U'_u, u') \rightsquigarrow^{\gamma_1;\gamma_2} (W, u')$ iff there exists W_0 such that $(U'_u, u') \rightsquigarrow^{\gamma_1} (W_0, u') \rightsquigarrow^{\gamma_2} (W, u')$
  iff there exists T_0 such that $(U, u) \rightsquigarrow^{\gamma_1} (T_0, u)$ and $(T_{0u}', u') \rightsquigarrow^{\gamma_2} (W, u')$ (by $H(\gamma_1)$).
  iff there exists T_0, T such that $(U, u) \rightsquigarrow^{\gamma_1} (T_0, u) \rightsquigarrow^{\gamma_2} (T, u)$ and $W = T'_u$ (by $H(\gamma_2)$).
  iff there exists T such that $(U, u) \rightsquigarrow^{\gamma_1;\gamma_2} (T, u)$ and $W = T'_u$.

- $\underline{\gamma = \gamma'^*}$:

  $(U'_u, u') \rightsquigarrow^{\gamma'^*} (W, u')$ iff there exists $k \geq 0$ such that $(U'_u, u') \rightsquigarrow^{\gamma'^k} (W, u')$
  iff there exists T such that $(U, u) \rightsquigarrow^{\gamma'^k} (T, u)$ and $W = T'_u$ (by repeating case $\gamma = \gamma_1;\gamma_2$).
  iff there exists T such that $(U, u) \rightsquigarrow^{\gamma'^*} (T, u)$ and $W = T'_u$.

- $\underline{\gamma = \bullet!}$: this case amounts to $\gamma = \bullet!_{\{f\}}$ with f an omniscient agent, meaning that $\pi_f = \top?$. We apply the case $\gamma = \bullet!_G$ to conclude.

- $\underline{\gamma = \bullet!_G}$:

  – Let T be such that $(U, u) \rightsquigarrow^{\langle \bullet!_G \rangle} (T, u)$ and $T'_u = W$ then there exists formulas $(\psi_a)_{a \in Ag}$ such that $(U, u) \rightsquigarrow^{\bigwedge_{a \in G} K_a \psi_a !} (T, u)$. We cannot apply the induction hypothesis here because it may not be true that $AP(\psi_a) \subseteq AP'$. For any agent $a \in G$, let $\chi_a = \bigvee_{v \in post_{\pi_a, U'_u}(T'_u)} \left(\bigwedge_{p \in v \cap AP'} p \wedge \bigwedge_{p \in (AP \setminus v) \cap AP'} \neg p \right)$. Intuitively, χ_a describes exactly the π_a-successors of T'_u that are in U'_u. Notice that here all successors of elements in T'_u are in U'_u, so χ_a actually describes the successors of T'_u by π_a. Let W_0 be such that $(U'_u, u) \rightsquigarrow^{\bigwedge_{a \in G} K_a \chi_a !} (W_0, u)$. Let us show that $W_0 = T'_u$:

    * If $v_0 \in T'_u$ then each π_a successor of v_0 is in $post_{\pi_a, U'_u}(T'_u)$ so $U'_u, v_0 \models K_a \chi_a$. So $v_0 \in W_0$.
    * If $v_0 \in W_0$, let $v \in U$ defined by $v = v_0 \cup (u \setminus u')$. For any w such that $v \xrightarrow{\pi_a} w$, we have $v_0 \xrightarrow{\pi_a} w'$ so $w' \in post_{\pi_a, U'_u}(T'_u)$. Then there exists $v_1 \in T$ such that $v'_1 \xrightarrow{\pi_a} w'$, so $v_1 \xrightarrow{\pi_a} w$. Furthermore, because $v_1 \in T$, we have $U, v_1 \models K_a \psi_a$, so $U, w \models \psi_a$. We then obtain that $W, v \models K_a \psi_a$ for every agent $a \in G$, so $v \in T$.

    then $(U'_u, u) \rightsquigarrow^{\langle \bullet!_G \rangle} (T'_u, u)$ and then $(U'_u, u) \rightsquigarrow^{\langle \bullet!_G \rangle} (W, u)$.

  – If $(U'_u, u) \rightsquigarrow^{\langle \bullet!_G \rangle} (W, u)$ then there exist formulas $(\psi_a)_{a \in G}$ such that $(U'_u, u) \rightsquigarrow^{\bigwedge_{a \in G} K_a \psi_a}$ (W, u). For any agent $a \in G$, we take the same χ_a than before. Let W_0 be such that $(U'_u, u) \rightsquigarrow^{\bigwedge_{a \in G} K_a \chi_a !} (W_0, u)$. Let us show that $W_0 = T'_u$:

* If $v \in W$ then as before, $U'_u, v \models K_a\psi_a$ so $v \in W_0$.
* If $v \in W_0$ then for all w such that $v \xrightarrow{\pi_a} w$, there exists $v_1 \in W$ such that $v_1 \xrightarrow{\pi_a} w$, so $U'_u, v_1 \models K_a\psi_a$, so $U'_u \models \psi_a$, so $U'_u, v \models K_a\psi_a$ and so $v \in W$.

Now we have $AP(\chi_a) \subseteq AP'$ for all agents $a \in G$ so we conclude by case $\gamma = \varphi!$ that there exists T such that $(U, u) \xrightsquigarrow{\bigwedge_{a\in G} K_a\chi_a!} (T, u)$ and $T'_u = W$.

E.2 Proof of Proposition 16

We first prove that for any Agent a and any two sets of valuations $U \subseteq W \subseteq \mathscr{U}$, the two following statements are equivalent:

1. There exists an arbitrary-free formula ψ such that for any valuation $w \in W \cap U$, $(W, w) \xrightsquigarrow{K_a\psi!} (U, w)$;
2. $U = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$.

We prove first prove that 1. implies 2.

Suppose that there exists an arbitrary-free formula ψ such that for any valuation $w \in W \cap U$, $(W, w) \xrightsquigarrow{K_a\psi!} (U, w)$. We prove $U = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$ by double inclusion.

- First, notice that by definition of $post_{\pi_a,W}(U)$, any element u of U has all its π_a-successors in $post_{\pi_a,W}(U)$, so that $u \in \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$. Therefore $U \subseteq \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$.
- Second, let $u \in \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$. By definition of $\widetilde{pre}_{\pi_a,W}$, for any v π_a-successor of u, we have $v \in post_{\pi_a,W}(U)$. As a consequence, for each of those v, there exists $u' \in U$ such that $u' \xrightarrow{\pi_a} v$. Since by assumption, for any valuation $w' \in W \cap U$, $(W, w') \xrightarrow{K_a\psi!} (U, w')$, we have in particular $W, u' \models K_a\psi$. Since $u' \xrightarrow{\pi_a} v$, we also have $W, v \models \psi$. Because v is an arbitrary π_a-successor of u, we can state that $W, u \models K_a\psi$, so that u survives after the announcement $K_a\psi!$, that is $u \in U$. We have shown that $U \supseteq \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(W))$, which achieves the proof of that 1. implies 2.

We now show that 2. implies 1.

Suppose $U = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(W))$. We show that there exists an arbitrary-free formula χ_a such that $(W, w) \xrightsquigarrow{K_a\chi_a!} (U, w)$ for any $w \in W \cap U$. We prove that the arbitrary-free formula χ_a defined by

$$\chi_a \stackrel{\text{def}}{=} \bigvee_{v \in post_{\pi_a,W}(U)} \left(\bigwedge_{p\in v} p \wedge \bigwedge_{p\in AP\setminus v} \neg p \right) \text{ is a good candidate.}$$

Clearly, for any world v, we have $W, v \models \chi_a$ iff $v \in post_{\pi_a,W}(U)$. We have $w \in U$ iff $w \in \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$, iff for all π_a-successor $v \in W$ of w, we have $v \in post_{\pi_a,W}(U)$, iff for all $v \in W$, $w \xrightarrow{\pi_a} v$ implies $W, v \models \chi_a$ iff $W, w \models K_a\chi_a$. We conclude $(W, w) \xrightarrow{K_a\chi_a!} (U, w)$.

We have just shown that the existence of a $K_a\chi_a$-formula to announce is equivalent to the existence of a set of valuations U such that $U = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U))$.

With the semantics of $\langle \bullet!_G \rangle \varphi$, justifying the existence of $K_a\psi_a$-formulas for each $a \in G$ that can be announced is equivalent to justifying the existence of sets U_a such that $U_a = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U_a))$. Furthermore, $(W, u) \xrightarrow{(\bigwedge_{a\in G} K_a\psi_a)!} (\bigcap_{a\in G} U_a, u)$. To conclude, searching for a formula of the form $\bigwedge_{a\in G} K_a\psi_a!$ with $W, u \models \langle \bigwedge_{a\in G} K_a\psi_a! \rangle \varphi$ is equivalent to searching for U_a such that $U_a = \widetilde{pre}_{\pi_a,W}(post_{\pi_a,W}(U_a))$ and $\bigcap_{a\in G} U_a, u \models \varphi$.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre: Complexité théorique du raisonnement en logique épistémique dynamique et étude d'une approche symbolique

Mot clés : système multi-agents, vérification de modèles, logique épistémique, complexité de calcul, modèle symbolique, planification épistémique.

Résumé : Nous étudions la complexité théorique de tâches de raisonnement mettant en jeu la connaissance des agents dans les systèmes multi-agents. Nous considérons la logique épistémique dynamique (DEL) comme une façon naturelle d'exprimer la connaissance, qui permet d'exprimer la connaissance d'ordre supérieur des agents et des actions dynamiques partiellement observées. Nous montrons des résultats de complexité algorithmique pour la vérification de modèles et la satisfiabilité de formules de DEL, et définissons une approche symbolique pour ces mêmes problèmes. Nous étudions également la planification basée sur DEL ainsi que des quantifications sur certaines actions : les annonces publiques.

Title: Theoretical complexity of reasoning in dynamic epistemic logic and study of a symbolic approach.

Keywords : multi-agent system, model checking, epistemic logic, computational complexity, symbolic model, epistemic planning.

Abstract : We study the theoretical complexity of reasoning tasks involving knowledge in multi-agent systems. We consider dynamic epistemic logic (DEL) as a natural way of expressing knowledge, which allows to express nested knowledge of agents and partially observed dynamic actions. We show complexity results for model checking and satisfiability of DEL formulas, and define a symbolic approach for these problems. We also study DEL-based planning and quantification over specific actions: public announcements.