Synthesis of Interactive Reactive Systems

Rodica Bozianu

Université Paris-Est, Lab. d'Algorithmique, Complexité et Logique
Doctoral school MSTIC

Université Libre de Bruxelles,Département d'Informatique

PhD THESIS

proposed to obtain the degree of Doctor,
Speciality : Computer Science

Synthesis of Interactive Reactive Systems

Proposed by

Rodica BOZIANU CONDURACHE

Thesis presented on 12th of December in front of the jury composed by:

Mme.	Patricia Bouyer-Decitre	LSV, CNRS & ENS de Cachan	Reviewer
Mr.	Cătălin Dima	LACL, Université Paris Est, Créteil	Coordinator
Mr.	Emmanuel Filiot	Université Libre de Bruxelles	Coordinator
Mme.	Orna Kupferman	Hebrew University, Jerusalem	Reviewer
Mr.	Jean-François Raskin	Université Libre de Bruxelles	Examinator
Mr.	Daniele Varacca	LACL, Université Paris Est, Créteil	Examinator

Université Paris-Est, Lab. d'Algorithmique, Complexité et Logique
École doctorale MSTIC

Université Libre de Bruxelles,Département d'Informatique

THÈSE

présentée en vue d'obtenir le grade de Docteur,
specialité: Informatique

Synthèse des systèmes réactifs interactifs

Proposée par

Rodica BOZIANU CONDURACHE

Thèse soutenue le 12 Décembre devant le jury composé de:

Mme.	Patricia Bouyer-Decitre	LSV, CNRS & ENS de Cachan	Rapporteur
Mr.	Cătălin Dima	LACL, Université Paris Est, Créteil	Co-Directeur
Mr.	Emmanuel Filiot	Université Libre de Bruxelles	Co-Directeur
Mme.	Orna Kupferman	Hebrew University, Jerusalem	Rapporteur
Mr.	Jean-François Raskin	Université Libre de Bruxelles	Examinateur
Mr.	Daniele Varacca	LACL, Université Paris Est, Créteil	Examinateur

Doctorat réalisé en co-tutelle entre:

- **Université Paris-Est**

  L'École Doctorale Mathématiques et Sciences et Technologies de l'Information et de la Communication (MSTIC)

  Laboratoire d'Algorithmique, Complexité et Logique (LACL)

  LACL, Département d'Informatique
  Faculté des Sciences et Technologie
  61 avenue du Général de Gaulle
  94010 Créteil Cedex

et

- **Université Libre de Bruxelles**

  Département d'Informatique
  ULB - Campus de la Plaine
  1050 Bruxelles Belgium

Title: Synthesis of Interactive Reactive Systems

Summary

We study the problem of automatic synthesis of programs in multi-component architectures such that they fulfill the specifications by construction. The main goal of the thesis is to develop procedures to solve the synthesis problem that may lead to efficient implementations.

Each component has partial observation on the global state of the multi-component system. The synthesis problem is then to provide observation-based protocols for the components that have to be synthesized that ensure that specifications hold on all interactions with their environment.

The environment may be antagonist, or may have its own objectives and behave rationally. We first study the synthesis problem when the environment is presumed to be completely antagonist. For this setting, we propose a "Safraless" procedure for the synthesis of one partially informed component and an omniscient environment from KLTL$^+$specifications. It is implemented in the tool Acacia-K.

Secondly, we study the synthesis problem when the components in the environment have their own objectives and are rational. For the more relaxed setting of perfect information, we provide tight complexities for particular ω-regular objectives. Then, for the case of imperfect information, we prove that the rational synthesis problem is undecidable in general, but we gain decidability if is asked to synthesize one component against a rational omniscient environment.

Key words: synthesis, verification, reactive systems, games, KLTL

Titre: Synthèse des systèmes réactifs interactifs

Résumé

Nous étudions le problème de la synthèse automatique de programmes dans des architectures multi-composants teles qu'elles respectent les spécifications par construction. Le principal objectif de cette thèse est de développer des procédures pour résoudre le problème de synthèse qui peut conduire à des implémentations efficaces.

Chaque composant a une observation partielle sur l'état global du système multi-composants. Le problème est alors de fournir des protocoles basés sur les observations afin que les composants synthétisés assurent les spécifications pour tout le comportement de leur environnement.

L'environnement peut être antagoniste, ou peut avoir ses propres objectifs et se comporter de façon rationnelle. Nous étudions d'abord le problème de synthèse lorsque l'environnement est présumé antagoniste. Pour ce contexte, nous proposons une procédure "Safraless" pour la synthèse d'un composant partiellement informé et un environnement omniscient à partir de spécifications KLTL^{+}. Elle est implémentée dans l'outil Acacia-K.

Ensuite, nous étudions le problème de synthèse lorsque les composants de l'environnement ont leurs propres objectifs et sont rationnels. Pour le cadre plus simple de l'information parfaite, nous fournissons des complexités serrées pour des objectifs ω-réguliers particuliers. Pour le cas de l'information imparfaite, nous prouvons que le problème de la synthèse rationnelle est indécidable en général, mais nous regagnons la décidabilité si on demande à synthétiser un composant avec observation partielle contre un environnement multi-composante, omniscient et rationnel.

Mots clés: synthèse, vérification, systèmes réactifs, jeux, logiques

Résumé détaillé

Motivation

Les systèmes réactifs sont des systèmes sans terminaison qui interagissent continuellement avec leur environnement. Ils apparaissent à la fois comme matériel et logiciel, et font habituellement partie des systèmes pour lesquelles la sécurité est critiques, par exemple, microprocesseurs, systèmes informatiques responsables de la gestion du trafic aérien, contrôle des centrales nucléaires, ou programmes de surveillance pour les dispositifs médicaux. Il est donc crucial de garantir leur comportement correct. Il existe des modalités différentes pour s'assurer que ces systèmes ont un comportement correct.

Vérification des Programmes La vérification des programmes est un domaine d'informatique qui, dans le processus de conception des systèmes, aide a détecter plus facilement à la fois les erreurs dans la partie logiciel et dans le matériel. L'une des principales réalisations dans la vérification des systèmes est la théorie de *model-checking* [27], qui permet de vérifier automatiquement la correction du système.

Il s'applique aux *modèles d'interaction* (ou systèmes de transitions) qui modélisent l'interaction entre un programme et son environnement ainsi que aux spécifications exprimé comme des *formules temporelles* [60] qui décrivant les comportements admissibles de programme. Le problème de model-checking consiste en répondre à la question *"Le programme satisfait-il les spécifications?"*. Dans le cas d'une réponse négative, les procédures de model-checking fournissent un contrexemple qui consiste en une interaction entre le système et l'environnement ne satisfaisant pas les spécifications. Ensuite, en utilisent le contrexemple, il faut réparer l'erreur dans le système, ou même dans les spécifications et procéder dès le début avec le processus de model-checking.

Actuellement, les techniques de model-checking représente une partie importante dans le processus de conception des systèmes dans entreprises comme Intel et IBM. Cependant, la conception des systèmes en utilisent le model-checking et le raffinement guidé peut prendre beaucoup de temps et dépend de la compétence des programmeurs. Autrement dit, le processus de vérification suivi d'une correction d'erreurs peut être répété plusieurs fois avant d'obtenir un programme correct.

Synthèse des Programmes Étant donné une spécification, le but de la *synthèse* est de synthétiser automatiquement un programme ayant comportement défini par la spécification. Par conséquent, les contraintes ne doivent pas être vérifiées après, ce qui permet au concepteur de se concentrer sur la définition de spécifications de haut niveau, plutôt que de concevoir des modèles computationnelles complexes des systèmes.

Le problème de synthèse a été introduit par Church [26] pour les circuits. Elle exige que, étant donné une spécification logique $S(I, O)$ entre les signaux d'entrée I et les signaux de sortie O, déterminer s'il existe un circuit qui, pour toutes les entrées reçues, il émet des signaux de telle sorte que la spécification est satisfaite.

Un des principaux défis de la synthèse des systèmes réactifs est de satisfaire la spécification quel que soit le comportement de son environnement. Par exemple, prenons un système informatique qui contrôle un ascenseur. Il a des actions pour faire monter ou descendre l'ascenseur, ouvrir et fermer les portes. À chaque étage il y a des boutons pour appeler l'ascenseur et à l'intérieur de l'ascenseur il y a quelques boutons qui servent à demander le déplacement de l'ascenseur a un certain étage. La personne qui appuie les boutons représente l'environnement du contrôleur de l'ascenseur. Ensuite, une spécification à satisfaire par l'ascenseur est la suivante. L'ascenseur doit éventuellement s'arrêter et ouvrir la porte à un étage, à chaque fois que il est appelé à cet étage. En outre, à chaque fois que la personne à l'intérieur de l'ascenseur demande à l'ascenseur d'aller à un certain étage, il devrait finalement l'atteindre. A noter que le comportement de la personne dans l'ascenseur n'est pas connu à l'avance et n'est pas restreint. Par conséquent, il peut pousser n'importe quels boutons et les spécifications doivent être satisfaites pour tout comportement possible de la personne.

L'environnement incontrôlable entraîne habituellement des problèmes de décision plus difficiles en termes de calcul, par rapport à la vérification des systèmes. Par exemple, le problème de model-checking pour des propriétés exprimées comme des formules de logique temporelle linéaire (LTL) est PSpace-complète [83] alors que le problème de la synthèse à partir des spécifications LTL est 2ExpTime-complète [74].

Le problème de synthèse des systèmes réactifs à partir des spécifications LTL a été introduite et initialement étudiée par Pnueli et Rosner [74] et aussi par Abadi, Lamport et Wolper [2]. Néanmoins, la solution proposée implique des automates nondéterministes qui doivent être déterminisé en utilisant la construction complexe de Safra[79], qui est résistante à des implementations efficaces [85]. Piterman [73], Muller et Schupp [70] ont proposé des optimisations de la construction de Safra, qui, malheureusement, continuent à ne pas conduire à des implémentations efficaces. La principale raison de la résistance à des implémentations de ces constructions est que ils ne peuvent pas être implémenté symboliquement et par conséquent il faut construire explicitement l'ensemble des états des automates.

Récemment, ont été développés des procédures qui évite la construction de

Safra (appelé *"Safraless"*). Une première solution a été proposé par Kupferman et Vardi dans [55] et utilise des automates coBüchi universelles pour caractériser les solutions du problème de synthèse. Cette approche a suscité beaucoup d'intérêt dans le développement des procédures faisables pour résoudre le problème de la synthèse. Depuis, ont été définis plusiours autres procedures "Safraless" [55, 81, 41, 36]. Dans [81, 36], il est démontré que la synthèse LTL se réduit aux tester le vide d'un automate d'arbres coBüchi universel, qui à son tour peut être réduit à résoudre un jeu de sûreté. La structure du jeux de sûreté peut être exploité et définir un algorithm symbolique, basé sur des antichaînes, pour résoudre le jeux [36]. Lily[49], ANZU[50], RATSY[11] et Acacia+[12] sont quelques exemples d'outils qui implémente efficacement des algorithmes pour résoudre le problème de synthèse LTL.

Systèmes Multi-Composants Dans les travaux précédemment présentés, le système à synthétiser est présumé avoir des informations parfaites sur l'état de l'environnement. Néanmoins, dans nombreux situations pratiques, cette supposition n'est pas réaliste car nombreuses systèmes ont des composants multiples. Dans le cas des systèmes distribué, typiquement, chaque composant a ses variables privées, qui ne sont pas visibles de l'extérieur. Par conséquent, les composants doivent agir sur la base des données locales et doivent se coordonner pour assurer certaines propriétés globales du système qui est exécuté dans un certain environnement.

L'interaction entre les composants d'un système et l'environnement est modélisée en utilisent des *modèles d'interaction* (ou systèmes de transitions). Chaque état d'un modèle d'interaction représente la configuration globale du système distribué (environnement compris) et les transition entre les états est fait en function de les actions des composants (environnement inclus). Alors, chaque composant a une observation partielle sur les états du modèle d'interaction, en function de ses données locales. Les composants ne voient pas quelles actions sont jouées par les autres, mais ils obtiennent une certaine observation sur les états dans lesquels le modèle d'interaction peut être. Une exécution du modèle d'interaction, du point de vue d'un composant du système, est alors une sequence infini de ses propres actions et des observations.

Un *protocole* (ou comportement, ou *stratégie*) d'un composant consiste à indiquer les actions à jouer dans le modèle d'interaction, selon les observations passées qu'il a reçues. Par conséquent, la synthèse d'un (ensemble) des composant(s) est équivalent à trouver une (set de) stratégie(s) pour le(s) composants à synthétiser. Quand est demandé de synthétiser un ensemble des composants, les autres composants du système distribué sont considérés comme faisant partie de l'environnement.

Les Logiques Temporales Épistémiques [44] sont des logiques formatées pour raisonner sur des situations multi-agent. En plus de la description formelle de l'ordre

temporelle des événements, ils permettent aussi le raisonnement sur la connaissance des composants. On peut formuler des propriétés comme "si le Processus i sait que la transaction sera abandonnée, il devrait annuler sa contribution locale et terminer immédiatement"[90]. L'importance de la connaissance des composants est mise en évidence par différents problèmes. L'un d'entre eux est le problème d'attaque coordonnée. Il s'agit de deux généraux qui tentent de coordonner une attaque en communiquant par un messager douteux. Ils doivent choisir le moment du attaque et se mettre d'accord pour attaquer en même temps. De plus, chaque général doit savoir que l'autre général sait qu'ils sont d'accord sur le plan d'attaque.

Les logiques temporelles épistémiques sont des extensions de la logique temporelle avec des opérateurs de connaissance K_i pour chaque agent. Une formule $K_i\varphi$ intuitivement dit que l'agent i sait que la propriété φ est vrai. Les logiques épistémiques ont été utilisées avec succès pour la vérification des diverses systèmes distribués dans lesquelles la connaissance des agents est essentielle dans la correction des spécifications du système.

Synthèse avec des objectifs temporelles épistémiques et environnement antagoniste Vardi et van der Meyden [89] ont étudié le problème de synthèse à partir de spécifications exprimées comme des formules de logique temporelle épistémique. Ils définissent le problème de synthèse dans un environnement multi-agent, pour les spécifications écrites en LTL étendu avec les opérateurs de connaissances K_i pour chaque agent (KLTL).

L'objectif du problème de synthèse KLTL est de générer automatiquement des protocoles (si ils existent) pour les systèmes (multi-agent) qui indique les actions à prendre, en function de les histoires finies, tel que pour toutes actions de l'environnement, toutes les exécutions (concrètes) infinies résultant compatibles avec cette stratégie satisfont la spécification KLTL. Grâce à [75], ce problème est indécidable déjà pour objectifs LTL quand le système est constitué de deux composants partiellement informés qui agissent contre l'environnement. D'autre part, pour un système avec un seul composant contre un environnement antagoniste, le problème est montré 2Exptime-complet [89], par réduction au problème de vide des automates d'arbres Büchi alternantes. Cette construction théoriquement élégante est, malheureusement, difficile à implémenter et à optimiser, car elle est basée sur des opérations complexes de type Safra sur les automates (construction Muller-Schupp).

Synthèse avec un environnement multi-composant et rationnel Plus récemment, Kupferman et. al [38, 53] ont introduit le problème de la synthèse contre un environnement multi-composants rationnel. Le problème de la synthèse rationnelle considère que chaque composant du modèle d'interaction a son propre objectif, autre que faire les autres échouer. Ils sont rationnel dans le sens que ils visent d'abord la satisfaction

de leurs propres objectifs, puis essaient de nuire aux autres.

Le problème du synthèse rationnelle a été introduit dans le cadre plus simple de l'information parfaite pour chaque composant dans le modèle d'interaction (environnement et les processus à synthétiser). C'est-à-dire, ils savent quel est l'état (globale) exacte du système. L'objectif de chaque processus est exprimé comme formules LTL et leur rationalité est modélisé en utilisent des équilibres de Nash, des stratégies dominantes, etc.

Le problème a été étudié dans deux cas différentes. Dans le cas de *synthèse rationnelle non-coopérative*[53], étant donné le fait que les composants du modèle d'interaction sont rationnels, ils peuvent jouer n'importe quel profil de stratégie qui est en équilibre. En conséquence, on demande de synthétiser un protocole pour un processus tel que il assure son objective contre n'importe quel comportement rationnel de les composants de l'environnement. D'autre part, la *synthèse rationnelle coopérative* assume que l'environnement est plus coopératif et accepte de adhérer à un profil de stratégies que est un équilibre. Alors, le problème est de trouver un tel profil de stratégies où l'objectif de protagoniste est satisfait.

La principale contribution des papiers originaux est de mettre en avant et de motiver les définitions ci-dessus. Les seules résultats de complexité donnés sont les suivants: les problèmes de synthèse rationnelle coopérative et non-coopérative sont 2ExpTime-complets pour objectifs donnés comme formules LTL. Ce résultat correspond exactement à la synthèse LTL avec information parfaite et un environnement antagoniste avec un seul composant. La borne inférieure est due au fait que le problème de synthèse LTL est un cas particulier du problème de synthèse rationnelle. La limite supérieure est obtenue par réduction au problème de model-checking pour des formules dans un fragment de Strategy Logic (SL[NG]) [65].

Objectifs de cette Thèse

L'objectif de cette thèse est d'étudier le problème de synthèse des systèmes réactifs interactifs dans le cas antagoniste ainsi que dans le cas rationnel. Nous considérons également différents types d'observations pour les processus. Autrement dit, nous étudions les deux problèmes quand les composants (processus) à synthétiser et les composantes de l'environnement ont l'information parfaite ou partielle.

Plus précisément, nous sommes intéressés de trouver des procédures "Safraless" pour résoudre le problème de synthèse et qui peut ramener à des implémentations efficaces. D'abord, pour le cas de la Synthèse KLTL avec un environnement antagoniste à un seul composant, notre objectif est de utiliser les automates coBüchi universelles et développer une procedure similaire à celle de [36].

Ensuite, dans le cas de Synthese Rationelle, la solution propose dans [37, 53] pour

le cas d'information parfaite pour tous composants est par réduction au problème de model-checking d'un fragment de Strategy Logic. Cela, est ensuite résolu par réduction au problème de tester le vide des automates d'arbres alternantes, qui implique l'utilisation des constructions similaires à construction de Safra. Notre objectif est de comprendre finement les complexités du problème de synthèse rationnelle et comment manipuler algorithmiquement le jeu sous-jasent. Ensuite, nous visons une procédure "Safraless" pour résoudre le problème du synthèse rationnelle sous information parfaite et imparfaite.

Contributions

Définition Générale du Problème de Synthèse

Dans cette thèse nous fournissons une définition plus générale du problème de synthèse qui généralise les deux problèmes mentionnés ci-dessus. Plus précisément, nous considérons des modèles d'interaction $\mathcal{M}$ sur lequel agissent $k+1$ processus partiellement informées. L'information partielle du chaque agent modélisée par une relation d'équivalence (d'indistinguibilité) sur les états du modèle d'interaction. Chaque classe d'équivalence représente une certaine observation du l'agent, correspondent à la relation considérée. Cella capture la situation où chaque état de $\mathcal{M}$ représente une état globale d'un système distribué, et chaque processus modélise l'un de ses composants qui ne voient pas les données locales des autres composants. La transition entre les états est fait, comme dans le modèle proposé en [89], en function des actions des processus. Aussi, au cours des exécutions, chaque processus n'est informé que de ses propres actions et de ses observations sur les états.

L'ensemble $\Omega = \{0, 1, ..., k\}$ des processus est partitionné en deux ensembles. Il y a un ensemble P des processus qui représente les composants dans le système distribué à synthétiser et les processus dans l'ensemble $\Omega \setminus P$ représente l'environnement des processus à synthétiser.

L'approche générale pour résoudre le problème de synthèse est de utiliser des concepts de théorie des jeux. C'est-à-dire, chaque composante du modèle d'interaction représente un joueur dans le jeux et, alors, résoudre le problème de synthèse se réduit à trouver des stratégies pour les joueurs correspondant aux processus à synthétiser tel que ils satisfont certaines spécifications. En général, les conditions à respecter sont donnés comme des *spécifications du profil de stratégies*. Plus précisément, il est fourni un ensemble Φ de profils de stratégies (une stratégie pour chaque processus) qui sont désirables. Étant donné l'ensemble Γ_i des stratégies possibles pour Processus $i \in \Omega$, Φ est un ensemble des tuples appartenant à $\Gamma_0 \times ... \times \Gamma_k$.

Ensuite, le **problème de synthèse** demande de trouver tuples $(\sigma_i)_{i \in P}$ de stratégies pour les processus dans P tel que pour n'importe quel tuple $(\sigma_i)_{i \notin P}$ de stratégies pour les processus dans l'environnement $(\Omega \setminus P)$, le profil de stratégies résultant $\langle \sigma_0, ..., \sigma_k \rangle$ est

inclus dans la spécification Φ. Autrement dit, on considère le *problème de réalisabilité* suivant:

$$\begin{aligned}&\textsf{ENTRÉE} : \mathcal{M},\ P \subseteq \Omega,\ \Phi \subseteq \Gamma_0 \times \cdots \times \Gamma_k\\&\textsf{SORTIE} : \text{Oui ssi } \exists(\sigma_i)_{i\in P}\ \forall(\sigma_i)_{i\notin P}\ \langle \sigma_0, ..., \sigma_k\rangle \in \Phi\end{aligned}$$

Si la réponse pour le problème ci-dessus est positive, le problème de synthèse demande de fournir un témoin. Dans cette thèse, tous les tests de réalisabilité sont constructifs (fournissent un témoin) et par suite nous nous référons au problème de réalisabilité comme le problème de synthèse.

Une vue d'ensemble des résultats de décidabilité de ce problème est dans le Tableau 2.1.

Synthèse KLTL et Environnement Antagoniste

D'abord, nous traitons le *Problème de synthèse KLTL* avec environnement antagoniste défini dans [89]. Dans ce cas, les processus dans l'ensemble P ont comme objectif commun la satisfaction d'une formule KLTL φ contre un environnement complètement antagoniste (ayant l'objectif $\neg\varphi$) qui n'a qu'un seul composant. Par conséquent, la spécifications du profil de stratégies se compose de tous profils de stratégies pour lesquels l'exécution résultante satisfait φ.

Le problème de synthèse KLTL est indécidable en général et alors nous nous concentrons sur la restriction de [89] où il y a un processus partialement informé ($|P| = 1$) à synthétiser contre un environnement complètement informé. La solution proposé dans [89] est basé sur le vide des automates d'arbres alternantes de Rabin, qui est testé par appliquent d'abord des constructions de type Safra pour éliminer l'alternance et après vérifier le vide de l'automate d'arbres nondeterministe résultant.

La principale contribution concernant la synthèse KLTL avec environnement antagoniste est de définir et implémenter une procédure Safraless pour le fragment positif de KLTL (KLTL$^+$), c'est à dire, formules KLTL où l'opérateur K apparaît sous un nombre pair de négations. La procédure proposé dans la thèse utilise des automates d'arbres coBüchi universelles (UCT). Plus précisément, étant donné une formule KLTL$^+$ φ et un modèle d'interaction $\mathcal{M}$, nous construisons un UCT $\mathcal{T}_\varphi$ qui accepte exactement l'ensemble de stratégies qui réalise φ dans l'environnement $\mathcal{M}$.

Malgré le fait que notre procédure a la complexité 2ExpTime dans le pire des cas, nous l'avons implémenté et montré sa faisabilité pratique à travers un ensemble d'exemples. En particuliers, basé sur l'idée de [36], nous réduisons le problème de tester le vide d'un UCT $\mathcal{T}_\varphi$ à résoudre un jeux de sûreté dont l'espace d'états peut être ordonné et représenté de manière compacte en utilisent des antichaînes. L'implémentation est basée sur l'outil Acacia [12] et, à notre connaissance, est la première implémentation d'une procédure de synthèse pour des spécifications temporelles épistémiques. En plus, cette implémentation

peut être utilisée pour résoudre des jeux à deux joueurs avec information imparfaite et objectifs LTL. Cette contribution est détaillé en Chapitre 4 et poubliee dans [14].

Synthèse avec Environnement Rationnel

Nous étudions aussi la *Synthèse Rationnelle*. La rationalité de l'environnement est modélisée en utilisant une notion d'équilibre. Ainsi, la spécifications du profil de stratégies pour le contexte non-coopératif contient tous profils de stratégies qui, si ils sont en équilibre de point de vue du l'environnement (P-fixé équilibre), alors ils satisfont tous les objectifs des processus dans P. Pour le cas coopératif, la spécifications du profil de stratégies contient tous les profils de stratégies qui sont équilibres P-fixées et satisfont les objectifs de tous les processus dans P.

Cas de l'Information Parfaite Nous d'abord étudions le problème dans le cas *d'information parfaite pour les deux processus à synthétiser et l'environnement.* On considère des variantes du problème de synthèse pour objectifs de accessibilité, sûreté, Büchi, coBüchi, parité, Rabin, Streett et Muller quand la rationalité des processus de l'environnement est modélisé en utilisent des équilibres de Nash. Nous étudions également la complexité de résoudre ces problèmes lorsque le nombre de joueurs est fixe. Cette analyse paramétrée a du sens car le nombre des composants de l'environnement peut être limité dans les situations pratiques. Nos résultats (illustrés dans le Tableau 5.1) montre que la complexité diminue pour les objectifs particuliers que nous considérons.

Les *bornes inférieures* pour les différents objectifs sont obtenus par des réductions (souvent complexes) de différents problèmes qui sont bien connus d'être difficile. Quand le nombre des processus est fixé, les bornes inférieures vient en général de la complexité de résoudre les jeux à deux joueurs à somme nulle, avec le même type d'objectif pour le protagoniste. Toutefois, les objectifs de Rabin et Streett (dans les deux cas, coopératif et non-coopératif) et objectifs de Parité (pour le cas non-coopératif) nécessitent une réduction plus complexe.

La synthèse rationnelle *coopérative* avec information parfaite est un cas particulier du problème d'existence d'un équilibre de Nash contraint dans les jeux multi-joueurs, où les objectifs des processus dans le set P doivent être satisfaits. La complexité de ce problème a été étudié par Ummels [87] pour quelques classes d'objectifs. L'idée principale dans [87] est de caractériser les équilibre de Nash par des formules LTL sur les exécutions dans le modèle d'interaction. Cette solution nous donne des bornes supérieures pour la synthèse rationnelle coopérative et objectifs de Büchi, coBüchi, Parité et Streett. Pour les autres objectifs, nous étendons cette caractérisation.

Les solutions pour le cas *non-coopératif* sont plus compliqué et sont basés sur une application fine des techniques sur les automates d'arbres. Nous définissons des automates non déterministes qui ont une taille exponentielle, mais nous montrons comment tester

leur vide en PSpace pour obtenir des algorithmes pour des objectives de Streett, Rabin et Muller et un nombre fixé de joueurs. Plus précisément, l'automate as un nombre exponentiel d'états dans le nombre d'agents et as des propriétés de monotonicité. Ça nous permet de réduire le test du vide des automates au problème de résoudre un jeux de durée fini (connu dans la littérature comme des jeux de premier cycle [8]) ayant une durée polynomiale dans le cas des objectifs de Sûreté, Accessibilité, Büchi et coBüchi et durée exponentielle pour les objectifs de Parité, Streett, Rabin et Muller. Pour un nombre fixée d'agents, l'automate nondéterministe d'arbres a une taille polynomiale dans la taille du modèle d'interaction fourni.

La procédure pour résoudre le Problème de Synthèse Rationnelle avec information parfaite et les preuves détaillées pour les complexités serres pour les différents types d'objectifs sont présentées dans Chapitre 5 et une version courte est publié dans [28].

Cas de l'Information Imparfaite Finalement, nous étudions le cas *d'information imparfaite* pour la synthèse rationnelle. Dans le cas général, le problème est indécidable car la synthèse distribué avec objectifs LTL et environnement antagoniste est déjà indécidable [75]. Toutefois, nous regagnons la décidabilité pour le cas non-coopératif si on demande à synthétiser un composant ($|P| = 1$) avec observation partielle contre un environnement milti-composant, omniscient et rationnel. En autre, un répons positif pour le cas où seulement le processus à synthétiser a l'information imparfaite, est aussi un répons positif pour le cas plus restrictif où l'environnement a aussi observation partielle. Effectivement, si Processus 0 a une stratégie pour gagner contre un environnement omniscient et rationnel, il peut appliquer la même stratégie pour assurer son objectif quand l'environnement a une observation partielle.

La preuve de décidabilité pour la synthèse d'un processus contre un environnement multi-composant, omniscient, rationnel et non-coopératif utilise l'idée du construction de l'automate d'arbre construit dans le cas d'information parfaite. Ensuite, le problème se réduit au problème de synthèse de deux processus avec observation hiérarchique et objectif LTL contre un environnement omniscient et antagoniste avec un seul composant. Le dernière problème est démontré être décidable dans [90].

Les résultats de décidabilité pour la synthèse rationnelle avec information imparfaite sont détailles dans Chapitre 6 et sont de nouveaux résultats qui ne sont pas encore publiés.

Organisation de cette Thèse

Dans le Chapitre 2, nous définissons le modèle d'interaction et donne la définition générale du problème de synthèse. Également, nous définissons et présentons les résultats existantes pour les deux problèmes de synthèse qui nous étudions.

Le Chapitre 3 présente outils et techniques utilisées pour résoudre le problème de

synthèse. D'abord nous rappelons quelques logiques que permet le raisonnement sur les stratégies et ensuite nous définissons les automates sur mots et arbres et fournissons les procédures classiques pour tester leur vide.

Chapitre 4 révise les procédures existantes pour résoudre le problème de synthèse avec environnement antagoniste et propose une solution Safraless pour la synthèse KLTL$^+$.

Dans Chapitre 5, nous fournissons des solutions pour résoudre le problème de synthèse rationnelle avec information parfaite. Dans Chapitre 6 nous étudions le problème du synthèse rationnelle dans le cas d'information imparfaite.

Acknowledgements

I would like to thank all the people who made this thesis possible.

In am especially grateful to my supervisors Cătălin Dima and Emmanuel Filiot for guiding my research and for all the time and interest they invested in making my PhD a great experience. Thank you for encouraging my research and for allowing me to grow as a research scientist.

I would like to thank each member of my thesis committee for reading my PhD memoir. I am honored to having you as part of my thesis committee.

Many thanks to my co-authors Cătălin Dima, Emmanuel Filiot, Raffaela Gentilini and Jean-François Raskin for the role they played in my research. I learned a lot from them about the way the research is done and how to present the work so that the audience can easily follow. I would also like to mention Youssouf Oualhadj with whom I had interesting and stimulating discussions related to my research, and who is always motivating.

To all members of LACL, I am thankful for the extraordinary time spent during the last years, for all the shared lunches, TPs and TDs. I have a special thought for Nicolas Herniou who helped me whenever I had problems with my computer.

I thank the members of Computer Science department of ULB for all nice moments spent during the time I was at ULB.

Last but not the least, I would like to thank my family: my husband and my parents and sister for supporting me spiritually throughout writing this thesis and my life in general.

Contents

1. Introduction

1.1 Motivation

Reactive systems are non-terminating systems that interact with their environment. They arise both as hardware and software, and are usually part of *safety-critical systems*, for example microprocessors, computer systems responsible of air traffic management, control of nuclear power plant or programs to monitor medical devices. It is therefore crucial to guarantee their correctness. There are different approaches to ensure the correctness of such systems.

Program Verification Program verification is a domain of computer science that facilitates the early detection of defects in the hardware and software systems for which failure is unacceptable. One of the major achievements in system verification is the theory of *model-checking* [27], that allows to automatically check for system correctness.

It applies to *interaction models* (or transition systems) that model the interaction between the program and its environment together with specifications typically expressed as *temporal logic formulas*[60] describing the allowable behaviors of the program. The model-checking problem consists of answering the question *"Does the program satisfy the specification?"*. In the case of a negative answer, some procedures provide a counterexample consisting of an interaction between the system and the environment that does not satisfy the specification. Then, using the counterexample (if provided), one has to fix the bug in the system or even in the specifications and proceed from the beginning with the model-checking process.

Nowadays, model-checking techniques represent an important part of the design process in companies such as Intel and IBM. However, designing the system using model-checking and guided refinements may be time consuming and remains dependent on the skills of programmers. That is, the process of verification followed by bug fixing may be repeated several times before a correct program is obtained.

Synthesis Given a specification, the *synthesis* problem aim is to automatically synthesize a program that fulfills the behavior defined by the specification. Therefore, the constraints do not need to be checked afterwards, and this allows the designer to

focus on defining high-level specifications, rather than designing complex computational models of the systems.

The synthesis problem was introduced by Church [26] for circuits. It requires that if given a logical specification $S(I, O)$ between input signals I and output signals O, to determine whether there is a circuit that, for any received inputs, it outputs signals so that the specification is satisfied.

One of the main challenges of synthesis of reactive systems is to satisfy the specification regardless of the behavior of its environment. For example, let us take a computer that controls an elevator. It has actions to make the elevator go up, go down, and open and close the doors. At each floor there are buttons that can be pushed to call the elevator and there are some buttons inside the elevator that serve to demand the destination floor of some person in the elevator. The person that pushes the buttons represents the environment of the controller of the elevator. Then, a specification to be satisfied by the elevator is the following. Whenever the elevator is called at a certain floor, it has to eventually stop and open the door at that floor. Also, whenever the person inside the elevator asks the elevator to go at a certain floor, it should eventually reach it. Note that the behavior of the person in the elevator is not known in advance and is also not restricted. Thus, they can push any buttons and the specifications have to be satisfied for any possible behavior of the person.

The uncontrollable environment usually leads to computationally harder decision problems, compared to system verification. For instance, model-checking properties expressed in *linear time temporal logic* (LTL) is PSPACE-complete[83] while LTL synthesis is 2EXPTIME-complete [74].

The synthesis problem of reactive systems from LTL specifications was first introduced and studied by Pnueli and Rosner [74] and also by Abadi, Lamport and Wolper [2]. However, the solution proposed involves nondeterministic automata that have to be determinized using the complex Safra's construction[79] that was shown to be resistant to efficient implementations[85]. Piterman [73] and Muller and Schupp [70] proposed optimizations of Safra's construction, that unfortunately continue not to lead to efficient implementations. The main reason of the resistance to implementation of these constructions is that they cannot be implemented symbolically and therefore one needs to explicitly build the entire set of states of the automata.

Recently, procedures that avoid Safra-like constructions (called *Safraless*) were developed. A first solution was proposed by Kupferman and Vardi in [55] and uses universal coBüchi automata to characterize solutions for the synthesis problem. This approach has aroused a lot of interest in developing feasible procedures to solve the synthesis problem that can lead to implementations. Since then, several other "Safraless" procedures have been defined [55, 81, 41, 36]. In [81, 36], it is shown that LTL synthesis reduces to testing the emptiness of a universal coBüchi tree automaton, that in turn can

be reduced to solving a safety game. The structure of the safety games can be exploited to define a symbolic game solving algorithm based on compact antichain representations [36]. Lily[49], Anzu[50], Ratsy[11] and Acacia+[12] are some examples of tools that efficiently implement algorithms to solve the LTL synthesis problem.

Multi-Component Systems In the previously presented works, the system is presumed to have perfect information about the state of the environment. However in many practical scenarios, this assumption is not realistic since several systems have multiple components. In distributed or multi-component systems, it is typical that each component has its private variables, which are not visible from exterior. Therefore, components must act on the basis of the local data and coordinate to ensure some global properties of the system in some environment.

The interaction of the distributed systems is classically modeled using interaction models (or transition systems). Each state of the interaction model represents the global configuration of the distributed system and the transition between states is made depending on the action of each component (including the environment of the system that can be considered an extra component). Then, each component has a partial visibility on the state of the interaction model corresponding to its local data. The components don't see which actions are played by the others but get some observation on the states in which the interaction model may be. An execution of the interaction model, from the system's perspective, is therefore an infinite sequence alternating between its own actions and observations.

A protocol (or behavior or strategy) of some component consists in indicating the actions to be run in the interaction model, depending on the past observations it received. Therefore, the synthesis of a (set of) component(s) is equivalent to finding a (set of) protocol(s) for the component(s) to synthesize. When a set of components is asked to be synthesized, the rest of the components of the distributed system are considered to be part of the environment.

Epistemic Temporal Logics [44] are logics formatted for reasoning about multi-agent situations. Besides the formal description of the temporal ordering of events, they also allow the reasoning about the knowledge of the components. One can formulate statements as "if process i knows that the transaction will be aborted, it should rollback its local contribution and terminate immediately"[90]. The importance of knowledge of components is highlighted by different problems. One of them is the coordinated attack problem. It involves two generals that attempt to coordinate an attack by communicating through an unreliable messenger. They must agree on a time to attack and to agree to attack at the same time. Moreover, each general must know that the other general knows that they have agreed on the attack plan.

Epistemic temporal logics are extensions of temporal logics with knowledge operators K_i for each agent. A formula $K_i\varphi$ intuitively says that the Agent i knows that the property φ holds. Epistemic logics have been successfully used for verification of various distributed systems in which the knowledge of the agents is essential for the correctness of the system specification.

Synthesis with Temporal Epistemic Objectives and Antagonist Environment Vardi and van der Meyden [89] have studied the synthesis problem from specifications expressed in epistemic temporal logic formulas. They define the synthesis problem in a multi-agent setting, for specifications written in LTL extended with knowledge operators K_i for each agent (KLTL).

The goal of the KLTL synthesis problem is to automatically generate protocols for the (multi-component) system (if it exists) that tell which action should be taken, depending on finite histories, so that whatever the environment does, all the (concrete) infinite executions resulting from this strategy satisfy the KLTL formula. Due to [75], this problem is undecidable already for LTL objectives when the system consists of two partially informed components against their environment. On the other hand, for a single-component system against antagonist environment, the problem is shown 2EXPTIME-complete[89], by reduction to the emptiness of alternating Büchi tree automata. This theoretically-elegant construction is, however, difficult to implement and optimize, as it relies on complex Safra-like automata operations (Muller-Schupp construction).

Synthesis with Multi-Component Rational Environment More recently, Kupferman et. al [38, 53] introduced the synthesis problem against a multi-component rational environment. The rational synthesis problem assumes that each component of the interaction model has its own objective, other then making the others fail. They are rational in the sense that first they target the satisfaction of their own objectives and then try to harm the others.

The Rational Synthesis problem was introduced in the more relaxed setting of perfect information of each component in the interaction model (environment and process to be synthesized). That is, they know which is the global state of the system. The objectives of each process is expressed as some LTL formula and their rationality is modeled using solution concepts as dominant strategies, Nash Equilibria, and the like.

The problem was studied in two different settings. In the *non-cooperative rational synthesis*[53], since the components of the environment are rational, they may play any strategy profile that is an equilibrium. Therefore, the problem asks to synthesize a protocol for some component such that it ensures its specification against any rational behaviors of the other components making up its environment. On the other hand, the

cooperative rational synthesis problem assumes that the environment is more cooperative in the sense that it agrees to adhere to a strategy profile that is an equilibrium. Therefore, the problem is to find such a strategy profile where the system satisfies its objective.

The main contribution of the original papers is to put forward and to motivate the definitions above. The only computational complexity results given in those papers are as follows: the cooperative and non-cooperative rational synthesis problems are 2ExpTime-complete for specifications expressed in linear temporal logic (LTL). This result matches exactly the perfect information LTL synthesis within a single-component antagonist environment. The lower bound is due to the fact that the later problem is a particular case of the rational synthesis problem. The upper bound is obtained by reduction to the model-checking problem of formulas in a Nested-Goal fragment of Strategy Logic (SL[NG]) [65].

1.2 Objectives of the Thesis

The objective of this thesis is to study the synthesis problem of reactive systems in both antagonist and rational settings. We also consider different kinds of observations for the processes. That is, we study the two problems when the components(processes) to be synthesized and the ones consisting their environment have perfect or imperfect information.

More precisely, we are interested in "Safraless" decision procedures to solve the synthesis problem that may lead to efficient implementations. First, for the case of KLTL Synthesis Problem with a one-component antagonist environment, our goal is to make use of universal coBüchi automata and develop a procedure similar to the one in [36].

Then, for the case Rational Synthesis Problem, the solution proposed in [37, 53] for the case of perfect information for all components is by reduction to model-checking of some Strategy Logic formulas. This, is further done by reduction to testing the emptiness of alternating tree automata that involves Safra-like constructions. Our goal is to finely understand the computational complexities of the rational synthesis problem and how to manipulate the underlying game algorithmically. Then, we target "Safraless" procedures to solve the rational synthesis problems under both perfect and imperfect information.

1.3 Contributions

General Definition of Synthesis Problem

In this thesis, we provide a more general definition of the synthesis problem that encapsulates both problems mentioned above. More precisely, we consider interaction models $\mathcal{M}$ on which act $k+1$ partially informed processes. The partial information of

each process is modeled by some equivalence indistinguishability relation on the states of the interaction model. Each equivalence class represents some observation of the corresponding process. This captures the situation where each state in $\mathcal{M}$ represents some global state of a distributed system, and each process models one of its components that do not see the local data of the others. The transition between states is done, as in the model proposed in [89], depending on the actions of processes. Also, along executions, each process is informed only on its own actions and the observations on states.

The set Ω of processes is partitioned in two. There is a set P of processes that represent the components in the distributed system to be synthesized and the processes in $\Omega \setminus P$ represent the environment of the processes to synthesize.

The general approach to solve the synthesis problem is by using game theory concepts. That is, each component of the interaction model represents a player in the game and then solving the synthesis problem consists in finding strategies for the players corresponding to processes to synthesize so that they ensure some specifications. We consider that the requirements are given as strategy profile specifications that describe the acceptable strategy profiles. That is, it is provided a set Φ of strategy profiles (one strategy for each process) that are desirable. Given the set Γ_i of possible strategies of Process $i \in \Omega$, Φ is a set of tuples in $\Gamma_0 \times ... \times \Gamma_k$.

Then, the **synthesis problem** asks to synthesize tuples $(\sigma_i)_{i \in P}$ of strategies for processes in P such that for any tuple $(\sigma_i)_{i \notin P}$ of strategies for the processes in the environment $(\Omega \setminus P)$, the resulting strategy profile $\langle \sigma_0, ..., \sigma_k \rangle$ is included in the specification Φ. That is, we consider the following *realizability problem*:

$$\begin{aligned} &\mathsf{INPUT}: \mathcal{M},\ P \subseteq \Omega,\ \Phi \subseteq \Gamma_0 \times \cdots \times \Gamma_k \\ &\mathsf{OUTPUT}: \text{Yes iff } \exists (\sigma_i)_{i \in P}\ \forall (\sigma_i)_{i \notin P}\ \langle \sigma_0, ..., \sigma_k \rangle \in \Phi \end{aligned}$$

If the answer for the previous problem is positive, the synthesis problem asks to provide a witness. In this thesis, all our realizability tests are constructive (provide a witness) and therefore we may refer to the realizability problem as the synthesis problem.

An overview of the decidability results for this problem is sketched in Table 2.1.

KLTL Synthesis with Antagonist Environment

First, we consider the *KLTL Synthesis Problem* with antagonist environment defined in [89]. In this case, the processes in the set P have as joint objective the satisfaction of some KLTL formula φ against a completely antagonist environment (having the objective $\neg\varphi$) that consists of a single process. Therefore, the strategy profile specification consists of all strategy profiles for which the resulting execution satisfies φ.

The KLTL synthesis problem is undecidable in the general case and therefore we focus on the restriction from [89] where one partially informed process ($|P| = 1$) is synthesized against a perfectly informed environment. The solution proposed in [89] is

based on emptiness of alternating Rabin tree automata, which is tested by first applying Safra-like constructions to remove alternation and then checking the emptiness of resulting nondeterministic tree automata.

Our main contribution regarding the KLTL synthesis against antagonist environment is to define and implement a Safraless synthesis procedure for the positive fragment of KLTL (KLTL$^+$), i.e., KLTL formulas where the operator K does not occur under an odd number of negations. Our procedure relies on universal coBüchi tree automata (UCT). More precisely, given a KLTL$^+$ formula φ and some interaction model $\mathcal{M}$, we show how to construct a UCT $\mathcal{T}_\varphi$ whose language is exactly the set of strategies that realize φ in the model $\mathcal{M}$.

Despite the fact that our procedure has 2ExpTime worst-case complexity, we have implemented it and shown its practical feasibility through a set of examples. In particular, based on ideas of [36], we reduce the problem of checking the emptiness of the UCT $\mathcal{T}_\varphi$ to the problem of solving a safety game whose state space can be ordered and compactly represented by antichains. Our implementation is based on the tool Acacia [12] and, to the best of our knowledge, it is the first implementation of a synthesis procedure for epistemic temporal specifications. As a byproduct, this implementation can be used to solve two-player games of imperfect information whose objectives are given as LTL formulas. The procedure is detailed in Chapter 4 and a short version is published in [14].

Synthesis with Rational Environment

We also study the *Rational Synthesis Problem*. The rationality of the environment is modeled by using some notion of equilibria. Thus, the strategy profile specification for the non-cooperative setting contains all strategy profiles that, if they are in equilibrium from the environment's perspective (P-fixed equilibrium), then they also satisfy all the objectives of the processes in P. For cooperative setting, the strategy profile specification contains all strategy profiles that are P-fixed equilibrium and satisfy the objectives of processes in the set P.

Perfect Information Setting We first consider the problem under *the assumption of perfect information for both processes to be synthesized and the environment*. We then solve variants of the problem for reachability, safety, Büchi, coBüchi, parity, Rabin, Streett and Muller objectives when the rationality of processes is modeled using Nash equilibria and also study the computational complexity of solving those problems when the number of players is fixed. This parameterised analysis makes sense as the number of components forming the environment may be limited in practical applications. Our results (illustrated in Table 5.1) show that the complexity decreases for the particular objectives we consider.

The *lower bounds* for the different objectives are obtained by (often intricate) reductions from different problems that are well known to be hard. For the case of

fixed number of processes, the lower bounds generally come from the complexity of solving two-players zero-sum games with the same type of objective for the protagonist. However, Streett and Rabin objectives (in both cooperative and non-cooperative settings) and Parity objectives (for non-cooperative setting) need more involved reductions.

Cooperative rational synthesis under perfect information assumption is a particular case of the more general problem of checking the existence of a constrained Nash equilibrium in a multiplayer game, where the strategies of processes in P are required to be winning. The complexity of constrained Nash equilibria has been studied by Ummels [87] for some classes of objectives. The main idea in [87] is to characterize Nash equilibria by means of LTL properties on executions in the interaction model. This directly gives us upper-bounds for cooperative synthesis and Büchi, coBüchi, parity and Streett objectives. For the other objectives, we extend this characterization.

The solutions for the *non-cooperative* case are more involved and are based on a fine tuned application of tree automata techniques. We define nondeterministic tree automata that have exponential size but we show how to test their emptiness in PSpace to obtain optimal algorithms for Streett, Rabin and Muller objectives and fixed number of players. More precisely, the automaton has an exponential number of states only in the number of processes and has monotonic properties. This allows us to reduce the emptiness test of the automaton to the problem of solving some finite duration games (known in the literature as first cycle games[8]) that have polynomial length in the case of Safety, Reachability, Büchi and coBüchi objectives and exponential length for Parity, Streett, Rabin and Muller objectives. For a fixed number of players, the nondeterministic tree automaton has polynomial size in the size of the given interaction model $\mathcal{M}$.

The procedures solving the Rational Synthesis Problem for the perfect information setting and detailed proofs for the tight complexity bounds for different types of objectives are presented in Chapter 5 and a short version is published in [28].

Imperfect Information Setting Finally, we consider the *imperfect information setting* for Rational Synthesis. In the general setting, the problem is undecidable due to the undecidability of LTL distributed synthesis with antagonist environment [75]. However, we gain decidability for non-cooperative setting if we consider only one partially informed process to synthesize ($|P| = 1$) against an omniscient environment. Moreover, a positive answer to the setting when only the system to be synthesized has imperfect information, is also a positive answer for the more restricted environment that has to play observational strategies. Indeed, if Process 0 has a strategy to win against the omniscient rational environment, then it can apply the same strategy to ensure its objective when the environment's observation is more restricted.

The decidability proof for the synthesis of one process against a multi-component omniscient rational non-cooperative environment uses the idea in the construction of the

tree automaton built for the case of perfect information. The problem then reduces to the problem of synthesizing two processes with hierarchical observations and LTL objectives against an omniscient one-component antagonist environment. The later problem is proven in [90] to be decidable.

Decidability results for the Imperfect Information Rational Synthesis are detailed in Chapter 6 and are new results that are not published yet.

1.4 Organization of the Thesis

In Chapter 2, we define the interaction model and give the general definition of the synthesis problem. We also define and present existent results and contributions for the two considered settings of synthesis problem.

Chapter 3 presents tools and techniques used for solving the synthesis problem. We first recall some logics that allow the reasoning about the strategies and then define the automata on words and trees and give the classical procedures for testing their emptiness.

Chapter 4 revises the existing procedures to solve the synthesis problem with antagonist environment and provides a Safraless solution for the KLTL$^+$synthesis.

In Chapter 5, we provide solutions for the rational synthesis problem under perfect information and in Chapter 6 we study the rational synthesis problem in the case of imperfect information.

2. Realizability and Synthesis

In this section we define the realizability and synthesis problems in a multi-component interaction model where each process (or agent) represents a component in the distributed system.

2.1 Interaction Model

We assume that the possible behaviours of the processes that interact and the changes that appear in the entire system are modelled as a transition system. Each state of the transition system corresponds to a global configuration in the system and the change of state is done depending on the events appearing in the system, modelled as actions of processes. Therefore, assuming that the processes are numbered from 0 to $k \in \mathbb{N}$, the transition system is defined over $k+1$ sets $\Sigma_0, \Sigma_1, ..., \Sigma_k$ of actions for the component processes and the transition relation from state to state is defined with respect to tuples of actions in $\Sigma_0 \times \Sigma_1 \times \cdots \times \Sigma_k$. Additionally, each state v of the interaction model carries an interpretation $\tau(s)$ over a (finite) set of properties modeled as a (finite) set of atomic propositions $\mathcal{P}$. Formally,

Definition 2.1.1. *An interaction model is a tuple* $\mathcal{M} = \langle \mathcal{P}, \Omega, (\Sigma_i)_{i\in\Omega}, V, V_0, E, \tau \rangle$ *where*

- $\Omega = \{0, 1, ..., k\}$ *is the finite set of processes (also called agents or players in the following)*
- $\mathcal{P}$ *is the finite set of propositions,* $\Sigma_0, \cdots \Sigma_k$ *are finite sets of actions for the processes*
- V *is the set of states,* V_0 *is the set of initial states,*
- $\tau : V \to 2^{\mathcal{P}}$ *is the labeling function (interpretation),*
- $E : V \times \Sigma_0 \times \Sigma_1 \times \cdots \Sigma_k \to V$ *is the (total) transition function*

Note that by the definition we consider, the model is assumed to be deadlock-free and complete for all actions, i.e., from any state and any tuple of actions there is one outgoing transition. At each round, each player chooses an action and according to the choices of all players, the system moves in the successor state given by E.

Turn-based interaction model In the above definition, the interaction model is concurrent. In each state, each process proposes one action and the transition is made depending on the tuple of actions of the processes. In the *turn-based* setting, there is a partition $(V_i)_{i\in\Omega}$ of the set of states, one set for each process. Intuitively, this corresponds to the setting when only one process is active at a time. Then, at each state, the process that is active (that controls the current state) chooses the next state in the interaction model. Therefore, we can assume that there are no actions on transitions and that the transition relation is formally defined as a set $E \subseteq V \times V$ of edges that indicates the possible choices from each state. Therefore, at state $v \in V$, the Agent i for which $v \in V_i$ chooses a state v' such that $(v, v') \in E$. In this thesis we refer to concurrent setting, unless is explicitly said otherwise.

Executions We denote by $E(v)$ the set of successor states of v, i.e., $E(v) = \{v' \in V \mid \exists(a_0, ..., a_k) \in \Sigma_0 \times ... \times \Sigma_k \text{ s.t. } v' = E(v, a_0, a_1, ..., a_k)\}$. An *execution*(or *path*) ρ in $\mathcal{M}$, is an infinite sequence of states $\rho = v_0 v_1 ... \in V^\omega$ such that $v_0 \in V_0$ and for all $n > 0$, $v_{n+1} \in E(v_n)$. We let $\mathsf{trace}(\rho) = \tau(v_0)\tau(v_1)\tau(v_2)... \in (2^{\mathcal{P}})^\omega$ be the *trace* of the execution ρ in $\mathcal{M}$. A *prefix* (or history) of ρ up to v_n is written $\rho[:n]$ and its last state is $\rho[n]$. We denote by $\sqsubset$ the prefix relation over $V^* \cup V^\omega$. We let $\mathsf{exec}(\mathcal{M})$ stand for the set of executions in $\mathcal{M}$, and $\mathsf{Prefs}(\mathcal{M})$ for its closure under $\sqsubset$. For $\rho \in V^\omega$, we write $\rho[n:]$ for the *suffix* of ρ starting from position n. Also, $\mathsf{visit}(\rho)$ stands for the set of states that appear at least once along ρ, i.e. $\mathsf{visit}(\rho) = \{v \in V \mid \exists n \geq 0 \text{ s.t. } \rho[n] = v\}$, $\mathsf{visit}(\rho, v)$ for the number of times the state v is visited along ρ (or ∞ if ρ visits infinitely often the state v) and $\inf(\rho)$ for the set of states occurring infinitely often along ρ, i.e. $\inf(\rho) = \{v \in V \mid \forall n \geq 0, \exists m \geq n \text{ s.t. } \rho[m] = v\}$. Finally, $\mathcal{M}[v]$ stands for the interaction model obtained from $\mathcal{M}$ by replacing the initial set of states by $\{v\}$.

Example 2.1.1. *We illustrate the notion of interaction model on the following toy example. Let us consider two processes, one that sends requests for some resource and the other grants it. We assume that Process* 1 *makes the requests by making the action* R *(*$\neg R$ *if no request) and that Process 0 grants it by making the action* G *(*$\neg G$ *if no grant).*

The interaction model has two states. The initial state is labeled with the proposition g *that stands for "all requests are granted". There is a second state labeled with* r *that corresponds to the case in which there are some requests that are not granted yet.*

We assume that, with action G*, Process 0 grants all the requests that were made up to the present, including the ones that are made in the same time with the grant. Therefore, the transition relation in the interaction model is the one depicted in Figure 2.1. Whenever a grant is given, the transitions lead to state* v_0 *labeled with* g*. However, if a request is made by Process 1 and there is no grant given in the same time, the transition leads to the state* v_1*. The interaction model remains in* v_1 *until a new grant is provided. Formally,*

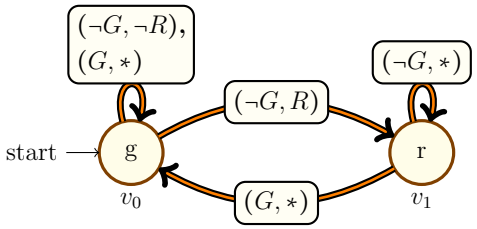


Figure 2.1: Example: Request-Grant interaction model

we write

$$E(v_0,(a,b)) = \begin{cases} v_1 & \text{if } (a,b) = (\neg G, R) \\ v_0 & \text{otherwise} \end{cases} \qquad E(v_1,(a,b)) = \begin{cases} v_0 & \text{if } a = G \\ v_1 & \text{otherwise} \end{cases}$$

Example 2.1.2 (Light Bulb)**.** *Another example that is more related to the real world is the one of [89], that describes the behavior of two processes acting on a timed toggle switch with two positions* (on,off) *and a light bulb. The light is intended to indicate the position of the toggle. That is, the light can be* on *only if the toggle is* on*. However, the light bulb may be broken, so the light may be* off *even if the toggle is* on*. Also, the toggle may switch off after an arbitrary time.*

Process 0 can act on the switch by performing one of the actions "toggle" or "skip". Whenever he toggles, the position of the toggle changes. He proposes the action "skip" whenever he doesn't want to change the position of the toggle.

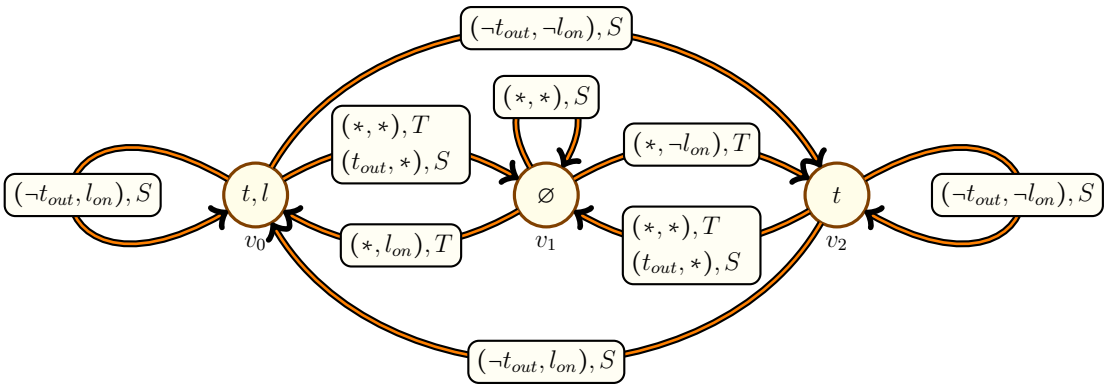


Figure 2.2: Interaction model $\mathcal{M}$ of Example 2.1.2

Formally, the configurations of the system are modeled by states labeled with atomic propositions t (true iff the toggle is on) and l (true iff the light is on) that encode the position of the toggle and the light respectively. Therefore, the set of atomic propositions is $\mathcal{P} = \{t, l\}$. The interaction model is illustrated in Figure 2.2. The state v_0 (labeled with both l and t) is the only state in which the light is on *and also requires the toggle to be* on*.*

But, since the light bulb can be broken, the state v_2 models the situation when the toggle is on *(is labeled with t) but not the light (it is not labeled by l). State v_1 corresponds to the situation when both the light and the toggle switch are* off*. Therefore, the labeling function $\tau : V \to 2^{\mathcal{P}}$ is such that $\tau(v_0) = \{t, l\}$, $\tau(v_1) = \varnothing$ and $\tau(v_1) = \{t\}$.*

Actions of the Process 0 are $\Sigma_0 = \{T, S\}$ for "toggle" and "skip" respectively. To model the fact the toggle times out at any time and that the light may be faulty, we consider another process (Process 1) that has special actions to model these situations. He may decide at any time to time out the toggle switch by doing the action t_{out}. Also, he controls the light and decides when it turns on the light by means of action l_{on}. Note that since the light can be on only if the toggle switch is on, the action l_{on} is ignored if the toggle is not on. Formally, the set of actions of Process 1 (which may be seen as the environment in which Process 0 runs) is $\Sigma_1 = \{(t_{out}, l_{on}) \mid t_{out}, l_{on} \in \{0, 1\}\}$. The Boolean variables t_{out} and l_{on} indicate that Process 1 times out the toggle and that he switches on the light. In Figure 2.2, we write t_{out} and l_{on} when the variables are set to true and $\neg t_{out}$ and respectively $\neg l_{on}$ when they are set to false. The star $*$ means "any action".

The transition function is depicted on the figure. Formally, $E(v, (a, (b_1, b_2)) = v'$ s.t.

$$\begin{cases} t \in \tau(v') \Leftrightarrow t \notin \tau(v) & \text{if } a = T \text{ or } (t \in \tau(v) \text{ and } b_1 = t_{out}) \\ t \in \tau(v') \Leftrightarrow t \in \tau(v) & \text{otherwise} \end{cases} \quad \text{and}$$

$$l \in \tau(v') \quad \text{iff} \quad t \in \tau(v') \text{ and } b_2 = l_{on}$$

2.1.1 Strategies

The processes are supposed to act depending to the current history. That is, they act according some *strategies* (called protocols in [89]) that are mappings from a sequence of states and actions of the corresponding process to an new action. Formally, a *strategy* (or protocol) for Process $i \in \Omega$ is a total function $\sigma_i : V(\Sigma_i V)^* \to \Sigma_i$. An execution $\rho = v_0 v_1 ... \in \mathsf{exec}(\mathcal{M})$ is said to be *compatible* with σ_i if there is an infinite sequence of actions $a_0^0 a_1^0 ... a_k^0 a_0^1 .... \in (\Sigma_0.\Sigma_1.\cdots\Sigma_k)^\omega$ such that $v_n = E(v_{n-1}, a_0^{n-1}, ..., a_k^{n-1})$ and for all $n \geq 0$, $a_i^n = \sigma_i(a_i^0 v_0 a_i^1 v_1 ... a_i^{n-1} v_{n-1})$. We denote by $\mathsf{exec}(\mathcal{M}, \sigma_i)$ the set of executions of $\mathcal{M}$ compatible with σ_i and by Γ_i the set of strategies of Agent i.

For a strategy σ_i and a history $h \in V(\Sigma_i V)$, the strategy σ_i^h denotes the strategy of Process i after the history h and is defined as $\sigma_i^h(h_1) = \sigma_i(h \cdot h_1)$.

In this thesis, the agents are assumed to have *perfect recall* and therefore the reason why in the above definition of strategies they are assumed to play actions depending on the entire history of states and actions of the corresponding process. However, in the literature there are several different settings. It can happen that agents do not remember their entire past experience. They may forget some previous states they visited (have only finite memory or even no memory at all) or they may forget the actions they took

in the past. In these cases, we speak about the notion of *finite memory* and *memoryless* strategies respectively. A strategy σ_i of Agent i is called *memoryless* if for any two sequences $h, h' \in (V\Sigma_i)^*$ and $v \in V$, $\sigma_i(hv) = \sigma_i(h'v)$.

Example 2.1.3. *One example in which memoryfull strategies are needed is to solve the absent-minded driver paradox described in [46]. It is assumed there is an absent-minded individual planning his trip home. In order to get home, he has to take the highway and get off at the second exit. Since the driver is absent-minded, whenever he arrives at an intersection, he cannot say if it is the first or the second intersection and cannot remember how many he has passed. Note that there is no strategy that gets the driver home in the setting we consider in this thesis. This is because if the driver decides to take the exit when he arrives at an intersection, he will take the first exit. On the other hand, if he decides to continue, he will not take the second exit either and he will arrive at the end of the highway. We will give more details about objectives of agents and existence of strategies in the remaining of the thesis.*

In this thesis we call a finite memory strategy one for which the Moore machine encoding it has a finite number of states.

Strategies as Moore Machines We use *Moore machines* to represent strategies $\sigma_i : V(\Sigma_i V)^* \to \Sigma_i$. A Moore machine M with input alphabet V and output alphabet Σ_i is a tuple $M = \langle V, \Sigma_i, S_M, s_0, \delta_M, g_M \rangle$ where S_M is the set of states, the initial state is $s_0 \in S_M$, $\delta_M : S_M \times V \to S_M$ is the (total) transition function and $g_M : S_M \to \Sigma_i$ is the (total) output function. The transition function δ_M is then extended to $\delta_M^* : V^* \to S_M$ inductively as follows: $\delta_M^*(\epsilon) = s_0$ and $\delta_M^*(xc) = \delta_M(\delta_M^*(x), v)$ for all $x \in V^*$ and $v \in V$.

We then say that a Moore machine M encodes a strategy σ_i for Process i if and only id for any history $h = v_0 a_i^0 v_1 a_i^1 ... v_n \in V(\Sigma_i V)^*$, holds that $g_M(\delta_M^*(v_0 v_1 ... v_n)) = \sigma_i(h)$.

Strategy Profiles A *strategy profile* is defined as a tuple of strategies $\bar{\sigma} = \langle \sigma_0, \sigma_1, ... \sigma_k \rangle$ and by $\bar{\sigma}_{-i}$ we denote a strategy profile $\bar{\sigma}$ from which the strategy of Process i is ignored and by $(\bar{\sigma}_{-i}, \sigma')$ the strategy profile in which Process i changes to strategy σ'. That is, $\bar{\sigma}_{-i} = \langle \sigma_0, ..., \sigma_{i-1}, \cdot, \sigma_{i+1}, ..., \sigma_k \rangle$ and $(\bar{\sigma}_{-i}, \sigma') = \langle \sigma_0, ..., \sigma_{i-1}, \sigma', \sigma_{i+1}, ..., \sigma_k \rangle$. We define similarly the strategy profile for a subset $P \subseteq \Omega$ of processes and write $\bar{\sigma}_P$ for it. In the strategy profile $\bar{\sigma}_P$, the strategies for processes in P are defined and the strategies for the others are ignored(are replaced by $\cdot$ in $\bar{\sigma}$).

Then, the interaction between processes that follow a certain strategy profile $\bar{\sigma}$ defines an execution in the interaction model $\mathcal{M}$. We write $\mathsf{exec}(\mathcal{M}, \bar{\sigma})$ for the execution in $\mathcal{M}$ that is compatible with all the strategies in the strategy profile $\bar{\sigma}$. Also, for a subset $P \subseteq \Omega$, $\mathsf{exec}(\mathcal{M}, \bar{\sigma}_P)$ consists of the executions in $\mathcal{M}$ that are compatible with all the strategies σ_i for $i \in P$.

Example 2.1.4. *Let us come back to Example 2.1.2. A strategy for Process 0 could be such that he plays "toggle" from states v_0 and v_1 and "skip" whenever the interaction model is in the state v_2. This is formally written as $\sigma_0(hv) = T$ for any $h \in (V\Sigma_0)^*$ if $v \in \{v_0, v_1\}$ and $\sigma_0(hv) = S$ if $v = v_2$.*

If the initial state of the interaction model is v_0, then the strategy described above for Process 0 is represented by the Moore machine in Figure 2.3.

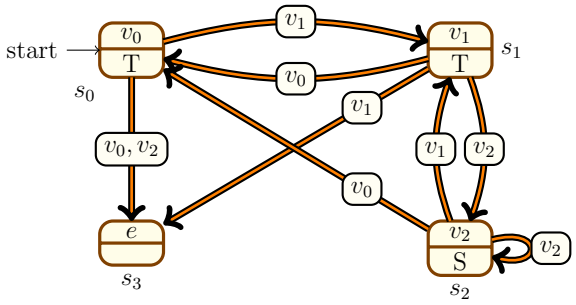


Figure 2.3: Example of strategy as Moore machine

The state s_j for $j \leq 2$ corresponds to the current state v_j in the interaction model. There is an extra error state s_3 reached in case the transition taken in the interaction model is not compatible with the last action of Process 0. The upper label of the states represents the memory needed for the strategy. In this case, since the strategy is memoryless, it is the current state. The lower label of the state is the action taken by Process 0. Finally, the labels on the edges represent the new state in the interaction model after taking the action of Process 0 and some action of the environment.

Let us now consider that Process 1 wants to always turn on the light, but never time out the toggle. This means that he always plays $(\neg t_{out}, l_{on})$ for any history. Formally, $\sigma_1(h) = (\neg t_{out}, l_{on})$ for any $h \in V(\Sigma_1 V)^*$.

Then, from a starting state v_0, the interaction between the two processes defines the execution $\mathsf{exec}(\mathcal{M}, \bar{\sigma}) = (v_0 v_1)^\omega$ alternating between the two states v_0 and v_1.

Remark 2.1.1. *Note that the interaction model is a game arena in which each process represents a player (or agent). Therefore the set $\mathsf{exec}(\mathcal{M})$ corresponds to the plays in the game arena and $\mathsf{exec}(\mathcal{M}, \sigma_i)$ is called the outcome of the strategy σ_i of Player i. In the following, we may mix the two notations and use the name Player or Agent for Process, game arena for the interaction model or play for an execution in the interaction model.*

In some settings, agents do not dispose of the same set of actions on all the states of the interaction model. Therefore, there is defined a function, called protocol, that says which actions are available to an agent at each state. In this case, a strategy is a mapping as presented above, but it has to output an action from the set of actions available at the current state in the model. Such a setting is adopted in the model checker MCMAS[59] where, besides the evolution of the configuration of the system, one is also asked to provide

the possible actions a process can take in each configuration. However, we can always equivalently model an interaction model like this using the definition we provided in this thesis by adding some extra sink state to which transitions are made whenever a process takes an action that is not in his protocol.

2.1.2 Observations

It is not realistic that the processes have perfect information about the state of the interaction model. This may be because, for example, some processes have some private variables that are not visible for the exterior. Therefore, the interaction model where agents have partial observation on states was introduced in [77] and [78]. That is, each process i is equipped with an *indistinguishability* equivalence relation $\sim_i$ over V. Each equivalence class of V induced by $\sim_i$ is an *observation* of Agent i. The equivalence class of a state $v \in V$ from the point of view of Agent i is denoted by $o^i(s)$ and the set of observations of Agent i is denoted by $\mathcal{O}_i$. The relation $\sim_i$ is naturally extended over (finite or infinite) executions: $\rho_1 \sim_i \rho_2$ if $|\rho_1| = |\rho_2|$ and $\forall 0 \leq n \leq |\rho_1|$, $\rho_1[n] \sim_i \rho_2[n]$. We say that an Agent i has *perfect information (is omniscient)* if each equivalence class induced by the relation $\sim_i$ is a singleton.

Two finite sequences $h = v_0 a_i^0 v_1 a_i^1 v_2 a_i^2 ... v_n \in V(\Sigma_i V)^*$ and $h' = v'_0 b_i^0 v'_1 b_i^1 v'_2 b_i^2 ... v'_m \in V(\Sigma_i V)^*$ of states and actions of Agent i are indistinguishable for him (and write $h \sim_i h$) if $n = m$ and $\forall 0 \leq r \leq n$, $v_r \sim_i v'_r$ and $a_i^r = b_i^r$. Then, a strategy σ_i of Agent i is *observation-based* if for any two finite sequences $h, h' \in V(\Sigma_i V)^*$ such as $h \sim_i h'$, we also have $\sigma_i(\pi) = \sigma_i(\pi')$. Therefore, observation-based strategies can be seen as mappings from history of his own actions and the observations he got so far to a next action, i.e., $\sigma_i : \mathcal{O}_i(\Sigma_i \mathcal{O}_i)^* \to \Sigma_i$, where $\mathcal{O}_i$ denotes the set of observations of Agent i over the states of the interaction model $\mathcal{M}$.

Note that in the case of observation-based strategies $\sigma_i : \mathcal{O}_i(\Sigma_i \mathcal{O}_i)^* \to \Sigma_i$, the Moore machine that encodes it, has as input alphabet the set $\mathcal{O}_i$ of observations and outputs actions in Σ_i. Therefore, the state update of the Moore machine is done depending on the new observation of Agent i, i.e., it is defined by $\delta_M : S_M \times \mathcal{O}_i \to S_M$.

Example 2.1.5. *Let us now consider that in Example 2.1.2 Process 0 sees only the light, i.e. $v_1 \sim_0 v_2$ and his environment (Agent 1) is assumed to have perfect information. Note that the strategy σ_0 we defined in Example 2.1.4 for Agent 0 is not observation-based since it proposes different actions in v_1 and v_2 which are indistinguishable for Process 0.*

An observation-based strategy for Process 0 is such that he "toggles" all the time ($\sigma_0(h) = T$ for any $h \in V(\Sigma_0 V)^*$). Therefore, for any indistinguishable history, he plays the same action.

Given an infinite sequence $\pi = o_0^i a_i^0 o_1^i a_i^1 o_2^i a_i^2 ... \in \mathcal{O}_i(\Sigma_i \mathcal{O}_i)^\omega$ of actions and observations of Agent i, we associate with it the set of possible executions of $\mathcal{M}$ that are compatible

with π. Formally, we write $\mathsf{exec}(\mathcal{M}, \pi)$ for the set of executions $\rho = v_0 v_1 \cdots \in \mathsf{exec}(\mathcal{M})$ such that for all $n \geq 0$, $o^i(v_n) = o^i_n$ and there are actions a^n_j for all other agents $j \neq i$ such that $E(v_{n-1}, a_0, \cdots a_k) = v_n$. We also define the traces of π as the set of traces of all executions of $\mathcal{M}$ compatible with π, i.e. $\mathsf{traces}(\pi) = \{\mathsf{trace}(\rho) \mid \rho \in \mathsf{exec}(\mathcal{M}, \pi)\}$.

2.1.3 Synthesis problem: goals and contributions

In this thesis, we are interested in the synthesis problem. Informally, we want to synthesize protocols of some processes such that the interaction with the other processes satisfies some desired properties. These properties are given as a *strategy profile specification*, which is a set of acceptable strategy profiles (i.e. $\Phi \subseteq \Gamma_0 \times \cdots \times \Gamma_k$). We will properly define this notion in the following sections for the particular cases we study.

Let $P \subseteq \Omega$ be the set of processes for which protocols have to be synthesized. We call *environment* the set of players that do not belong to the set P, as we have to take into account all their possible behaviors.

Problem 2.1.1 (Synthesis problem)**.** *Given an interaction model $\mathcal{M}$, a set of processes $P \subseteq \Omega$ and a strategy profile specification Φ, are there strategies for processes in P such that all the strategy profiles containing them are acceptable for Φ? Formally,*

$$\mathsf{INPUT} : \mathcal{M},\ P \subseteq \Omega,\ \Phi \subseteq \Gamma_0 \times \cdots \times \Gamma_k$$
$$\mathsf{OUTPUT} : \textit{Yes}\ \ \textit{iff}\ \ \exists(\sigma_i)_{i \in P}\ \forall(\sigma_i)_{i \notin P}\ \langle \sigma_0, ..., \sigma_k \rangle \in \Phi$$

Note that in the literature, the above problem is referred as the *realizability* problem. Then, the synthesis problem asks to generate strategies that realize the strategy profile specification. In this thesis, the algorithms we provide also output a witness in the case the answer is "yes". Therefore, since we treat both problems in the same time, we only refer to *synthesis* problem.

Definition 2.1.2 (Realizability)**.** *Given an interaction model $\mathcal{M}$ and a set P of processes, the strategy profile specification Φ is realizable iff the output of the synthesis problem is affirmative.*

In this thesis we are interested in studying the Synthesis problem for two kinds of strategy profile specifications. We describe them intuitively in the following.

KLTL Synthesis Problem First, we consider the *KLTL Synthesis Problem* where processes in the set P have a joint objective φ expressed in some epistemic temporal logic that allows one to reason about the temporal evolution of the system, but also express what the agents know about the current state of the system. Therefore, the logic we consider in this case extends the linear temporal logic(LTL) with a knowledge operator $K_i\psi$ which stands for "Agent i knows that the property ψ holds at the current state in

the interaction model". In this setting, the environment is assumed to be completely antagonist having as only objective to prevent the satisfaction of the objective φ. Then, the question is to find a strategy profile for the agents in P so that their joint objective is satisfied against all the possible strategies of the other agents. We give the formal definition of the problem in Section 2.3.

Rational Synthesis Problem Secondly, we consider the case of *Rational Synthesis Problem* where each process has individual objectives and plays rationally. This means that the goal of the processes in the environment is first to satisfy their objectives and then to harm the other processes. For this setting we consider objectives express as LTL formulas and also study some restrictions on the agent's objectives by considering that they are Safety, Reachability, Büchi, Streett or Muller conditions. The rationality of the agents is modeled using solution concepts such as Nash Equilibria, Subgame Perfect Equilibria or Dominant Strategies. Then, synthesis problem asks if there is a strategy profile for the agents in P so that all their objectives are satisfied when the other agents play whatever strategies that are in some equilibrium. We call this setting the *Non-Cooperative* Rational Synthesis Problem.

We also study the *Cooperative* Rational Synthesis Problem where the processes in the environment agree to adhere to some strategy profile which is in equilibrium. In this case, the synthesis problem asks if there is a strategy profile which is in equilibrium (with respect to the considered solution concept) and for which the processes in P satisfy their objectives.

Known Results and Contributions

In the rest of the thesis we define the strategy specifications accordingly and study the two problems considering different kinds of observations for the processes. That is, we will study the two problems when the processes in the set P (to be synthesized) and the ones making up the environment having perfect or imperfect information. An overview of the (decidability) results is sketched in Table 2.1.

Let us first take the case of **KLTL Synthesis Problem**. The synthesis problem with antagonist environment is undecidable already for LTL objective when considering imperfect information for the processes to be synthesized and perfect information for the environment [75]. Since LTL is a fragment of KLTL, it implies the undecidability of KLTL synthesis. To gain decidability, several restrictions have been considered. If there is only one partially informed process to synthesize against an omniscient environment, Van der Meyden and Vardi [89] proved that the KLTL synthesis problem is decidable in 2ExpTime. However, the algorithm they present uses Safra's construction, notoriously known to be resistant to efficient implementations[85]. Therefore, we were interested in avoiding this construction and come with a Safraless approach for a positive fragment of

Synthesis Problem : $\exists(\sigma_i)_{i\in P}\ \forall(\sigma_i)_{i\notin P}\ \langle\sigma_0,...,\sigma_k\rangle \in \Phi$?			
Information		**Strategy Profile Specification**	
Agents in P	**Environment**	**KLTL Synthesis**	**Rational Synthesis**
Perfect info	Perfect info	[54, 36] : 2ExpTime-c	[37, 53]: 2ExpTime-c [28] : Particular objectives
Perfect info	Imperfect info	?	?
Imperfect info	Perfect info	[75]: Undecidable	Undecidable
Imperfect info $\|P\| = 1$	Perfect info	2ExpTime-c [89] : Safra [14] : Safraless	Decidable (on-going work)
Imperfect info	Imperfect info	[75]: Undecidable	Undecidable

Table 2.1: Decidability results for the Synthesis Problem

this logic [14] that leads to an efficient implementation [1]. The algorithm will be detailed in Chapter 4.

When considering perfect information for both processes in P and the processes composing the environment, the KLTL synthesis problem is equivalent to the one when considering only one process to synthesize and one antagonist process for the environment. Also, the KLTL formula can be equivalently written as an LTL formula. This leads to the problem studied in [54] and [36] for linear temporal objectives (LTL) and proven to be 2ExpTime-complete.

Let us now take the case of **Rational Synthesis Problem**. It was introduced by Kupferman et. al [38, 53] in the setting of perfect information for all processes and proved to be 2ExpTime-complete. However, the solution provide is based on strategy logic and it does not allow one to fully understand the computational complexity aspects.Therefore, our objective in [28] is to take particular classes of objectives for the processes (like Safety, Reachability, Büchi, coBüchi, Streett, Rabin, parity and Muller) and better understand the difficulties when solving the problem. Then, through a fine analysis we also obtained better complexity results. The complete results on this topic will be presented in Chapter 5.

Further, we study the rational synthesis problem when some processes have imperfect information. We prove that it is undecidable when all processes have imperfect information. Complete proofs are provided in Section 6.1. The same happens when it is asked to synthesize protocols for two or more processes with imperfect information against an omniscient environment. However, if we are interested in synthesizing the protocol of only one process with imperfect information and the environment is perfectly informed, the problem is decidable and an algorithm to solve this case is given in Chapter 6.

2.2 Execution Specifications

Towards the definition of strategy profile specifications for which we studied the synthesis problem, we first define the execution specifications.

As already mentioned, the allowable behaviors (or executions) are generally characterized using temporal formulas that have to be satisfied or, more concrete, as sets of states that have to be reached, avoided or visited infinitely often. We can use epistemic linear temporal logic (KLTL) in the case of imperfect information, LTL for perfect information case, more particular specifications as safety, reachability, Büchi[16], coBüchi, Strett[84], Rabin[62, 76], parity[67, 32], Muller[68] conditions or even other temporal logics not used in this thesis. We first define some particular conditions and then we give a formal definition of KLTL and LTL logics.

2.2.1 ω-regular objectives

In order to define a "good execution" in the interaction model, one can ask that some states of the interaction model be avoided (safety), eventually reach a certain configuration (reachability) or can express liveness conditions asking that something good happens over and over again (Büchi).

Let $\overline{X}$ stand for the complement of a set $X \subseteq V$ in V, i.e., $\overline{X} = V \setminus X$. In the following we revise the definition of several classes of ω-regular objectives:

- *Safety*: Given a set $S \subseteq V$ of safe states, $\mathsf{Safe}(S) = \{\rho \in V^\omega \mid \mathsf{visit}(\rho) \subseteq S\} = S^\omega$ is the set of infinite sequences of states in S.

- *Reachability*: Given a set $T \subseteq V$ of target states, the dual of safety condition is $\mathsf{Reach}(T) = \{\rho \in V^\omega \mid \mathsf{visit}(\rho) \cap T \neq \varnothing\} = \overline{\mathsf{Safe}(\overline{T})}$.

- *Büchi*: Given a set $F \subseteq V$, $\mathsf{Buchi}(F) = \{\rho \in V^\omega \mid \inf(\rho) \cap F \neq \varnothing\}$ is the set of sequences in which some state in F appears infinitely many times.

- *coBüchi*: Given a set $F \subseteq V$, $\mathsf{coBuchi}(F) = \{\rho \in V^\omega \mid \inf(\rho) \cap F = \varnothing\} = \overline{\mathsf{Buchi}(F)}$ is the set of sequences in which all the states in F appear finitely many times.

- *Streett*: Given a set $\Psi \subseteq 2^V \times 2^V$, the Streett condition for Ψ asks that for all pairs $(R, G) \in \Psi$, if $\rho[k] \in L$ for infinitely many k then $\rho[k] \in R$ for infinitely many k, i.e. $\mathsf{Streett}(\Psi) = \bigcap_{(R,G)\in\Psi}(\mathsf{coBuchi}(R) \cup \mathsf{Buchi}(G))$.

- *Rabin*: Given a set $\Psi \subseteq 2^V \times 2^V$, the Rabin condition for Ψ asks that there is one pair $(L, R) \in \Psi$, such that $\rho[k] \in L$ for infinitely many k and $\rho[k] \in R$ for finitely many k, i.e. $\mathsf{Rabin}(\Psi) = \bigcup_{(L,R)\in\Psi}(\mathsf{Buchi}(L) \cap \mathsf{coBuchi}(R)) = \overline{\mathsf{Streett}(\Psi)}$.

- *Parity*: For a priority mapping $p : V \to \mathbb{N}$, $\mathsf{Parity}(p)$ is the set of infinite sequences such that the least color appearing infinitely often is even, i.e., $\mathsf{Parity}(p) = \{\rho \in V^\omega \mid \min\{p(v) \mid v \in \inf(\rho)\} \text{ is even}\}$.

- *Muller*: Given a Boolean formula μ over V, the Muller condition for μ is the set of infinite sequences such that the set of states appearing infinitely often satisfy the formula μ, i.e., $\mathsf{Muller}(\mu) = \{\rho \in V^\omega \mid \inf(\rho) \vDash \mu\}$.

Note that Büchi and coBüchi conditions are Parity conditions with two priorities. Also, a Büchi (resp. coBüchi) condition F is a Streett condition (V, F) (resp. $(F, \varnothing)$) or a Parity condition $p : V \to \{0, 1\}$ with $p(v) = 0$ if $v \in F$ and $p(v) = 1$ otherwise (resp. $p : V \to \{1, 2\}$ with $p(v) = 1$ if $v \in F$ and $p(v) = 2$ otherwise).

As defined above, in this thesis we consider Muller conditions given implicitly as boolean formulas over the set V of states. However, in the literature, the Muller condition may also be equivalently given explicitly as a set $\Psi \subseteq 2^V$. In this case, the Muller condition Ψ is satisfied on all infinite sequences such that the set of states appearing infinitely often belongs to ψ. That is, $\mathsf{Muller}(\Psi) = \{\rho \in V^\omega \mid \inf(\rho) \in \Psi\}$.

The above conditions can be also expressed in the interaction model $\mathcal{M}$ for sets of atomic propositions. For example, in the case of safety (resp. coBüchi) objectives, the set S (F resp.) can be defined using a set of propositions Y. That is, $S_Y = \{v \in V \mid \tau(v) \subseteq Y\}$. For reachability objectives, the target set of states can be defined by means of a subset of propositions that have to be eventually seen. That is, $T_Y = \{v \in V \mid \tau(v) \cap Y \neq \varnothing\}$.

2.2.2 Epistemic Linear Temporal Logic (KLTL)

In the case of distributed systems, the components may not have access to the global state of the system. Therefore, they must act on the basis of local data to coordinate and ensure the global properties of the system. The incomplete information of each component in distributed synthesis leads then to the notion of what the agent knows about the global state, given its local state.

Epistemic Temporal Logics[44] are logics formatted for reasoning about this kind of multi-agent situations and allow one to express reasoning about the knowledge of agents in distributed systems. One can formulate statements as "if Process i knows that the transaction will be aborted, it should rollback its local contribution and terminate immediately"[90]. They are extensions of temporal logics with knowledge operators K_i for each agent and have been successfully used for verification of various distributed systems in which the knowledge of the agents is essential for the correctness of the system specification.

The logic KLTL extends LTL with the epistemic operator $K_i\phi$, expressing that Agent i knows that the formula ϕ holds.

Definition 2.2.1 (KLTL)**.** *KLTL formulas are defined over the set of atomic propositions $\mathcal{P}$ by the following syntax:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi \mid K_i\varphi$$

in which $p \in \mathcal{P}$ and $\bigcirc$ and $\mathcal{U}$ are the "next" and "until" operators from linear temporal logic. Formulas of the type $K_i\varphi$ are read as "agent i knows that φ holds". The following macros can be defined:

$$\begin{aligned}
\varphi_1 \wedge \varphi_2 &= \neg(\neg\varphi_1 \vee \neg\varphi_2) && (\varphi_1 \text{ and } \varphi_2)\\
\varphi_1 \rightarrow \varphi_2 &= \neg\varphi_1 \vee \varphi_2 && (\varphi_1 \text{ implies } \varphi_2)\\
\varphi_1 \leftrightarrow \varphi_2 &= (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) && (\varphi_1 \text{ equivalent to } \varphi_2)\\
\Diamond\varphi &= true\ \mathcal{U}\varphi && (\text{ eventually } \varphi)\\
\Box\varphi &= \neg \Diamond \neg\varphi && (\text{ always } \varphi)\\
\varphi_1 \mathcal{R} \varphi_1 &= \Box\varphi_1 \vee \varphi_1 \mathcal{U} \varphi_2 && (\varphi_1 \text{ releases } \varphi_2)
\end{aligned}$$

Note that the operator $\Box$ is the dual of $\Diamond$ and $\mathcal{R}$ is the dual of $\mathcal{U}$. Finally, the dual of K_i is the operator P_i which reads as "agent i considers that φ is possible" and is definable as

$$P_i\varphi = \neg K_i \neg\varphi$$

KLTL Semantics The semantics of a epistemic LTL formula φ is defined for an interaction model $\mathcal{M} = \langle \mathcal{P}, \Omega, (\Sigma_i)_{i\in\Omega}, V, V_0, E, \tau \rangle$, a set of executions $R \subseteq \mathsf{exec}(\mathcal{M})$, an execution $\rho = v_0 v_1 \cdots \in R$ and a position $n \geq 0$ in ρ. This is because when the operator K_i is used, the formula $K_i\varphi$ holds on an execution ρ if φ holds on all indistinguishable executions up to position n in the set R. It is defined inductively by:

- $R, \rho, n \vDash p$ if $p \in \tau(v_n)$,
- $R, \rho, n \vDash \neg\varphi$ if $R, \rho, n \nvDash \varphi$,
- $R, \rho, n \vDash \varphi_1 \vee \varphi_2$ if $R, \rho, n \vDash \varphi_1$ or $R, \rho, n \vDash \varphi_2$,
- $R, \rho, n \vDash \bigcirc\varphi$ if $R, \rho, n+1 \vDash \varphi$,
- $R, \rho, n \vDash \varphi_1 \mathcal{U} \varphi_2$ if $\exists m \geq n$ s.t. $R, \rho, m \vDash \varphi_2$ and $\forall n \leq p < m$, $R, \rho, p \vDash \varphi_1$,
- $R, \rho, n \vDash K_i\varphi$ if for all $\rho' \in R$ s.t. $\rho[:n] \sim_i \rho'[:n]$, we have $R, \rho', n \vDash \varphi$.

In particular, Agent i knows φ at position n in the execution ρ, if all other executions in R whose prefix up to position n are indistinguishable from that of ρ, also satisfy φ. We write $R, \rho \vDash \varphi$ if $R, \rho, 0 \vDash \varphi$, and $R \vDash \varphi$ if $R, \rho \vDash \varphi$ for all executions $\rho \in R$. We also write $\mathcal{M} \vDash \varphi$ to mean $\mathsf{exec}(\mathcal{M}) \vDash \varphi$. Note that $\mathcal{M} \vDash \varphi$ iff $\mathcal{M} \vDash K\varphi$.

LTL Linear Temporal Logic(LTL) is a included in KLTL logic and is obtained by removing the knowledge operator K_i. Note that for the LTL formulas, as we do not need to refer to other executions in the set R, we can ignore it and define the semantics on execution ρ in the interaction model $\mathcal{M}$ and positions n. We can also define the semantics of LTL formulas over infinite words $w \in (2^{\mathcal{P}})^{\omega}$. The only difference is that $w, n \vDash p$ if $p \in w[n]$.

Positive KLTL The positive fragment of KLTL (KLTL^{+}) is equivalent to the fragment of KLTL in which formulas contain the knowledge operator K occurring under an even number of negations. This is obtained by straightforwardly pushing the negations down towards the atoms. We denoted this fragment of KLTL by KLTL^{+}. Formally, it is defined as follows.

Definition 2.2.2 (KLTL^{+})**.** *Positive KLTL formulas are defined by the following grammar:*

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \Box \varphi \mid \varphi \mathcal{U} \varphi \mid K_i \varphi$$

for all $0 \leq i \leq k$.

Example 2.2.1. *Consider Example 2.1.2 where Agent 0 doesn't distinguish between states* v_1 *and* v_2 *and the set* R *of executions that eventually loops in* v_0*. Pick any* ρ *in* R*. Then* $R, \rho, 0 \vDash \Box K_0 \Diamond l$*. Indeed, take any position* n *in* ρ *and any other executions* $\rho' \in R$ *such that* $\rho[:n] \sim_0 \rho'[:n]$*. Then since* ρ' *will eventually loop in* v_0*, it will satisfy* $\Diamond l$*. Therefore* $R, \rho, n \vDash K_0 \Diamond l$*, for all* $n \geq 0$.

Using the logic of knowledge, one also can express specifications in security protocols [45, 88]. For example, for sending a message anonymously, it can be specified that all the parties come to know some fact, but all the agents except the sender should not know the identity of the sender [88]. Also, in a voting system, the voter should know the value of his vote, but none of the other participants in the voting protocol should come to know how one voted. Therefore, from the point of view of an attacker a, we can express the secrecy of a vote by the KLTL formula $\Box(\neg\varphi_{same} \rightarrow \bigwedge_v \bigwedge_{vote_v}(vote_v \rightarrow \neg K_a vote_v))$ where v ranges over all voters, the proposition $vote_v$ ranges over all the possible votes of voter v and the formula φ_{same} is a boolean formula that expresses the fact that all the voters voted in the same way. The formula reads as "at any time, if the voters did not vote in the same way, the attacker does not know the vote of the voter v".

Other kind of specifications may be expressed using other temporal logics like CTL or CTL^{*} that are temporal logics allowing to express branching specifications. For example, the CTL formula $\exists \Box \neg \mathit{down}$ may formalize the property that possibly the system never goes down, where the atomic proposition *down* is true in the case the system would fail. This can be expressed using KLTL formula $P_i \Box \neg \mathit{down}$ that has to hold at the initial

state of the interaction model, where P_i is the "possibly" operator. However, there are formulas like $\forall \diamond \forall \Box a$ that assert that on any computation, eventually some state is reached so that all the continuations satisfy $\Box a$. This formula is not expressible in KLTL.

2.3 Synthesis with antagonist environment

Interactive reactive systems (or open systems) are systems that interact with other systems making up their environment. Therefore, the synthesis problem considers there are some processes for which strategies have to be synthesized, and the other processes make up the environment.

We recall that, as defined in Section 2.1.3, given an interaction model $\mathcal{M}$, a set P of processes and a set Φ of acceptable strategy profiles, the synthesis problem asks if there is a strategy profile for the processes in the set P such that for any strategies of the other processes, the resulting strategy profile belongs to Φ.

The classical setting for the synthesis problem considers the environment (the processes that are not in P) that act against the other agents forming the system to synthesize. Its only objective is to falsify the condition that has to be ensured by the processes in P.

The synthesis problem was first formalized by Church [26] in 1962 and asks to synthesize circuits from synthesis requirements expressed in monadic second order theory of one successor. Later, the synthesis problem was extended to distributed systems by Pnueli and Rosner [75]. It asks to synthesize a system made of several components that interact with each other and with the surrounding environment. The (joint) objective of the composed system is to ensure the satisfaction of some requirement expressed in terms of a linear temporal logic formula. This problem has been proven in [75] to be undecidable.

Van der Meyden and Vardi [89] formalize the synthesis problem of a distributed system when the specifications, besides the temporal evolution of the system, may also speak about the knowledge of the involved agents. Intuitively, given the set $P \subseteq \Omega$ to synthesize and considering an antagonist environment (all processes in $\Omega \setminus P$ play against the processes in the set P), the *strategy specification* for a tuple $(\sigma_i)_{i \in P}$ of strategies of processes in P consists on all the strategy profiles that include $(\sigma_i)_{i \in P}$ and whose outcome satisfy some joint objective expressed as an KLTL formula φ. The formula is realizable if there are strategies $(\sigma_i)_{i \in P}$ for processes in P such that the formula holds in the set of executions compatible with $(\sigma_i)_{i \in P}$.

Formally, assuming that is given the set of processes $P \subseteq \Omega$ to be synthesized, the *realizability* problem is defined as follows.

Definition 2.3.1 (Synthesis with antagonist environment)**.** *A KLTL formula φ is realizable in the model $\mathcal{M}$ by processes in $P \subseteq \Omega$ if there is a strategy profile $\bar{\sigma}_P = (\sigma_i)_{i \in P}$*

such as $\mathsf{exec}(\mathcal{M}, \bar{\sigma}_P) \vDash \varphi$.

$$\begin{aligned}&\mathsf{INPUT}: \mathcal{M},\ \varphi,\ P \subseteq \Omega,\\ &\mathsf{OUTPUT}: \textit{Yes}\ \ \textit{iff}\ \ \exists (\sigma_i)_{i\in P}\ :\ \mathrm{exec}(\mathcal{M}, \bar{\sigma}_P) \vDash \varphi\end{aligned}$$

We recall that since $\mathrm{exec}(\mathcal{M}, \bar{\sigma}_P)$ is a set of executions, by $\mathrm{exec}(\mathcal{M}, \bar{\sigma}_P) \vDash \varphi$ we mean that for any execution ρ in $\mathrm{exec}(\mathcal{M}, \bar{\sigma}_P)$, holds $\mathrm{exec}(\mathcal{M}, \bar{\sigma}_P), \rho, 0 \vDash \varphi$.

Note that since the interaction model corresponds to a game arena, the synthesis problem asks to find a joint strategy for the players constituting the distributed system such that they enforce the satisfiability of the given specification φ without communicating their knowledge to each other.

The synthesis problem with antagonist environment and KLTL specifications has been introduced by van der Meyden and Vardi for the processes of the environment having perfect information and only the processes to synthesize being partially informed. Therefore, we can consider that we only have one environment process (let say Process k) and it is to synthesize protocols for processes 0 to $k-1$.

In the literature, the environment processes in the synthesis problem with antagonist environment are always considered to be perfectly informed about the current state of the interaction model. Indeed, whenever one can realize a specification against an omniscient environment, he can realize it against an partial informed environment too.

In the following, we also consider a perfectly informed environment for the KLTL synthesis. Therefore, whenever we speak about the synthesis with antagonist environment *we always consider that there is only one omniscient environment process.*

Example 2.3.1. *Considering again Example 2.1.2, the formula* $\varphi = \Box(K(t) \vee K(\neg t))$ *expresses the fact that the system knows at each step the position of the toggle. As argued in [89], this formula is realizable if the initial set of states in the interaction model is* $\{v_0, v_2\}$ *since both states are labeled with* t*. Then, one strategy of Process 0 is to play first time* T*, action that will lead to* v_1*, and then always play* S *in order to stay in that state. Following this strategy, in the first step the formula* $K(t)$ *is satisfied and in the future states* $K(\neg t)$ *becomes true forever. However, the formula is not realizable if the set of initial states of the interaction model is* $\{v_1, v_2\}$ *since from the beginning the system does not know the value of the toggle.*

Theorem 2.3.1 ([75])**.** *The KLTL synthesis problem is undecidable for two partially informed processes to synthesize and one antagonist omniscient environment.*

The above holds since the problem is already undecidable for two processes to synthesize and specifications expressed using only linear-time temporal logic [75]. The proof is done by Pnueli and Rosner for distributed systems where the processes to synthesize communicate with each other by means of some variables. Each variable has at most one reading and at most one writing process and it is assumed that each variable

which is written by one process is read by another (if any). The variables that have no writing process associated are called input variables and the ones that have no process to read them are called output variables.

There is also an environment that provides some input to the distributed system by writing in some input variables. Then, the processes to synthesize respond by setting some output variables. Each process sets the variables in which it can write depending on the values of variables he reads.

The joint objective of the processes in the distributed system is then to satisfy some specification given as linear time logic formula on the set of input and output variables.

This architecture can also be modeled using an interaction model as presented in this thesis. One configuration of the variables at an instance of time consists of the state of the interaction model. Then, the transition between states is done according to the changes the processes do on the controlled variables.

The undecidability proof is done by reducing from the halting problem on the empty input-tape of deterministic Turing machine. More precisely, for an instance of distributed system and a given deterministic Turing machine M, an LTL specification φ_M is defined, such that it is realizable by the distributed system if and only if M halts on the empty-tape. The reduction is as follows: the configuration of the output variables at an instance of time encodes a legal configuration of M (the tape symbols, the internal state and the special symbol); a configuration of the input variables encode the vocabulary $\{S, N\}$ of M where S stands for "start"(a new configuration) and N for "next"(symbol of the current configuration); and the formula φ_M encodes the behavior of the Turing machine together with the requirement that eventually a terminating configuration is reached.

The above result is based on Peterson and Reif's [77, 72] work on multiplayer games with imperfect information where they prove that, in the games with three or more players, the existence of winning strategies for a set P of players is undecidable if there is no restriction on the information they may have and the players outside P have complete information.

Another immediate proof for the undecidability of LTL realizability in distributed systems results from [31]. The authors prove the undecidability of ATL model checking under the assumption of imperfect information and perfect recall (ATL_{iR} for short). The proof is done by reduction from the non-halting problem of deterministic Turing machines to the existence of strategies for two agents in a three-agent imperfect information interaction model so that the formula $\Box\, ok$ is satisfied, where ok is some atomic proposition corresponding to the non-halting of the Turing machine. The same proof applies to prove the undecidability of LTL synthesis under imperfect information.

Decidable Restrictions for Synthesis with Antagonist Environment

Despite the undecidability result for the KLTL realizability problem, a number of restrictions on the specifications, the number of agents or their information have been identified, for which the problem becomes decidable.

One Process with KLTL Objective Van der Meyden and Vardi [89] proved the decidability of KLTL realizability problem for one process to synthesize and one perfectly informed antagonist process making up the environment. In this case, the problem is viewed as a two-player zero-sum game with an imperfectly informed protagonist and an omniscient adversary.

Theorem 2.3.2 ([89])**.** *The KLTL realizability problem for one agent to synthesize and one omniscient antagonist environment is* 2ExpTime*-complete.*

Hierarchical Systems Later, van der Meyden and Wilke [90] studied some restrictions of the synthesis problem, formalized in [89], that consider the synthesis of several processes. In particular, they study the cases of *hierarchical systems* and *broadcast systems.*

In *hierarchical* model, the processes to be synthesized can be linearly ordered such that each process in the sequence observes at least as much as the preceding ones. This setting is motivated by the models in which there is a hierarchy between agents. It is the case of unclassified, secret and top-secret documents and agents allowed to access a specific level of security. The agents from one level of security clearance are also allowed to read the files from the below levels. Considering that there is some information p that is accessible only to a high level (let us say a top-secret document), we can express properties that ask if the top-secret information is passed to a lower level(say secret) by means of the KLTL formula $\Diamond(K_L(p) \vee K_L(\neg p))$ where K_L is the knowledge operator with respect to the agents in the lower ("secret") level.

In [90] is then proved that synthesis problem with respect to LTL specifications and imperfect information is decidable in hierarchical systems, but it is undecidable even for two processes to synthesize having KLTL specifications. It is the case even if one process to synthesize is omniscient and the other blind.

Theorem 2.3.3 ([90])**.** *The synthesis problem for distributed systems with respect to LTL specifications is decidable in hierarchical systems.*

Van der Meyden and Wilke prove the above theorem by providing an iterative automata-based procedure that receives an instance of the synthesis problem with k processes to synthesize having hierarchical observations. Then, at each step, it transforms the current model in a model with one agent less by removing the less informed process and

passing its decisions to the ones with a more accurate observation. Finally, the problem is reduced to the synthesis of one imperfectly informed process against one antagonist environment and epistemic specification.

Theorem 2.3.4 ([90]). *The synthesis problem for distributed systems with respect to KLTL specifications is undecidable in a system with two processes to synthesize, the first being omniscient, the second being blind. It remains undecidable for the case where the protocol of the blind agent is fixed (so only the protocol for the omniscient agent needs to be synthesized).*

The later theorem is proven by reduction from an undecidable problem in lossy counter machines. Namely, given a lossy counter machine L, is there a natural number n such that there is an infinite run on L starting from the initial configuration $(q_I, 0, ..., 0, n)$ that avoids the forbidden state q_f?

However, the synthesis problem is decidable in hierarchical models when the specification is given as a positive KLTL formula.

Theorem 2.3.5 ([90]). *For specifications in positive KLTL (*KLTL^+*), the synthesis problem in distributed systems is decidable in hierarchical models.*

The algorithm reduces the problem to an instance of the synthesis problem with specifications given as LTL formulas. Thanks to the fact that the knowledge operators have only positive occurrences in the given specification, for any (possibly non-hierarchical) interaction model $\mathcal{M}$, the authors define the new interaction model $\mathcal{M}'$. In $\mathcal{M}'$, at each state, each agent has to say which is the knowledge subformulas he thinks are true by choosing a corresponding action. Then, whenever one agent proposed a knowledge subformula, the next states are labeled with an corresponding atomic proposition and the newly defined LTL specification requires that the subformula is indeed satisfied. Moreover, if in the initial interaction model $\mathcal{M}$ the agents have hierarchical observations, it is also the case in the newly defined model $\mathcal{M}'$.

Broadcast Systems Another restriction for which the synthesis problem is decidable is in the case of broadcast systems. The *broadcast* model implies agents that have some private information that can be shared with the other agents only by means of synchronous simultaneous broadcast to all agents. This is motivated by systems in which components communicate by means of shared bus.

Theorem 2.3.6 ([90]). *The synthesis problem for specifications as KLTL formulas is decidable in broadcast systems.*

***CTL*, *CTL** and *LTL* Objectives** More restricted cases of the synthesis problem were studied in [54] and [36] considering one process to synthesize and one antagonist

environment. In [54] the realizability problem for CTL and CTL^* specifications is studied, independently of the presence of perfect or imperfect information. It is proven to be decidable using automata-theoretic constructions.

Theorem 2.3.7 ([74, 54])**.** *The synthesis problem for one process with either complete or incomplete information against an omniscient environment is* 2ExpTime*-complete for LTL and CTL^* specifications and* ExpTime*-complete for CTL formulas.*

The algorithm proposed in [54] to prove the above theorem uses alternating tree automata. In [36], Filiot, Jin and Raskin propose an antichain based solution for the LTL synthesis under perfect information that avoids alternating tree automata and leads to an efficient implementation in the tool *Acacia*$^+$ [12].

Objectives with Boxes and Diamonds Other restrictions for the synthesis problem transfer from solving two-player zero-sum games. This is because the interaction model corresponds to a game arena and each agent is a player in the corresponding game. Therefore, motivated by the results improvements in model checking of LTL specifications when removing *next*($\bigcirc$) and *until*($\mathcal{U}$) operators, Alur et. al [7, 6] proposed even more restricted specifications in synthesis of one perfectly informed process against one omniscient antagonist environment (two-player zero-sum games with perfect information for both players). That is, they define fragments of linear temporal logic that don't use the next and until operators and obtain Pspace, Exptime and Expspace complexities.

Let X be a set of formulas (a fragment of LTL) and $L_{op_1,...,op_n}(\mathsf{X})$ be the logic obtained from the formulas in X as atomic propositions and the operators $op_1, ..., op_n$. Also, let $\mathcal{B}(\mathsf{X})$ be the logic obtained using only boolean combinations of elements in X. Then, the following holds:

Theorem 2.3.8 ([6],[7])**.** *The synthesis problem of one perfectly informed process against one antagonist omniscient environment is*

- Pspace*-complete for specifications given as $\mathcal{B}(L_{\Diamond,\wedge}(\mathcal{P}))$, $\mathcal{B}(L_{\Box\Diamond}(\mathcal{P}) \cup L_{\Diamond,\wedge}(\mathcal{P}))$ or $L_{\Diamond,\wedge,\vee}(\mathcal{P})$ formulas;*
- Exptime*-complete for specifications given as $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge}(\mathcal{P}))$ formulas;*
- Expspace*-complete for specifications given as $\mathcal{B}(L_{\Diamond,\wedge,\vee}(\mathcal{P}))$ or $\mathcal{B}(L_{\Diamond,\bigcirc,\wedge,\vee}(\mathcal{P}))$ formulas and*
- 2Exptime*-complete for specifications given as $L_{\Box,\Diamond,\wedge,\vee}(\mathcal{P})$ formulas.*

Request-Response Objectives Finally, we remind the restriction in [20] where specifications are given as request-response conditions. That is, a specification is given as a set $\psi \in 2^V \times 2^V$ of pairs (Q, P). We say that the specification is satisfied along an

execution if for each pair $(Q, P) \in \psi$, whenever a state in Q is visited, either the respective state also belongs to set P, or later a state in P is also visited. In terms of LTL formula, assuming that states are labeled with propositions Q and P corresponding to the sets in ψ, the synthesis specification is written as $\bigwedge_{(Q,P)\in\psi} \square(Q \rightarrow \diamond P)$.

Theorem 2.3.9 ([20]). *The realizability problem of one perfectly informed process is* EXPTIME*-complete for request-response conditions.*

2.4 Multi-component rational environment

In the previous section, we assumed that there is only one "hostile" environment whose only target is to prevent the realizability of the specification.

However, in real life, systems are made of several components having their own goals, other then making the system fail. For example, there may be clients that interact with a server. The approach taken in the field of game theory is to assume that agents that interact in a computational system are *rational*. That is, their first goal is to achieve their objectives and then to harm the others.

Therefore, in [38] the notion of *rational synthesis* under perfect information was introduced. It asks to synthesize a strategy for one process that interacts with a multi-component rational environment. The objectives of each process is expressed as a LTL formula and the rationality of the processes is modeled using solution concepts as dominant strategies, Nash Equilibria, and subgame perfect equilibria.

In this thesis we generalize the definition to the case of several processes to synthesize and discuss the decidability of the synthesis problem also when agents have imperfect information.

Let us first recall several solution concepts that are used in game theory to model the rationality of the processes and then define the rational synthesis problem in two different settings (cooperative and non-cooperative).

2.4.1 Modeling Rationality

Let $(\varphi_i)_{i\in\Omega}$ be the tuple of LTL objectives for Processes $0, 1, ..., k$, where φ_i is the objective of Process i. The *payoff* of a strategy profile $\bar{\sigma}$ is the vector $pay(\bar{\sigma}) \in \{0, 1\}^{k+1}$ defined by $pay(\bar{\sigma})[i] = 1$ if and only if $\mathsf{exec}(\mathcal{M}, \bar{\sigma}) \vDash \varphi_i$. We write $pay_i(\bar{\sigma})$ for Process i's payoff $pay(\bar{\sigma})[i]$. Intuitively, a Process i has payoff 1 if the execution(s) compatible with the strategy profile satisfies the objective φ_i. We also use the game theory vocabulary and say that Process i wins.

For a strategy profile $\bar{\sigma}$ and a history h, the strategy σ_i^h denotes the strategy of Process i after the history h and is defined as $\sigma_i^h(h_1) = \sigma_i(h \cdot h_1)$. Then, $pay_i(\bar{\sigma})_h = pay_i(\bar{\sigma}^h)$ is the payoff of Process i when the strategy profile $\bar{\sigma}$ is played after history h happened.

Also, W_i denotes the set of *winning states* (also called *winning set* or *winning region*) for Process i to achieve objective φ_i. That is, W_i is the set of states v so that if the initial state of the interaction model is v, Process i has a strategy to ensure his objective against all the other agents that would play against him to satisfy the objective $\neg\varphi_i$.

As the notions we use to model the rationality of Processes come from game theory, we also use the notion of players to refer to processes. There are several solution concepts that can be used to model rationality, as follows.

Nash Equilibria A strategy profile $\bar{\sigma} = (\sigma_i)_{i\in\Omega}$ is a *Nash equilibrium* in the model $\mathcal{M}$ if no player can improve his payoff by (unilaterally) switching to a different strategy. Formally,

Definition 2.4.1 (Nash Equilibrium)**.** *Given an interaction model $\mathcal{M}$, a strategy profile $\bar{\sigma}$ is a Nash equilibrium iff for all players $i \in \Omega$ and all strategies σ_i' of Player i,*

$$pay_i(\bar{\sigma}_{-i}, \sigma_i') \leq pay_i(\bar{\sigma})$$

Given a strategy profile $\bar{\sigma}$, the strategy σ_i' of Player i is a *profitable deviation* from $\bar{\sigma}$ iff Player i can improve his payoff by unilaterally deviate from $\bar{\sigma}$ and play σ_i'. Formally, σ_i' is a profitable deviation for Player i from $\bar{\sigma}$ iff $pay_i(\bar{\sigma}_{-i}, \sigma_i') > pay_i(\bar{\sigma})$.

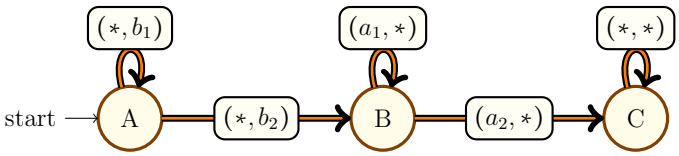


Figure 2.4: Example of interaction model

Example 2.4.1. *For simplicity, let us take the setting where the players have perfect information. Consider the two-player interaction model (or game arena) in Figure 2.4 where Player 0 plays the actions $\{a_1, a_2\}$ and Player 1 plays actions $\{b_1, b_2\}$ and both players have as objective to reach state C (infinitely often). As observed, Player 0 has complete control of state B while Player 1 controls state A by choosing the next state. Let us consider the following strategy profiles:*

- *Player 1 plays action b_2 in state A and Player 0 plays action a_2 in state B. Hence, both players win.*
- *Player 1 plays action b_2 in state A and Player 0 plays action a_1 in state B. Hence, both players lose since the execution remains in state B forever.*
- *Player 1 plays action b_1 in state A and Player 0 plays action a_1 in state B. Hence, both players lose.*

Note that only the first and the last strategy profiles are Nash equilibria. In the first case, this is because both players win and therefore none of them can improve the payoff. In the third case however, both of the players lose, but they cannot improve their payoff since if Player 1 changes the strategy and plays b_2 in state A, the play loops in state B forever.

The second strategy profile is not a Nash equilibrium since Player 0 could deviate and play a_2 from state B making both players win.

Chatterjee et al. [25] showed that any infinite multiplayer game with ω-regular winning conditions and perfect information has a Nash equilibrium in pure strategies, and they also gave an algorithm for computing one. Moreover, the Nash Equilibria can be computed in nondeterministic polynomial time when players have parity objectives.

Theorem 2.4.1 ([25])**.** *Every turn-based perfect information deterministic game with ω-regular objectives has a Nash equilibrium with pure strategy profile.*

Subgame Perfect Equilibria The notion of Nash equilibrium is the classical solution concept that models the rationality of agents. However, in the definition of a Nash equilibrium it is not taken into account that players can change their strategies during a play. A solution concept that considers this is the *subgame perfect equilibrium* for which the choice of strategies should be optimal from all possibles histories of the game, including the histories not reachable by the strategy profile.

We say that a strategy profile $\bar{\sigma} = (\sigma_i)_{i\in\Omega}$ is a *Subgame Perfect equilibrium* if for every history of the game, no agent $i \in \Omega$ has an incentive to unilaterally deviate from the strategy σ_i. Formally,

Definition 2.4.2 (Subgame Perfect Equilibrium)**.** *Given an interaction model $\mathcal{M}$, a strategy profile $\bar{\sigma}$ is a Subgame Perfect Equilibrium if $\forall h \in \mathsf{Prefs}(\mathcal{M})$, $\forall i \in \Omega$, $\forall \sigma_i'$ a strategy of Player i,*

$$pay_i(\bar{\sigma}_{-i}, \sigma_i')_h \leq pay_i(\bar{\sigma})_h$$

Example 2.4.2. *Going back to Example 2.4.1, the first strategy profile defined, which is a Nash equilibrium, is also a subgame perfect equilibrium. However, the last strategy profile defined (which is a Nash equilibrium too) is not a subgame perfect equilibrium since Player 0 can improve his payoff after the play reaching state B by deviating and playing the action a_2.*

Theorem 2.4.2. *Any multiplayer perfect information game with ω-regular winning objectives has a subgame perfect equilibrium.*

Theorem 2.4.2 is a direct result of [86] that proves that every ω-regular multiplayer game is equivalent to a multiplayer parity game. In the case of LTL formulas, it has double exponential size. Finally, for any multiplayer parity game, there is a finite-state subgame perfect equilibrium that can be computed in exponential time [86].

Dominant Strategies A *dominant strategy* is a strategy that, by playing it, the corresponding player can never lose, regardless of the strategies of the other players. A *dominant strategy profile* is a profile in which all the strategies are dominant. Formally,

Definition 2.4.3. *Given an interaction model $\mathcal{M}$, $\bar{\sigma} = (\sigma_i)_{i\in\Omega}$ is a dominant strategy profile if $\forall i \in \Omega$ and $\forall \bar{\sigma}'$ an strategy profile with $\sigma'_i \neq \sigma_i$, it holds that*

$$pay_i(\bar{\sigma}') \leq pay_i(\bar{\sigma}'_{-i}, \sigma_i)$$

Example 2.4.3. *In the game from Example 2.4.1, we can see that there is no dominant strategy for any player and therefore no dominant strategy profile. Indeed, if Player 1 plays the action b_2 and go to state B, Player 1 may play action a_1 and make both players lose. The same holds for the point of view of Player 0. If he chooses the strategy that plays a_2 from state B, Player 1 may play the action b_1 from state A and never give the chance to Player 0 to win.*

Non-deviating processes to synthesize Since the problem studied in this section is to synthesize some processes (the ones in some given set $P \subseteq \Omega$) against a multi-component environment, Processes in the set P in the interaction model are supposed to not deviate from a given strategy. Therefore, we introduce the following notions.

Definition 2.4.4 (P-fixed equilibrium)**.** *A strategy profile $\bar{\sigma} = (\sigma_i)_{i\in\Omega}$ is a P-fixed Nash equilibrium if $pay(\bar{\sigma}_{-i}, \sigma'_i) \leq pay(\bar{\sigma})$ for all players $i \in \Omega \setminus P$ and all strategies σ'_i of Process i.*

In other words, it is a Nash equilibrium in which processes in P are not changing their strategy. Observe that any Nash equilibrium $(\sigma_i)_{i\in\Omega}$ is a P-fixed equilibrium, but the converse may not hold. Similarly, we define the notions of P-fixed Subgame perfect equilibrium and P-fixed dominant strategies profile.

2.4.2 Rational Synthesis

Rational synthesis has been introduced in [38, 53] in two different settings for the solution concepts considered above and perfect information for all players. Let γ be a solution concept from above (i.e., γ is "Nash equilibrium", "subgame perfect equilibrium" or "dominant strategies").

The *non-cooperative rational synthesis*[53] assumes that, since the components of the environment are rational, they may play any strategy that is a γ-equilibrium. The problem follows the general definition we gave for the synthesis problem in Problem 2.1.1. That is, one has to output (if it exists) a strategy profile $(\sigma_i)_{i\in P}$ for agents in P which has to be winning against all the possible strategy profiles which are γ-equilibria and include strategies $(\sigma_i)_{i\in P}$ for players in P. In this setting, the strategy specification for the

synthesis problem consists of all the strategy profiles that if they are P-fixed γ-equilibria, they are satisfy the objectives of all processes in P.

Definition 2.4.5 (**Non-Cooperative Rational Synthesis (NCRSP)**). *Given an interaction model $\mathcal{M}$ with winning objectives $(\varphi_i)_{i\in\Omega}$, a set $P \subseteq \Omega$ of processes to synthesize and a solution concept γ, are there strategies $(\sigma_i)_{i\in P}$ for Player in P s.t. for any P-fixed γ-equilibrium $\bar{\sigma} = \langle \sigma_0, \ldots, \sigma_k \rangle$, we have $pay_i(\bar{\sigma}) = 1$ for all $i \in P$?*

$$\begin{aligned}&\mathsf{INPUT}: \mathcal{M},\ (\varphi_i)_{i\in\Omega},\ P \subseteq \Omega,\\ &\mathsf{OUTPUT}: \textit{Yes}\ \ \textit{iff}\ \ \exists(\sigma_i)_{i\in P}\ \forall(\sigma_i)_{i\notin P}\ (Eq_{\gamma,P}(\bar{\sigma}) \rightarrow \bigwedge_{i\in P} pay_i(\bar{\sigma}) = 1)\end{aligned}$$

where the function $Eq_{\gamma,P}(\bar{\sigma})$ outputs "true" if $\bar{\sigma}$ is P-fixed γ-equilibrium.

In [38] *cooperative rational synthesis* is also considered. It assumes that the environment cooperates with the processes to be synthesized in the sense that its components agree to play a γ-equilibrium that is winning for processes in P (if it exists). In other words, in the cooperative setting, one assumes that once a γ-equilibrium winning for agents in P is proposed, all the players will adhere to the suggested strategies. In the cooperative setting, the synthesis problem is defined by the formula $\exists(\sigma_i)_{i\in P}\ \exists(\sigma_i)_{i\notin P}\ \langle \sigma_0, ..., \sigma_k \rangle \in \Phi$ where Φ is the strategy profile specification consisting of all the P-fixed γ-equilibria for which the objectives of all processes in P are satisfied.

Definition 2.4.6 (**Cooperative Rational Synthesis (CRSP)**). *Given an interaction model $\mathcal{M}$ with winning objectives $(\varphi_i)_{i\in\Omega}$, a set $P \subseteq \Omega$ of processes to synthesize and a solution concept γ, is there a P-fixed γ-equilibrium $\bar{\sigma}$ s.t. $pay_i(\bar{\sigma}) = 1$ for all $i \in P$?*

$$\begin{aligned}&\mathsf{INPUT}: \mathcal{M},\ (\varphi_i)_{i\in\Omega},\ P \subseteq \Omega,\\ &\mathsf{OUTPUT}: \textit{Yes}\ \ \textit{iff}\ \ \exists(\sigma_i)_{i\in P}\ \exists(\sigma_i)_{i\notin P}\ (Eq_{\gamma,P}(\bar{\sigma}) \wedge \bigwedge_{i\in P} pay_i(\bar{\sigma}) = 1)\end{aligned}$$

where the function $Eq_{\gamma,P}(\bar{\sigma})$ outputs "true" if $\bar{\sigma}$ is P-fixed γ-equilibrium.

Example 2.4.4. *As an example, consider again the two-agent interaction model of Figure 2.4 with Alice (Agent $0 \in P$) and Bob (Agent $1 \notin P$) having perfect information and playing the actions $\{a_1, a_2\}$ and $\{b_1, b_2\}$ respectively. Let us take reachability objectives for them given by the set $R_0 = \{B\}$ and $R_1 = \{C\}$ and Nash equilibrium as solution concept. Remember that Alice has complete control over the state B and Bob controls the states A and C. Consider Alice's strategies σ_0 which consists in looping forever in state B, and σ_0' which eventually goes to state C.*

Let Bob cooperate by playing the strategy σ_1 that goes to state B (making Alice win). Both strategy profiles $\langle \sigma_0, \sigma_1 \rangle$ and $\langle \sigma_0', \sigma_1 \rangle$ are solutions to the cooperative setting: under the first strategy profile Bob loses but cannot get better payoff by deviating, and under the second one Bob wins. Strategy σ_0 is not a solution to the non-cooperative setting, because

Bob could stay forever in state A (according to a strategy σ_1'): The profile $\langle \sigma_0, \sigma_1' \rangle$ is a 0-fixed Nash equilibrium because even by deviating and going to state B Bob would still lose, and it is losing for Alice. However, σ_0' is a solution to the non-cooperative setting: The only 0-fixed Nash equilibria in that case are when Bob eventually move to state B, making him and Alice win.

Perfectly informed agents

In [38] and [53], Kupferman et al. introduce and study the problem of rational synthesis assuming that all agents have perfect information. It is equivalent to considering that the set P consists of only one process (say Process 0) and there are k environment components that play according to a solution concept γ.

The main contribution of the original papers is to propose and to motivate the definitions above. The only computational complexity results given in those papers are for the setting when all agents composing the system and the environment have perfect information over the sets of the interaction model and they are as follows:

Theorem 2.4.3 ([38, 53])**.** *The cooperative and non-cooperative rational synthesis problems are* 2ExpTime*-complete for specifications expressed in linear-time temporal logic (LTL) and all processes having perfect information.*

The result above matches exactly the complexity of classical zero-sum two-player LTL synthesis [74]. The lower bound is due to the fact that the synthesis problem of one process against one antagonist environment when both agents have perfect information is a particular case of the rational synthesis. Indeed, consider an interaction model with two agents and Agent 0 has the objective φ to achieve whatever the will of the other agents. We can translate this problem in an instance of the rational synthesis problem on the same interaction model, but associate the objective $\neg\varphi$ to the environment (Agent 1). Then, asking if there is a strategy σ_0 for Agent 0 that it is winning for all 0-fixed γ-equilibria containing σ_0 is the same with asking if there is a winning strategy for Agent 0 against all strategies of the environment.

The upper bound is obtained by reductions to the model-checking problem of formulas in a Nested-Goal fragment of Strategy Logic(Sl[ng]) [65]. More precisely, the authors define some Sl[ng] formulas that characterize the 0-fixed γ-equilibria. Then, they define polynomial length Sl[ng] formula that hold in the interaction model if and only if there is a solution for the rational synthesis problem. In the case of cooperative setting, the formula expresses that there is a strategy profile $\bar{\sigma}$ that is 0-fixed γ-equilibrium and satisfies the specification φ_0. On the other hand, the formula for the non-cooperative setting expresses the existence of a strategy σ_0 for Agent 0 so that all executions compatible with a strategy profile that contains the strategy σ_0 for Agent 0 and is a

0-fixed γ-equilibrium, satisfy the objective φ_0. We give more details about the reduction in Chapter 5 after we have formally defined the fragment of Strategy Logic in Chapter 3.

Imperfect information for Processes

Following the general definition of the interaction model, let us consider the rational synthesis when agents have imperfect information. The goal is to synthesize some partially informed processes from a given set $P \subseteq \Omega$ that act in the rational environment made of the other partially informed processes in the interaction model $\mathcal{M}$.

This case is motivated by the situation when there is a distributed system made of several components and the goal is to introduce some other components that have their own objectives. In distributed systems, it may be the case that each system has some private local states that are not visible to the exterior.

In this setting, the rational synthesis problem gets easily undecidable. In Chapter 6 we study the problem considering different capabilities (imperfect/perfect information) for both processes in P and the processes making up the environment. We prove that the problem is undecidable in case when both the processes to be synthesize and the environment have imperfect information. Also, the problem remains undecidable if there are at least two processes to be synthesized against a perfectly informed environment. This is because it corresponds to distributed synthesis problem (Definition 2.3.1) against one perfectly informed antagonist environment which, according to Theorem 2.3.1, is undecidable.

However, we gain decidability if we consider only one partially informed process to synthesize against a multi-component omniscient but rational environment. In Chapter 6 we provide an algorithm that reduces the problem to the distributed synthesis problem with hierarchical observations which, by Theorem 2.3.3, is decidable.

3. Games, Logics and Automata: Tools and Techniques

In this section, we shall first revise some results regarding two-players zero-sum games that are used in this thesis.

Then, we present some temporal logics that contain modalities allowing one to reason about agents' strategies in a multi-component interaction model. These logics can also be used to express the synthesis problems in terms of model-checking problems. That is, one can reduce the synthesis problem to the model-checking problem of some formulas in these logics.

Finally, we revise the notions of automata on words and trees with the accepting conditions used in this thesis. We also provide some complexity results and algorithms used to test the emptiness of tree automata that are the main tool used in this thesis. The main use of the tree automata is in the classical approach to solve the synthesis problem. Also, they are used by model-checking algorithms that verify that some formula in a logic of strategies holds.

3.1 Two-players Zero-sum Games

Two-players zero-sum games consist of an interaction model as defined in Section 2.1 with two players: a protagonist having some objective $\Theta \subseteq V$ and an opponent that tries to falsify Θ. Therefore, on each execution in the game, only one of the two players win (zero-sum).

We consider here perfect information turn-based games where there is only one initial state. Formally a game is defined as $\mathcal{G} = \langle \Omega = \{A, B\}, V = V_A \uplus V_B, v_0, E, \Theta \rangle$.

As already mentioned in Remark 2.1.1, the set of plays in the game $\mathcal{G}$ corresponds to the set of executions in the underlying interaction model. We denote by $\mathsf{exec}(\mathcal{G})$ the set of plays in $\mathcal{G}$. For a play $\rho \in \mathsf{exec}(\mathcal{G})$, we say that Player A (the protagonist) *wins* if $\rho \in \Theta$ and Player B wins otherwise. A player has a *winning strategy* σ from a state $v \in V$, if all plays (executions) compatible with σ are won by him. We define the *winning region* for Player $X \in \{A, B\}$, denoted W_X, as the set of all states $v \in V$ from which Player X has a winning strategy. Clearly, since the game is zero-sum, at most one player has a winning

strategy from a state v. Therefore, $W_A \cap W_B = \varnothing$.

A game is called *determined* if the winning regions for both players form a partition of the set V. That is, $V = W_A \uplus W_B$. Therefore, from any state in the two-player zero-sum game, it is exactly one player that has a strategy to win against any strategy of the other player. Martin showed in [61] that the two-players zero-sum games where the protagonist's objective is a Borel subset of V^ω are determined.

Theorem 3.1.1. *[61] Every two-player zero-sum game with Borel winning condition is determined.*

We do not give the definition of Borel winning conditions, since they are not used in this thesis. We will use the determinacy for objectives given Safety, Reachability, Büchi, coBüchi, Streett, Rabin, Muller conditions (crf. Section 2.2.1) or, more generally, as LTL formulas over the set of states V. All of the mentioned winning conditions are particular Borel conditions.

Theorem 3.1.2. *Given a two-player zero-sum game $\mathcal{G}$ and a state $v \in V$, deciding if Player A has a winning strategy from v is*

- *[23, 42, 18]* PTIME*-complete for Safety, Reachability, Büchi or coBüchi conditions.*
- *in* UP $\cap$ coUP *for Parity conditions*
- *[33]* NP*-complete for Rabin conditions.*
- *[33]* coNP*-complete for Streett conditions.*
- *[48]* PSPACE*-complete for Muller conditions.*
- *[74]* 2EXPTIME*-complete for LTL conditions.*

Moreover, in the games where the condition is given as Safety, Reachability, Buchi, coBuchi,Rabin and Parity, if Player A has a winning strategy in the game $\mathcal{G}$ to win, he also has a memoryless winning strategy. The property does not hold for Streett and Muller conditions. However, if Player A has a winning strategy in a Muller game, he has one that can be represented by a finite-state Moore machine.

First cycle games First cycle games [8] (FCG) are played on a finite interaction model by two players until a state is repeated, and a simple cycle is formed. Player A wins the play if the sequence of labels of nodes of the cycle satisfies some fixed cycle property, and otherwise Player B wins.

Theorem 3.1.3. *[8] The problem of Player A has a winning strategy in a first cycle game is* PSPACE*-complete.*

3.2 Logics of Strategies

The components of a system are formally modeled as modules that interact with their environment. Then, a desired property is verified on all such interactions (executions).

When studying interaction models (or multiplayer games), one may also be interested in analyzing the behavior of individual components and sets of components. That is, reasoning about the strategies of the agents/players and also analyzing the outcome of these strategies. Therefore, logics that focus on the strategic behavior of agents in multi-agent systems were developed[5, 71, 64].

In this section, we recall some temporal logics that, besides the properties on executions, contain modalities that allow one to reason about the strategies of the agents.

We first revise the alternating-time temporal logic (ATL^*)[5] that allows one to implicitly quantify on the strategies of agents. Then, we recall the nested-goal strategy logic (SL[NG])[24, 66] that explicitly quantifies on strategies and allows one to bind exactly one strategy to each player at a time. Finally, we recall the epistemic strategy logic (SLK)[17] that also allow one to reason about the knowledge of imperfectly informed agents.

3.2.1 Alternating-Time Temporal Logic

One of the most important development in this field is *Alternating-Time Temporal Logic* (ATL^* for short), introduced in [5], that allows one to reason about strategies of agents with temporal goals. It disposes of *strategy modalities* $\langle\langle P \rangle\rangle$ and $[\![P]\!]$ where $P \subseteq \Omega$ is a set of agents. These strategy modalities are used to encode cooperation and competition between agents in order to achieve some objectives.

Operator $\langle\langle P \rangle\rangle$ existentially quantifies on the strategies of agents in the set P. It intuitively say that the players in P can cooperate and ensure some property. For example, one can express properties such as "Agents 0 and 1 cooperate to ensure that the system (with more than two agents) never fails" by the formula $\langle\langle 0, 1 \rangle\rangle \Box \neg\, fail$.

On the other hand, the operator $[\![P]\!]$ is the dual of $\langle\langle P \rangle\rangle$ and universally quantifies on the strategies of agents in P. The formula $[\![P]\!]\varphi$ intuitively says that the coalition P cannot avoid some property φ.

In ATL^* there are two types of formulas: *state formulas*, whose satisfaction is related to specific states, and *path formulas*, that are related to specific computations. Formally, the formulas in ATL^* are defined by the following syntax:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle P \rangle\rangle\psi$$
$$\psi := \varphi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi \mathcal{U} \psi$$

where $p \in \mathcal{P}$ is an atomic proposition, φ is a state formula and ψ is a path formula. The operator $[\![P]\!]$ is the dual of $\langle\langle P \rangle\rangle$ and is defined by $[\![P]\!]\varphi = \neg\langle\langle P \rangle\rangle\neg\varphi$. We also define macros

$\Box\psi$, $\varphi_1 \wedge \varphi_2$, $\psi_1 \wedge \psi_2$ or $\psi_1 \mathcal{R} \psi_2$ as in the case of KLTL formulas (Section 2.2.2).

The semantics of ATL^* formulas is defined over states v and executions $\rho \in \mathsf{exec}(\mathcal{M})$ respectively (depending on the type of formula) in some interaction model $\mathcal{M}$ as follows:

- $v \vDash p$ iff $p \in \tau(v)$
- $v \vDash \neg\varphi$ iff $v \nvDash \varphi$
- $v \vDash \varphi_1 \vee \varphi_2$ iff $v \vDash \varphi_1$ or $v \vDash \varphi_2$
- $v \vDash \langle\!\langle P \rangle\!\rangle \psi$ if there exist a strategy profile $\bar{\sigma}_P$ for the agents in P such that for all paths $\rho \in \mathsf{exec}(\mathcal{M}[v], \bar{\sigma}_P)$, holds $\rho \vDash \psi$.
- $\rho \vDash \varphi$ iff $\rho[0] \vDash \varphi$
- $\rho \vDash \neg\psi$ iff $\rho \nvDash \psi$
- $\rho \vDash \psi_1 \vee \psi_2$ iff $\rho \vDash \psi_1$ or $\rho \vDash \psi_2$
- $\rho \vDash \bigcirc\psi$ iff $\rho[1:] \vDash \psi$
- $\rho \vDash \psi_1 \mathcal{U} \psi_2$ iff there is $i \geq 0$ s.t $\rho[i:] \vDash \psi_2$ and for all $0 \leq j < i$, holds $\rho[j:] \vDash \psi_2$.

Note that the semantics for the path formulas is defined as for LTL formulas (Section 2.2.2) with respect to executions in the interaction models.

Model-Checking problem for ALT^* asks, given an interaction model $\mathcal{M}$ and an ATL^* (state) formula φ, for the set of states of $\mathcal{M}$ that satisfy φ. Formally,

Definition 3.2.1 (Model-Checking of ATL^*)**.** *Given $\mathcal{M} = \langle \mathcal{P}, \Omega, (\Sigma_i)_{i\in\Omega}, V, V_0, E, \tau \rangle$ and an ATL^* (state) formula φ,*

$$INPUT \ : \ \mathcal{M}, \varphi$$
$$OUTPUT \ : \ X \subseteq V \ \textit{s.t.} \ \forall v \in X, \ \textit{holds} \ v \vDash \varphi$$

The model-checking problem for ATL^* is closely related to synthesis problem for linear-time formulas and, in the case of perfect information setting, it requires doubly exponential time.

Depending on the perfect/imperfect information of the agents in the interaction model and their capabilities to recall the past (perfect/imperfect recall), different complexity results for the model-checking problem are obtained. We recall that in this thesis we consider agents to have perfect recall. That is, their strategies depend on the entire history.

Theorem 3.2.1 ([5])**.** *Model-Checking problem for ATL^* formulas under perfect information and perfect recall for agents is* 2ExpTime*-complete, even in turn-based interaction models. For ATL^* formulas of bounded length, the model checking problem is* Ptime*-complete.*

The strategies in [5] are strategies with perfect recall. That is, as in the setting of this thesis, they are mappings that associate with every finite prefix in the interaction model, an action. The algorithm that solves the ATL^* model-checking problem under perfect information and perfect recall translates each instance of the problem to the problem of checking the nonemptiness of some tree automaton.

Unfortunately, if agents have imperfect information and perfect recall, the model-checking problem is undecidable [31].

Theorem 3.2.2 ([31]). *The model-checking problem for ATL^* under imperfect information and perfect recall is undecidable.*

The result directly comes from [31] where the authors prove the undecidability of ATL (a fragment of ATL^*) model-checking. The proof is based on a direct simulation of Turing machines by three-agent interaction models $\mathcal{M}$ where agents have imperfect information and perfect recall. Then, the Turing machine does not halt on the empty word if and only if the formula $\langle\langle 0,1 \rangle\rangle \Box\, ok$ holds in the initial state of the interaction model. The atomic proposition ok holds as long as the transitions in the interaction model correspond to some computation step in the Turing machine.

However, if only the existence of memoryless strategies is asked, the model-checking problem becomes decidable in PSPACE[82].

Theorem 3.2.3 ([82]). *The model-checking problem for ATL^* with imperfect information and memoryless strategies is* PSPACE*-complete.*

The algorithm for Theorem 3.2.3 simply enumerates all the memoryless strategies with imperfect information and solves a CTL^* model-checking problem for them.

Due to the syntax of LTL, the synthesis problem for some set P of processes against an antagonist environment and an LTL objective φ can be modeled as the model-checking problem of the formula $\langle\langle P \rangle\rangle \varphi$. However, this does not improve the already known results because when agents have perfect recall, the model-checking problem is 2EXPTIME for perfect information setting and undecidable if agents have imperfect information.

Limitations Unfortunately, ATL^* suffers from the strong limitation that strategies are treated only implicitly. That is, one can express the fact that some agent (or coalition) has a strategy to enforce some property, but cannot refer explicitly to some strategy. A consequence is the impossibility to ask that two players share in different contexts the same strategy, assuming that they have the same actions available.

Moreover, the alternating temporal logic does not allow one to express properties like "For all strategies of Agent 0, there is a strategy of Agent 1 such that for any strategy of Agent 2 a formula φ holds". This is since ATL^* is forgetful, in the sense that each quantifier deletes the previously selected strategies. The previous statement

is not equivalent to writing $v_0 \vDash [\![0]\!]\langle\!\langle 1\rangle\!\rangle[\![2]\!]\varphi$. According to the semantics of ATL^*, the fact that the initial state v_0 of $\mathcal{M}$ satisfies $[\![0]\!]\langle\!\langle 1\rangle\!\rangle[\![2]\!]\varphi$ is equivalent to the fact that for any strategy σ_0 of Agent 0 there is at least one executions $\rho \in \text{exec}(\mathcal{M}, \sigma_0)$, such that $\rho \vDash \langle\!\langle 1\rangle\!\rangle[\![2]\!]\varphi$. This is equivalent to $v_0 \vDash \langle\!\langle 1\rangle\!\rangle[\![2]\!]\varphi$ and therefore there is a strategy σ_1 of Agent 1 such that for all $\rho' \in \text{exec}(\mathcal{M}, \sigma_1)$, $\rho' \vDash [\![2]\!]\varphi$, i.e., $v_0 \vDash [\![2]\!]\varphi$. The last formula says that Agent 2 cannot avoid φ from the initial state of the interaction model, but it does not ask that this happens for any strategy of Agent 0, which was the case in the informal description of the requirement. The quantification over the strategies of previous agents is lost when a subformula containing another quantification is evaluated.

Extensions of ATL^* to Remember Strategies Due to the *forgetfull* semantics of ATL^*, in the literature several extensions of the logic were proposed, that make strategy quantifiers not to "forget" the strategies fixed by the previous quantifiers.

First, $IATL$ [3] extends ATL (fragment of ATL^*) with strategy contexts that store the previously selected strategies. The strategies of agents are considered *irrevocable* in the sense that it requires agents to commit to a strategy, which they are not allowed to modify in the sequel. However, the strategies of the agents are *memoryless*. For this setting, the following complexity result is proved.

Theorem 3.2.4 ([3])**.** *Model-checking problem for $IATL$ under perfect information is in P^{NP} w.r.t. the size of the model and the formula.*

The logic ATL^*_{sc} introduced in [15] relaxes the restrictions considered in $IATL$ in the following sense. A new operator that removes the strategies of the agents from the current context is introduced. Then, agents keep on following their strategies until the formula explicitly removes the strategy from the current context, or it is replaced by a new one. Moreover, the authors consider *memoryfull* strategies for agents that depend on the entire history of the current execution.

To solve the model-checking problem for ATL^*_{sc}, in [29] an algorithm based on alternating tree automata is proposed, that gives the following complexity result.

Theorem 3.2.5 ([29, 56])**.** *Model-checking problem for ATL^*_{sc} under perfect information is $(d+1)$*ExpTime*-complete even for turn-based models, where d is the number of nested strategy quantifiers in the formula being checked.*

Using this fragment, one can express properties as the one above by the intuitive formula $[\cdot 0\cdot]\langle\cdot 1\cdot\rangle[\cdot 2\cdot]\varphi$. As in the case of ATL^*, the operator $\langle\cdot P\cdot\rangle$ expresses the existence of a strategy for the agents in $P \subseteq \Omega$ and the operator $[\cdot P\cdot]$ quantifies universally on the strategies of agents in P.

Also, ATL^*_{sc} can express the existence of a Nash Equilibrium as

$$\langle\cdot\Omega\cdot\rangle\Big[\bigwedge_{i\in\Omega} \neg\varphi_i \to \neg\langle\cdot i\cdot\rangle\varphi_i\Big]$$

where φ_i is the LTL objectives for Agent $i \in \Omega$ in the interaction model $\mathcal{M}$.

3.2.2 Nested-Goal Strategy Logic (SL[NG])

All limitations of ATL^* have led to the introduction of *Strategy Logic*(SL) that treats strategies as explicit first order objects. It was first introduced in [24] for two agents and then extended in [66] for multiagent architectures. Considering the set Γ of strategies of agents, [24] introduces the two operators $\langle\langle x \rangle\rangle$ and $[\![x]\!]$ that quantify over the strategy variable $x \in \mathsf{Var}$. Also, the "binding" operator (i, x) is introduced to allow to associate with Agent i the strategy bound to variable x. The model-checking problem for SL is non-elementary hard.

Theorem 3.2.6 ([64])**.** *The model-checking problem for (perfect information) Strategy Logic is d-*ExpSpace*-hard, where d is the alternation depth of the formulas, i.e., d is the maximum number of quantifiers switches $\langle\langle x \rangle\rangle [\![y]\!]$ and $[\![x]\!]\langle\langle y \rangle\rangle$.*

Therefore, in [64] fragments of strategy logic that are expressive enough were considered, for which are obtained better complexities. One of these fragments is the Nested-Goal Strategy Logic (SL[NG]). The idea is that, when there is a quantification over a variable used in a goal, we are forced to quantify over all free variables of the inner subformula containing the goal itself, by using a quantification prefix. In this way, the subformula is build only by nesting and Boolean combinations of goals.

Syntax Formally, the set of SL[NG] formulas is defined over the set of atomic propositions $\mathcal{P}$ and the set Var_i of strategy variables of each agent $i \in \Omega$ by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \langle\langle x_i \rangle\rangle \varphi \mid [\![x_i]\!]\varphi \mid \flat(\vec{x})\varphi$$

where a formula $\langle\langle x_i \rangle\rangle \varphi$, where $x_i \in \mathsf{Var}_i$ ranges over Γ_i(the strategies of Agent i), states that there is a strategy for Agent i such that the formula φ holds; the formula $[\![x_i]\!]\varphi$ states that for all strategies of Agent i formula φ holds; and, finally, for a tuple $\vec{x} \in \Gamma_0 \times \cdots \times \Gamma_k$, $\flat(\vec{x})$ is a *binding prefix* that intuitively binds each Agent i to a strategy variable $x_i \in \mathsf{Var}_i$.

Semantics To define the semantics of a Sl[ng] formula, we need the notion of *assignment* which is a partial function $\chi : \mathsf{Var} \cup \Omega \rightharpoonup \Gamma$ that maps variables and agents to strategies. For an assignment χ, $\chi[l \mapsto \sigma]$ denotes a new assignment so that, for the element $l \in \mathsf{Var} \cup \Omega$, $\chi[l \mapsto \sigma](l) = \sigma$ and $\chi[l \mapsto \sigma](l') = \chi(l')$ for all $l' \in (\mathsf{Var} \cup \Omega) \setminus \{l\}$.

Then, given an interaction model $\mathcal{M}$, a Sl[ng] formula φ, a state $v \in V$ and an assignment χ, we write $\mathcal{M}, \chi, v \vDash \varphi$ to indicate that the formula φ holds at state v in $\mathcal{M}$ under the assignment χ. The semantics of Sl[ng] logic for the newly introduced operators is inductively defined as follows:

- $\mathcal{M}, \chi, v \vDash \langle\langle x_i \rangle\rangle \varphi$ if there is a strategy σ_i for Agent i s.t. $\mathcal{M}, \chi[x_i \mapsto \sigma_i], v \vDash \varphi$

- $\mathcal{M}, \chi, v \vDash [\![x_i]\!]\varphi$ if for all strategies σ_i for Agent i, it holds $\mathcal{M}, \chi[x_i \mapsto \sigma_i], v \vDash \varphi$
- $\mathcal{M}, \chi, v \vDash \flat(\vec{x})\varphi$ if $\mathcal{M}, \chi[0 \mapsto \chi(x_0)]...[k \mapsto \chi(x_k)], v \vDash \varphi$

Intuitively, by means of operator $\flat(\vec{x})$, the agents in Ω are committed to the strategies assigned to the variables in $\vec{x}$. Then, we say that a formula φ is satisfied in $\mathcal{M}$ if there is an assignment χ such that $\mathcal{M}, \chi, v_0 \vDash \varphi$ for all initial states $v_0 \in V_0$.

Theorem 3.2.7 ([64]). *The model-checking problem for* SL[NG] *(under perfect information) can be solved in* $(d+1)$EXPTIME*, where* d *is the alternation depth of the specification, i.e., the maximum number of quantifiers switches* $\langle\!\langle x \rangle\!\rangle [\![y]\!]$ *and* $[\![x]\!] \langle\!\langle y \rangle\!\rangle$.

As already mentioned in [66], ATL^* corresponds to the one-alternation fragment of strategy logic. Therefore, it is also possible to express the synthesis problem with antagonist environment in terms of a model-checking problem. Let x_i be the second order variable that ranges over the strategies of Agent i. Then, the LTL specification φ is realizable by the set $P = \{0, 1, ..., k-1\}$ of agents in the interaction model $\mathcal{M}$ if and only if the SL[NG] formula

$$\langle\!\langle x_0 \rangle\!\rangle \langle\!\langle x_1 \rangle\!\rangle ... \langle\!\langle x_{k-1} \rangle\!\rangle [\![x_e]\!] \, \flat(x_0, ..., x_{k-1}, x_e)\varphi$$

is true in $\mathcal{M}$, where the environment consists of Agent e. Again, the above model-checking problem can be solved in 2EXPTIME since the alternation depth equals to 1.

Moreover, SL[NG] can easily express the existence of equilibria in multi-player games. That is, considering a multi-agent interaction model $\mathcal{M}$ and an LTL objective φ_i for each agent i, there exists a Nash equilibrium if the forumula

$$\langle\!\langle x_0 \rangle\!\rangle \langle\!\langle x_1 \rangle\!\rangle ... \langle\!\langle x_k \rangle\!\rangle \bigwedge_{i \in \Omega} [\![x_i']\!] \big(\flat(x_0, ..., x_{i-1}, x_i', x_{i+1}, ..., x_k)\varphi_i \rightarrow \flat(\vec{x})\varphi_i \big)$$

holds. We give more details about the use of this result in solving the rational synthesis problem in Chapter 5.

3.2.3 Epistemic Strategy Logic (SLK)

In [17] was introduced a variant of strategy logic that integrates the epistemic concepts. That is, the agents are considered to have imperfect information due to their limited access only to their local variables. Then, the syntax of the strategy logic with knowledge includes operators that allows one to reason about the knowledge of agents.

Unfortunately, since the ATL^* model-checking is undecidable for imperfect information and perfect recall (Thm. 3.2.2), the memoryless setting is adopted for the logic SLK, where the agent's local state do not necessary include the local history of the run. Consequently, the strategies are also memoryless.

Theorem 3.2.8 ([17]). *The model-checking problem for SLK in systems with memoryless strategies and imperfect information can be solved in* PSPACE.

In [17] the model checker MCMAS-SLK is also presented. The tool is based on MCMAS, which is an open-source model checker for ATL(a fragment of ATL^*) and epistemic operators.It reads a file containing the interaction model specified in terms of agents which are described by their local state, a protocol specifying the available actions at each local state and a local evolution function returning the state change according to the joint actions of all agents. Also, the file contains the SLK formula that has to be verified. The tool reads the input interaction model described in ISPL(Interpreted System Programming Language) and verifies the SLK formula. In the case the formula is not satisfied for the universally quantified variables, the algorithm implemented using OBDDs returns a counterexample that can be used to refine the interaction model or even the formula to be verified.

3.3 Automata on infinite Words

An *infinite word automaton* over some (finite) alphabet Λ is a tuple $\mathcal{A} = (\Lambda, Q, q_0, \delta, \alpha)$ where Q is the finite set of states, $q_0 \subseteq Q$ is the set of initial states, $\alpha \subseteq Q$ is the set of final states (accepting states) and $\delta \subseteq Q \times \Lambda \times Q$ is the transition relation.

For all $q \in Q$ and all $\ell \in \Lambda$, we let $\delta(q, \ell) = \{q' | (q, \ell, q') \in \delta\}$. We let $|\mathcal{A}| = |Q| + |\delta|$. We say that $\mathcal{A}$ is *deterministic* if $\forall q \in Q, \forall \ell \in \Lambda, |\delta(q, \ell)| \leq 1$. It is *complete* if $\forall q \in Q, \forall \ell \in \Lambda, \delta(q, \ell) \neq \varnothing$. Without loss of generality, in the rest of the thesis the word automata are always complete. This is because we can always complete an automaton by adding missing transitions to some sink state.

A *run* of the automaton $\mathcal{A}$ over an infinite input word $w = \ell_0\ell_1\ell_2...$, is a sequence $r = q_0q_1q_2... \in Q^\omega$ so that $(q_i, \ell_i, q_{i+1}) \in \Delta$ for all $i \geq 0$. We denote by $Runs_{\mathcal{A}}(w)$ the set of runs of $\mathcal{A}$ on w and consider different accepting conditions for infinite word automata and name the infinite word automata according to the used accepting condition. Let $B \in \mathbb{N}$. A word $w \in \Lambda^\omega$ is accepted by $\mathcal{A}$ if (according to the accepting condition):

Büchi	$: \exists r \in Runs_{\mathcal{A}}(w)$ s.t. $\inf(r) \cap \alpha \neq \varnothing$
Universal coBüchi	$: \forall r \in Runs_{\mathcal{A}}(w),\ \inf(r) \cap \alpha = \varnothing$
Universal B-coBüchi	$: \forall r \in Runs_{\mathcal{A}}(w), \forall q \in \alpha,\ \mathsf{visit}(r, q) \leq B$

where, as for executions in the interaction model, $\inf(r) = \{q \in Q \mid \forall n \geq 0, \exists m \geq n \text{ s.t. } r[m] = q\}$ is the set of states that appear infinitely often in r, $\mathsf{visit}(r) = \{q \in Q \mid \exists n \geq 0 \text{ s.t. } r[n] = q\}$ is the set of states that appear at least once along r and $\mathsf{visit}(r, q)$ is the number of times the state r appears along the rune r.

The set of words accepted by $\mathcal{A}$ with the non-deterministic Büchi (resp. universal coBüchi and B-coBüchi) accepting condition is denoted by $\mathcal{L}_{nb}(\mathcal{A})$ (resp. $\mathcal{L}_{uc}(\mathcal{A})$ and

$\mathcal{L}_{uc,B}(\mathcal{A})$). We say that $\mathcal{A}$ is a *non-deterministic* Büchi word automaton (NBW) if the first acceptance condition is used, a *universal coBüchi word automaton* (UCW) for the second condition and that $(\mathcal{A}, B)$ is an *universal B-coBüchi word automaton* (UBCW) if the last one is used.

Theorem 3.3.1 ([93])**.** *For any LTL formula φ over the set of atomic propositions Λ, there is an effective construction of a nondeterministic Büchi word automaton $\mathcal{A}_\varphi$ such that $\forall w \in (2^\Lambda)^\omega$,*

$$w \in \mathcal{L}_{nb}(\mathcal{A}_\varphi) \text{ iff } w \vDash \varphi$$

The first procedure to translate LTL formulas to word automata is proposed in [93] and in the worst case outputs automata with $\mathcal{O}(2^n)$ states, where n is the number of subformulas in the given formula. The algorithm starts by generating a node for each (maximally consistent) set of subformulas of the property φ. Therefore, the procedure is not reasonable for implementation, since it immediately reaches the worst case complexity.

Other more efficient algorithms were proposed in [52] and [40]. The algorithm in [52], proposed as a basis for an implementation, is an improvement of the procedure in [93] by using an incremental algorithm that allows the construction of only reachable states. Although the worst case remains exponential, the construction often achieves a substantial reduction of the number of generated states.

The algorithm proposed in [40] is the first algorithm that works on-the-fly in the sense that the automaton is generated as needed in the automatic protocol verification process. It is based on the algorithm in [52] and translates an LTL formula into a Büchi automaton. The algorithm was implemented in Standard ML of New Jersey and statistics showing that the improved algorithm constructs substantially smaller automata are provided. A similar algorithm is used in the model checker SPIN [47] in the on-the-fly verification program. Later, in [30] improvements (LTL2AUT) based on simple syntactic techniques of the existing algorithm from [40] were proposed.

In [39] (later improved in [9]) another efficiently implemented algorithm was proposed, which is motivated by the impossibility to use SPIN to generate a Büchi automaton from a formula containing several fairness conditions. The procedure first builds a generalized Büchi automaton which is a Büchi automaton, with labels and accepting conditions on transitions instead of states. This, and some additional simplifications on the intermediary automata, allow the construction of a Büchi automaton with significantly less states then by using the classical direct construction. The algorithm is implemented in the tool LTL2BA which seems to be much more efficient then the previous existent implementations, in computational time and used memory.

LTL to Universal coBüchi Automata Using the above theorem, given an LTL formula φ, we can translate it into an equivalent universal co-Büchi word automaton $\mathcal{A}_\varphi$. This can be done with a single exponential blow-up by first negating φ, then translating

$\neg\varphi$ into an equivalent non-deterministic Büchi word automaton, and then dualizing it into a universal coBüchi word automaton [36, 55]. That is, the states and transitions of the automaton remain unchanged and only the set of final states is seen as a coBüchi condition. Indeed, there is a path in the automaton that visits infinitely often the set F of accepting states if and only if it does not hold that all the paths visit finitely often the set F of rejecting states.

Corollary 3.3.1 ([36, 55])**.** *For any LTL formula φ over the alphabet Λ, there is an effective construction of a universal coBüchi word automaton $\mathcal{A}_\varphi$ that accepts exactly the infinite words over Λ satisfying φ.*

In the literature there are also used other accepting conditions for the word automata as Streett, Rabin, parity or Muller and algorithms to *determinize* a nondeterministic automaton are studied. More precisely, Safra proposed in [79] a procedure to transform a nondeterministic Büchi word automaton with n states into an equivalent deterministic Rabin word automaton having $2^{\mathcal{O}(n \log n)}$ states and n pairs. However, Safra's construction was shown quite resistant to efficient implementations [85]. The tool presented in [85] implements Safra's construction for determinizing Büchi word automata and also provides determinization routines for Streett and Rabin automata by converting them first to Büchi automata.

Later, Piterman proposes in [73] an algorithm that takes a nondeterministic Streett word automaton and produces an equivalent deterministic parity word automaton. The construction is very similar to Safra's construction, but thanks to some optimization, the total number of states of the resulting automaton slightly reduces.

Muller and Schupp procedure in [70], which was developed to remove alternation from tree automata (see Section 3.4.3), can also be applied to determinize nondeterministic Streett word automata and obtain deterministic Streett automata. As we do not use results on word automata determinization in this thesis, we don't detail them here, but refere the reader to the cited papers. More insight will be given on Muller-Schupp construction for tree automata in the following section.

3.4 Trees and Tree Automata

3.4.1 Infinite Trees

Given a finite set Υ of directions, a Υ*–tree* is a prefix-closed set $T \subseteq \Upsilon^*$, i.e., if $x \cdot c \in T$, where $c \in \Upsilon$, then $x \in T$. The elements of T are called *nodes* and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$, for $c \in \Upsilon$, are the *successors* of x. A node x is a *leaf* if it has no successor in T, formally written $\forall c \in \Upsilon, x \cdot c \notin T$. The tree T is *complete* if for all nodes, there are successors in all directions, formally, $\forall x \in T, \forall c \in \Upsilon, x \cdot c \in T$, that

is, $T = \Upsilon^*$. Finite and infinite *branches* π in a tree T are naturally defined, respectively, as finite and infinite sequences of directions in T starting from the root node.

Given an alphabet Λ, a Λ*−labeled* Υ*−tree* is a mapping $t : T \to \Lambda$ that maps each node of the Υ-tree T to a letter in Λ. We call T the support of the labeled tree t denoted $\mathsf{Supp}(t)$. Then, in a labeled tree t, an infinite (resp. finite) branch π induces an infinite (resp. finite) sequence of labels and directions in $(\Lambda.\Upsilon)^\omega$ (resp. $(\Lambda.\Upsilon)^*\Lambda$).

Strategies as trees We can use trees to encode strategies. For instance, for a set of Agent i's actions Σ_i and a set of observations $\mathcal{O}_i$, a strategy $\sigma_i : \mathcal{O}_i(\Sigma_i\mathcal{O}_i)^* \to \Sigma_i$ of Agent i can be seen as a complete $(\Sigma_i \cup \{\#\})$-labeled $\mathcal{O}_i$-tree where the label of the root is the special character $\#$ meaning that the Agent i does not play. This corresponds to the choice of the initial states in the interaction model and then the first observation is communicated to the agents. The rest of the nodes in the tree encoding the strategy are labeled with elements of Σ_i. More precisely, for a node $x = o_0^i o_1^i ... o_n^i \in \mathcal{O}_i^+$, $t(x) = \sigma_i(o_0 t(o_0) o_1 t(o_0 o_1) o_2 ... o_n)$. We call *strategy trees* the trees with this structure.

Trees as Moore Machines As it was defined for strategies in Chapter 2, a complete Λ-labeled Υ-tree t can be encoded by a (possible infinite state) Moore machine $M_t = \langle \Upsilon, \Lambda, S_M, s_0, \delta_M, g_M \rangle$ with input alphabet Υ and output alphabet Λ. Note that the composition of the two functions δ_M^* and g_M represents exactly a Λ-labeled Υ-tree. That is, for any Λ-labeled Υ-tree t, we can define the equivalent Moore machine M_t so that the function δ_M^* associates to each vertex x from t, a state in the Moore machine and it holds that $\forall x \in \mathsf{Supp}(t)$, $t(x) = g_M(\delta_M^*(x))$.

3.4.2 Alternating Automata on Infinite Trees

A finite *alternating* tree automaton[69, 91] that runs over Λ-labeled Υ-trees is a tuple $\mathcal{T} = \langle \Lambda, \Upsilon, Q, q_0, \delta, \alpha \rangle$ where Λ is the finite alphabet, Υ is the set of directions, Q is the set of states, $q_0 \in Q$ is the initial state, $\alpha \subseteq Q^\omega$ is the accepting condition and $\delta : Q \times \Lambda \to \mathcal{B}^+(Q \times \Upsilon)$ is the transition relation (assumed to be total) where $\mathcal{B}^+(Q \times \Upsilon)$ is the set of positive boolean formulas over $Q \times \Upsilon$, i.e., boolean formulas over $Q \times \Upsilon$ using only operators $\wedge$ and $\vee$ where we also allow the formulas $true$ and $false$. Therefore, the transition relation maps a state and an input letter to a formula that suggests a new configuration of the automaton. As in this thesis we use tree automata to accept strategy trees, we assume that the tree automata run over complete trees.

Example 3.4.1. *In order to exemplify an alternating tree automaton, let us take the example of [58]. Let the alphabet $\Lambda = \{a, b\}$ and the language consisting of binary trees (the set of directions is $\Upsilon = \{0, 1\}$) so that below each node labeled with "a" eventually appears some node labeled with "b" on some branch. More precisely, whenever there is $x \in \Upsilon^*$ s.t. $t(x) = a$, there is an $y \in \Upsilon^*$ s.t. $t(xy) = b$.*

To accept the trees defined above, we build an alternating tree automaton that has two states $Q = \{q, q_b\}$ where the state q is used to run over the tree and whenever a node is labeled with the letter "a", the automaton duplicates itself to the state q_b that is used to guess a path on which "b" appears below the current node. The automaton has as initial stat $q_0 = q$ and the transition relation is defined as follows:

$$\delta(q, a) = (q, 0) \wedge (q, 1) \wedge ((q_b, 0) \vee (q_b, 1))$$
$$\delta(q, b) = (q, 0) \wedge (q, 1)$$
$$\delta(q_b, a) = (q_b, 0) \vee (q_b, 1)$$
$$\delta(q_b, b) = true$$

The state q always reproduces itself in both successor nodes by the formula $(q, 0) \wedge (q, 1)$. This means that it continues to run on both subtrees from the current node. The state q_b on the other hand, before a node labeled with the letter "b" appears, guesses a path by choosing a successor on which it continues the search for a "b" using the transition $\delta(q_b, a) = (q_b, 0) \vee (q_b, 1)$. Whenever a "b" appears in a label along the guessed path in the tree, the automaton stops searching using the formula true.

Then, the accepting condition of the automaton is the coBüchi condition defined by the set $F = \{q_b\}$ meaning that on each branch of the tree, the state q_b should appear a finite number of times. In other words, whenever the automaton is duplicated in a state q_b, it eventually has to reach the letter "b" and stop the search.

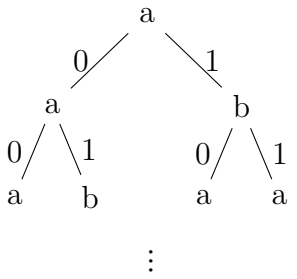
Runs in a Tree Automaton The alternating tree automata runs on Λ-labeled complete Υ-trees. A *run* of $\mathcal{T}$ on a Λ-labeled Υ-trees t is a $(Q \times \Upsilon^*)$-labeled $\mathbb{N}$-tree r such that

- $r(\epsilon) = (q_0, \epsilon)$ and
- for all nodes $x \in \mathsf{Supp}(r)$ such that $r(x) = (q, h)$, there is a possible empty set $S = \{(q_1, c_1), ..., (q_n, c_n)\}$ such that S satisfies $\delta(q, t(h))$ and for all $1 \leq m \leq n$, $x \cdot m \in \mathsf{Supp}(r)$ and $r(x \cdot m) = (q_m, h \cdot c_m)$.

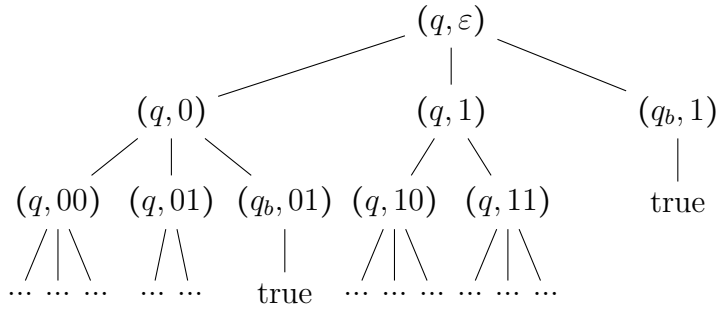
A run is called *accepting* if and only if all its branches are in the accepting set α. Particular Büchi, coBüchi, Rabin or Parity conditions can be defined. Intuitively, the Büchi (resp. coBüchi) condition asks that each branch of the run r passes through an accepting set $F \subseteq Q$ infinitely (resp. finitely) often. The B-coBüchi condition is similar to the co-Büchi condition, but there is a bound B that is imposed on the maximal number of times some bad states are seen. A Rabin condition is defined by a set $\Psi \subseteq 2^Q \times 2^Q$ and asks that along all branches π of a run, there is at least one pair $(L, R) \in \Psi$ so that π visits infinitely often L and finitely often R. Finally, a Parity condition is defined by a priority function $p : Q \to \mathbb{N}$ that associates to each state $q \in Q$ a priority (also called color). Then,

it asks that, along each branch, the smallest priority appearing infinitely often is even. Also, a tree automaton is a *safety* automaton if the winning condition consists on all the sequences in Q^ω that avoid a certain set S of states, i.e., $\alpha = Q^\omega \setminus Q^* S Q^\omega$. Depending on the accepting condition X used on the branches of the trees, the automaton is called a X tree automaton.

Then, the automaton $\mathcal{T}$ accepts a tree t if there is an accepting run on t in $\mathcal{T}$. We denote by $\mathcal{L}_\alpha(\mathcal{T})$ the trees for which there is an accepting run in $\mathcal{T}$.



(a) A labeled binary tree t



(b) A run of the automaton from Example 3.4.1 on the tree from 3.1a

Figure 3.1: Example of a tree and a run on it

Example 3.4.2. *In Figure 3.1 is an example of a tree and one run on it in the automaton defined in Example 3.4.1. From the initial state q in the root of the accepting run, is taken the transition $(q, a, \{(q,0),(q,1),(q_b,1)\})$ meaning that the automaton continues to explore the tree on both directions with $(q,0)$ and $(q,1)$ and guesses that by taking direction 1 it will find a "b" (since the label of the root of the tree t is a). Then, from the node 0 of the tree, from the state q (node $(q,0)$) the run uses the same transition $(q, a, \{(q,0),(q,1),(q_b,1)\})$ since $t(0) = a$. From the node 1 and the state q (the node $(q,1)$ in the run) the automaton simply continues to explore the tree t since $t(1) = b$. Finally, from the node 1 in t and state q_b the automaton stops the search since it is already in a node labeled with "b". The rest of the run is built in the same way depending on the labels of the tree t on which the automaton runs.*

Then, the automaton accepts the tree t if on each branch of the run in Figure 3.1b we see a finite number of states q_b.

Non-deterministic Tree Automata *Non-deterministic* tree automata are particular case of alternating tree automata. A tree automaton is non-deterministic if for each state $q \in Q$ and each letter $c \in \Lambda$, the transition relation $\delta(q,c)$ in disjunctive normal form does not contain two pairs (c, q_1) and (c, q_2) in the same disjunct. Therefore, the transition relation is a mapping $\delta : Q \times \Lambda \to 2^{\Upsilon \to Q}$ that maps any pair of states and labels to a set

of mappings from directions to states (states in which are sent the children of the current node).

Example 3.4.3. *An example of nondeterministic tree automaton is one that has to check that in a binary tree as in Figure 3.1a exists at least one node labeled with the letter "b". The automaton needs two states q and q_b. The initial state is q_b with the meaning that the automaton waits for a letter "b". The state q is the "good" state from which nothing is expected.*

From the state q_b the automaton has to guess the subtree in which it will find the letter "b" and chooses the transition that goes to state q_b on the corresponding direction and to state q in the other one. When the letter "b" is reached, from the state q_b, the automaton goes in both directions to the state q. Once the state q is reached, the automaton just continues to explore the tree with the state q. Formally, the transition relation is

$$\begin{aligned}
&\delta(q_b, a) = ((q_b, 0) \wedge (q, 1)) \vee ((q, 0) \wedge (q_b, 1)) \\
&\delta(q_b, b) = (q, 0) \wedge (q, 1) \\
&\delta(q, \ell) = (q, 0) \wedge (q, 1) \quad , \ \forall \ell \in \{a, b\}
\end{aligned}$$

The automaton accepts if there is a run on the input tree such that on each branching of the run the state q appears infinitely often. I.e., the Büchi condition $F = \{q\}$ is satisfied.

Universal Tree Automata A tree automaton $\mathcal{T}$ is called *universal* if for each state $q \in Q$ and each letter $c \in \Lambda$, the transition relation $\delta(q, c)$ does not contain disjunctions. That is, $\delta : Q \times \Lambda \times \Upsilon \to 2^Q$ assumed to be total and if the tree automaton is in some state q at some node x labeled by some $\ell \in \Lambda$, it will evaluate, for all $c \in \Upsilon$, the subtree rooted at $x \cdot c$ in parallel from all the states of $\delta(q, \ell, c)$.

Example 3.4.4. *To illustrate an universal tree automaton, let take a $\{a, b\}$-labeled binary tree and ask that whenever one "a" appears, it is followed later by a "b" on all branches of the subtree. The difference from Example 3.4.1 is that the automaton has to check in both directions for a letter "b" after the letter "a" appears. That is,*

$$\begin{aligned}
&\delta(q, a) = (q, 0) \wedge (q, 1) \wedge (q_b, 0) \wedge (q_b, 1) \\
&\delta(q_b, a) = (q_b, 0) \wedge (q_b, 1) \\
&\delta(q, b) = (q, 0) \wedge (q, 1) \\
&\delta(q_n, b) = true
\end{aligned}$$

The accepting condition is, as in Example 3.4.1, the coBüchi set $F = \{q_b\}$.

Deterministic Tree Automata We say that a tree automaton $\mathcal{T}$ is *deterministic* if the transition relation is of the form $\delta : Q \times \Lambda \to (\Upsilon \to Q)$, i.e., it maps any pair of states

and labels to one mappings from directions to states. In this case, we equivalently say that the transition relation is of the form $\delta : Q \times \Lambda \times \Upsilon \to Q$.

Example 3.4.5. *In order to verify that on each branch of a given $\{a, b\}$-labeled binary tree exists at least one letter "b", we define the deterministic automaton with two states q_b and q with the initial state q_b meaning that the letter "b" was not seen yet. Then, if the label on the current node of the tree is "a", the automaton duplicates itself in both children with the same state q_b. Whenever the label "b" appears in a node, the automaton goes to state q where it remains forever. Formally, the transition relation is defined by:*

$$\begin{aligned}
&\delta(q_b, a) = (q_b, 0) \wedge (q_b, 1) \\
&\delta(q_b, b) = (q, 0) \wedge (q, 1) \\
&\delta(q, \ell) = (q, 0) \wedge (q, 1) \quad , \ \forall \ell \in \{a, b\}
\end{aligned}$$

Then, the automaton accepts if on all branches, the state q appears an infinite number of times (or equivalently, the state q_b appears a finite number of times).

Similarly to word automata, when the automaton $\mathcal{T}$ has a Büchi condition, we may also write $\mathcal{L}_{ub}(\mathcal{T})$, $\mathcal{L}_{nb}(\mathcal{T})$ or $\mathcal{L}_{ab}(\mathcal{T})$ for its language, depending if the Büchi tree automaton is universal (UBT), nondeterministic (NBT) or alternating (ABT). For the other conditions, we proceed similarly. For example $\mathcal{L}_{uc}(\mathcal{T})$ stands for the language of the universal co-Büchi tree automaton (UCT) $\mathcal{T}$ and $\mathcal{L}_{uc,B}(\mathcal{T})$ stands for the language of the universal B-co-Büchi tree automaton (UBCT) $(\mathcal{T}, B)$ where the final set can be visited at most B times.

Theorem 3.4.1. *The Muller, Rabin, Streett and parity tree conditions are equivalent, in the sense that a tree automaton using any condition may be converted into another automaton using one of the others, but conversion to any condition, except Muller, costs states.*

Proof. Since the parity condition is a special case of Muller, Rabin and Streett conditions, it suffices to explain how to transform a Muller tree automaton $\mathcal{T}$ into another tree automaton that uses parity conditions. The construction is based on *least appearance record (LAR)*[43], which allows us to identify states in Q appearing infinitely often. It consists of a deterministic parity word automaton $\mathcal{A}_\alpha$ that accepts the infinite sequences of states in the automaton $\mathcal{T}$ that satisfy the Muller accepting condition. Then, we take the product between the tree automaton $\mathcal{T}$ and the deterministic word automaton $\mathcal{A}_\alpha$ built for the accepting condition in $\mathcal{T}$. The acceptance condition in the resulting automaton is given by the parity condition of the automaton $\mathcal{A}_\alpha$. However, the automaton $\mathcal{A}_\alpha$ has an exponential number of states in the number of states of the automaton $\mathcal{T}$. ∎

3.4.3 Emptiness of Alternating Tree Automata

Different approaches in solving the synthesis problem use alternating tree automata to characterize the possible solutions of the problem. The synthesis problem of a reactive system that satisfies a desired specification is often reduced to checking the emptiness of an alternating tree automaton accepting all possible strategies that realize the specification [74]. However, in order to test the emptiness of an alternating tree automaton, the removal of alternation is needed, which is usually done by reductions to non-deterministic tree automata. In general, this reduction involves an exponential blow-up.

Muller-Schupp Construction

In solving the synthesis problem, the most common are alternating Rabin and parity tree automata.

Theorem 3.4.2 ([70]). *Given an alternating Rabin tree automaton $\mathcal{T}$ on infinite trees with n states and m pairs, there is an effective construction that produces a non-deterministic Rabin tree automaton $\mathcal{N}$ having $(mn)^{\mathcal{O}(mn)}$ states and mn pairs that accepts the same language.*

Theorem 3.4.3 ([70]). *For each alternating parity tree automaton with n states and d priorities one can construct a nondeterministic parity tree automaton, accepting the same language, with $2^{\mathcal{O}(nd\log(nd))}$ states.*

In order to construct the nondeterministic tree automaton in the above theorems, Muller and Schupp[70] propose a Safra-like construction. The algorithm is given for Streett conditions, but thanks to Theorem 3.4.1, it also applies for Muller, Rabin and parity conditions and also to Büchi and co-Büchi conditions since they are particular cases of Streett conditions. We recall that a Streett condition $\Psi \in 2^Q \times 2^Q$ asks that for each pair $(R_i, G_i) \in \Psi$, if a "red" state in R_i is seen an infinite number of times, then also a "green" state in G_i is seen infinitely often. Therefore, the condition can be rewritten as "either we see finitely often states in R_i, or states in G_i appear infinitely often".

For an alternating tree automaton $\mathcal{T}$ with a Streett condition with m pairs, the nondeterministic tree automaton $\mathcal{N}$ that simulates it has as states m-tuples of ternary trees, one tree for each pair in the Streett set. The nodes in each tree are named with natural numbers and additionally are labeled by two items: a subset of states in the initial alternating automaton and a color from the set {red, yellow, green}.

As observed in [4] for the case of Büchi conditions, in the Muller-Schupp trees the sons of each node are partitioned into three classes: the ones that carry states in G_i, the ones that carry states in R_i and the ones that carry states that are neither in G_i nor in R_i.

The set of successors in G_i are collected in a set as the label of the left son, the ones in R_i form the label of the right son and the last set labels the middle son.

The initial state in the nondeterministic automaton $\mathcal{N}$ is the m-tuple of trees where each tree consists of a single vertex named 1_i, colored green and labeled with the initial state of the input Streett alternating tree automaton. Then, for an input letter a (as label of an input tree for the automaton), the automaton nondeterministically chooses a term in the transition $\delta(q, a)$ in $\mathcal{T}$ (in disjunctive normal form) for each state q in the set of states that label the leafs of the current tree. The nondeterministic choice indicates, for each state, the possible successors that will be used in the update operation of the trees contained by the next state. Then, for each direction $c \in \Upsilon$, the m-tuple of trees is updated accordingly as follows.

We explain how the tree from position i corresponding to the pair (R_i, G_i) is updated. First, the tree at position i from the current state is copied and all the red vertices are made yellow. All the other vertices keep their color. Then, the tree is extended from left to right by attaching sons (indicated by the nondeterministic choice) to the leaves according to the subset construction, starting from the set of states in each leaf. No son is introduced to a leaf if from none of its states a continuation with the letter "a" is possible. In this case, the whole path is deleted up to the last branching point. In the remaining cases, the sons (left, middle and/or right) are introduced depending on the membership of the continuations to the set G_i, $Y_i = Q \setminus (R_i \cup G_i)$ or R_i. The new vertices are then colored green except the right successors that are colored red. Then, the threes are kept from becoming too large by contracting edges vw to the vertex v in the case w is the only successor of v. Each time a vertex is added in the new tree, it is named with the smallest number available and whenever one vertex is removed, its name is made available to be used later as names for other vertices.

Finally, the acceptance condition of the resulting nondeterministic tree automaton is given as a Streett condition defined by pairs $(\hat{R}_{i,j}, \hat{G}_{i,j})$, for all i corresponding to the ith pair in the condition of the alternating autonaton $\mathcal{T}$ and each $j \in N_i$ where N_i is the set of all numbers used as vertex names of the trees corresponding to the pair i. The set $\hat{R}_{i,j}$ consists of those m-tuples in which the vertex j of the ith tree is red and $\hat{G}_{i,j}$ consists of those m-tuples in which the vertex j in the ith tree is green. Thus, every vertex name is forced to be green infinitely often or red only finitely often.

Piterman's algorithm proposed in [73] can also be applied on alternating Streett tree automata in order to obtain nondeterministic parity tree automata. However, as already discussed in Section 3.3, it is very similar to Safra's construction. It improves the number of states by means of some optimizations in the labeling of the trees used as states in the resulting automaton.

The construction of the nondeterministic tree automaton starting from an alternating tree automaton involves Safra-like determinizations of automata that was shown resistant

to efficient implementations. The Muller-Schupp procedure is an alternative to Safra's construction, but, as already seen, it still uses trees as states of the resulting automata. One tool that implements both Muller-Schupp and Safra's constructions and proves once again their resistance to efficient implementations is OmegaDet[4]. It receives Büchi nondeterministic word automata given as a well-structured text file and outputs the equivalent deterministic Rabin automaton. This tool cannot be used for application examples of any serious scale, the statistics shown in [4] giving results for the determinizations of some Büchi automata that have only a few states.

Avoiding Safra-like Constructions for Alternating Parity Tree Automata

Kupferman and Vardi [55] proposed an alternative procedure for testing the emptiness of alternating Parity tree automata by using universal coBüchi automata instead of deterministic tree automata. The emptiness problem for the universal coBüchi tree automaton is solved by translation to nondeterministic Büchi tree automata. The translation goes through alternating weak tree automata. Finally, the nonemptiness problem of nondeterministic Büchi tree automata is much simpler than the nonemptiness problem of alternating Parity tree automata and can be solved symbolically in quadratic time in the size of the transition function [92]. All these steps have been implemented and optimized in the tool Lily [49] that solves the synthesis problem (under perfect information) from LTL formulas by reducing it to the emptiness of universal coBüchi tree automata.

Theorem 3.4.4. *[55] For each alternating Parity tree automaton $\mathcal{T}$ with n states, transition function of size m and d priorities, there is an universal coBüchi tree automaton $\mathcal{U}$ with $\mathcal{O}(nd)$ states and alphabet size $2^{\mathcal{O}(m)}$ such that $\mathcal{L}_{ap}(\mathcal{T}) \neq \varnothing$ iff $\mathcal{L}_{uc}(\mathcal{U}) \neq \varnothing$.*

The universal coBüchi tree automaton built in the proof of the above Theorem accepts annotated trees with restrictions on the transition functions to be taken. That is, the trees to accept are annotated with partial functions $\nu : Q \to 2^{Q \times \Upsilon}$ so that a vertex (ℓ, ν) appears in a label if for all $q \in Q$, ν satisfies $\delta(q, \ell)$. Intuitively, if the transitions function is given in disjunctive normal form, the function ν is used to remove the nondeterminism by choosing a term from all the possible current states in the automaton $\mathcal{T}$. Assuming that the automaton $\mathcal{T}$ uses d priorities (from 1 to d) in the acceptance condition, automaton $\mathcal{U}$ consists of $h/2$ copies of the automaton $\mathcal{T}$, the ith copy checking that if the priority $2i$ is visited finitely often, then so is the priority $2i+1$. It has the coBüchi accepting condition $(F_{2i+1}, \{i\})$ where F_{2i+1} is the set of states with priority $2i+1$ and the label $\{i\}$ marks the fact that is the ith copy. Then, following the disjunct chosen by ν, from each state the automaton $\mathcal{U}$ branches universally to all copies of the automaton $\mathcal{T}$.

Theorem 3.4.5. *[55] Let $\mathcal{U}$ be an universal coBüchi tree automaton with n states. There is an nondeterministic Büchi tree automaton $\mathcal{N}$ over the same alphabet such that $\mathcal{L}_{uc}(\mathcal{U}) \neq \varnothing$ iff $\mathcal{L}_{nb}(\mathcal{N}) \neq \varnothing$, and the number of states in $\mathcal{N}$ is $2^{\mathcal{O}(n^2 \log n)}$.*

The algorithm that proves Theorem 3.4.5 builds an nondeterministic Büchi tree automaton $\mathcal{N}$ whose states are of the form (S, R, g) where S and R are sets of states in the universal automaton $\mathcal{U}$ and g is a partial function from the set of states in $\mathcal{U}$ to the set $\{0, ..., b\}$. The bound $b = (2n!)n^{2n+1}3^n(n+1)/n!$ is a consequence of Safra's determinization[79, 55] and $g(q)$ is even for all final states in $\mathcal{U}$.

Using the first set S, the nondeterministic automaton computes a subset construction on a run in $\mathcal{U}$ labeled nondeterministically with some ranks in the set $\{0, ..., b\}$. On each labeled path of $\mathcal{U}$, the successors of a vertex have smaller or equal rank then the current vertex and if a label is not correct (a final state is associated with an odd rank), the automaton rejects immediately.

Using the second set of states, the automaton makes sure that every infinite path visits infinitely often states that are associated to odd ranks. That is, since there is a finite number of ranks and the ranks decrease along paths of the run, an accepting run in the automaton $\mathcal{U}$ gets trapped in a set of states labeled with odd ranks. Then, because of the restriction of "correct labeling", this is possible only if the final sets are visited at most b times. Finally, the acceptance condition in the constructed nondeterministic tree automaton is the Büchi set consisting of the states having the second set empty.

Hayashi-Miyano Construction for Alternating Buchi Tree Automata

Other approaches reduce the synthesis problem to the emptiness of alternating Büchi tree automata. It is the case of the procedure that solves the synthesis problem from CTL specifications proposed in [54]. The following result, proven first by Miyano and Hayashi [63] for word automata and then addapted by Kupferman and Vardi [55] for tree automata, helps one to solve the emptiness of alternating Büchi tree automata.

Theorem 3.4.6 ([63, 55])**.** *For each alternating Büchi tree automaton with n states, one can construct an equivalent nondeterministic Büchi tree automaton with at most 3^n states.*

The idea behind the construction of the nondeterministic Büchi tree automaton from the proof of Theorem 3.4.6 is an extension of subset construction. It essentially merges all the vertexes, in a run of the given alternating tree automaton, that correspond to the same node in the input tree. Therefore, the states of the nondeterministic tree automaton are sets of states from the alternating automaton together with some additional information that helps keeping track of the accepting condition on individual paths.

3.4.4 Emptiness Game for Nondeterministic Tree Automata

Once the nondeterministic tree automaton obtained, its emptiness is classically tested by solving a two-player zero-sum game between the protagonist Eve (that wants to prove that there is a tree and an accepting run on it) and Adam, the opponent, that tries to

identify a non-accepting path in the run by choosing directions in the tree that falsify the accepting condition.

Now, we define the game $\mathcal{G}_\mathcal{T}$ corresponding to the nondeterministic tree automaton $\mathcal{T} = \langle \Lambda, \Upsilon, Q, q_0, \delta, \alpha \rangle$ where the transition from a state q with the letter ℓ is given as a set of functions $f : \Upsilon \to Q$. Let us denote by $\mathsf{Range}(f)$ the range of such a function f.

Formally, we define the turn-based game $\mathcal{G}_\mathcal{T} = \langle V_E, V_A, E', q_0, \Theta \rangle$ where

- $V_E = Q$ is the set of states controlled by Eve and
- $V_A = \{\mathsf{Range}(f) \mid \exists q \in Q, \ell \in \Lambda, f \in \delta(q, \ell)\}$ is the set of states controlled by Adam.

Then, the transition relation is defined for all $q \in V_E$ and all $Y \in V_A$ by

- $(q, Y) \in E'$ if there exist $\ell \in \Lambda$ and $f \in \delta(q, \ell)$ such that $Y = \mathsf{Range}(f)$ and
- $(Y, q) \in E'$ if $q \in Y$.

In other words, to go from q to Y, Eve chooses a symbol ℓ and a function $f : \Upsilon \to Q$ in $\delta(q, \ell)$. Then, Adam chooses a direction in Υ, but since he wants to construct a sequence of states not in α, one only needs to remember $\mathsf{Range}(f)$. Adam then picks a state in this set. Finally, Eve's objective is the set $\Theta = \{\pi = q_1 Y_1 q_2 Y_2 ... \in (V_E V_A)^\omega \mid q_1 q_2 ... \in \alpha\}$.

Proposition 3.4.1. *Eve has a wining strategy in the emptiness $\mathcal{G}_\mathcal{T}$ iff $\mathcal{L}(\mathcal{T}) \neq \varnothing$.*

Note that the resulting game $\mathcal{G}_\mathcal{T}$ is linear in the size of the nondeterministic tree automaton $\mathcal{T}$. Therefore, depending on the accepting condition, we obtain the complexity results for the emptiness of the nondeterministic tree automata using the complexity of solving the underlying game. Thanks to this, we have the following results:

Theorem 3.4.7 ([33, 74])**.** *Emptiness of a nondeterministic Rabin tree automaton with n states and m pairs over an alphabet with l letters can be tested in time $(lmn)^{\mathcal{O}(m)}$.*

Theorem 3.4.8 ([51])**.** *Emptiness of a nondeterministic Parity tree automata with n states and d priorities can be decided in NP $\cap$ co-NP. There is a deterministic subexponential algorithm solving the problem in $\mathcal{O}(n^{\sqrt{n}})$.*

3.4.5 Antichain Algorithm of UCT Automata Emptiness

As we already saw, there is a general trend to avoid Safra's construction when desiring to obtain algorithms that may lead to efficient implementations. Having as final objective an implementation to solve the LTL synthesis problem under perfect information, in [36] and [81] it was noted that testing the emptiness of an universal coBüchi automaton reduces to testing the emptiness of a universal B-coBüchi word automaton for a sufficiently large bound B, which in turn reduces to solving a safety game.

We detail and slightly adapt here the algorithm proposed in [36, 35] for solving the LTL synthesis under perfect information so that we test the emptiness of universal coBüchi tree automata. First, to prove the existence of a bound as described above for the universal coBüchi tree automata, we use the following results from [57, 73] to prove that if a UCT automaton with n states is not empty, then it accepts a tree encoded by a finite state Moore machine.

Theorem 3.4.9 ([73, 57])**.** *For every NBW $\mathcal{N}$ with n states, there is a DPW $\mathcal{D}$ with $2n(n!)^2$ states and index $2n$ such hat $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{N})$.*

Theorem 3.4.10. *A UCT $\mathcal{T}$ with n states is not empty iff there is a non-empty Moore machine with at most $2n(n!)^2$ states that represents a tree accepted by $\mathcal{T}$.*

Proof. The proof of the theorem follows the steps of one in [55] proving the theorem for a different bound, but passes through parity automata. The interesting part of the proof is from left to right. Let consider that the language of $\mathcal{T}$ is not empty.

We first prove that there is a DPT $\mathcal{N}$ automaton equivalent to $\mathcal{T} = \langle \Lambda, \Upsilon, Q, q_0, \delta, \alpha \rangle$. Note that when the automaton $\mathcal{T}$ rejects some tree t, there is a branch $\pi = \ell_0 c_0 \ell_1 c_1 ... \in (\Lambda \cdot \Upsilon)^\omega$ for which there is a branch η in the run r on t in $\mathcal{T}$ which is compatible with π and does not satisfy the coBüchi condition. Let us call such a branch rejecting.

We define a NBW automaton $\mathcal{A}$ that accepts branches in the tree t that are rejecting. That is, it accepts words $w \in (\Lambda \times \Upsilon)^\omega$. The automaton $\mathcal{A}$ has the same state space as $\mathcal{T}$. Assuming that the transition relation in $\mathcal{T}$ is given as $\delta : Q \times \Lambda \times \Upsilon \to 2^Q$, the transition relation in the automaton $\mathcal{A}$ is defined by $\delta_{\mathcal{A}}(q, (\ell, c)) = \delta(q, \ell, c)$. By Theorem 3.4.9, there is a DPW $\mathcal{B}$ with $2n(n!)^2$ states equivalent to $\mathcal{A}$. Let $\mathcal{B} = \langle Q_B, \Lambda' = \Lambda \times \Upsilon, \delta_B, p \rangle$ where p is the priority function. Then, we complement $\mathcal{B}$ by incrementing the priorities by 1 and get the DPW $\mathcal{B}' = \langle Q_B, \Lambda', \delta_B, p' \rangle$. Note that the automaton $\mathcal{B}'$ accepts branches π in t that are not rejecting. That is, all the branches in the run in $\mathcal{T}$ that are compatible with π that are accepting.

We now define the DPT automaton $\mathcal{D}$ that accepts Λ-labeled Υ-trees t whose branches are accepted by the automaton $\mathcal{B}'$. It has the same set of states as $\mathcal{B}'$ and the same priority function. The transition relation is defined by $\delta_{\mathcal{D}}(q_B, \ell) = \{f\}$ where $f : \Upsilon \to Q_B$ such that $f(c) = \delta_B(q_B, (\ell, c))$.

We now continue the proof showing that if the language of the automaton $\mathcal{D}$ is not empty, there is a Moore machine encoding an accepted tree. We see the DPT automaton $\mathcal{D}$ as a two-player parity game $\mathcal{G}_{\mathcal{D}}$ with perfect information and $2n(n!)^2$ states such that $\mathcal{L}(\mathcal{D}) \neq \varnothing$ iff there is a winning strategy for the protagonist in $\mathcal{G}_{\mathcal{D}}$. Further, it is known that parity games admit memoryless strategies. Therefore, if $\mathcal{L}(\mathcal{D}) \neq \varnothing$, there is a memoryless strategy for the protagonist in $\mathcal{G}_{\mathcal{D}}$ that is to play the labels of the accepted tree in $\mathcal{D}$. This strategy can be transformed into a Moore machine with the same states as $\mathcal{D}$, and therefore $2n(n!)^2$ states. ∎

Theorem 3.4.11. *Given a UCT $\mathcal{T}$ that accepts Λ-labelled Υ-trees with n states and a Moore machine $\mathcal{M}_t$ with m states representing a Λ-labelled Υ-tree t,*

$$t \in \mathcal{L}_{uc}(\mathcal{T}) \text{ iff } t \in \mathcal{L}_{uc,mn}(\mathcal{T})$$

Proof. From right to left, the proof is obvious since $\mathcal{L}_{uc,mn}(\mathcal{T}) \subseteq \mathcal{L}_{uc}(\mathcal{T})$. In the other direction, let $\mathcal{M}_t = \langle \Lambda, \Upsilon, S_M, s_0, \delta_M, g_M \rangle$ be the Moore machine representing the tree t.

First, we equivalently transform $\mathcal{M}_t$ into a nondeterministic Büchi word automata $\mathcal{A}_M = \langle \Lambda' = \Lambda \times \Upsilon, S_M, s_0, \delta'_M, \alpha' = S_M \rangle$ that runs over the alphabet $\Lambda' = \Lambda \times \Upsilon$ and for which $\mathcal{L}_{nb}(\mathcal{A}_M)$ is the set of traces of $\mathcal{M}_t$. It accepts all the branches of t since the tree t is the "tree unfolding" of $\mathcal{M}_t$. Note that $\mathcal{A}_M$ has m states.

Let $\mathcal{T} = \langle \Lambda, \Upsilon, Q, q_0, \delta, \alpha \rangle$. We define the product $\mathcal{T} \times \mathcal{A}_M = \langle \Lambda, \Upsilon, \tilde{Q}, \tilde{q_0}, \tilde{\delta}, \tilde{\alpha} \rangle$ where $\tilde{Q} = Q \times S_M$, $\tilde{q_0} = (q_0, s_0)$, $\tilde{\alpha} = \alpha \times S_M$ and

$$\tilde{\delta}((q,s),\ell,c) = \{(q',s') \mid q' \in \delta(q,\ell,c) \text{ and } s' = \delta'_M(s,(\ell,c))\}$$

Now, since $t \in \mathcal{L}_{uc}(\mathcal{T})$, for all accepting run r_t on t in $\mathcal{T}$, for all branches η of r_t, the restriction $\eta{\restriction}Q = q_0 q_1 q_2 \ldots$ of η on Q satisfies $\inf(\eta{\restriction}Q) \cap \alpha = \varnothing$. That is, $\nexists\ 0 \leq i \leq k < j$ s.t. $q_i = q_j$ and $q_k \in \alpha$ (there are no cycles in r_t containing a final state). Then, from the construction of $\mathcal{T} \times \mathcal{A}_M$, there is no cycle reachable from $\tilde{q_0}$ that contains a final state. Therefore, all branches η of runs r_t on t visit at most $m \times n$ final states. That is, $t \in \mathcal{L}_{uc,mn}(\mathcal{T})$. ∎

Using this results we can turn the emptiness problem for the UCT automaton $\mathcal{T}$ into the emptiness problem of the UBCT automaton $(\mathcal{T}, B)$ as follows:

Corollary 3.4.1. *Given an UCT $\mathcal{T}$ that accepts Λ-labelled Υ-trees with n states and the bound $B = 2n^2(n!)^2$,*

$$\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing \text{ iff } \mathcal{L}_{uc,B}(\mathcal{T}) \neq \varnothing$$

Proof. If $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$, by Theorem 3.4.10, there is a regular tree $t \in \mathcal{L}_{uc}(\mathcal{T})$ represented by a finite state machine with at most m states$(m < 2n(n!)^2)$. Then, by Theorem 3.4.11, $t \in \mathcal{L}_{uc,nm}(\mathcal{T})$ and therefore $\mathcal{L}_{uc,B}(\mathcal{T}) \neq \varnothing$. In the other sense, the proof is obvious since $\mathcal{L}_{uc,B}(\mathcal{T}) \subseteq \mathcal{L}_{uc}(\mathcal{T})$. ∎

Clearly, for all $b \geq 0$, if $\mathcal{L}_{uc,b}(\mathcal{T}) \neq \varnothing$, then also holds $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$. This observation has led to an incremental algorithm by starting with some small bound b and increment it until a value for which the language of the b-coBüchi tree automaton is not empty is reached.

Given a bound $b \geq 0$ and a UCT $\mathcal{T}$, the idea is to construct a safety game $\mathcal{G}(\mathcal{T}, b)$ so that Player 0 (the protagonist) has a winning strategy in $\mathcal{G}(\mathcal{T}, b)$ if and only if $\mathcal{L}_{uc,b}(\mathcal{T})$ is non-empty. The game $\mathcal{G}(\mathcal{T}, b)$ is obtained by extending the classical automata subset construction with counters which count, up to b, the maximal number of times all the

runs, up to the current point, have visited final states. If Q is the set of states of $\mathcal{T}$, the set of states of the safety game $\mathcal{G}(\mathcal{T}, b)$ is the set of functions $F : Q \to \{-1, 0, \ldots, b+1\}$. The value $F(q) = -1$ means that no run have reached q and $F(q) \in \{0, \ldots, b\}$ means that the maximal number of final states that has been visited by some run reaching q is $F(q)$. We set $F(q) = b+1$ if there is a run that reaches q and visits more then b times the set of final states. The safe states are all the functions F so that $F(q) \leq b$ for all $q \in Q$. The set of states can be partially ordered by the pairwise comparison between functions and it is shown that the sets of states manipulated by the fixpoint algorithm are downward closed for this order.

The emptiness of the UCBT automaton $\mathcal{T}$ is first reduced to the emptiness of a deterministic safety tree automaton $Det(\mathcal{T}, b)$. Then, the emptiness of the later automaton is tested by reducing to the emptiness game $\mathcal{G}(\mathcal{T}, b)$ associated to it. The game $\mathcal{G}(\mathcal{T}, b)$ is a two-player zero-sum safety game, meaning that the protagonist has as objective that the play stays into a safe set and the antagonist's objective is to reach an unsafe state. As we will use the construction in this thesis, we give the formal definition here and also provide the proofs.

Determinization of UBCT Given the UBCT $(\mathcal{T}, b)$ with $\mathcal{T} = \langle \Lambda, \Upsilon, Q, Q_0, \delta, \alpha \rangle$, the deterministic safety tree automaton $Det(\mathcal{T}, b)$ is formally defined as the tuple $Det(\mathcal{T}, b) = \langle \mathcal{F}, F_0, \alpha', \delta' \rangle$ where $\mathcal{F}$ is the set of states, F_0 is the initial state, α' is the set of final states and δ' is the transition relation where

- $\mathcal{F} = \{F | F : Q \to \{-1, 0, ..., b+1\}\}$
- $\forall q \in Q$, $F_0(q) = -1$ if $q \neq q_0$ and $F_0(q) = (q \in \alpha)$ otherwise
- $\alpha' = \{F \in \mathcal{F} | \exists q \in Q \text{ s.t. } F(q) > b\}$
- $\forall c \in \Upsilon$, $\delta'(F, \ell, c) = (F', c)$ if
  $F'(q') = max\{min\{b+1, F(q) + (q' \in \alpha) \mid (q', c) \in \delta(q, \ell, c) \text{ and } F(q) \neq -1\}\}$

where $max(\varnothing) = -1$ and, for all states $q \in Q$, $(q \in \alpha) = 1$ if q is in α and 0 otherwise. We say that a state F is unsafe if there exists $q \in Q$ such that $F(q) = b+1$. Intuitively, this happens whenever there is a path from the initial state to q that visits more then b final states.

Proposition 3.4.2. *Let $\mathcal{T}$ be a UCT and $b \in \mathbb{N}$. Then, $Det(\mathcal{T}, b)$ is complete, deterministic and $\mathcal{L}_{uc,0}(Det(\mathcal{T}, b)) = \mathcal{L}_{uc,b}(\mathcal{T})$.*

Proof. It is obvious by construction that the automaton $Det(\mathcal{T}, b)$ is complete since $\mathcal{T}$ already is. Also, it is deterministic by construction: for each state F, one letter $\ell \in \Lambda$ and one direction $c \in \Upsilon$, there is only one successor state F'.

We prove the language equality by double inclusion. From right to left, if t is a Λ-labeled Υ-tree accepted by $(\mathcal{T}, b)$, there is an accepting run r_t on t in $(\mathcal{T}, b)$. Since the run is accepting, each branch of r_t is such that it visits at most b times the set of final states. Then, by construction, $Det(\mathcal{T}, b)$ uses a sort of subset construction on the states reached with the same sequence of letters and directions keeping the biggest number of final states visited before a state q is reached. Therefore, there is an corresponding run in $Det(\mathcal{T}, b)$ for r_t that runs on t. The run in $Det(\mathcal{T}, b)$ is also accepting because the set α' is never reached, each branch in r_t visiting at most b times the set of final states.

From left to right, let t be a Λ-labeled Υ-tree accepted by $Det(\mathcal{T}, b)$ and r_t the accepting run on it. Since r_t is accepting, it never visits a state F such that there is q with $F(q) \geq b + 1$. Then, we can build a corresponding run for r_t in $(\mathcal{T}, b)$ which is accepting, since all the branches visit at most b times the final set of states α (otherwise, r_t is not accepting). ■

Reduction to safety game The turn-based safety game is defined similarly to the emptyness game used to test emptyness of nondeterministic tree automata as: $\mathcal{G}(\mathcal{T}, b) = \langle V_E, V_A, E', q_0, \mathsf{safe}\rangle$ where $V_E = \mathcal{F}$ is the set of states controlled by Eve, $V_A = \{(F, \ell) \mid F \in \mathcal{F} \text{ and } \ell \in \Lambda\}$ is the set of states controlled by Adam. The transition relation is defined by $(F, (F, \ell)) \in E'$ for all $F \in \mathcal{F}$ and $\ell \in \Lambda$ and $((F, \ell), F') \in E'$ for all $F' \in \delta'(F, l)$. Finally, Eve's objective $\mathsf{safe} = (\mathcal{F} \cup V_A) \setminus \alpha'$ asks that all the states visited along a play are not in the state α'.

Theorem 3.4.12 ([35, 36]). *Let $\mathcal{T}$ be a UCT and $b \in \mathbb{N}$. $\mathcal{L}_{uc,0}(Det(\mathcal{T}, b)) \neq \varnothing$ iff there is a winning strategy σ_E for Eve in $\mathcal{G}(\mathcal{T}, b)$.*

Fixpoint algorithm to solve the Safety game Symbolic techniques that are also exploited in this thesis have been used to solve the safety games. In [35] and [36] is shown that the safety games $\mathcal{G}(\mathcal{T}, b)$ can be solved on-the-fly without constructing them explicitly, and that the fixpoint algorithm used to solve these safety games could be optimized by using some antichain representation of the sets constructed during the fixpoint computation.

The fixpoint algorithm solving the game $\mathcal{G}(\mathcal{T}, b)$ computes the set of controllable predecessors for Eve using the following predecessor functions. The functions $Pre_U : 2^{V_E} \to 2^{V_A}$ and $Pre_C : 2^{V_A} \to 2^{V_E}$ compute the controllable and respectively uncontrolable predecessors of states in a set X that are also in the set of safe states of the game $\mathcal{G}(\mathcal{T}, b)$. Formally, they are defined by

$$Pre_U(X) = \{(F, \ell) \in V_A \mid \forall F' \in V_E, \text{ if } ((F, \ell), F') \in E', \text{ then } F' \in X\} \cap \mathsf{safe}$$
$$Pre_C(X) = \{F \in V_E \mid \exists (F, \ell) \in V_A\} \cap \mathsf{safe}$$

Intuitively, the function $Pre_U(X)$ outputs the set of safe states belonging to Adam that have all the successors in the set X. Therefore, even if Eve cannot control Adam's choice, from some state in $Pre_U(X)$, Adam cannot avoid the set X. On the other hand, the function $Pre_C(X)$ compute the set of safe states controlled by Eve that have at least one successor in the state X.

Then, the controllable predecessor operator used by the fixpoint algorithm is

$$CPre = Pre_C \circ Pre_U$$

Since the function $CPre$ is monotone over the complete latice $(2^{V_E}, \subseteq)$, it has a greatest fixpoint denoted by $CPre^*$. That is, $CPre^*$ is the fixpoint of the $\subseteq$-descending chain defined by: $X_0 = V_E \cup V_A$ and $\forall i \geq 0,\ X_{i+1} = CPre(X_i) \cap X_i$. Intuitively, the set X_i consists of all of Eve's states from which she has a winning strategy to stay in the safe region during i steps. Let us unfold the defined chain to better understand the meaning of X_i. The chain starts with the entire set of states of the game. Then, $X_1 = Pre_C(Pre_U(X_0)) \cap X_0 = V_E \cap \mathsf{safe}$ is the set of Eve's safe states. At the next step, X_2 consists of all states from X_1 from which Eve has a strategy (can choose a successor) to enforce the game to stay in the safe set X_1. Therefore, as the following theorem says, the fixpoint $CPre^*$ is the set of states from which Eve has a strategy such the game remains forever in the safe set safe.

Theorem 3.4.13 ([35]). *The set of states from which Eve has a winning strategy in $\mathcal{G}(\mathcal{T}, b)$ is equal to $CPre^*$.*

Partial order and antichains To optimize the algorithm, in [35] and [36] is defined a partial order $\preccurlyeq$ on states such that $\forall F, F' \in \mathcal{F}$, $F \preccurlyeq F'$ iff $\forall q$, $F(q) \leq F'(q)$. We can extend this ordering to the states belonging to Adam by defining $(F, \ell) \preccurlyeq (F', \ell)$ iff $F \preccurlyeq F'$ since none of Adam's states is final (if (F, ℓ) is seen as a function, $(F, \ell)(q) = F(q)$). Intuitively, if Eve can win from a state F', then she can also win from a state F with $F \preccurlyeq F'$ since F records less visited final states of $\mathcal{T}$ then F'.

A set X is *closed* for $\preccurlyeq$ if $\forall F \in X$, $\forall F' \preccurlyeq F$, holds $F' \in X$. Then, since the image of a closed set by any functions defined above is also a closed set, instead of keeping all the states in $\mathcal{G}(\mathcal{T}, b)$, the antichain algorithm manipulates *closed* sets with respect to the partial order $\preccurlyeq$ that are carefully represented and manipulated.

The antichain algorithm described is implemented for universal coBüchi word automata in the tool Acacia+[12] that solves the synthesis problem for LTL formulas under the assumption of perfect information for the agents. We refer to [35] and [36] for further details.

4. Solving the $KLTL$ Synthesis Problem

In this section we study the synthesis problem of one partially informed process against an antagonist omniscient environment starting from KLTL specifications. We provide a *Safraless procedure* to solve the Synthesis Problem for *positive KLTL* specifications. A short version of this chapter was published in [14].

We recall that the KLTL synthesis problem asks, given an interaction model $\mathcal{M} = \langle \mathcal{P}, \Omega, (\Sigma_i)_{i\in\Omega}, V, V_0, E, \tau \rangle$, a set $P \subseteq \Omega$ and a KLTL formula φ, to synthesize strategies for processes in P such that the specification φ is satisfied for any strategies of the environment (processes not in P). Formally,

$$\begin{aligned} &\mathsf{INPUT}: \mathcal{M},\ \varphi,\ P \subseteq \Omega, \\ &\mathsf{OUTPUT}: \text{Yes} \ \text{ iff } \ \exists (\sigma_i)_{i\in P} \ :\ \text{exec}(\mathcal{M}, \bar{\sigma}_P) \vDash \varphi \end{aligned}$$

Contributions Our main contribution in this chapter, is a Safraless synthesis procedure for the positive fragment of KLTL (KLTL$^+$), i.e., KLTL formulas where the operator K occurs under an even number of negations. We consider the setting where it is asked to synthesize only one partially informed process ($|P| = 1$) against an omniscient antagonist environment, since the more general case where several partially informed processes have to be synthesized was proved to be undecidable. Our procedure relies on universal coBüchi tree automata (UCT). More precisely, given a KLTL$^+$ formula φ and some interaction model $\mathcal{M}$, we show how to construct a UCT $\mathcal{T}$ whose language is exactly the set of strategies that realize φ in $\mathcal{M}$.

Despite the fact that our procedure has 2ExpTime worst-case complexity, we have implemented it and shown its practical feasibility through a set of examples. In particular, based on ideas of [36] also recalled in Chapter 3, we reduce the problem of checking the emptiness of the automaton $\mathcal{T}$ to solving a safety game whose state space can be ordered and compactly represented by antichains. Our implementation is based on the tool *Acacia+* [12] and, to the best of our knowledge, it is the first implementation of a synthesis procedure for epistemic temporal specifications considering perfect recall for processes. Our tool synthesizes small strategies that correspond to the intuitive strategies

we would expect, although it goes through a nontrivial automata construction. As an application, this implementation can be used to solve two-player games with imperfect information whose objectives are given as LTL formulas.

4.1 Preliminaries

Perfect Information Case (LTL Synthesis)

Let us first note that when all the agents in the interaction model have perfect information, each KLTL formula reduces to an LTL formula by simply removing the knowledge operator. Further, since all processes have perfect information and the interaction model is known to everyone, the synthesis problem in Definition 2.3.1 reduces to the synthesis of one process against one antagonist environment. This problem was already studied in [36, 35, 54] without an interaction model specified. In the setting considered in the above papers, the alphabet over which the specifications are formalized is partitioned into input and output signals. The output alphabet $\mathcal{P}_O$ belongs to the process to synthesize (Agent 0) and the input alphabet $\mathcal{P}_I$ is for the environment (Agent 1). Then, the realizability game is played in turns. At each round Agent 0 gives a subset o of output signals and then Agent 1 answers with a subset i of input signals. The play lasts forever and defines an infinite word $w = (i_0 \cup o_0)(i_1 \cup o_1)... \in (2^{\mathcal{P}})^\omega$. Then, the problem is to find a strategy for the first agent such that, for any input from the adversary, the LTL specification is satisfied on the defined infinite word.

This problem was first studied by Pnuli and Rosner [74] and Abadi, Lamport and Wolper in [2]. The classical algorithm that solves the perfect information LTL realizability problem translates the specification φ into a nondeterministic Büchi word automaton equivalent to it that accepts all infinite words satisfying φ, applies Safra's construction to get a deterministic Rabin automaton for it and then extends it to a deterministic Rabin tree automaton that accepts all complete $2^{\mathcal{P}_O}$-labeled $2^{\mathcal{P}_I}$-trees whose branches satisfy the specification φ and hence encode winning strategies. Finally, the algorithm checks the non-emptiness of the later tree automaton.

The algorithm proposed in [54] for solving LTL and CTL^* realizability problems under both perfect or imperfect information relies on the construction of alternating Rabin tree automata and the testing of the emptiness of the accepted language. This is done using the Muller-Schupp translation [70] of alternating tree automata into nondeterministic tree automata recalled in Chapter 3.

Safraless Solutions Unfortunately, the above theoretically nice procedures turn to be difficult to implement due to the fact that they use Safra-like procedures that, as we have seen in Section 3.4.3, generate deterministic automata with a very complex state space. The approach proposed in [36] by Filiot, Jin and Raskin for solving the LTL

realizability under perfect information is based on antichains and works as following. The LTL specification φ is first transformed into a universal coBüchi word automaton $\mathcal{A}_\varphi$ as described in Section 3.3. Then, the automaton $\mathcal{A}_\varphi$ over the alphabet $\mathcal{P}$ is equivalently transformed into a *turn-based* UCW (tbUCW) by splitting transitions $(q_1, \ell, q_2) \in \delta_\mathcal{A}$ into two transitions $(q_1, \ell \cap \mathcal{P}_O, q_1')$ and $(q_1', \ell \cap \mathcal{P}_I, q_2)$. The remaining steps for testing the emptiness of the universal automaton follow the line described in Section 3.4.5. The incremental algorithm starts with a small bound b and transforms the later automaton into a b-coBüchi tbUCW which then is reduced to a safety game solved on the fly using the antichain algorithm. This algorithm was efficiently implemented in the tool *Acacia*$^+$ [12]. Because in practice the bound b turns to be quite small when the formula is realizable, and because the unrealizability of a LTL formula φ under perfect information is equivalent to the realizability of $\neg\varphi$ from the point of view of the environment, the tool *Acacia*$^+$ tests the realizability of both formulas and stops when one of the problems has a positive answer.

The above procedure proposed in [36] can also be applied on architectures where the interaction model is given. That is, one has to take the product between the interaction model and the universal coBüchi word automaton equivalent with the specification and apply the remaining of the algorithm on the resulting universal coBüchi automaton. More details are provided in the following section.

Imperfect Information for the Process to Synthesize

As already stated in Section 2.3, the synthesis problem is undecidable for two or more partially informed processes to synthesize. Therefore, in [89] Van der Meyden and Vardi studied the synthesis problem of one partially informed process against one omniscient antagonist environment with specifications given by KLTL formulas. In this case, the interaction model is formally defined by $\mathcal{M} = \langle \mathcal{P}, \Omega = \{0,1\}, \Sigma_0, \Sigma_1, V, V_0, E, \tau \rangle$ and then, assuming that Process 0 is the process to be synthesized and Process 1 represents the environment, the synthesis problem resumes to the following:

> A KLTL formula φ is *realizable* in $\mathcal{M}$ if there is a protocol σ_0 for Process 0 such that for all executions $\rho \in \mathsf{exec}(\mathcal{M}, \sigma_0)$, holds $\mathsf{exec}(\mathcal{M}, \sigma_0), \rho, 0 \vDash \varphi$.

They prove that this problem is 2ExpTime-complete by reducing it to the emptiness problem for alternating Rabin tree automata that accept $\mathcal{O}$-trees (where $\mathcal{O}$ is the set of observations of Process 0) whose vertices are labeled with actions of the agent for which the protocol is synthesized and a set K intended to be the knowledge of Process 0. The set K consists of pairs (X, v) of a state v in the interaction model and a mapping X (called atom) that associates to each subformula of φ a truth value. The projection of the labeling of such trees on the actions of the agent encodes the strategy to follow in the interaction model to realize φ. Note that the alphabet of the tree automaton built by the algorithm

is exponential in the number of states of the interaction model and doubly exponential in the size of the input formula φ.

The KLTL realizability is characterized by some restrictions that have to be satisfied by the accepted tree. First of all, the atom X after any initial observation in the tree has to map the formula φ to 1. Then, the authors define some conditions expressing that the states appearing in K represent the possible states in which the interaction model may be. The set is computed by subset construction depending on the actions of the agent and the received observation. Finally, there are two more conditions that express the fact that whenever (X, v) appears in the labeling of some node in the tree, the atoms X map to *true* exactly the subformulas that hold true on some executions in the interaction model starting from v. This implies that whenever X maps to *true* a subformula $K\psi$, all the atoms X' that appear in some pair (X', v') at the current node in the tree map ψ to *true*.

The alternating Rabin tree automaton that accepts exactly trees satisfying the above conditions is then obtained as the intersection of the automata checking each property. Then, the alternating automaton is equivalently transformed using Muller-Schupp construction (Theorem 3.4.2) into a nondeterministic Rabin tree automaton with $2^{|\mathcal{M}|\cdot 2^{\mathcal{O}(|\varphi|)}}$ states and $|\mathcal{M}| \cdot 2^{\mathcal{O}(|\varphi|)}$ pairs. The 2ExpTime complexity is finally obtained by applying Theorem 3.4.7. The 2ExpTime lower bound holds since the synthesis from LTL specifications under the assumption of perfect information is a particular case of KLTL synthesis and it is already 2ExpTime-hard.

4.2 Safraless Synthesis Procedure for Positive KLTL Specifications

In this section we explain our Safraless automata-based procedure to solve the synthesis problem from positive KLTL formulas in an interaction model $\mathcal{M}$ where the process to be synthesized (Process 0) has imperfect information. We first define the procedure to deal with LTL formulas and then extend this procedure to handle the knowledge operator K.

4.2.1 LTL Synthesis under Imperfect Information

As already mentioned in the previous chapter, given an interaction model $\mathcal{M}$ and the set of observations $\mathcal{O}$ of Agent 0, a complete $(\Sigma_0 \cup \{\#\})$-labeled $\mathcal{O}$-tree t where the root is labeled with # and the other nodes have labels in Σ_0 defines a strategy of the process to synthesize. Further, all $(\Sigma_0 \cup \{\#\})$-labeled $\mathcal{O}$-trees that we consider have this structure and are called *strategy trees*.

Any infinite branch π of t defines an infinite sequence of actions and observations of Agent 0 in $\mathcal{M}$, which in turn corresponds to a set of possible executions in $\mathcal{M}$. Further,

$\mathsf{traces}(\pi)$ denotes the set of traces of all executions compatible with π (the set of traces of a sequence of actions and observations has been defined in Chapter 2).

Given an LTL formula ψ over the set $\mathcal{P}$ of propositions labeling the states of the interaction model $\mathcal{M}$, we construct a universal coBüchi tree automaton $\mathcal{T} = \langle \Sigma_0, \mathcal{O}, Q, Q_0, \delta, \alpha \rangle$ that accepts all the strategies of Agent 0 (the protagonist) that realize ψ in the interaction model $\mathcal{M}$. First, we convert ψ into an equivalent universal coBüchi word automaton(UCW) $\mathcal{A}_\psi = \langle 2^{\mathcal{P}}, Q^{\mathcal{A}}, Q_0^{\mathcal{A}}, \delta^{\mathcal{A}}, \alpha^{\mathcal{A}} \rangle$. This is done by constructing the nondeterministic Büchi word automaton for the negation of the formula ψ. The later automaton always exists thanks to algorithms presented in [40, 39]. Then, as a direct consequence of the definition of KLTL realizability:

Proposition 4.2.1. *Given a complete $(\Sigma_0 \cup \{\#\})$-labeled $\mathcal{O}$-tree t, t defines a strategy that realizes ψ under $\mathcal{M}$ iff for all infinite branches π of t and all executions $\rho \in \mathrm{exec}(\mathcal{M}, \pi)$, holds $\mathsf{trace}(\rho) \in \mathcal{L}_{uc}(\mathcal{A}_\psi)$.*

We now show how to construct a universal tree automaton that verifies the property mentioned in the previous proposition for all branches of the trees. We use universal transitions to check, on every branch of the tree, that all the possible traces in $\mathcal{M}$ compatible with the sequence of actions in Σ_0 and observations in $\mathcal{O}$ defined by the branch satisfy the formula ψ.

Based on the sequence of observations that the process has received and his own actions, Agent 0 can define its *knowledge* I of possible states in which the interaction model can be, as a subset of states of V. Given an action $a \in \Sigma_0$ and some observation $o \in \mathcal{O}$, we define by $\mathrm{post}_a(I, o)$ the new knowledge that Agent 0 can infer from observation o, its action a and the previous information I. Formally,

$$\mathrm{post}_a(I, o) = \{v \in V \cap o \mid \exists b \in \Sigma_1, \exists v' \in I \text{ s.t. } E(v', a, b) = v\}$$

UCT $\mathcal{T}$ for LTL realizability The states of the universal tree automaton $\mathcal{T}$ are pairs of states of $\mathcal{A}_\psi$ and knowledge sets. We also have some extra initial state (q_{init}, V_0) and two sink states $(q_w, \varnothing)$ and $(\bot, \varnothing)$. That is, $Q = (Q^{\mathcal{A}} \times 2^V) \cup \{(q_{init}, V_0), (q_w, \varnothing), (\bot, \varnothing)\}$ where the state $(\bot, \varnothing)$ is reached when the tree to accept does not encode properly a strategy and therefore the automaton rejects and $(q_w, \varnothing)$ is added for the completeness. The initial set of states is then $Q_0 = \{(q_{init}, V_0)\}$ and the accepting condition is the coBüchi set $\alpha = \{(\bot, \varnothing)\} \cup (\alpha^{\mathcal{A}} \times 2^V)$. This means that the associated path in the word automaton $\mathcal{A}_\psi$ is used to verify the acceptance condition along all the possible executions in $\mathcal{M}$ defined by the sequence of knowledge sets.

Let us take an action $a \in \Sigma_0$ and some observation $o \in \mathcal{O}$ of Agent 0. First, we define the transition relation from the initial state as

$$\delta((q_{init}, V_0), \#, o) = \{(q_0^{\mathcal{A}}, V_0 \cap o) \mid q_0^{\mathcal{A}} \in Q_0^{\mathcal{A}}\} \text{ and}$$
$$\delta((q_{init}, V_0), a, o) = \{(\bot, \varnothing)\}, \ \forall a \in \Sigma_0$$

This reads as: the automaton asks that the root of the tree to be accepted is labeled with the special symbol # and the transition goes to the set of states consisting of an initial state of the automaton $\mathcal{A}_\psi$ and the knowledge set updated according the received observation. The following equations define the transition relation in teh case of the sink states or a label # in a node of the tree other than the root:

$$\delta((\bot,\varnothing),\ell,o) = \{(\bot,\varnothing)\}, \forall \ell \in \Sigma_0 \cup \{\#\};$$
$$\delta((q_w,\varnothing),\ell,o) = \{(q_w,\varnothing)\}, \forall \ell \in \Sigma_0 \cup \{\#\} \text{ and}$$
$$\delta((q,I),\#,0) = \{(\bot,\varnothing)\} \text{ for } (q,I) \in Q^{\mathcal{A}} \times 2^V$$

We now define $\delta((q,I),a,o)$ for a state $q \in Q^{\mathcal{A}}$ and a knowledge set $I \subseteq V$.

It can be the case that there is no transition in $\mathcal{M}$ from a state in I to a state of o, i.e., $\text{post}_a(I,o) = \varnothing$. In this case, all the paths from the next vertex of the tree should be accepting. This situation corresponds to Agent 0 receiving a wrong observation and it is modeled by going to the extra state $(q_w,\varnothing)$, i.e., $\delta((q,I),a,o) = \{(q_w,\varnothing)\}$.

Now, suppose that $\text{post}_a(I,o)$ is not empty. Since the automaton must check that all the traces of $\mathcal{M}$ that are compatible with actions in Σ_0 and observations are accepted by $\mathcal{A}_\psi$, intuitively, one would define $\delta((q,I),a,o)$ as the set of states of the form $(q',\text{post}_a(I,o))$ for all q' such that there exists $v \in I$ with $(q,\tau(v),q') \in \delta^{\mathcal{A}}$. However, it is not correct for several reasons. First, it could be that v has no successor in o for action a, and therefore one should not consider it because the traces up to state v die at the next step after getting the observation o. Therefore, one should only consider states of I that have a successor in o. Second, it is not correct to associate the new knowledge $\text{post}_a(I,o)$ with q' because it could be that there exists a state $v' \in \text{post}_a(I,o)$ so that for all its predecessors v in I, there is no transition $(q,\tau(v),q')$ in $\delta^{\mathcal{A}}$, and therefore, one would also take into account sequences of interpretations of propositions that do not correspond to any trace in $\mathcal{M}$.

Taking into account these two remarks, we define, for all states q', the set

$$I_{q,q'} = \{v \in I \mid (q,\tau(v),q') \in \delta^{\mathcal{A}}\}$$

Then, the transition $\delta((q,I),a,o)$ in the automaton $\mathcal{T}$ is defined as the set

$$\delta((q,I),a,o) = \{(q',\text{post}_a(I_{q,q'},o)) \mid \exists v \in I, (q,\tau(v),q') \in \delta^{\mathcal{A}}\}$$

Note that with a state q' is not associated the entire knowledge of Agent 0. However, because the automaton is universal and $\bigcup_{q' \in Q^{\mathcal{A}}} \text{post}_a(I_{q,q'},o) = \text{post}_a(I,o)$, the protagonist does not have better knowledge by restricting the knowledge sets in one state of the automaton.

In the following, since the labeling of the root of trees encoding strategies and the initial state in the automaton $\mathcal{T}$ have no impact on the actual executions in the interaction model $\mathcal{M}$, we ignore them when speaking about branches in a tree or an accepting run.

Lemma 4.2.1. *The LTL formula ψ is realizable in $\mathcal{M}$ iff $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$.*

Proof. If ψ is realizable in $\mathcal{M}$, there is a strategy $\sigma_0 : \mathcal{O}(\Sigma_0\mathcal{O})^* \to \Sigma_0$ for Agent 0 such that $\mathsf{exec}(\mathcal{M}, \sigma_0) \vDash \psi$. Let us see this strategy as the $(\Sigma_0 \cup \{\#\})$-labeled $\mathcal{O}$-tree t_{σ_0} that encodes it and prove that $t_{\sigma_0} \in \mathcal{L}_{uc}(\mathcal{T})$.

The run on t_{σ_0} in the automaton $\mathcal{T}$ is a $(Q \times \mathcal{O}^*)$-labeled $\mathbb{N}$-tree r. Therefore, each branch π of r induces an infinite sequence $r(\pi) = ((q_0, I^{(0)}), o_0)((q_1, I^{(1)}), o_0o_1)...$ where, by the definition of $\mathcal{T}$, $q_0 \in Q_0^{\mathcal{A}}$, $I^{(0)} = V_0 \cap o_0$ and $I^{(i+1)} = \mathrm{post}_{t(o_0...o_i)}(I^{(i)}_{q_i,q_{i+1}}, o_{i+1})$ for all $i \geq 0$.

Since $I^{(i)}_{q_i,q_{i+1}}$ is a a subset of the knowledge set $I^{(i)}$ consisting of the states in the interaction model whose labels may fire transitions from q_i to q_{i+1} in $\mathcal{A}_\psi$, and since the transitions in $\mathcal{T}$ are universal, $\eta = q_0q_1q_2...$ is a run in $\mathcal{A}_\psi$ on the traces of the executions $\rho = v_0v_1v_2... \in \mathrm{exec}(\mathcal{M}, \sigma_0)$ where $v_i \in I^{(i)}$. Hence, because $\mathsf{exec}(\mathcal{M}, \sigma_0) \vDash \psi$ and $\rho \in \mathsf{exec}(\mathcal{M}, \sigma_0)$, η is an accepting run and then $r(\pi)$ visits a finite number of times the final states of $\mathcal{T}$ along paths. Therefore, $t_{\sigma_0} \in \mathcal{L}_{uc}(\mathcal{T})$ and $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$.

In the other direction, if $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$, there exists an complete $(\Sigma_0 \cup \{\#\})$-labeled $\mathcal{O}$-tree t_{σ_0} encoding a strategy such that $t_{\sigma_0} \in \mathcal{L}_{uc}(\mathcal{T})$. We prove that the strategy σ_0 encoded by the tree t_{σ_0} realizes ψ, i.e., for all executions $\rho \in \mathsf{exec}(\mathcal{M}, \sigma_0)$, holds $\mathsf{exec}(\mathcal{M}, \sigma_0), \rho, 0 \vDash \psi$.

Let r be the accepting tun in $\mathcal{T}$ on the tree t_{σ_0} and π be a branch of r with $r(\pi) = ((q_0, I^{(0)}), o_0)((q_1, I^{(1)}), o_0o_1)((q_2, I^{(2)}), o_0o_1o_2)...$. Since r is an accepting run in $\mathcal{T}$, the sequence $\eta = q_0q_1q_1...$ is an accepting run in $\mathcal{A}_\psi$. Therefore, because of the definition of the set $I^{(i+1)}$ as $\mathrm{post}_{t(o_0...o_i)}(I^{(i)}_{q_i,q_{i+1}}, o_{i+1})$, the sequence η is an accepting run in $\mathcal{A}_\psi$ on the traces of all executions $\rho = v_0v_1v_2...$ s.t. $v_i \in I^{(i)}$ and therefore $\mathsf{trace}(\rho) \in \mathcal{L}_{uc}(\mathcal{A}_\psi)$.

Moreover, since the automata $\mathcal{A}_\psi$ and $\mathcal{T}$ are universal and $\forall i \geq 0$, $\bigcup_{q' \in Q^{\mathcal{A}}} \mathrm{post}_a(I^{(i)}_{q_i,q'}, o) = \mathrm{post}_a(I^{(i)}, o)$, it holds that $\mathsf{trace}(\rho) \in \mathcal{L}_{uc}(\mathcal{A}_\psi)$ for all executions ρ compatible with a branch $u = o_0a_0o_1a_1o_2...$ of t. But since the set of such executions ρ is exactly $\mathsf{exec}(\mathcal{M}, \sigma_0)$, we conclude that the formula ψ is realizable in $\mathcal{M}$. ∎

Moreover, thanks to Theorem 3.4.10, if a UCT has a non-empty language, it accepts a tree that is the unfolding of a finite graph, or equivalently, that can be represented by a finite state Moore machine. Therefore if ψ is realizable, it is realizable by a finite-memory strategy. In this thesis we will also use the notation $\mathcal{T}_{\psi,X}$ for the UCT built for the LTL formula ψ where the executions of $\mathcal{M}$ start from the set $X \subseteq V$,i.e., $Q_0 = \{(q_{init}, X)\}$.

4.2.2 Positive KLTL Synthesis

In this section we extend the construction of Section 4.2.1 to the positive fragment of KLTL. We recall that the Positive KLTL ($KLTL^+$) (defined in Section 2.2.2) extends LTL with the knowledge operator K_i that appears under a even number of negations. Moreover, we assume that the formulas are in negative normal form, meaning that all

negations are pushed down towards the atoms. Since in this section we consider only one partially informed process, the operator K is intended to reason about its knowledge.

Intuition on the construction

Given an $KLTL^+$ formula φ and an interaction model $\mathcal{M} = \langle \mathcal{P}, \Omega = \{0,1\}, \Sigma_0, \Sigma_1, V, V_0, E, \tau \rangle$, we show how to construct a UCT $\mathcal{T}$ such that $\mathcal{L}_{uc}(\mathcal{T}) \neq \varnothing$ if and only if φ is realizable in $\mathcal{M}$.

The construction is compositional and follows, for the basic blocks, the construction of Section 4.2.1 for LTL formulas. The main idea is to replace iteratively the innermost subformulas of the form $K\gamma$ by fresh atomic propositions p_γ so that we get an LTL formula for which the realizability problem can be transformed into the emptiness of an universal coBüchi tree automaton. The realizability of the subformulas $K\gamma$ that have been replaced by p_γ is then checked by branching universally to an UCT constructed for γ as in Section 4.2.1 (γ is an LTL formula over an extended alphabet). Since transitions are universal, it will ensure that all infinite branches of the tree from the current vertex where the new automaton has been triggered also satisfy the formula γ.

The UCTs we construct are defined over an extended alphabet that contains the new atomic propositions, but we show that we can safely project the final universal tree automaton on the alphabet Σ_0.

The assumption on positivity of KLTL formulas implies that there is no subformulas of the form $\neg K\gamma$. If subformulas $\neg K\gamma$ would appear, the construction would not work since one would need to check that there is one execution compatible with one of the branches of the tree to accept that does not satisfy the formula γ. This would require nondeterministic transitions and therefore the automaton $\mathcal{T}$ would become alternating.

Formal definition

We now describe the construction formally. We inductively define the sequence of formulas associated with φ as:

$\varphi^0 = \varphi$ and

φ^{i+1} is obtained from φ^i in which the *innermost* subformulas $K\gamma$ are replaced by fresh atomic propositions p_γ.

Let d be the smallest index such that φ^d is an LTL formula. In other words, d is the maximal nesting level of the knowledge operators K. Also, we identify by

$$\mathbb{K} = \bigcup_{i=0}^{d} \{p_\gamma \mid K\gamma \text{ is subformula of } \varphi^i\}$$

the set of new atomic propositions and let $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$ be the extended set of atomic propositions. Note that by the definition of the formulas φ^i, for all atomic propositions p_γ occurring in φ^i, γ is an LTL formula over $\mathcal{P}'$.

Example 4.2.1. *Let us take the KLTL$^+$ formula $\varphi = p \rightarrow K(q \rightarrow (Kr \vee Kz))$ and $\mathcal{P} = \{p, q, r, z\}$. The sequence of formulas φ^i is:*

$$\varphi^0 = \varphi$$
$$\varphi^1 = p \rightarrow K(q \rightarrow (p_r \vee p_z))$$
$$\varphi^2 = p \rightarrow p_\gamma \text{ where } \gamma = q \rightarrow (p_r \vee p_z)$$

Also, the extended set of atomic propositions is $\mathcal{P}' = \{p, q, r, z, p_r, p_z, p_\gamma\}$ where $\gamma = q \rightarrow (p_r \vee p_z)$.

Then, we construct incrementally a chain of universal coBüchi tree automata $\mathcal{T}^d, ..., \mathcal{T}^0$ such that

$$\mathcal{L}_{uc}(\mathcal{T}^d) \supseteq \mathcal{L}_{uc}(\mathcal{T}^{d-1}) \supseteq ... \supseteq \mathcal{L}_{uc}(\mathcal{T}^0)$$

and the following invariant is satisfied: for all $i \in \{0, ..., d\}$, $\mathcal{T}^i$ accepts exactly the set of strategies that realize φ^i in $\mathcal{M}$

Intuitively, the automaton $\mathcal{T}^i$ is defined by adding new transitions in $\mathcal{T}^{i+1}$, so that for all atomic propositions p_γ occurring in φ^{i+1}, $\mathcal{T}^i$ will ensure that $K\gamma$ is satisfied. This is done by branching to a UCT verifying γ whenever the atomic proposition p_γ is met.

Since the formulas φ^i are defined over the extended alphabet $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$ and the interaction model $\mathcal{M}$ is defined over $\mathcal{P}$, we now make clear what we mean by realizability of a formula φ^i in $\mathcal{M}$. It uses the notion of *extended model executions* and *extended strategies*.

Extended actions, model executions and strategies We extend the actions of Agent 0 to $\Sigma_0' = \Sigma_0 \times 2^{\mathbb{K}}$ and call them *e-actions*. Informally, Agent 0 plays an e-action (a, K) if he considers formulas $K\gamma$ for all $p_\gamma \in K$ to be true. An *extended execution* (e-execution) of $\mathcal{M}$ is an infinite sequence $\rho = (v_0, K_0)(v_1, K_1)... \in (V \times 2^{\mathbb{K}})^\omega$ such that $v_0 v_1 ... \in \mathsf{exec}(\mathcal{M})$. We denote $v_0 v_1 \dots$ by $\mathrm{proj}_1(\rho)$ and $K_0 K_1 \dots$ by $\mathrm{proj}_2(\rho)$. The *extended labeling* function τ' is a function from $V \times 2^{\mathbb{K}}$ to $\mathcal{P}'$ defined by $\tau'(s, K) = \tau(s) \cup K$. The indistinguishability relation between extended executions is defined, for any two extended executions ρ_1, ρ_2, by $\rho_1 \sim_0 \rho_2$ if and only if $\mathrm{proj}_1(\rho_1) \sim_0 \mathrm{proj}_1(\rho_2)$ and $\mathrm{proj}_2(\rho_1) = \mathrm{proj}_2(\rho_2)$, i.e., the propositions in $\mathbb{K}$ are visible to Agent 0. Given the extended labeling functions and indistinguishability relation, the KLTL satisfiability notion $R, \rho, i \vDash \psi$ can be naturally defined for a set of e-execution R, $\rho \in R$ and ψ a KLTL formula over $\mathcal{P}' = \mathcal{P} \cup \mathbb{K}$.

An extended strategy is a strategy defined over e-actions, i.e. a function from $\mathcal{O}(\Sigma_0'\mathcal{O})^*$ to Σ_0'. For an infinite sequence $u = o_0(a_0, K_0)o_1(a_1, K_1)o_2 \cdots \in \mathcal{O}(\Sigma_1'\mathcal{O})^\omega$, we define

$\text{proj}_1(u)$ as $o_0a_0o_1a_1o_2\dots$. The sequence u defines a set of compatible e-executions $\mathsf{exec}(\mathcal{M}, u)$ as follows: it is the set of e-executions $\rho = (v_0, K_0)(v_1, K_1)... \in (V \times 2^{\mathbb{K}})^\omega$ such that $\text{proj}_1(\rho) \in \mathsf{exec}(\mathcal{M}, \text{proj}_1(u))$. Similarly, we define for e-strategies σ'_0 the set $\mathsf{exec}(\mathcal{M}, \sigma'_0)$ of e-executions compatible with σ'_0. Then,

> A KLTL formula ψ over $\mathcal{P}'$ is realizable in $\mathcal{M}$ if there is an e-strategy σ'_0 such that for all runs $\rho \in \mathsf{exec}(\mathcal{M}, \sigma'_0)$, we have $\mathsf{exec}(\mathcal{M}, \sigma'_0), \rho, 0 \vDash \psi$.

Proposition 4.2.2. *There is an e-strategy $\sigma'_0 : \mathcal{O}(\Sigma'_0\mathcal{O})^* \to \Sigma'_0$ realizing φ^0 in $\mathcal{M}$ iff there exists a strategy $\sigma_0 : \mathcal{O}(\Sigma_0\mathcal{O})^* \to \Sigma_0$ realizing φ^0 in $\mathcal{M}$.*

Proof. Let us see e-strategies and strategies as the $(\Sigma'_0 \cup \{\#\})$-labeled (resp. $(\Sigma_0 \cup \{\#\})$-labeled) $\mathcal{O}$-trees encoding them. Given a tree representing σ'_0, we project its labels on Σ_0 to get a tree representing σ_0. The strategy σ_0 defined in this way realizes φ^0, as φ^0 does not contain any occurrence of propositions in $\mathbb{K}$. Conversely, given a tree representing σ_0, we extend the labels of all nodes except the root with $\varnothing$ to get a tree representing σ'_0. It can be shown for the same reasons that σ'_0 realizes φ^0. ∎

Incremental tree automata construction The invariant mentioned above can now be stated more precisely: for all i,

$$\mathcal{T}^i \text{ accepts the } e\text{-strategies } \sigma'_0 : \mathcal{O}(\Sigma'_0\mathcal{O}) \to \Sigma'_0 \text{ that realize } \varphi^i \text{ in } \mathcal{M}.$$

Therefore, the trees accepted by the UCT $\mathcal{T}^i$ are labeled with e-actions in Σ'_0 and have as directions elements of $\mathcal{O}$. We now explain how they are constructed.

Since φ^d is an LTL formula, we follow the construction of Section 4.2.1 to build the UCT $\mathcal{T}^d$. Then, we construct $\mathcal{T}^i$ from $\mathcal{T}^{i+1}$, for $0 \leq i < d$. The invariant tells us that $\mathcal{T}^{i+1}$ defines all the e-strategies that realize φ^{i+1} in $\mathcal{M}$. It is only an over-approximation of the set of e-strategies that realize φ^i in $\mathcal{M}$ since the subformulas of φ^i of the form $K\gamma$ correspond to atomic propositions p_γ in φ^{i+1}, and therefore $\mathcal{T}^{i+1}$ does not check that they are satisfied. Therefore to maintain the invariant, $\mathcal{T}^i$ is obtained from $\mathcal{T}^{i+1}$ such that whenever an action that contains some formula $p_\gamma \in \mathsf{Sub}(\varphi^{i+1})$ occurs on a transition of $\mathcal{T}^{i+1}$, we trigger (universally) a new transition to a UCT $\mathcal{T}_{\gamma,I}$, for the current information set I in $\mathcal{T}^{i+1}$, that will check that $K\gamma$ indeed holds.

As already mentioned, the assumption on positivity of KLTL formulas is necessary here as we do not have to check for formulas of the form $\neg K\gamma$, which could not be done without an involved "non Safraless" complementation step. Since γ is necessarily an LTL formula over $\mathcal{P}'$ by definition of the formula φ^{i+1}, we can apply the construction of Section 4.2.1 to build $\mathcal{T}_{\gamma,I}$.

Formally, from the incremental way of constructing the automata $\mathcal{T}^j$ for $j \geq i$, we know that $\mathcal{T}^{i+1}$ has a set of states Q_{i+1} where all states are of the form (q, I) where $I \subseteq V$ is some knowledge set. In particular, it can be verified to be true for the state space of $\mathcal{T}^d$

by definition of the construction of Section 4.2.1. Let also δ_{i+1} be the transition relation of $\mathcal{T}^{i+1}$. For all formulas γ such that p_γ occurs in φ^{i+1}, we let Q_γ be the set of states of $\mathcal{T}_{\gamma,I}$, δ_γ its set of transitions and α_γ be the set of its final states. Again from the construction of Section 4.2.1, we know that $Q_\gamma = Q^{\mathcal{A}_\gamma} \times 2^V \cup \{(\bot, \varnothing), (q_w, \varnothing), (q^\gamma_{init}, I)\}$ where $Q^{\mathcal{A}_\gamma}$ is the set of states of a UCW associated with γ (assumed to be disjoint from that of $\mathcal{T}^{i+1}$) and $Q^0_\gamma = \{(q^\gamma_{init}, I)\}$ is the set of initial states of $\mathcal{T}_{\gamma,I}$. However, since all the states in I have the same observation and the current observation is known, we can consider the initial set of states in the automaton $\mathcal{T}_{\gamma,I}$ being $\{(q^\gamma_0, I) \mid q^\gamma_0 \in Q^0_\gamma\}$ and that it accepts Σ'-labeled $\mathcal{O}$-trees.

The automaton $\mathcal{T}^i$ is then formally defined as $\mathcal{T}^i = \langle \Sigma'_0, \mathcal{O}, Q_i, Q^i_0, \delta_i, \alpha_i \rangle$ where, assuming without loss of generality that there is a unique initial state $q^\gamma_0 \in Q^{\mathcal{A}_\gamma}$ in the UCW $\mathcal{A}_\gamma$, we have

- $Q_i = Q_{i+1} \cup \bigcup_{p_\gamma \in \mathsf{Sub}(\varphi^{i+1})} Q_\gamma$
- $Q^i_0 = Q^{i+1}_0 = \{(q_{init}, V_0)\}$
- $\delta_i((q, I), (a, K), o) = \delta_{i+1}((q, I), (a, K), o) \cup \bigcup_{p_\gamma \in \mathsf{Sub}(\varphi^{i+1})} \delta_\gamma((q^\gamma_0, I), (a, K), o)$
- $\alpha_i = \alpha_{i+1} \cup \alpha_\gamma$

The transition relation for the other cases is the same as in the automaton $\mathcal{T}^{i+1}$.

In the following, we prove that the automaton $\mathcal{T}^i$ accepts exactly the strategies that realize φ^i in the extended runs of the interaction model $\mathcal{M}$. An e-strategy of the system in this case can be seen as an appropriate $((\Sigma_0 \times 2^{\mathbb{K}}) \cup \{\#\})$-labeled $\mathcal{O}$-tree t as explained in Chapter 3 and then we extend the notation for the set of executions induced by the strategy tree in the interaction model by $\mathrm{exec}(\mathcal{M}, t)$. Note that a branch of t is a sequence $\pi = o_0 \tilde{a}_1 o_1 \tilde{a}_2 o_2 \ldots$ where $\forall i$, $\tilde{a}_i \in \Sigma_0 \times 2^{\mathbb{K}}$ and $o_i \in \mathcal{O}$.

As we mentioned before, we start our construction with the LTL formula φ^d and construct the universal coBüchi tree automaton $\mathcal{T}^d$ for it. Then, we build the sequence of automata $\mathcal{T}^d$, $\mathcal{T}^{d-1}$,... $\mathcal{T}^0$ such that $\mathcal{T}^i$ checks for the satisfaction of φ^i on the extended executions compatible with the branches of accepted trees. In other words,

$$\forall t \in \mathcal{L}(\mathcal{T}^i),\ \forall \rho \in \mathrm{exec}(\mathcal{M}, t),\ \text{holds } \mathrm{exec}(\mathcal{M}, t), \rho, 0 \vDash \varphi^i.$$

In order to prove that the invariant holds, let us define the set of all extended executions that are compatible with the branch π of the tree t up to position j as:

$$\mathrm{exec}(\mathcal{M}, t, \pi, j) = \bigcup_{\pi':\ \pi'[0\ldots j] = \pi[0\ldots j]} \mathrm{exec}(\mathcal{M}, \pi')$$

This executions are indistinguishable from the ones in $\mathrm{exec}(\mathcal{M}, \pi)$ up to position j. Observe that, by definition, we have $\mathrm{exec}(\mathcal{M}, t, \pi, 0) = \mathrm{exec}(\mathcal{M}, t)$.

Definition 4.2.1. *A $(\Sigma_0 \times 2^{\mathbb{K}} \cup \{\#\})$-labelled $\mathcal{O}$-tree t encoding a strategy is called fair with respect to an one-agent $KLTL^+$ formula ϕ if:*

$$\begin{aligned}&\forall \pi = o_0(a_0, K_0)o_1(a_1, K_1)o_2..., \forall j, \forall p_\gamma \in K_j \cap \mathsf{Sub}(\phi),\\ &\forall \pi' \ s.t. \ \pi'[0...j] = \pi[0...j], \ \forall \rho \in \text{exec}(\mathcal{M}, \pi'),\\ &\ holds \ \text{exec}(\mathcal{M}, \pi'), \rho, j \vDash \gamma\end{aligned}$$

Intuitively, whenever p_γ appears in the label of a node of the tree along a branch π, the formula γ holds at that position j on all the executions compatible with π up to that position. These executions are indistinguishable up to position j. Because t is a tree, the branches π' that pass by the node $x = o_0o_1...o_j$, have the same prefix as π up to position j. Therefore, the set of e-executions indistinguishable up to position j from an execution compatible with π consists of the e-executions that are compatible with some π' having the same prefix as π up to position j.

Lemma 4.2.2. *If a $(\Sigma_1 \times 2^{\mathbb{K}} \cup \{\#\})$-labeled $\mathcal{O}$-tree t encoding a strategy is fair with respect to ϕ, then*

$$\begin{aligned}\forall \psi \in \mathsf{Sub}(\phi), \forall \pi \ &branch \ of \ t, \forall j, \forall \rho \in \text{exec}(\mathcal{M}, t, \pi, j),\\ &if \ \text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi,\\ &then \ \text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi^{-1}\end{aligned}$$

where ψ^{-1} is obtained from ψ by replacing in one step the atomic propositions p_γ with the formula $K\gamma$.

Proof. Let us fix a branch $\pi = o_0(a_0, K_0)o_1(a_1, K_1)o_2(a_2, K_2)o_3...$ of t. Then, the proof is done by induction on the structure of ψ.

- if $\psi = p \in \mathcal{P}$, $\psi^{-1} = \psi$ and then, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash p$.
- if $\psi = p_\gamma$, since $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash p_\gamma$, holds $p_\gamma \in \text{proj}_2(\rho[j]) = K_j$. Then, because t is a fair tree with respect to ϕ and $\psi \in \mathsf{Sub}(\phi)$, holds that $\forall \pi' \in t$ s.t.$\pi[0...j] = \pi'[0...j]$, for all runs $\rho' \in \text{exec}(\mathcal{M}, \pi')$, we have that $\text{exec}(\mathcal{M}, t), \rho', j \vDash \gamma$. Then, since $\text{exec}(\mathcal{M}, t, \pi, j) = \bigcup_{\pi' : \pi[0...j] = \pi'[0...j]} \text{exec}(\mathcal{M}, \pi')$, $\text{exec}(\mathcal{M}, t, \pi, j), r, j \vDash K\gamma$.
- if $\psi = \psi_1 \wedge \psi_2$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_1 \wedge \psi_2$. That is, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_1$ and $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_2$. From the induction hypothesis, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_1^{-1}$ and $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_2^{-1}$ which means that $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_1^{-1} \wedge \psi_2^{-1}$. The proof is the same for $\psi = \psi_1 \vee \psi_2$.
- if $\psi = \psi_1 \mathcal{U} \psi_2$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \psi_1 \mathcal{U} \psi_2$. This means that $\exists j' \geq j$ s.t. $\forall j \leq k < j'$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, k \vDash \psi_1$ and $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j' \vDash \psi_2$.

Note that $\text{exec}(\mathcal{M}, t, \pi, k) \subseteq \text{exec}(\mathcal{M}, t, \pi, j)$ and $\text{exec}(\mathcal{M}, t, \pi, j') \subseteq \text{exec}(\mathcal{M}, t, \pi, j)$. Therefore, $\forall j \leq k < j'$, $\text{exec}(\mathcal{M}, t, \pi, k), \rho, k \vDash \psi_1$ and $\text{exec}(\mathcal{M}, t, \pi, j'), \rho, j' \vDash \psi_2$.

Then, from the inductive hypothesis, $\forall j \leq k < j'$, $\text{exec}(\mathcal{M}, t, \pi, k), \rho, k \vDash \psi_1^{-1}$ and $\text{exec}(\mathcal{M}, t, \pi, j'), \rho, j' \vDash \psi_2^{-1}$ which means $\forall j \leq k < j'$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, k \vDash \psi_1^{-1}$ and $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j' \vDash \psi_2^{-1}$. This is because $\text{exec}(\mathcal{M}, t, \pi, j)$ is the set of runs that are consistent with π up to position j and $\text{exec}(\mathcal{M}, T, \pi, j')$ and $\text{exec}(\mathcal{M}, T, \pi, k)$ are subsets of $\text{exec}(\mathcal{M}, T, \pi, j)$(since $j \leq k < j'$). But, all the runs in $\text{exec}(\mathcal{M}, T, \pi, j')$ (and $\text{exec}(\mathcal{M}, T, \pi, k)$ respectively) are distinguishable from the rest of words in $\text{exec}(\mathcal{M}, T, \pi, j)$. Then, $\text{exec}(\mathcal{M}, T, \pi, j), \rho, j \vDash \psi_1^{-1} \mathcal{U} \psi_2^{-1}$.

- if $\psi = \bigcirc\psi_1$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \bigcirc\psi_1$ and therefore $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j+1 \vDash \psi_1$. Because $\text{exec}(\mathcal{M}, t, \pi, j+1) \subseteq \text{exec}(\mathcal{M}, t, \pi, j)$ and from the induction hypothesis, we have $\text{exec}(\mathcal{M}, t, \pi, j+1), \rho, j+1 \vDash \psi_1^{-1}$. Using the same argument as before, holds that $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j+1 \vDash \psi_1^{-1}$ and therefore $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \bigcirc\psi_1^{-1}$.

- if $\psi = \Box\psi_1$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \Box\psi_1$. Then,$\forall k \geq j$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, k \vDash \psi_1$. Because $\forall k \geq j$ $\text{exec}(\mathcal{M}, t, \pi, k) \subseteq \text{exec}(\mathcal{M}, t, \pi, j)$ and from the inductive hypothesis, $\forall k \geq j$, $\text{exec}(\mathcal{M}, t, \pi, k), \rho, k \vDash \psi_1^{-1}$. Using the same argument as before, we get $\text{exec}(\mathcal{M}, t, \pi, j), \rho, k \vDash \psi_1^{-1}$, $\forall k \geq j$. That is, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \Box\psi_1^{-1}$

- if $\psi = K\psi_1$, the fact that $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash K\psi_1$ means that $\forall \rho' \in \text{exec}(\mathcal{M}, t, \pi, j)$, holds $\text{exec}(\mathcal{M}, t, \pi, j), \rho', j \vDash \psi_1$. From the induction hypothesis applied for ρ', we have that $\forall \rho' \in \text{exec}(\mathcal{M}, t, \pi, j)$, $\text{exec}(\mathcal{M}, t, \pi, j), \rho', j \vDash \psi_1^{-1}$. That is, $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash K\psi_1^{-1}$.

- if $\psi = \neg\psi_1$, then $K\gamma$ is not a subformula of ψ_1 because $K\gamma$ does not occur under negations in ψ. This means that $\psi_1 = \psi_1^{-1}$ and then $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \neg\psi_1$ implies that $\text{exec}(\mathcal{M}, t, \pi, j), \rho, j \vDash \neg\psi_1^{-1}$.

■

Then, denoting by $\text{exec}(\mathcal{M}[X], \pi)$ the set of e-executions in $\mathcal{M}$ that are compatible with π and start when the set of current possible states in $\mathcal{M}$ is X, we can prove that:

Lemma 4.2.3. *For all $i \in \{0, ..., d-1\}$, $\forall t \in \mathcal{L}_{uc}(\mathcal{T}^i)$, t is fair with respect to φ^{i+1}.*

Proof. The proof of this theorem comes directly from the construction of $\mathcal{T}^i$ from $\mathcal{T}^{i+1}$.

Let us fix $t \in \mathcal{L}_{uc}(\mathcal{T}^i)$. By construction, $\mathcal{T}^i$ is $\mathcal{T}^{i+1}$ to which are "plugged" the automata $\mathcal{T}_{\gamma,I}$ for the LTL formulas γ where $p_\gamma \in \mathsf{Sub}(\varphi^{i+1})$ appears on a transition of $\mathcal{T}^{i+1}$.

Therefore, if at position j on a branch $\pi = o_0(a_0, K_0)o_i(a_1, K_1)o_2...$ of t holds $p_\gamma \in K_j \cap \mathsf{sub}(\varphi^{i+1})$, there also starts the execution of $\mathcal{T}_{\gamma,I}$ that accepts the "maximal" subtree of t with the root at position j on branch π. Let us call this subtree $t_{\pi,j}$ and observe that it is formed from the suffixes $\pi'[j...]$ of all the branches π' of t such that $\pi'[:j] = \pi[:j]$.

Therefore, since $t_{\pi,j} \in \mathcal{L}_{uc}(\mathcal{T}_{\gamma,I})$ for some information set I and γ is an LTL formula, using the results in Section 4.2.1, we have that $\text{exec}(\mathcal{M}[I], \pi'[j...]) \vDash \gamma$. That is, $\forall \pi'$ branch of t such that $\pi'[:j] = \pi[:j]$ and $\forall \rho \in \text{exec}(\mathcal{M}, \pi')$, we have $\text{exec}(\mathcal{M}, \pi'), \rho, j \vDash \gamma$. This means that t is fair with respect to φ^{i+1}. ■

Lemma 4.2.4. *For all $i \geq 0$, $\mathcal{L}(\mathcal{T}^i)$ accepts the set of e-strategies that realize φ^i in $\mathcal{M}$.*

Proof. An e-strategy σ'_0 realizes φ^i if $\forall \rho \in \text{exec}(\mathcal{M}, \sigma'_0)$, holds $\text{exec}(\mathcal{M}, \sigma'_0), \rho, 0 \vDash \varphi^i$. That is, if we look at σ'_0 as a $(\Sigma'_0 \cup \{\#\})$-labeled $\mathcal{O}$-tree $t_{\sigma'_0}$ encoding it and remember that each branch of $t_{\sigma'_0}$ has different sequence of observations, we equivalently have that

$$t_{\sigma'_0} \text{ realizes } \varphi^i \text{ iff } \forall \pi \text{ a branch of } t_{\sigma'_0},\ \forall \rho \in \text{exec}(\mathcal{M}, \pi),\ \text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^i.$$

Therefore, we prove that a tree t encoding a strategy is accepted by $\mathcal{T}^i$ if and only if it satisfies the above property. We do the proof of the theorem by induction on i.

For the base case, if $i = d$, $\mathcal{T}^d$ is the automaton built for the LTL formula φ^d as in Section 4.2.1 and the proof is done with a sample adaptation of the proof of Lemma 4.2.1.

For the inductive step, we assume that the desired property holds for $\mathcal{T}^{i+1}$ and prove it for $\mathcal{T}^i$. From the inductive hypothesis, we have that $t \in \mathcal{L}_{uc}(\mathcal{T}^{i+1})$ iff $\forall \pi$ branch of t, $\forall \rho \in \text{exec}(\mathcal{M}, \pi)$, holds $\text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^{i+1}$. But, since $\mathcal{L}_{uc}(\mathcal{T}^{i+1}) \supseteq \mathcal{L}_{uc}(\mathcal{T}^i)$, it also holds that $t \in \mathcal{L}_{uc}(\mathcal{T}^i)$ iff $\forall \pi$ branch of t, $\forall \rho \in \text{exec}(\mathcal{M}, \pi)$ we have $\text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^{i+1}$.

Let us first take $t \in \mathcal{L}_{uc}(\mathcal{T}^i)$ and a branch π of t. By Lemma 4.2.3, t is fair with respect to φ^{i+1}. Also, since all the runs $\rho \in \text{exec}(\mathcal{M}, \pi)$ are distinguishable from the other runs from $\text{exec}(\mathcal{M}, t, \pi, 0) \setminus \text{exec}(\mathcal{M}, \pi)$ (because they are compatible with different branches π' that have different observations o_i), we have that $\forall \rho \in \text{exec}(\mathcal{M}, \pi)$ holds $\text{exec}(\mathcal{M}, t, \pi, 0), \rho, 0 \vDash \varphi^{i+1}$. Then, by applying Lemma 4.2.2, results that $\text{exec}(\mathcal{M}, t, \pi, 0), \rho, 0 \vDash \varphi^i$ and because of the inclusion $\text{exec}(\mathcal{M}, \pi) \subseteq \text{exec}(\mathcal{M}, t, \pi, 0)$, we have that $\text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^i$.

Let us now take t such that $\forall \pi$ a branch of t, $\forall \rho \in \text{exec}(\mathcal{M}, \pi)$, holds $\text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^i$. We can label t with p_γ for all innermost subformulas $K\gamma$ of φ^i if and only if γ holds on all branches passing through that node. Therefore, it also holds $\text{exec}(\mathcal{M}, \pi), \rho, 0 \vDash \varphi^{i+1}$ and from the inductive hypothesis, $t \in \mathcal{L}_{uc}(\mathcal{T}^{i+1})$.

Then, $\mathcal{T}^i$ is built from $\mathcal{T}^{i+1}$ by triggering the automaton $\mathcal{T}_{\gamma,I}$ for all propositions p_γ that appear in φ^{i+1}. But, from the way we extended the labeling of t, the automaton $\mathcal{T}_{\gamma,I}$ always accepts the subtree it is triggered for. Hence, $t \in \mathcal{L}_{uc}(\mathcal{T}^i)$. ■

From Lemma 4.2.4, we know that $\mathcal{L}(\mathcal{T}^0)$ accepts the set of e-strategies that realize $\varphi^0 = \varphi$ in $\mathcal{M}$. Then by Proposition 4.2.2 we get:

Theorem 4.2.1. *For any* KLTL$^+$ *formula φ, one can construct a UCT $\mathcal{T}^0$ such that*

$$\mathcal{L}_{uc}(\mathcal{T}^0) = \{t_{\sigma_0} \mid \sigma_0 \text{ realizes } \varphi \text{ in } \mathcal{M}\}$$

Proof. Let us first take a tree t that is accepted by $\mathcal{T}^0$ and that encodes a e-strategy $\sigma'_0 : \mathcal{O}(\Sigma'_0\mathcal{O})^* \to \Sigma'_0$ of Agent 0. By Lemma 4.2.4, σ'_0 realizes φ^0. Then, from Proposition 4.2.2, there exists a strategy $\sigma_0 : \mathcal{O}(\Sigma_0\mathcal{O})^* \to \Sigma_0$ of Agent 0 that realizes $\varphi^0 = \varphi$ in $\mathcal{M}$.

Now, if φ is realizable in $\mathcal{M}$, there is a strategy $\sigma_0 : \mathcal{O}(\Sigma_0 \times \mathcal{O})^* \to \Sigma_0$ of Agent 0 that realizes $\varphi^0 = \varphi$ in the model $\mathcal{M}$. From Proposition 4.2.2, there is an e-strategy $\sigma'_0 : \mathcal{O}(\Sigma'_0\mathcal{O})^* \to \Sigma'_0$ that realizes φ^0 in $\mathcal{M}$. Then, again by Lemma 4.2.4, the tree encoding σ'_0 is accepted by $\mathcal{T}^0$. ■

Therefore, the automaton searched in this section to accept exactly strategies that realize the specification φ is the automaton $\mathcal{T}^0$ obtained at the end of the constructed chain of automata. The number of states of $\mathcal{T}^0$ is (in the worst-case) $2^{|V|}.(2^{|\varphi^d|} + \sum_{p_\gamma \in \mathbb{K}} 2^{|\gamma|} + 1) + 1$, and since $|\varphi^d| + \sum_{p_\gamma \in \mathbb{K}} |\gamma|$ is bounded by $|\varphi|$, the number of states of $\mathcal{T}^0$ is $O(2^{|V|+|\varphi|})$. In the rest of the chapter we refer to automaton $\mathcal{T}^0$ as the automaton $\mathcal{T}$ that accept the set of strategies that realize φ.

Applying the Antichain algorithm to Test the Emptiness As Theorem 4.2.1 says, in order to test the $KLTL^+$ realizability, we have to test the emptiness of the universal tree automaton $\mathcal{T}$. For this, we use the antichain algorithm presented in Section 3.4.5. We recall that the incremental algorithm starts with a small bound b and first equivalently transforms a universal coBüchi tree automaton into an universal b-coBüchi automaton for some bound $b \leq 2n^2(n!)^2$ where n is the number of states of the tree automaton. Then,it builds a safety game that is solved on-the-fly using antichains. In the case the protagonist of the game has a strategy to win the game, his strategy also realizes the formula φ in the interaction model $\mathcal{M}$. If there is no winning strategy, the bound b is incremented and the procedure is repeated. In the case the formula is not realizable, the algorithm stops when the bound reaches the value $2n^2(n!)^2$.

4.3 Implementation and Test Cases

In this section we briefly present our prototype implementation *Acacia–K*[1] for positive KLTL synthesis, and provide some interesting examples on which we tested the tool, on a laptop equipped with an Intel Core i7 2.10Ghz CPU. *Acacia–K* extends the LTL synthesis tool *Acacia*$^+$[12]. As Acacia+, the implementation is made in Python together with C for the low level operations that need efficiency.

4.3.1 Implementation

As Acacia+, the tool is available in one version working on both Linux and MacOsX and can be executed using the command-line interface. As parameters, *Acacia–K* requires a file containing the KLTL$^+$ formula, one defining the sets of actions for each process and the

partition of the atomic propositions in visible and invisible signals (atomic propositions labeling states of the interaction model) that defines the indistinguishability relation for Agent 0. Finally, it also requires a file with the description of the interaction model. The output of the tool is a winning strategy, if the formula is realizable, given as a Moore machine described in Verilog and, if this strategy is small, *Acacia–K* also outputs it as a picture.

In order to have a more efficient implementation, the construction of the automata for the LTL formulas γ is made on demand. That is, we construct the UCT $\mathcal{T}_\gamma$ incrementally by updating it as soon as it needs to be triggered from some state (q, I) which has not been constructed yet.

As said before, the synthesis problem is reduced to the problem of solving a safety game for some bound b on the number of visits to accepting states. The tool is incremental: it tests the realizability for small values of b first and increments it as long as it cannot conclude for realizability. In practice, we have observed, as for classical LTL synthesis, that small bounds b are sufficient to conclude for realizability. However if the formula is not realizable, we have to iterate up to a large upper bound, which in practice is too large to give an efficient procedure for testing unrealizability. We leave as future work the implementation of an efficient procedure for testing unrealizability.

4.3.2 Light Bulb Controller

Let us come back to the Light Bulb Example (Example 2.1.2). We recall that there are two agents. The first one, Agent 0, controls the switch by playing "toggle"(T) to change its position or "skip"(S) for doing nothing. The second agent, Agent 1, is used to model the environment that may time out the toggle switch or may decide that the light bulb is broken. Let us also consider that Agent 0 has partial information and only sees the light and the specification is $\varphi = \Box(K(t) \vee K(\neg t))$. We have seen in Example 2.3.1 that the formula is realizable if the initial set of states is $\{v_0, v_1\}$.

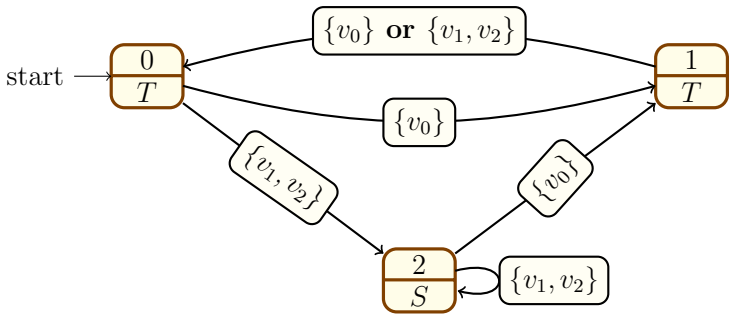


Figure 4.1: Winning strategy synthesized by *Acacia* – K for Example 2.1.2

We tested the tool *Acacka* – K on this example and the strategy provided by the tool is depicted in Figure 4.1. The labels of the states are the actions that have to be played

by the agent and the labels of the transitions are the possible observations that the agent may receive from the environment. The provided strategy by the tool asks to play first "toggle" and then keep on playing "skip". Depending on the observation he gets, the system (Agent 0) goes in a different state. The state 0 is for the start, state 1 is the "error" state in which the system goes if he receives a wrong observation. That is, the environment gives an observation even if he cannot go in a state having that observation. Then, if the observation is correct, after playing the action "toggle" from the initial states $\{v_0, v_1\}$, the interaction model is forced to go in v_2 and by playing the action "skip", the system forces the interaction model to stay in v_2 and he will know that t is false. In the strategy, this situation corresponds to the state 2. For this example, *Acacia−K* constructed a UCT with 31 states and the total running time is 0.2s.

4.3.3 The 3-Coins Game

Example 4.3.1 (The 3-Coins Game)**.** *Another example that we tried is a game played using three coins which are arranged on a table with either head or tail up. The system (Agent 0) does not see the coins, but knows at each time the number of tails and heads. Then, the game is infinitely played as follows. At the beginning the environment (Agent 1) chooses an initial configuration and then at each round, the system chooses a coin and the environment has to flip that coin and inform the system about the new number of heads and tails. The objective of the system is to reach, at least once, the state in which all the coins have the heads up and to avoid all the time the state in which all the coins are tails. Depending on the initial number of tails up, the system may or may not have a winning strategy.*

In order to model this, we considered an interaction model whose states are labeled with atomic propositions c_1, c_2, c_3 for the three coins, which are not visible for the system, and two other variables b_1, b_0 which are visible and represent the bits encoding the number of *heads* in the configuration. The actions of the system are C_1, C_2, C_3 with which he chooses a coin and the environment has to flip the coin chosen by the system by playing only the action *done*. A picture of the environment generated by the tool is in Figure 4.2. The first action on transitions belongs to Agent 1 (the environment) and the second one is the action of Agent 0 (the system).

The specification is translated into the KLTL$^+$formula $\Diamond K(c_1 \wedge c_2 \wedge c_3) \wedge \Box K(c_1 \vee c_2 \vee c_3)$. Then, assuming that the initial state of the interaction model has two *heads* (the set of initial states is $\{s_1, s_2, s_3\}$), the synthesized strategy proposes to "check" the position of every coin by double flipping. If after one flip, the winning state is not reached, the system flips back the coin and at the third round he chooses another coin to check. In the Moore machine in Fig. 4.3 representing the strategy of Agent 0, the state 2 corresponds to the configuration in which there are only *heads* and the states 3 and 7 are the states

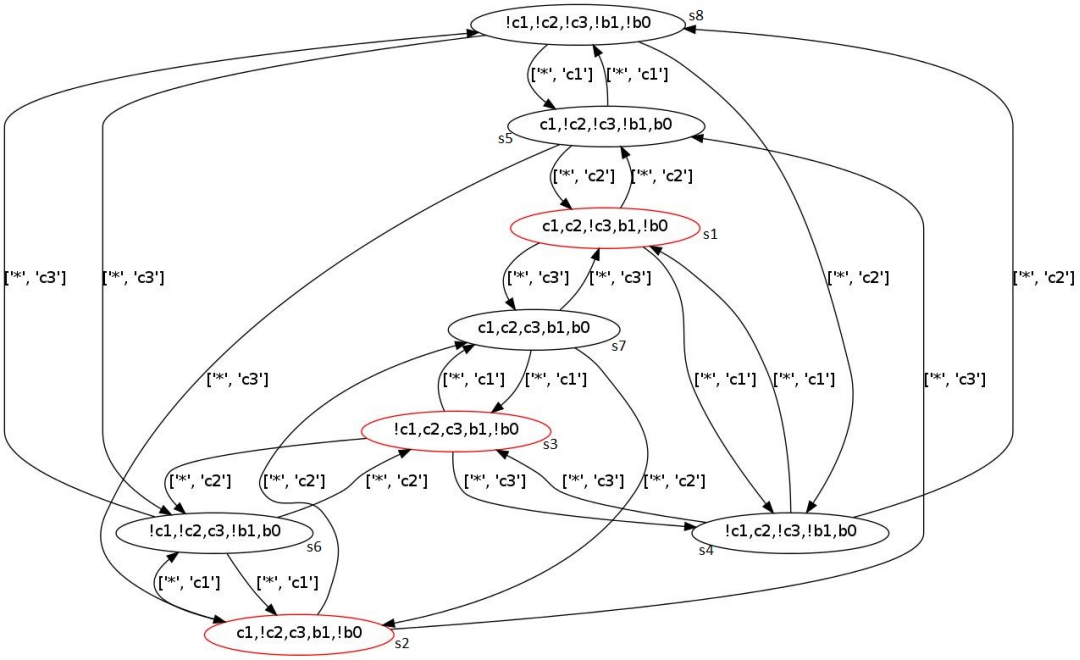


Figure 4.2: The interaction model for 3-Coins Game

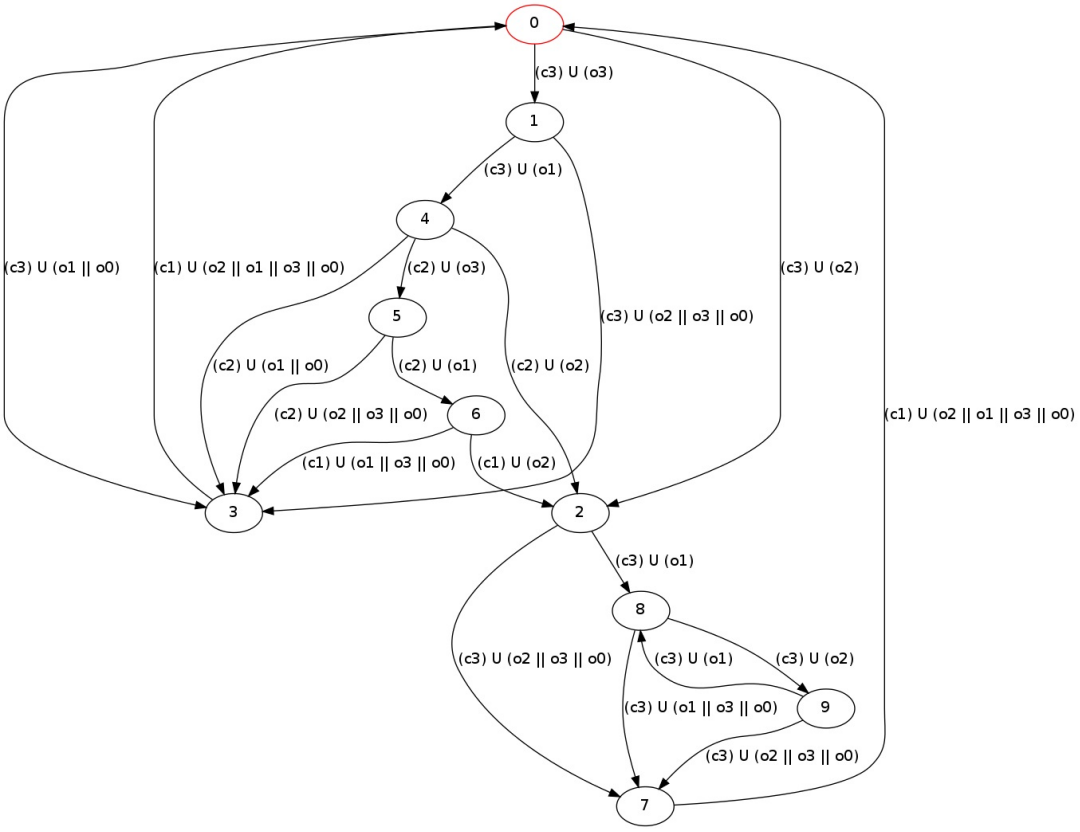


Figure 4.3: The winning strategy for 3-Coins Game

in which he goes when the environment cheated. Then, the strategy has two parts. One that leads to the state 2 by checking the coins and the second part in which the system plays only c_3 and alternates between the states s_1 and s_7 in the interaction model.

The strategy reads as follows: first (state 0) flip coin C_3 and if it gets observation o_2(won) go in state 2. Otherwise, if observation o_3(one *head*) is received, go in state 1. Finally, if the agent receives observations o_0(lost) or o_1(two heads), he goes in the error state 3. Then, the action of the following turns is depending on the current state. If the agent did not won (state 1), he turns again the coin C_3 and if he receives a correct observation goes in state 4 from which plays C_2 and proceeds in the same way as with the first coin. After the agent wins (state 2), he can always play the same action, as his objective to turn the three coins with heads up is already reached. It only remains to ensure that the three coins are not tails-up simultaneously.

For this example, *Acacia* – K constructs a UCT with 79 states, synthesizes a strategy with 10 states, and the total running time is 3.9s.

4.3.4 n-Prisoners Enigma

Example 4.3.2 (n-Prisoners Enigma)**.** *Finally, the last example is about n prisoners in a prison, each one in his own cell and they cannot communicate. Also, there is a room with a light bulb and a switch and a policeman that, at each moment of time, sends only one prisoner in that room and gives him the possibility to turn on or off the light. The prisoners can observe only the light when they are in the room. The guardians ensure that each prisoner is sent into the room an infinite number of times (fairness assumption). Before the game starts, the prisoners are allowed to communicate, and they know the initial state of the light. The goal of the prisoners is to learn whether all of them have visited the room at least once – more specifically, whenever all prisoners have visited the room, one specially designated prisoner must know that fact.*

We assume that the light is initially off. Then the winning strategy is that the special prisoner, say prisoner n, will count up to $n-1$*. For all* $1 \leq j \leq n-1$*, the fairness assumption ensures that prisoner j will visit the room again and again until the game stops. The first time j visits the room and the light is off, he turns it on, otherwise he does nothing. Prisoner n will turn the light off next time he enters the room, and increment his counter by* 1*. When the counter reaches* $n-1$*, prisoner n will be sure that all prisoners have visited the room at least once.*

To model this problem, it is natural to represent the guardians by the environment and the prisoners by multi-agents. Then, as the strategies of all prisoners except prisoner n are fixed, it remains the prisoner n (the agent to synthesize) that must figure out a winning strategy (ideally the counting strategy described above). We have modeled this example in Acacia-K and indeed, our tool find the strategy described above. Let us now

give more details about the formalization.

For three prisoners, $\mathcal{P} = \{on, x_1, x_2, p_1, p_2, p_3\}$ where the atomic proposition on corresponds to the light, values of x_i for $i \in \{1, 2\}$ is $true$ if the prisoner i already turned the light on, and the proposition p_i for $i \in \{1, 2, 3\}$ indicates the prisoner that is inside the room. Then, the special prisoner sees only the propositions $\{on, p_3\}$ indicating that the last prisoner sees all the time the light and can observe when he is inside the special room but cannot see what the other prisoners do.

We assume that at the beginning there is no one in the room. Then, the environment can propose an action in $\Sigma_1 = \{P_1, P_2, P_3\}$ deciding which prisoner is going in the room and prisoner 3 will decide if he wants the light on or off by choosing an action in $\Sigma_0 = \{t_{on}, t_{off}\}$. Note that the action of P_3 is ignored if he is not chosen by the environment. The transition relation asks that if the prisoner $p_i, i \in \{1, 2\}$ finds for the first time the light off ($x_1 = false$ and $on = false$), he turns on the light and the value of x_i changes and remains $true$ for all the reachable states from there.

Then, assuming that the environment is restricted to send all the prisoners in the special room infinitely many times, the KLTL$^+$formula that translates the goal is

$$\Box \bigwedge_{i=1}^{n} (\Diamond p_i) \rightarrow \Diamond K(\bigwedge_{i=1}^{n-1} x_i)$$

A winning strategy for the prisoner n would be to turn off the light whenever he is sent to the special room and to let it off if it already is. Then, after he finds the light on $n-1$ times when he is sent in that room, thanks to the strategy of the other prisoners, he will know that all of them passed by that room, and even more, all of them switched an the light. Assuming that the observations set $\mathcal{O} = \{o_0, o_1, o_2, o_3\}$ where $o_0 = \{s \in S \mid on \in \tau(s) \text{ and } p_3 \notin \tau(s)\}$, $o_1 = \{s \in S \mid on \notin \tau(s) \text{ and } p_3 \in \tau(s)\}$, $o_2 = \{s \in S \mid on \in \tau(s) \text{ and } p_3 \in \tau(s)\}$ and $o_3 = \{s \in S \mid on \notin \tau(s) \text{ and } p_3 \notin \tau(s)\}$, the strategy synthesized by $Acacia-K$ for three prisoners and corresponds to the intuitive strategy is illustrated in Figure 4.4 where the state 10 in the generated Moore machine corresponds to the moment when the prisoner p_3 knows that all the other prisoners passed through the special room and turned on the light.

We have tried 3/4/5/6 prisoners versions (including the protagonist) of this problem, obtaining a one hour timeout for 6 agents. The statistics we obtained are depicted in Table 4.1.

Again, $Acacia-K$ generates strategies that are natural, the same that one would synthesize intuitively. This fact is remarkable itself since, in synthesis, it is often a difficult task to generate small and natural strategies.

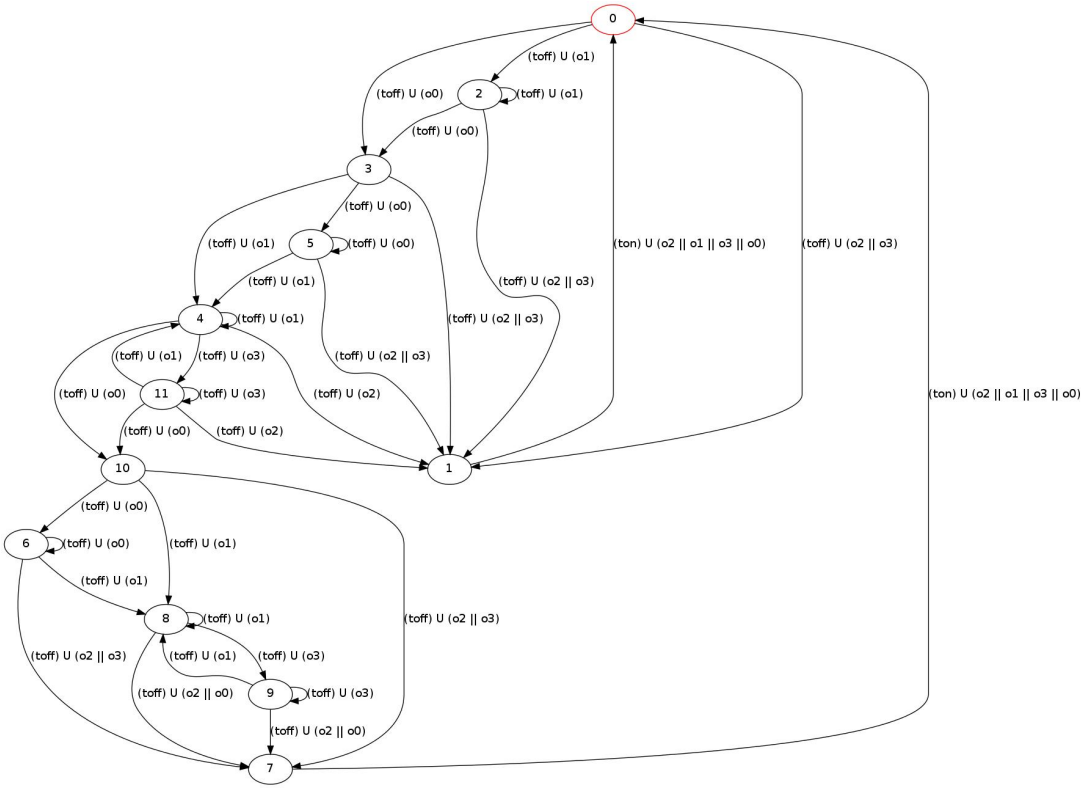


Figure 4.4: The winning strategy for 3-Prisoners Game

Pris #	$\|\mathcal{M}\|$	$\|UCT\|$	$\|tb-UCT\|$	Aut constr (s)	$\|\mathcal{M}_\lambda\|$	Total time(s)
3	21	144	692	1.79s	12	1.87s
4	53	447	2203	1.98s	16	13.20s
5	129	1310	6514	199.06s	20	553.45s ($\simeq$ 9 min)
6	305	3633	18125	6081.69s	N/A	N/A

Table 4.1: Statistic Results for n-Prisoners Example

4.3.5 Comparing Acacia-K with other tools

For the sake of comparison, we took some examples on which we tested the tool Acacia-K and tried to solve them in other existing tools.

MCMAS-SLK[17] is an open-source model-checker supporting the verification of interaction models against specifications in a variant of strategy logic (SLK) that includes epistemic operators (Section 3.2.3). MCMAS-SLK takes in input an interaction model specification and a set of SLK formulas to be verified. It evaluates the truth value of these formulas using algorithms based on Ordered Binary Decision Diagrams (OBDDs).

Whenever possible, MCMAS-SLK produces counterexamples for false formulas and witnesses for true formulas.

The interaction model is described in ISPL (Interpreted Systems Programming Language) that characterizes the Agents by means of local variables and their evolution in time conditioned by Boolean expressions. In the .ispl file two kinds of agents are described: "standard" agents, and the environment agent, being used to express constraints on systems or the infrastructure shared by the standard agents. This model naturally expresses systems with incomplete information.

The systems that are considered in [17] are memoryless where the agents local state do not include the local history of a run, implying the fact that the strategies are also memoryless. This semantics is chosen due to the fact that, as seen in Section 3.2.3, the model-checking problem becomes undecidable when considering more then two memoryfull agents with incomplete information.

Problems expressible in MCMAS-SLK Even if MCMAS-SLK does not allow memory for the agents, there are some (memoryfull) examples that can be modeled using memoryless semantics. It is the case of models in which agents need a finite memory that can be encoded in the current state of the agent. Some toy examples on which MCMAS-SLK was tested are dining cryptographers protocol, the cards game and muddy children puzzle on which are used the epistemic operators to translate the specifications.

Acacia-K vs. MCMAS-SLK Even if in Acacia-K we solve the synthesis problem under the assumption of imperfect information and perfect recall, we took some examples on which we tested our tool and tried to describe and verify them in MCMAS-SLK.

Let us first take the **timed toggle** example. In order to model this example in MSMAS-SLK, we need to use a variable *tog* that helps the protagonist deduce the current position of the toggle depending on the initial state and his actions. This variable changes between *on* and *off* if the Player knows the position of the toggle and his action is T (toggle) and takes a special value in case the protagonist lost this information. The formula to be verified for this example is

$$\langle\!\langle x \rangle\!\rangle (Agent, x) [\![y]\!] (Environment, y) \big(K(Agent, t) \vee K(Agent, \neg t) \big)$$

Then, the constructed model has 6 reachable states and MCMAS-SLK runs 0.011s before giving the winning strategy in Figure 4.5 that says to play all the time *toggle*. Also, it uses 22 BDD variables and 9047840 BDD memory.

In this case, MCMAS-SLK was faster then Acacia-K, but provided different strategy. Namely, the strategy is to play all the time T ("toggle").

Another example we consider is the **n-Prisoners** puzzle. We recall that the strategy of the special prisoner was to count how many times the light is turned on by the other

```
   Strategies:
---------- Strategy x [Player] ----------
Agent Player
   Environment.light=off, tog=NA (0): N
   Environment.light=off, tog=off (1): T
   Environment.light=off, tog=on (2): T
   Environment.light=on, tog=NA (3): T
   Environment.light=on, tog=on (4): T
-----------------------------------------
```

Figure 4.5: Strategy for timed toggle game given by McMAS-SLK

prisoners. Therefore, when modeling the problem in MCMAS-SLK, one needs a variable in the local state of the prisoner to count. After running this example for three prisoners, MCMAS-SLK outputs the winning strategy after 0.725s when the model has 63 reachable states and the BDD constructed uses 74 variables and 14438832 BDD memory.

We also tried the tool while increasing the number of agents. Then, we obtained the correct strategies for up to 4 prisoners. The obtained results are illustrated in Table 4.2. Even if MCMAS-SLK synthesized faster strategies for 3 and 4 prisoners, it did not output any strategy for 5 prisoners, problem solved in 9 minutes with Acacia-K.

Pris	Reach St	BDD var	BDD mem	Cache use	$\lvert\mathcal{M}_\lambda\rvert$	Total time(s)
3	63	74	14438832	94.71%	16	0.725s
4	173	146	62269216	100.00%	22	47.529s
5	N/A	434	2972996000	100.00%	N/A	N/A

Table 4.2: Statistics of n-Prisoners Puzzle in MCMAS-SLK

We stress out that the encoding of the above problems in MCMAS-SLK requires knowledge of the expected strategy for the protagonists. Using it, one encodes the states of the winning strategy in the states of the agent. This shows that Acacia-K is comparable with other tools.

4.4 Conclusions

In this chapter, we defined a Safraless procedure for the synthesis from KLTL$^+$specifications of a process with imperfect information and perfect recall against an omniscient antagonist environment. This problem is 2ExpTime-complete but we have shown that our procedure, based on universal coBüchi tree automata, can be implemented efficiently thanks to an antichain symbolic approach. We have implemented a prototype and run some preliminary experiments that prove the feasibility of our method. While

the universal coBüchi tree automata constructed by the tool are not small (around 1300 states), our tool can handle them, although in theory, the safety game could be exponentially larger than the automaton from which it is derived. Moreover, the tool synthesizes small strategies that correspond to the intuitive strategies we would expect, although it goes through a non-trivial automata construction.

The results obtained represent an encouraging (and necessary) step towards developing implementable procedures for the entire logic KLTL. A first step towards a procedure for KLTL specifications would be to consider assume-guarantees specifications $K\phi \to \psi$, where ϕ is an LTL formula and ψ a KLTL$^+$ formula. Already in order to solve the n-Prisoners Enigma such an assumption is needed. Namely, it is important that all prisoners are sent infinitely often in the special room so that the special prisoner have a strategy to eventually know that all the prisoners passed through that room. In this case the assumption is an LTL formula and the procedure for KLTL$^+$ applies.

When considering formulas of the type $K\phi \to \psi$ with ϕ is an LTL formula and ψ a KLTL$^+$ formula, not only that the environment should act *correctly* on the current execution, but he has to do the same thing on all the other (indistinguishable) executions.

5. Rational Synthesis with Perfect Information

In this section we study the rational synthesis problem when all agents have perfect information on the states of the interaction model. We first present the solution proposed by Kupferman et. all in [37, 53] which is based on a reduction to model-checking of some SL[NG] formula that can be verified in 2ExpTime.

Our contribution in this chapter is to study the problem and provide tight results for most of the classical omega-regular objectives, and show how to solve the cooperative and non-cooperative rational synthesis optimally. A short version of this chapter was published in [28].

We recall that the rational synthesis problem asks to synthesize strategies for a set P of processes that are executed in an rational environment represented by several processes with their own objectives.

As already mentioned in Section 2.4.2, there are two settings in which the rational synthesis problem is studied. In the first setting, the environment is considered to be *cooperative* and agree to adhere to a strategy profile as far as it is an equilibrium. Then, the problem to solve the cooperative rational synthesis is to find a strategy profile that is an equilibrium and for which the objectives of processes to synthesize are satisfied.

In the *non-cooperative* setting, the environment may not cooperate. Therefore, the problem is to synthesize strategies only for processes in the set P such that their objectives are satisfied on all executions that are compatible with these strategies and are outputs of some strategy profile which is in equilibria.

Turn-Based Setting Due to the equivalence of the interaction model with a game arena and the use of notions from game theory, in this chapter we often refer to processes as players and to executions as plays. We also consider that the interaction model has only one initial state. Moreover, we consider *turn-based interaction models* (at each step there is exactly one player that determines the next state) and give away the actions of the agents that label the transitions between states. Finally, we also remove the labeling on states and consider that each state $v \in V$ has associated one atomic proposition v that

is true only in state v. That is, $\mathcal{M} = \langle \Omega, V, (V_i)_{i \in \Omega}, v_0, E \rangle$ with $E \subseteq V \times V$ and $(V_i)_{i \in \Omega}$ a partition of V. Therefore, the action of Agent i from a state $v \in V_i$ is to choose a next state v' such that $(v, v') \in E$. Since there is only one initial state, there is *only one execution compatible with each strategy profile*. We often also introduce the objectives $\Theta_i \subseteq V^\omega$ in the definition of the interaction model. That is, $\mathcal{M} = \langle \Omega, V, (V_i)_{i \in \Omega}, v_0, E, (\Theta_i)_{i \in \Omega} \rangle$.

We discuss it in Section 5.5 the generalization of the procedures presented in this chapter to the concurrent setting and the resulting complexities.

Contributions Note first that in the case of perfect information, the problem is equivalent to considering only one process to synthesize (modeled by Agent 0) against the rational environment (modeled by Agents 1 ... k).

To better understand the computational complexity of the rational synthesis problems and how to manipulate their underlying games algorithmically, we consider variants of those problems for games played on turn-based graph structures for reachability, safety, Büchi, coBüuchi, parity, Rabin, Streett and Muller objectives and the rationality of the environment modeled as Nash equilibria. We also study the computational complexity of solving the rational synthesis problem when the number of players is fixed. This parameterised analysis makes sense as the number of components forming the environment may be limited in practical applications. The results we obtain are summarized in Table 5.1.

	Perfect Information Rational Synthesis Problem			
	Cooperative		Non-Cooperative	
	Unfixed k	Fixed k	Unfixed k	Fixed k
Safety	NP-c	Ptime-c	Pspace-c	Ptime-c
Reachability	NP-c	Ptime-c	Pspace-c	Ptime-c
Büchi	Ptime-c[87]	Ptime-c[87]	Pspace-c	Ptime-c
co-Büchi	NP-c[87]	Ptime-c	Pspace-c	Ptime-c
Parity	NP-c[87]	$UP \cap coUP$, parity-h	Exptime, Pspace-h	Pspace, NP-h, coNP-h
Streett	NP-c [87]	NP [87], NP-hard	Exptime,Pspace-h	Pspace-c
Rabin	P^{NP}, NP-h, coNP-h	P^{NP}, coNP-h	Exptime, Pspace-h	Pspace-c
Muller	Pspace-c	Pspace-c	Exptime, Pspace-h	Pspace-c
LTL	2Exptime-c[37]	2Exptime-c[37]	2Exptime-c[53]	2Exptime-c[53]

Table 5.1: Complexity of rational synthesis under perfect information for k processes.

On the positive side, our results show that for a *fixed* number of players, for objectives that admit a polynomial time solution in the two-player zero-sum case (reachability, safety, Büchi and coBüchi), cooperative and non-cooperative rational synthesis can be solved in PTime. On the negative side, for rich omega regular objectives defined by parity, Rabin, or Streett objective, the complexity increases. First, rational synthesis problem with parity objectives cannot be solved in polynomial time unless PTime equals NP. This is

because solving two-player zero-sum parity games is a particular case of rational synthesis and it is conjectured that this result does not hold for two-player zero sum parity games. Second, rational synthesis with Rabin or Streett objectives is PSPACE-complete for the non-cooperative setting while two-player zero-sum Parity and Streett games are solved in nondeterministic polynomial time.

When the number of players is not fixed, the complexity is usually substantially higher than for the two-player zero-sum case. For example, non-cooperative rational synthesis is PSPACE-hard for all objectives, so even for safety objectives.

Cooperative rational synthesis is a particular case of the more general problem of checking the existence of a constrained Nash equilibrium in a multiplayer game, where the strategy of Process 0 is required to be winning. That is, it asks for a strategy profile $\bar{\sigma}$ such that it is an Nash-equilibrium and $\langle 1, 0, ..., 0\rangle \leq pay(\bar{\sigma}) \leq \langle 1, 1, ..., 1\rangle$. The complexity of constrained Nash equilibria has been studied by Ummels in [87] for some classes of objectives, based on a characterisation of Nash equilibria by means of LTL formulas to be checked on the game arena. This directly gives us upper-bounds for cooperative synthesis and Büchi, coBüchi, parity and Streett objectives. Namely, the existence of a Nash equilibrium with a payoff between the given bounds is decidable in polynomial time for Büchi objectives, is in $UP \cap coUP$ for parity objectives and in NP for Streett and NP-complete for coBüchi objectives. So one immediately gets the upper bounds of Table 5.1 for these measures (and unfixed k). To establish the remaining upper bounds, we characterize the Nash equilibrium by means of LTL formulas. We extend the technique used in [87] for tail objectives to safety and reachability.

Solutions to the *non-cooperative* case are much more involved and are based on a fine tuned application of tree automata techniques. This is a central contribution of this chapter. In particular, our tree automata have exponential size but we show how to test their emptiness in PSPACE to obtain optimal algorithms for Streett, Rabin and Muller objectives and fixed number of players.

5.1 Preliminaries

Using SL[NG] to Solve Perfect Information Rational Synthesis As already mentioned, the rational synthesis problem was introduced in [37, 53] for perfect information setting. The main contribution of these papers is to propose and to motivate the definitions above. The only computational complexity results given in those papers are as follows: the cooperative and non-cooperative rational synthesis problems under perfect information for all agents are 2EXPTIME-complete for specifications expressed in linear temporal logic (LTL), thus matching exactly the complexity of classical zero-sum two-player LTL synthesis [74]. The upper bound is obtained by reductions to the satisfiability problem of formulas in Nested-Goal Strategy Logic (SL[NG]) [64].

Nested-Goal Strategy logic is first used to define a formula that characterizes strategy profiles $\bar{\sigma}$ that are 0-fixed γ-equilibria for every solution concept $\gamma \in \{$ Nash, Dominant Strategy, Subgame Perfect Nash $\}$. For example, in the case of Nash equilibria, the formula

$$\psi_{0Nash}(\bar{\sigma}) \coloneqq \bigwedge_{i=1}^{k} [\![\sigma_i']\!] \big(\flat(\bar{\sigma}_{-i}, \sigma_i')\varphi_i \rightarrow \flat(\bar{\sigma})\varphi_i \big)$$

says that for any agent in the environment, for any unilateral deviation σ_i', if the objective φ_i of Agent i is satisfied, then it is also satisfied when Agent i does not deviate.

Then, to solve the rational synthesis problem, Kupferman et. al reduce it to the model-checking problem of a SL[NG] formula with one alternation of strategy quantifiers. For the cooperative setting, one has to verify the formula ψ_{cRS} defined below, while in the non-cooperative setting, the synthesis problem is reduced to model-checking of the formula ψ_{noncRS}.

$$\psi_{cRS} \coloneqq \langle\!\langle \sigma_0 \rangle\!\rangle \langle\!\langle \sigma_1 \rangle\!\rangle ... \langle\!\langle \sigma_k \rangle\!\rangle \big(\psi_{0Nash}(\bar{\sigma}) \wedge \varphi_0 \big)$$
$$\psi_{noncRS} \coloneqq \langle\!\langle \sigma_0 \rangle\!\rangle [\![\sigma_1]\!] ... [\![\sigma_k]\!] \big(\psi_{0Nash}(\bar{\sigma}) \rightarrow \varphi_0 \big)$$

In the cooperative case, the formula intuitively says that there is a strategy profile $\bar{\sigma} = \langle \sigma_0, \sigma_1, ..., \sigma_k \rangle$ so that $\bar{\sigma}$ is a 0-fixed γ-equilibrium for which the outcome satisfies the objective of the process to be synthesized (Agent 0). For the non-cooperative case, the formula asks for the existence of a strategy σ_0 for Process 0 such that for all tuples $(\sigma_1, ..., \sigma_k)$ of strategies for the environment components, if the resulting strategy profile is a 0-fixed γ-equilibrium, then also the objective φ_0 of Process 0 is satisfied.

Using ATL^*_{sc} to Solve Perfect Information Rational Synthesis As already seen in Section 3.2.1, we can also use the logic ATL^*_{sc} to express the existence of Nash equilibria when processes have LTL objectives. Therefore, we can also write ATL^*_{sc} formulas that hold if and only if there is a solution for the cooperative and non-cooperative rational synthesis problem. That is, we reduce the rational synthesis problem to model-check the formulas ψ'_{cRS} for cooperative setting and ψ'_{noncRS} for the non-cooperative setting where:

$$\psi'_{cRS} \coloneqq \langle \cdot \Omega \cdot \rangle \Big[\varphi_0 \wedge \big(\bigwedge_{i=1}^{k} \neg\varphi_i \rightarrow \neg\langle \cdot i \cdot \rangle \varphi_i \big) \Big]$$
$$\psi'_{noncRS} \coloneqq \langle \cdot 0 \cdot \rangle [\cdot \{1, ..., k\} \cdot] \Big[\big(\bigwedge_{i=1}^{k} \neg\varphi_i \rightarrow \neg\langle \cdot i \cdot \rangle \varphi_i \big) \rightarrow \varphi_0 \Big]$$

Since the model-checking problem for ATL^*_{sc} is $(d+1)$ExpTime-complete (where d is the alternation depth of the quantifiers), the above reduction provides the 2ExpTime upper bound for the complexity of rational synthesis problem. As in the case of $SL[NG]$, this reduction does not provide more insight on the computational complexity of the rational synthesis problem.

5.2 LTL Characterization of Nash Equilibria

In this section we provide effective characterizations of the existence of 0-fixed Nash equilibria in interaction models where agents have safety, reachability, or tail objectives, through the existence of an execution satisfying certain properties. These properties will be expressed by LTL formulas.

Linear Temporal Logic on Interaction Models In order to define the LTL formulas that characterize Nash equilibria (NE for short), let us first make clear the way we make use of linear temporal logic. The alphabet over which LTL formulas are built represents the set of states in the interaction model. Given a state $v \in V$, we view v as an atomic proposition, true in the state v, and false otherwise. Given $S \subseteq V$, we may freely use S in an LTL formula, where it stands for the formula $\bigvee_{v \in S} v$. Therefore, we may write, for instance, $\Box\neg S$, to denote the set of infinite paths in $(V \backslash S)^\omega$.

We denote by LTL($\mathcal{M}$) the set of LTL formulas over the set of atomic propositions V. A set $\Theta \subseteq V^\omega$ of infinite executions is *definable* in LTL($\mathcal{M}$) if there exists an LTL($\mathcal{M}$) formula ϕ such that for all $\rho \in V^\omega$, $\rho \vDash \phi$ if and only if $\rho \in \Theta$. In [86] similar formulas were given for similar tale objectives (see Corollary[26]).

LTL characterization of (0-fixed) Nash equilibria For all winning objectives considered in this chapter, we characterize the existence of a (0-fixed) Nash equilibria in an interaction model by the existence of a path satisfying some LTL formula, that depends on the winning objectives. For tail objectives, we give a generic way of constructing such an LTL formula.

An objective expressed as a set $\Theta \subseteq V^\omega$ is *tail* if for all $\rho_1 \in V^*$ and $\rho_2 \in V^\omega$, $\rho_1\rho_2 \in \Theta$ iff $\rho_2 \in \Theta$. In other words, a path is winning if and only if one of its (infinite) suffix is winning. Büchi, coBüchi, parity, Streett, Rabin and Muller objectives are all tail. We use the notations defined in Section 2.2.1 to define the objectives of the agents. For example, we write $\Theta_i = \mathsf{Safe}(S_i)$ for the set of executions that satisfy the Safety objective of some agent $i \in \Omega$, where the set of safe states for him is $S_i \subseteq V$.

Let $\mathcal{M} = \langle \mathcal{P}, \Omega, (\Sigma_i)_{i\in\Omega}, V, V_0, E, \tau \rangle$ be an interaction model and Θ_i the objective of Agent i, for all $i \in \Omega$. Let $(W_i^{\mathcal{M}})_{0\leq i\leq k}$ be the winning sets of each agent (i.e., the set of states from which he has a winning strategy when the other agents play against him) in the interaction model $\mathcal{M}$ and $b \in \{0, 1\}$. Whenever the interaction model is understood from the context, we simply write W_i for the winning region of Agent i.

We define an LTL[$\mathcal{M}$]-formula $\phi_{b\text{Nash}}^{\mathcal{M}}$ that characterize Nash equilibria ($b = 1$) and 0-fixed Nash equilibria ($b = 0$) for safety, reachability and tail objectives. We give the characterization of Nash equilibria since it might be of independent interest for the reader.

$$\phi^{\mathcal{M}}_{b\text{Nash}} = \begin{cases} \bigwedge_{i=1-b}^{k}((\neg W_i^{\mathcal{M}}\ \mathcal{U}\ \neg S_i) \vee \Box S_i) & \text{if } \Theta_i \text{ are safety objectives of the form} \\ & \Theta_i = \mathsf{Safe}(S_i) \text{ for } S_i \subseteq V \\ \bigwedge_{i=1-b}^{k} \neg\varphi_i \rightarrow \Box\neg W_i^{\mathcal{M}} & \text{if } \Theta_i \text{ are either all reachability or all tail} \\ & \text{objectives definable by an LTL}[\mathcal{M}] \text{ formula } \varphi_i \end{cases}$$

Assume that $b = 1$, and consider the formula for safety objectives. Intuitively, it says that for all agents $i \in \{0, \ldots, k\}$, either Agent i always stays safe, or if eventually he visits an unsafe position, then he should never visit a winning position until he meets an unsafe position for the first time. This is because otherwise he could apply a winning strategy and satisfy his own objective, and therefore has some incentive to deviate.

The formula $\phi_{b\text{Nash}}$ characterises (0-fixed) Nash equilibria in the following sense:

Proposition 5.2.1 (Characterization of 0-fixed NE and NE ([87] for tail objectives))**.** *Let $\mathcal{M}$ be an interaction model in which agents have either all safety, all reachability, or all tail objectives, definable in LTL$[\mathcal{M}]$. Then, the following hold:*

1. *For all $\rho \in \mathsf{exec}(\mathcal{M})$, if $\rho \models \phi^{\mathcal{M}}_{0Nash}$ (resp. $\rho \models \phi^{\mathcal{M}}_{1Nash}$), then there exists a 0-fixed NE (resp. NE) $\bar{\sigma}$ in $\mathcal{M}$ such that $\text{exec}(\mathcal{M}, \bar{\sigma}) = \pi$,*
2. *For all 0-fixed NE (resp. NE) $\bar{\sigma}$ in $\mathcal{M}$, $\text{exec}(\mathcal{M}, \bar{\sigma}) \models \phi^{\mathcal{M}}_{0Nash}$ (resp. $\text{exec}(\mathcal{M}, \bar{\sigma})) \models \phi^{\mathcal{M}}_{1Nash}$).*

Before proceeding to the proof of Proposition 5.2.1, we illustrate the characterization of Nash equilibria on an example where agents have reachability objectives.

(a) Path not consistent with a 0-fixed NE

(b) Path consistent with 0-fixed NE

Figure 5.1: Example for LTL characterization of NE

Example 5.2.1. *Consider the interaction model in Fig. 5.1, containing three agents that control circle (Agent 0), square (Agent 1) and diamond (Agent 2) states respectively. The objective of each agent is to reach its target. Player 0 and Player 1 aim for the red-blue target ($R_0 = R_1 = \{\mathsf{TARGET}_{0,1}\}$) and Player 2 aims at the green target ($R_2 = \{\mathsf{TARGET}_2\}$).*

The winning sets for the three players (from which they have a winning strategy to satisfy their objective for any strategy of the other agents) are $W_0 = \{\mathsf{TARGET}_{0,1}\}$, $W_1 = \{v_1, \mathsf{TARGET}_{0,1}\}$ *and* $W_2 = \{v_2, \mathsf{TARGET}_2\}$.

First, consider the execution ρ *that starts in* v_0*, passes through* v_1 *and reaches the red-blue target. It satisfies the LTL characterization* $\bigwedge_{i=0}^{k} \Box\neg R_i \rightarrow \Box\neg W_i$ *of Nash equilibria. For this path, we can build a Nash equilibrium* $\bar{\sigma}$ *that is represented in Fig. 5.1 b) with bold arrows such that* $\mathrm{exec}(\mathcal{M}, \bar{\sigma}) = \rho$.

On the other hand, consider the path ρ' *that starts in the initial state and passes through* v_1 *and then* v_2 *before reaching the green target. It is highlighted in Fig.5.1 a). It does not satisfy* $\bigwedge_{i=0}^{k} \Box\neg R_i \rightarrow \Box\neg W_i$*. Also, it is not the outcome of any 0-fixed Nash equilibria. Indeed, Agent* 1 *does not follow the path* ρ' *from state* v_1*. He would better deviate and go to the red-blue target in order to reach his objective.*

Note that from a state $v \notin W_i$, Agent i does not have a winning strategy to ensure Θ_i against the other players that want to make him lose. Since we are in the perfect information setting, we can reduce to a two-player zero-sum game where Agent i plays against an opponent (representing all the agents in $\Omega \setminus \{i\}$) that wants to falsify the objective Θ_i. Since the objectives that we consider are particular Borel objectives, by the determinacy Theorem 3.1.1, there is a strategy for the opponent from the state $v \notin W_i$ such that it makes Agent i lose on any play. Therefore, we can define the strategy profile $(\mathsf{ret}_j^{v,i})_{j \neq i}$ for the agents in $\Omega \setminus \{i\}$ that makes Agent i lose. We say that this strategy profile played by the agents in $\Omega \setminus \{i\}$ is the *retaliating* strategy profile.

We now give the proof of Proposition 5.2.1 that characterizes the existence of (0-fixed) Nash Equilibria.

Proof of Proposition 5.2.1. We only write the proof for 0-fixed Nash equilibria, since it is used further in the thesis. The proof for Nash equilibria is the same, differing only in the agents that may deviate.

Statement (1)*, safety objectives* The strategy profile $\bar{\sigma}$ is intuitively defined as follows: as long as the current history is a prefix of ρ, then the agents play according to ρ. If at some point, some agent, say Agent i, decides to deviate from ρ, ending up in a state v, then if $v \notin W_i$, all agents except Agent i punish him by playing a strategy that will make him lose, otherwise, they play any strategy.

Let us give some arguments to justify that it is a 0-fixed equilibrium. The outcome of $\bar{\sigma}$ is ρ, and if an agent wins along ρ, then he has no incentive to deviate. If some agent, say Agent i, loses along ρ, then suppose that he decides eventually to deviate from ρ: either he has already lost before deviating and therefore his deviation is useless, or he deviates to a state v before visiting an unsafe state for the first time, but in that case, since $\rho \vDash \neg W_i \,\mathcal{U}\, \neg S_i$, we have $v \notin W_i$ (otherwise the previous state would be winning), and all the other agents retaliate, making his deviation useless, again.

Let us define $\bar{\sigma}$ formally. For a state $v \notin W_i$, we let $(\mathsf{ret}_j^{v,i})_{j \neq i}$ a retaliating profile for the players $j \neq i$ that make Player i lose from the state v. We also pick an arbitrary profile of strategies $(\beta_0, \ldots, \beta_k)$. Then, we define σ_j as follows, for $x \in V^*V_j$:

$$\bar{\sigma}_j(x) = \begin{cases} \rho[l+1] & \text{if } x = v_0 v_1 ... v_l \text{ is a prefix of } \rho \\ \mathsf{ret}_j^{v,i}(vx_2) & \text{if condition (1) is satisfied} \\ \beta_j(x) & \text{otherwise} \end{cases}$$

where condition (1) requires that x can be decomposed into $x = x_1 v x_2$ such that $x_1 \in V^*V_i$, $x_2 \in V^*$, x_1 is a prefix of ρ, $v \notin W_i$, $x_1 v$ is not a prefix of ρ (meaning that Agent i has deviated to a losing state).

Clearly, we have $\text{exec}(\mathcal{M}, \bar{\sigma}) = \rho$. We claim that $\bar{\sigma}$ is a 0-fixed Nash equilibrium. Towards a contradiction, we suppose that some agent i ($1 \leq i \leq k$) loses, and can win by playing another strategy σ_i' (when the other agents stick to their strategies $\bar{\sigma}_{-i}$). Then necessarily, $\text{exec}(\mathcal{M}, \bar{\sigma}_{-i}, \sigma_i')$ deviates from ρ after some prefix x_1 of ρ such that $x_1 \in V^\omega V_i$. Let $v \in V$ and $y \in V^\omega$ such that $\text{exec}(\mathcal{M}, \bar{\sigma}_{-i}, \sigma_i') = x_1 v y$. Then, if $v \notin W_i$, the other agents retaliate making $\text{exec}(\mathcal{M}, \bar{\sigma}_{-i}, \sigma_i')$ losing for Player i, which is a contradiction. Therefore $v \in W_i$. Let v' be the last state of x_1. Then $v' \in W_i$ since $v' \in V_i$ and it has a successor in W_i (v). Now, we consider two cases: (i) suppose that some state of x_1 is unsafe for Agent i, then it contradicts the fact that $\text{exec}(\mathcal{M}, \bar{\sigma}_{-i}, \sigma_i')$ is winning for Agent i; (ii) if $x_1 \in (S_i)^*$, then since ρ is losing for Agent i, there is an unsafe state for Agent i that occurs after the prefix x_1, contradicting the fact that $\rho \vDash \neg W_i \, \mathcal{U} \, \neg S_i$, since the last state of x_1 is in W_i. In all cases, we have found a contradiction, showing that such a strategy σ_i' cannot exist.

Statement (2)*, safety objectives* Assume that some agent, say i for $1 \leq i \leq k$ does not win, i.e. $\text{exec}(\mathcal{M}, \bar{\sigma}) \vDash \Diamond \neg S_i$. Towards a contradiction, assume that $\text{exec}(\mathcal{M}, \bar{\sigma}) \nvDash \neg W_i \mathcal{U} \neg S_i$. Consider the first occurrence j of state satisfying $\neg S_i$ in $\text{exec}(\mathcal{M}, \bar{\sigma})$, i.e. $j = argmin\{j \mid \text{exec}(\mathcal{M}, \bar{\sigma})[j] \notin S_i\}$ (it exists since $\text{exec}(\mathcal{M}, \bar{\sigma}) \vDash \Diamond \neg S_i$). Clearly, there is a position $0 \leq t < j$ such that $\text{exec}(\mathcal{M}, \sigma)[t] \in W_i$ (otherwise $\text{exec}(\mathcal{M}, \bar{\sigma})$ would satisfy $\neg W_i \mathcal{U} \neg S_i$). At that position, Agent i could have deviated and apply a winning strategy, thus getting a strictly better payoff, contradicting the fact that $\bar{\sigma}$ is a 0-fixed Nash equilibrium.

Statements (1) *and* (2)*, reachability and tail objectives* The proofs of these two statements are very similar to that of safety objectives. The only difference here is that the objectives are either all reachability or all tail, and therefore one has to make sure that on ρ, the agents that lose never visit their winning region, because if it is so, they would have an incentive to deviate: indeed, the satisfaction of their winning objective would be independent from the prefix up to a visit to their winning region. The formal definition of the strategy profile is as in the case of Safety objectives.

For statement (1), the profile of strategies $\bar{\sigma}$ is: follow the execution ρ as long as

the play stays in ρ, and the first time the play deviates (say Agent i deviates from ρ), then if ρ is losing for Agent i, then apply from that point on a retaliating strategy (as a coalition of all the players $j \neq i$), otherwise apply any strategy. If ρ is not winning for Agent i, the retaliating strategy exists by definition of $\phi_{0\text{Nash}}$, since the first position after the deviation would not be in W_i.

Conversely, any 0-fixed NE $\bar{\sigma}$ satisfies $\phi_{0\text{Nash}}$. Indeed, if it is not the case, then there is some player that satisfies $\neg\varphi_i$ and $\Diamond W_i$. When reaching its winning region, this player would better apply a winning strategy and strictly increase his payoff. ■

5.3 Cooperative Rational Synthesis (CRSP)

We recall that the Cooperative Rational Synthesis Problem (CRSP) asks for a strategy profile $\bar{\sigma}$ such that it is a Nash-equilibrium and $\langle 1, 0, ..., 0\rangle \leq pay(\bar{\sigma}) \leq \langle 1, 1, ..., 1\rangle$. In [87], Ummels proved that the existence of a Nash equilibrium with a payoff between the given bounds is decidable in polynomial time for Büchi objectives, is in $UP \cap coUP$ for parity objectives and in NP for Streett and NP-complete for coBüchi objectives.

In the following, we give a general solution for the cooperative rational synthesis problem that extends the solution given in [87] by Ummels. Then, we describe the algorithms and prove the tight complexity for the particular objectives.

5.3.1 General Solution for CRSP

Proposition 5.2.1 allows us to give a generic procedure to solve the cooperative rational synthesis problem, which is based on the following direct consequence:

Proposition 5.3.1. *Let $\mathcal{M}$ be an interaction model with $k+1$ processes having either all safety, all reachability, or all tail objectives, definable in LTL$[\mathcal{M}]$ by formulas $(\varphi_i)_{0\leq i\leq k}$. There is a solution to the cooperative synthesis problem iff there is an execution $\rho \in \mathsf{exec}(\mathcal{M})$ such that $\rho \vDash \phi_{0Nash} \wedge \varphi_0$.*

Proof. Indeed, if there is a solution $\bar{\sigma}$ for the cooperative rational synthesis problem, $\rho = \text{exec}(\mathcal{M}, \bar{\sigma})$ satisfies φ_0. Moreover, by Proposition 5.2.1, Statement 2, ρ also satisfies $\phi_{0\text{Nash}}$. Therefore, $\rho \vDash \phi_{0\text{Nash}} \wedge \varphi_0$.

On the other direction, if there is an execution ρ such that $\rho \vDash \phi_{0\text{Nash}} \wedge \varphi_0$, by Proposition 5.2.1, Statement 1, one can build a strategy profile $\bar{\sigma}$ so that it is a 0-fixed Nash equilibrium and $\text{exec}(\mathcal{M}, \bar{\sigma}) = \rho$. Since φ_0 holds along ρ, $\bar{\sigma}$ is a solution for the cooperative rational synthesis problem. ■

Based on the latter proposition, it is not difficult to design a procedure to decide CRSP: first, compute the winning sets W_i for $i = 1, \ldots, k$, and then check whether the interaction model $\mathcal{M}$ contains an execution which satisfies the formula $\phi^{\mathcal{M}}_{0\text{Nash}} \wedge \varphi_0$, where φ_0 is the

objective of Process 0, expressed in LTL[$\mathcal{M}$]. To establish precise upper bounds, one needs to consider the complexity of computing the winning sets, and then the complexity of model-checking these particular LTL formulas. One objective of this chapter is to give tight complexity bounds for the model-checking of this formula and, thus, to the cooperative rational synthesis problem.

5.3.2 Upper Bounds for CRSP

In this section we give the upper bounds complexities for solving the Cooperative Rational Synthesis Problem.

To obtain the tight complexity upper bounds for CRSP, we prove the following short witness property: if an execution of the interaction model satisfies the formula $\phi^{\mathcal{M}}_{0\text{Nash}} \wedge \varphi_0$, then there is a lasso path uw^ω satisfying it, such that u and w have polynomial length. Then, the nondeterministic algorithm solving the cooperative synthesis problem simply guesses such a path and verifies, in polynomial time, that it satisfies the desired property.

Remove Cycles from Paths To obtain the short witness, we remove cycles from infinite executions so that the satisfaction of the LTL characterization is preserved on the short path. By *removing all cycles* from a sequence $x \in V^*$, we refer to the following operation. We decompose $x = x_1 v x_2 v x_3$ with $x_1, x_2, x_3 \in V^*$ and $v \in V$ such that there are no repetitions in $x_1 v x_2$. We remove the sequence $v x_2$ from x and obtain the sequence $x' = x_1 v x_3$. We repeat the procedure for the newly obtained sequence until is obtained a sequence that has no repetitions of states.

Safety Objectives

In the case of safety condition, the characterization of a 0-fixed Nash equilibrium intuitively expresses the fact that either Process i always stays in its safe set of states, or is the case that he loses by eventually reaching a unsafe state, but he could not play better, i.e., the execution did not pass (before the unsafe state) through a state from which he has a winning strategy.

The formula to be satisfied in the case of Safety objectives is

$$\phi = \Box S_0 \wedge \bigwedge_{i=1}^{k} ((\neg W_i \mathcal{U} \neg S_i) \vee \Box S_i)$$

The following theorem establishes the NP membership of the cooperative rational synthesis problem with safety objectives and perfect information for the agents.

Theorem 5.3.1. *The perfect information cooperative rational synthesis problem for safety objectives is in NP.*

Proof. To solve the NP membership of this problem, it suffices to check the existence of an execution in the interaction model that satisfies the LTL formula $\phi = \Box S_0 \wedge \bigwedge_{i=1}^{k}((\neg W_i \mathcal{U} \neg S_i) \vee \Box S_i)$.

First, by Theorem 3.1.2, two-player safety games can be solved in polynomial time, and therefore the winning sets W_i can be computed in polynomial time.

Then, it is well-known from LTL model-checking that given a lasso path $\rho = uw^\omega$, it can be checked in polynomial time (in $|u|$ and $|w|$ and the size of the interaction model) whether $\rho \vDash \phi$. Indeed, using W_i and S_i as abbreviations for $\bigvee_{v \in W_i} v$ and $\bigvee_{v \in S_i} v$, one can easily construct a 5-state automaton equivalent to each of the subformula $((\neg W_i \mathcal{U} \neg S_i) \vee \Box S_i)$, for which checking the acceptance of uw^ω can be done in polynomial time.

It remains to show that we can bound the length of u and w polynomially.

Let $\rho \in V^\omega$ be an execution satisfying ϕ. For each $i \in \{1, \dots, k\}$, we consider the first occurrence of an unsafe state of Process i in ρ, and decompose ρ according to these positions as follows. Formally, ρ is decomposed as $\rho = x_1 v_{P_1} x_2 v_{P_2} ... x_l v_{P_l} x_{l+1}$ for all $j \in \{1, \dots, l\}$, $x_j \in V^*$, $P_j \subseteq \{1, \dots, k\}$ and $v_{P_j} \in V$ the first occurrence of a state which is unsafe for all the processes in P_j (P_j is maximal for that property).

First, we remove the cycles in all x_j, $j \in \{1, \dots, l\}$, leading to a new execution of the form $x'_1 v_{P_1} x'_2 v_{P_2} ... x'_l v_{P_l} x_{l+1}$ where the sequences x'_j are loop-free. This preserves the satisfaction of ϕ, i.e. $\rho' \vDash \phi$. Indeed, by doing so, the subformula $\Box S_0$ is still satisfied, and for all $i \in \{1, \dots, k\}$, if $\Box S_i$ was satisfied by ρ, then it is still satisfied in ρ'. If $\neg W_i \mathcal{U} \neg S_i$ was satisfied in ρ, then by the choice of our decomposition, removing the cycles still preserve the existence of an unsafe state for Process i in ρ', and all the states before its first occurrence in ρ' satisfies $\neg W_i$.

Secondly, we modify x_{l+1} into a short lasso path $x'_{l+1}(x'')^\omega$, where x'' is a simple loop, and x'_{l+1} is loop-free. This can be done by taking x'_{l+1} to be shortest prefix of x_{l+1} to a state v that repeats in the future, and to take x'' has any loop from v to v, shortened into a simple loop by removing all inner-cycles. All these operations preserve the properties of satisfying $\Box S_i$ for all $i \in \{0, \dots, k\}$.

Then, we set $u = x'_1 v_{P_1} x'_2 v_{P_2} ... x'_l v_{P_l} x'_{l+1}$ and $w = x''$. It holds that $uw^\omega \vDash \phi$ and $|uw| \leq n(k+2)$ which concludes the proof. ∎

Reachability Objectives

The algorithm to solve the Cooperative Rational Synthesis for reachability objectives is similar to the one in the case of safety objectives. However, observe that unlike the case of two-player zero-sum games, there is no duality between reachability and safety, and no natural reduction from reachability to safety.

Based on Proposition 5.3.1, in order to solve the rational cooperative synthesis problem for reachability objectives, it suffices to have a procedure that test the existence of the

path in the game that satisfies the formula

$$\phi = \Diamond R_0 \wedge \Big(\bigwedge_{i=1}^{k} \Box\neg R_i \rightarrow \Box\neg W_i \Big)$$

We prove that, as in the case of safety objectives, if such an execution exists, there is a short witness and the rational synthesis problem can be solved in NP.

Theorem 5.3.2. *The perfect information cooperative rational synthesis problem for reachability objectives is in NP.*

Proof. We start by showing that each execution ρ so that $\rho \vDash \phi$, with $\phi = \Diamond R_0 \wedge \big(\bigwedge_{i=1}^{k} \Box\neg R_i \rightarrow \Box\neg W_i \big)$ can be shortened into a lasso path ρ^* such that $\rho^* \vDash \phi$ and ρ^* can be decomposed into a prefix of length at most nk followed by a simple loop. In fact, let L be the maximal set of processes such that $\rho \vDash \bigwedge_{i \in L} \Diamond R_i$. Then, it is sufficient to enucleate the first occurrences of states in R_i for all $i \in L$ along ρ, and eliminate all the cycles between these occurrences. This leads to a new path ρ' where each process $i \in L$ accomplishes its reachability objective in at most $n|L|$ steps and such that $\rho' \vDash \bigwedge_{i \notin L} \Box\neg W_i$. Let j be the smallest position in ρ' such that each process $i \in L$ has accomplished its reachability objective, i.e. $j = min\{\ell \geq 0 \mid \rho'[:\ell] \vDash \bigwedge_{i \in L} \Diamond R_i\}$. Then, ρ^* is obtained from ρ' by considering the first cycle (reduced to a simple cycle) appearing from the vertex $\rho'[j]$ on.

Therefore, the NP algorithm works as follows: guess a lasso-path of length at most $n(k+1)$, check whether it satisfies ϕ, and use it to build a winning strategy that uses as much memory as the length of the path. This is correct by the small lasso property proved above, and Proposition 5.3.1. ∎

Büchi, co-Büchi, Streett and Parity Objectives

In the following, we show how to solve the cooperative rational synthesis problem in the case of ω-regular objectives, in particular for Büchi, coBüchi, Streett and Parity objectives. For these objectives, we rely on the results shown by Ummels in [87], following the next remark.

Remark 5.3.1. *In [87] the complexity of finding a Nash equilibrium $\bar{\sigma}$ in a game with $k+1$ processes having all Büchi, coBüchi, Streett or Parity objectives is studied, so that $x \leq pay(\bar{\sigma}) \leq y$, for two given threshold $x, y \in \{0,1\}^{k+1}$. The cooperative synthesis problem reduces to this setting, by taking $x = (1,0,\ldots,0)$ and $y = (1,1,\ldots,1)$.*

The following theoren is a direct consequence of [87].

Theorem 5.3.3. *The perfect information cooperative rational synthesis problem in multi-component interaction models is:*

- *in PTime for Büchi objectives,*
- *NP for co-Büchi, Streett and parity objectives.*

Rabin Conditions

For Rabin conditions the P^{NP} complexity can be obtained by applying the same general solution as for the other classes of objectives. This result is already mentioned in [87], but we provide here the complete proof.

Let us consider an interaction model $\mathcal{M}$ where the $k+1$ processes have the Rabin objectives $(\mathsf{Rabin}(\psi_i))_{0\leq i\leq k}$ with $\psi_i = \{(L_1, R_1), ..., (L_{m_i}, R_{m_i})\}$, for all $0 \leq i \leq k$. Based on Proposition 5.3.1 and the fact that the Rabin condition ψ_i can be equivalently expressed by the $LTL[\mathcal{M}]$ formula $\varphi_i = \bigvee_{j=1}^{m_i}(\Box \Diamond L_{ij} \wedge \Diamond \Box \neg R_{ij})$, solving the cooperative rational synthesis problem with Rabin objectives is equivalent to find an execution satisfying the formula

$$\phi^{\mathcal{G}}_{0\text{Nash}} \wedge \varphi_0 = \bigvee_{j=1}^{m_0}(\Box \Diamond L_{0j} \wedge \Diamond \Box \neg R_{0j}) \wedge \bigwedge_{i=1}^{k}\Big(\Box \neg W_i \vee \bigvee_{j=1}^{m_i}(\Box \Diamond L_{ij} \wedge \Diamond \Box \neg R_{ij})\Big)$$

Theorem 5.3.4. *The perfect information cooperative rational synthesis problem for Rabin objectives is in P^{NP}.*

Proof. We first show that given the winning regions W_i for $1 \leq i \leq k$, each path $\rho = uw^\omega$ so that $\rho \vDash \phi^{\mathcal{G}}_{0\text{Nash}} \wedge \varphi_0$ can be shortened into a lasso path $\rho' = u'(w')^\omega$ such that $\rho' \vDash \phi^{\mathcal{G}}_{0\text{Nash}} \wedge \varphi_0$ and $|u'w'| \leq |V|^2 + |V| \cdot k$.

First, we mark as red node the first occurrence along uw of a state in W_i for every process i not satisfying its objective along ρ. We also mark as red node the first occurrence along w of states in L_{ij} and R_{ij}(if any) for every process and every pair j in the Rabin condition ψ_i. Note that along u at most k red nodes are marked and along w we have marked at most $|V|$ states since we marked the first occurrence of a state.

Then, by removing all the loops in u and w that do not contain red nodes we obtain u' and w' such that $|u'| \leq |V| \cdot k$ and $|w'| \leq |V|^2$. Note that the property $\rho' \vDash \phi^{\mathcal{G}}_{0\text{Nash}} \wedge \varphi_0$ also holds since we didn't remove key nodes(red nodes on w) from ρ.

Then the P^{NP} algorithm will run as follows. First, it guesses a path $\rho = uw^\omega$ of polynomial length (as we saw $|uw| \leq |V|^2 + |V| \cdot k$ suffices) and mark the states in uw by W_i or $\neg W_i$ by checking in NP time if $v \in W_i$ for each state. This can be done in P^{NP} time. Then, in polynomial time check if $\rho \vDash \phi^{\mathcal{G}}_{0\text{Nash}} \wedge \varphi_0$ by checking if for each process is a pair (L_{ij}, R_{ij}) such that L_{ij} appears on w but not R_{ij}. If not, W_i should not be a label of a state in uw. If it is, the algorithm rejects. ∎

Muller Objectives

Let $\mathcal{M}$ be an interaction model on which the agents have the Muller objectives $(\mathsf{Muller}(\mu_i))_{i\in\Omega}$ where the formula μ_i is a boolean formula over the states of the interaction model, i.e., $\mu_i = l_1\ Op_1\ l_2\ Op_2\ ...\ l_{m_i}$ where $Op_j \in \{\wedge, \vee\}$ for all $1 \leq j < m_i$ and each literal l_j is either a state $v_j \in V$ or its negation $\neg v_j$.

Let us define the LTL formula φ_i from μ_i by replacing each v_j by the subformula $\Box \Diamond v_j$. Then, we claim that for any execution ρ, we have that ρ satisfies $\mathsf{Muller}(\mu_i)$ if and only if $\rho \vDash \varphi_i$. Intuitively, it holds since whenever $v \in \inf(\rho)$, it satisfies both the Muller condition $\mathsf{Muller}(v)$ and the LTL formula $\Box \Diamond v$. And if $v \notin \inf(\rho)$, both Muller condition $\mathsf{Muller}(\neg v)$ and LTL formula $\Diamond \Box \neg v \equiv \neg \Box \Diamond v$ are satisfied by ρ.

Then, using the characterization of 0-fixed Nash equilibria for ω-regular objectives, the problem is to decide the existence of an execution satisfying the LTL formula $\phi = \varphi_0 \vee \bigwedge_{i=1}^{k}(\varphi_i \vee \Box \neg W_i)$ where φ_i, $0 \leq i \leq k$ is defined as above. The formula ϕ is an LTL formula in the fragment $\mathcal{B}(L_{\Box\Diamond}(\mathcal{P}) \cup L_{\Diamond,\wedge}(\mathcal{P}))$ where $\mathcal{P}$ corresponds to the atomic propositions associated to the states of $\mathcal{M}$. For this fragment of LTL, it is shown in [6] that a two-player zero-sum game with the protagonist having an objective in this fragment of LTL can be solved in PSPACE and therefore also the cooperative rational synthesis for Muller objectives is.

5.3.3 Lower Bounds for CRSP

In this section we prove the lower bounds for the complexity of Cooperative Rational Synthesis Problem for the several objectives we consider.

Safety and Reachability Objectives

Theorem 5.3.5. *The perfect information cooperative rational synthesis problem for safety objectives is NP-hard.*

Proof. The proof for the NP-hardness of the CRSP problem for safety objectives is done by reduction from $3SAT$. Given a Boolean formula $\varphi = C_1 \wedge ... \wedge C_k$ in conjunctive normal form where each clause has at most three literals, we construct an interaction model $\mathcal{M}_\varphi$ with $k+1$ processes having safety objectives as follows (its arena is depicted in Fig. 5.2): Let $X = \{x_1, ..., x_m\}$ be the set of variables that appear in φ. The interaction model $\mathcal{M}_\varphi$ has three states $x, 1_x$ and 0_x controlled by the process to be synthesized (Process 0) for all variables $x \in X$. The two latter states correspond to the two possible truth values of x. For all $i \in \{1, \ldots, m-1\}$, there are edges from each x_i to both 1_{x_i} and 0_{x_i} and from 1_{x_i} and 0_{x_i} to x_{i+1}. There is a state C_j controlled by Process j for each clause in φ and two states U_s and U_e (which will be unsafe for Process 0 and processes in the environment respectively). We add edges from 1_{x_m} and 0_{x_m} to C_1, from C_j to C_{j+1} for all $1 \leq j < k$, and from every C_j to U_s and from C_k to U_e. In Fig. 5.2 the protagonist (Process 0) plays the circle states and each environment Process i plays the diamond state C_i.

We let V be the set of states (vertexes) of $\mathcal{M}_\varphi$, x_1 being the initial one. All states but U_s are safe for Process 0, i.e. $S_0 = V \setminus \{U_s\}$. The unsafe states for Process $j \neq 0$ are U_e, as well as the state 0_x if $\neg x$ appears in C_j, and the state 1_x if x appears in C_j, i.e. $S_j = V \setminus (\{0_x \mid \neg x \in C_j\} \cup \{1_x \mid x \in C_j\} \cup \{U_e\})$.

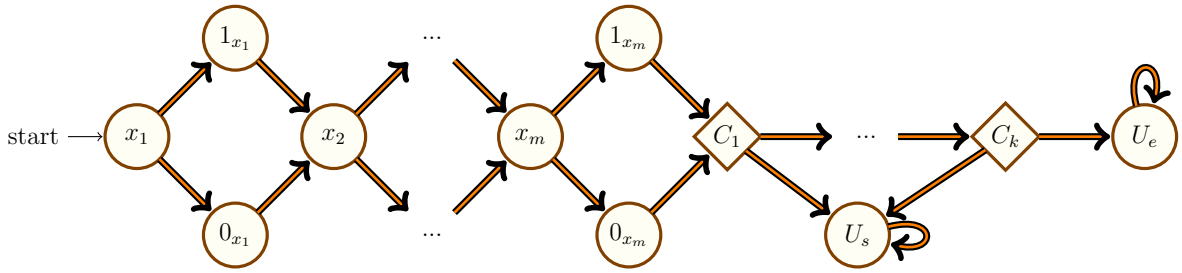


Figure 5.2: NP-h Safety CRSP: Reduction from 3-SAT

Let us now prove the correctness of this reduction, i.e. φ is satisfiable if and only if there is a 0-fixed Nash equilibrium winning for Process 0 in $\mathcal{M}_\varphi$. Suppose first that φ is satisfiable by a valuation $\nu : X \to \{0,1\}$ of its variables. The strategy σ_0 of Process 0 is then to choose the truth values of the literals according to this valuation: choose 1_x if $\nu(x) = 1$, and 0_x otherwise. By doing so, all processes of the environment visit at least one unsafe state before reaching C_1. Indeed, let $j \in \{1, \dots, k\}$. Since C_j is satisfied by ν, there is a literal ℓ in C_j such that $\nu(\ell) = 1$. If $\ell = x$ for some $x \in X$, then 1_x is unsafe for Process j, but that is exactly the choice of Process 0 to go to 1_x (and similarly when $\ell = \neg x$). After reaching C_1, we set the choices of the processes $j \neq 0$ such to follow the path $C_1, ..., C_k$ and then reach U_e. This profile is winning for Process 0 since the state U_s is not reached, and losing for all the other processes. They have no incentive to deviate since they have already lost before making any choice. Therefore, it is a 0-fixed Nash equilibrium.

Conversely, if there is a solution for the cooperative synthesis problem, the only way to obtain a Nash equilibrium $\bar{\sigma}$ winning for the system is to make all processes j, $1 \leq j \leq k$ lose before reaching C_1. Indeed, if $\bar{\sigma}$ is winning for the system, then $\text{exec}(\mathcal{M}, \sigma)$ eventually reaches U_e, which is losing for the components of the environment. In order to prevent the deviation of the processes in the environment to U_s (which is safe for them), it is necessary that all the processes, except Process 0, have lost before reaching C_1. By definition of their sets of unsafe states, the only way to make them lose before reaching C_1 is a to chose a valuation that satisfies the formula, if it exists. ■

With a similar proof as the one for safety objectives, one can prove the NP-hardness for Cooperative Rational Synthesis Problem when the agents have reachability objectives.

Theorem 5.3.6. *The perfect information cooperative rational synthesis problem for reachability objectives is NP-hard.*

Proof. The proof is similar to the one for Theorem 5.3.5. It uses the same interaction model, but it changes the objectives of the processes. We set the target of the processes composing the environment ($\{1, ..., k\}$) so that when the state C_1 is reached, all the processes reached their objective if and only if the formula φ is satisfiable. Then, we also add the state U_s as their target and set the state U_e as the only target for Process 0.

Formally, $R_0 = \{U_e\}$ and $R_j = \{0_x \mid \neg x \in C_j\} \cup \{1_x \mid x \in C_j\} \cup \{U_s\}$ for $j \in \{1, ..., k\}$. Therefore, if some process does not reach its objective until the state C_1, he prefers to play U_s which makes Process 0 lose. ■

co-Büchi, Streett, Parity and Muller Objectives

In [87] (Theorem 15) it is already proven that the problem of finding a Nash equilibrium in coBüchi multiplayer games with a payoff between the thresholds $x = (x_i)_{i\in\Omega}$ and $y = (y_i)_{i\in\Omega}$ with $x_0 = y_0 = 1$, $x_i = 0$ and $y_i = 1$ for $1 \leq i \leq k$ is NP-hard. Therefore, the CRSP problem is NP-hard.

Theorem 5.3.7. *The perfect information Cooperative Rational Synthesis Problem is*

- *NP-hard for coBüchi, Streett and Parity objectives.*
- PSPACE*-hard for Muller Objectives*

Proof. As already argued, the NP-hardness for coBüchi objectives comes from [87]. We now prove the NP-hardness of CRSP for Streett and parity objectives which can (polynomially) express coBüchi objectives.

Parity. It follows directly from the following two facts: (i) the problem is NP-hard for coBüchi objectives, (ii) a coBüchi objective given by a set of states F can be equivalently expressed by the priority function p_F such that $p_F(v) = 1$ if $v \in F$, and 2 otherwise.

Streett. As for parity, a Streett condition can easily express a coBüchi condition F, by taking the set of pairs $\{(F, \varnothing)\}$. The result follows from the NP-hardness of coBüchi objectives.

Muller. The PSPACE-hardness of CRSP with Muller objectives follows by reducing from the problem of solving two-players zero-sum Muller games with Muller objective μ that by Theorem 3.1.2 is PSPACE-hard. In the interaction model, we keep the same game arena and set the objective of Process 0 to be μ and the objective of Process 1 to be $\neg\mu$. Then, it is obvious that there is a 0-fixed Nash equilibrium (σ_0, σ_1) winning for Process 0 iff there is a winning strategy σ_0 for the Process 0 in the zero-sum two-player game. ■

Rabin Objectives

We already saw that the Cooperative Rational Synthesis Problem for Rabin objectives can be solved in P^{NP}. We now prove that the problem is both NP-hard and $coNP$-hard.

Theorem 5.3.8. *The perfect information cooperative rational synthesis problem with Rabin objectives is NP-hard and coNP-hard.*

Proof. As in the case of Parity and Streett games, the NP-hardness comes directly from the fact that we can express a coBüchi condition $F \subseteq V$ as a Rabin condition $\psi = \{(V, F)\}$ where V is the set of states of the interaction model.

To show the coNP-hardness, we reduce the problem of solving two-player zero-sum Rabin games (which are NP-hard) to the CRSP problem with Rabin objectives. Let $\mathcal{G} = \langle V, V_A, V_B, E, v_0, \psi \rangle$ be such a game where the protagonist (Player A) has the Rabin objective ψ. We construct the interaction model $\mathcal{M}$ by considering a copy of $\mathcal{G}$ together with two extra states v and v' and transitions from v to both v' and the initial state of $\mathcal{G}$ and a self loop on v'. A sketched image of the interaction model is in Figure 5.3. Then, Agent 0 in $\mathcal{M}$ controls the states belonging to Player B in $\mathcal{G}$ and Agent 1 controls the states belonging to Player A in $\mathcal{G}$ together with the newly introduced states v and v'.

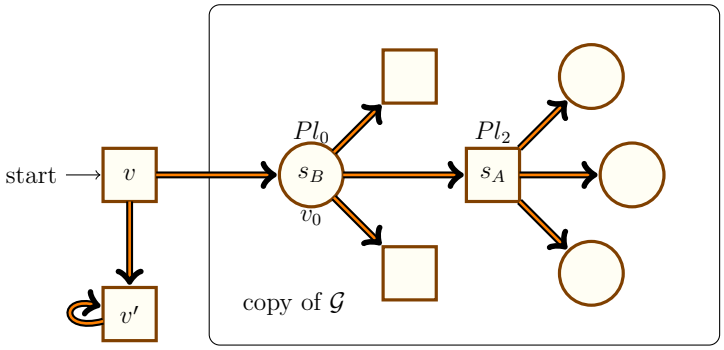


Figure 5.3: coNP-h Rabin CRSP: Reduction from 2-players Rabin games

Formally, $\mathcal{M} = \langle \{0,1\}, V', V_0, V_1, v_0', E', \psi_0, \psi_1 \rangle$ where $V' = V \cup \{v, v'\}$, $V_0 = V_B$, $V_1 = V_A \cup \{v, v'\}$, $E' = E \cup \{(v, v_0), (v, v'), (v', v')\}$, $v_0' = v$ and the objectives of the two agents are defined as $\psi_0 = \{(\{v'\}, \varnothing)\}$ and $\psi_1 = \psi$. That is, Agent 0 wins if the game goes in the state v' and Agent 1 wins if the winning condition of the protagonist in the game $\mathcal{G}$ is satisfied.

We claim that there is a solution to the rational synthesis problem in $\mathcal{M}$ if and only if there is no winning strategy for Player A in $\mathcal{G}$. Indeed, if there is a solution to the rational synthesis, there is a 0-fixed Nash equilibrium $\langle \sigma_0, \sigma_1 \rangle$ winning for Agent 0. The only possibility that this happens is if $\sigma_1(v) = v'$ in which case Agent 1 loses. But since $\langle \sigma_0, \sigma_1 \rangle$ is a 0-fixed Nash equilibrium, for any other strategy σ_1' s.t. $\sigma_1'(v) = v_0$, $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1' \rangle)$ does not satisfy $\mathsf{Rabin}(\psi_1)$. That is, there is a strategy σ_B for Player B in $\mathcal{G}$ so that $\forall \sigma_A$ a strategy of Player A, $\text{exec}(\mathcal{M}, (\sigma_A, \sigma_B)) \notin \mathsf{Rabin}(\psi)$ which means that Player A has no winning strategy in $\mathcal{G}$.

Suppose now that there is no solution to the cooperative rational synthesis. It means that there is no 0-fixed Nash equilibrium $\langle \sigma_0, \sigma_1 \rangle$ such that $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1 \rangle)$ satisfies $\mathsf{Rabin}(\psi_0)$. That is, whatever strategy σ_0 chooses Agent 0, Agent 1 prefers to go in the copy of $\mathcal{G}$ where he has a strategy to win. That is, Player A has a winning strategy σ_A in $\mathcal{G}$ to ensure $\mathsf{Rabin}(\psi)$. ■

5.3.4 CRSP with Fixed Number of Processes

Until now, we considered the general case where the number of agents composing the environment is not fixed. As announced, we restrict the rational synthesis problem to the particular case when the number of players is fixed and study the complexity of solving the cooperative rational synthesis problem.

Upper Bounds for k-fixed CRSP

The following theorems prove the upper bounds to solve CRSP with fixed $k+1$ number of agents provided in the second column of Table 5.1. In particular, Theorem 5.3.9 provides PTIME procedures to solve the k-fixed Cooperative Rational Synthesis Problem with respect to safety, reachability, Büchi and coBüchi objectives. Theorem 5.3.10 provides a UP∩ coUP algorithm for parity k-fixed CRSP.

Theorem 5.3.9. *The perfect information* CRSP *with respect to safety, reachability, Büchi and coBüchi objectives is in* PTIME *when the number of processes is fixed.*

Proof. As seen in the proof of Proposition 5.3.1, there is a solution for cooperative rational synthesis if and only if there is a path π such that $\pi \vDash \varphi$ where $\phi = \varphi_0 \wedge \phi^{\mathcal{M}}_{0\text{Nash}}$.

Given the above, the PTIME algorithm for the winning conditions $(\mathsf{X_i})_\mathsf{i}$ (where $\mathsf{X_i} \in \{\mathsf{Safe(S_i)}, \mathsf{Reach(R_i)}, \mathsf{Buchi(F_i)}, \mathsf{coBuchi(F_i)}\}$) first labels in polynomial time each node in the winning region W_i of each Agent i, $0 \le i \le k$, by W_i. Also, for the winning conditions of each process, we label in polynomial time nodes belonging to S_i (resp. R_i and F_i) with the corresponding atomic proposition v_{S_i} (resp. v_{R_i} and v_{F_i}). Note that since the number of processes is fixed, also the number of atomic propositions introduces is and the formula $\phi = \varphi_0 \wedge \phi^{\mathcal{M}}_{0\text{Nash}}$ becomes a constant formula(depends only on the number of processes).

Then, to check the existence of an execution so that $\rho \vDash \phi$, we build a constant size Büchi word automaton $\mathcal{A}_\phi$(since the LTL formula ϕ is constant for k constant), take the product with the interaction model and check in polynomial time the emptiness of the resulting automaton. ■

Theorem 5.3.10. *The perfect information* CRSP *with respect to parity objectives is in* UP∩ coUP *when the number of processes is fixed*

Proof. Given $0 \le i \le k$, let $p_i : V \to \{0, ..., 2n\}$ be the priority function for Agent i, where $n = |V|$. We provide a UP∩coUP algorithm to check if the interaction model $\mathcal{M}$ admits an execution such that $\rho \vDash \phi$, where $\phi = parity(p_0) \wedge \bigwedge_{1 \le i \le k}(parity(p_i) \vee \Box\neg W_i)$ and

$$parity(p_i) = \bigvee_{j=0}^{n}(\Box \Diamond C^i_{2j} \wedge \bigwedge_{p<2j} \Diamond \Box \neg C^i_p)$$

encodes the winning condition for Agent i, where C^i_j is an abbreviation for $\bigvee_{v:p_i(v)=j} v$ corresponding to color j associated by Agent i.

First, we prove that, if $\mathcal{M}$ admits an execution ρ such that $\rho \vDash \phi$, then $\mathcal{M}$ admits an execution $\rho' = \rho'_1(\rho'_2)^\omega$ such that $\rho' \vDash \phi$, $|\rho'_1| \leq n$ and ρ'_2 is a loop of size at most $(k+2)\cdot n$. We build the execution ρ^* as follows. If $\rho \vDash \Box\neg W_i$ for each $1 \leq i \leq k$, then ρ^* can be obtained by cutting ρ as soon as the first node repeats on it. Otherwise, we proceed to apply a pumping procedure. We mark the first occurrences if vertexes that have the smallest priority w.r.t. the parity function of each process. Then cut so that remains a short cycles that contains all the marked nodes. That is, for each $1 \leq i \leq k$ so that $\rho \vDash \mathit{parity}(p_i)$, let $m_i \in \{0 \dots n\}$ be the least priority w.r.t. p_i occurring infinitely often on ρ. For each node v, we label v by the vector $\bar{a} = (a_0 \dots, a_k)$, where for each $0 \leq i \leq k$, $\bar{a}[i] = m_i$ if $p_i(v) = m_i$, and $\bar{a}[i] = \bot$ otherwise. Since m_0 is the least priority w.r.t. p_0, there is a vertex u that appears infinitely often on ρ and is assigned a label $\bar{a}$ that has m_0 (rather than $\bot$) at index 0. Pick the first occurrence of such a vertex and color it by green. Repeat the above procedure for each $1 \leq i \leq k$ such that $\rho \vDash \mathit{parity}(p_i)$ (starting from the last green node colored along the path) in order to recover a green node on ρ for each $0 \leq i \leq k$ such that $\rho \vDash \mathit{parity}(p_i)$. Once detected the last green node, cut the remaining path as soon as is find a further occurrence of the vertex u. Therefore, we obtain an execution $\rho'' = \rho''_1\rho''_2$, where ρ''_2 is a loop (from u to u) witnessing that $\rho'' \vDash \phi$. Removing each simple loop on ρ''_1 as well as on each subpath of ρ''_2 without green nodes lead to a path $\rho' = \rho'_1(\rho'_2)^\omega$ such that $\rho' \vDash \phi$, $|\rho'_1| \leq n$, and ρ'_2 is a loop of size $(k+2)\cdot n$.

Given the above, it is sufficient to design UP$\cap$ coUP algorithm to check if $\mathcal{M}$ admits an execution $\rho' \vDash \phi$, where $\rho' = \rho'_1(\rho'_2)^\omega$ such that $|\rho'_1| \leq n$ and ρ'_2 is a loop of size $(k+2)\cdot n$. The UP algorithm works as follows. For each node v in $\mathcal{M}$, for each $1 \leq i \leq k$ guess if $v \in W_i$. Verify the guess applying the corresponding UP algorithm. If the guess was incorrect, then reject immediately. Otherwise check in NLOGSPACE if $\mathcal{M}$ contains an execution $\rho' \vDash \phi$, where $\rho' = \rho'_1(\rho'_2)^\omega$ such that $\rho' \vDash \phi$, $|\rho'_1| \leq n$ and ρ'_2 is a loop of size $(k+2)\cdot n$. This is possible by guessing on-the-fly a path ρ' and a node u on it where the loop should start, while maintaining (1) for each $0 \leq i \leq k$, the minimum priority seen along the loop w.r.t p_i; (2) for each $0 \leq i \leq k$, a bit to check if $\Box\neg W_i$ appears along ρ' (3) the length of ρ'_1, ρ'_2 and (4) the node u witnessing that ρ'_2 is a loop. Since k is a fixed constant, the priorities are bounded by n, and the length of the path is polynomial w.r.t. the size of the graph and therefore the amount of space required is logarithmic w.r.t. the size of the input graph.

The coUP algorithm verifies in UP if $\forall\rho$ holds $\rho \nvDash \phi$. This is done as follows. For each node v in $\mathcal{M}$ and for each $1 \leq i \leq k$, guess if $v \in W_i$. Verify the guess applying the corresponding UP algorithm. If the guess was incorrect, then reject immediately. Otherwise, verify in coNLOGSPACE if $\forall\rho$ holds $\rho \nvDash \phi$. This amounts to check in NLOGSPACE if $\exists\rho$ such that $\rho \vDash \phi$ that is done as above. ∎

Lower Bounds for k-fixed CRSP

Theorem 5.3.11 provides a reduction from two players zero-sum games to the cooperative rational synthesis problem in interaction models containing two agents. This allows one to infer the lower bounds on the complexities of CRSP for fixed number of processes.

Theorem 5.3.11. *The perfect information Cooperative Rational Synthesis problem for a fixed number of processes is as hard as solving the two-player zero-sum game with the same type of objective.*

Proof. The construction is similar to the one to prove the coNP-hardness of CRSP for Rabin objectives.

Let $\mathcal{G}$ be a two-players zero-sum game where the protagonist (Player A) has the objective ψ, and so Player B has objective $\neg\psi$. We construct the 2-processes CRSP interaction model $\mathcal{M}$ by considering a copy of $\mathcal{G}$ and two fresh states v and w. The state v is the initial state of $\mathcal{M}$ and has a transition to the initial state of $\mathcal{G}$ and a transition to w, which is equipped with a self-loop. The environment (Agent 1) controls v, w and the states belonging to Player A in $\mathcal{G}$, while Agent 0 controls the states belonging to Player B in $\mathcal{G}$. For the winning conditions, Agent 0 wins only if the play gets into w (and stays that forever), while the objective of the environment is ψ (i.e. the objective of Player A in $\mathcal{G}$).

$\mathcal{M}$ is a positive instance of the CRSP problem if and only if Agent 1 playing edge $v \to w$ is a Nash equilibrium. But clearly Agent 1 does not have an incentive to deviate if and only if Player A does not have a winning strategy in $\mathcal{G}$ for forcing ψ. ■

Since Safety, Reachability, Büchi and coBuchi two-player zero-sum games are PTime-complete (crf. Theorem 3.1.2), it follows that also the CRSP problem in models with a fixed number of agents is PTime-hard. For parity objectives, the problem is as hard as solving the two-player zero-sum parity games (parity-hard) since. The NP-hardness for Streett objectives comes from the co-NP-hardness of Streett objectives.

Finally, we already proved that CRSP with two processes is NP-hard for Rabin objectives (crf. Theorem 5.3.8) and the PSpace-hardness for Muller objectives (crf. Theorem 5.3.7).

Note that the general algorithm for solving the perfect information Cooperative Synthesis Problem for a fixed number of processes is the same as in the non-fixed case. However, the fact that the number of processes is fixed, makes easier to search for an execution that satisfies the formula $\varphi_0 \wedge \phi^{\mathcal{M}}_{0\text{Nash}}$.

5.4 Non-Cooperative Rational Synthesis (NCRSP)

In this section we study the complexity of non-Cooperative Rational Synthesis Problem. In this setting the environment may not cooperate with the Process 0 (to synthesize), and may play (rationally) any strategy profile providing it is a 0-fixed Nash equilibrium.

In the cooperative setting, in the cases where we could not rely on existing results [87], namely reachability and safety objectives, we get our upper bounds via a reduction to a model-checking problem. In the non-cooperative setting, we cannot rely on existing results.

Proposition 5.2.1 characterises 0-fixed Nash equilibria by means of an $\mathsf{LTL}[\mathcal{M}]$ formula $\phi^{\mathcal{M}}_{0\text{Nash}}$. This allowed us to solve cooperative rational synthesis problem by model-checking against the interaction model $\mathcal{M}$, the formula $\phi^{\mathcal{M}}_{0\text{Nash}} \wedge \varphi_0$, where φ_0 is Process 0's objective. It is tempting to think that non-cooperative rational synthesis reduces to a two-player zero-sum game between Process 0, whose objective is $\phi^{\mathcal{M}}_{0\text{Nash}} \rightarrow \varphi_0$, and the coalition of the other processes.

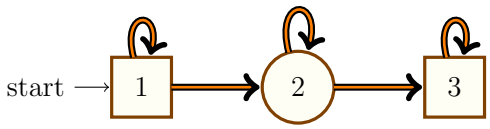


Figure 5.4: NCRSP: Example of interaction model

Consider the three states interaction model Figure 5.4 on which interact two processes. Process 0 owns circle states and Process 1 square states and they have reachability objectives given by the sets $R_0 = \{2\}$ and $R_1 = \{3\}$ respectively. This example shows that this intuitive reduction to two-player games is not true in general. Indeed, in this example there is a solution to non-cooperative rational synthesis problem, but no solution to the two-player game with objective $(\Box\bar{R}_1 \rightarrow \Box\bar{W}_1) \rightarrow \Diamond R_0$. Since $W_1 = \{3\}$, whatever the strategy of Process 0 is, if Process 1 stays in state 1 forever, the execution $\rho = (1)^\omega$ satisfies $(\Box\bar{R}_1 \rightarrow \Box\bar{W}_1)$ but not $\Diamond R_0$ and therefore Process 0 loses.

The intrinsic reason why the reduction to two-player games is incorrect lies in the definition of non-cooperative rational synthesis problem: once a Process 0's strategy σ_0 is fixed, only 0-fixed NE with respect to σ_0 are considered, while the formula $\phi_{0\text{Nash}}$ can be satisfied by paths which are outcomes of some 0-fixed Nash equilibrium, fixed for a different strategy of Process 0.

5.4.1 General Solution for NCRSP

The non-Cooperative Rational Synthesis Problem is more involved and requires automata based techniques to solve and provide tight complexities for problem when safety, reachability or the other classical tail objectives are considered.

We represent strategies σ_0 of Process 0 as a tree t_{σ_0} and use *tree automata* to define the set of strategies that are solutions to the non-cooperative rational synthesis in the interaction model $\mathcal{M} = \langle \Omega, V, (V_i)_{i\in\Omega}, v_0, E, (\Theta_i)_{i\in\Omega} \rangle$.

Theorem 5.4.1. *Let $\mathcal{M}$ be an interaction model with $k+1$ processes and n states. One can construct a non-deterministic tree automaton $\mathcal{T}_{\mathcal{M}}$ with an accepting condition α such that $\mathcal{L}_\alpha(\mathcal{T}_{\mathcal{M}})$ is the set of trees encoding solutions to NCRSP. The automaton has an exponential number of states in k, and polynomial in n. Moreover, for all runs r of $\mathcal{T}_{\mathcal{M}}$, for all branches η of r, the number of states appearing in η is polynomial in n and k.*

In the following, we provide the construction of the tree automaton that proves the above theorem. The emptiness of tree automata is then checked by solving a two-player zero-sum game, whose complexity is carefully analyzed for all the winning conditions considered in this chapter.

Strategies as trees

Let $\mathcal{M}$ be an interaction model in which acts the set $\Omega = \{0, ..., k\}$ of processes. Let $\sigma_0 : V^*V_0 \to V$ be a strategy for Process 0. As seen in Section 3.4.1, strategies can be encoded as strategy trees.

Since we consider here turn-based interaction model, the strategy trees are labeled with letters in $\Lambda = V \cup \{*_i \mid 1 \le i \le k\} \cup \{\#\}$ and the set of directions is $\Upsilon = V$. Therefore, any node of the tree is an history h in the interaction model $\mathcal{M}$.

Intuitively, the labeling of the strategy tree t_{σ_0} is as follows. The root is the only node labeled by $\#$. Then, for each node controlled by Process 0 (hv s.t. $v \in V_0$), the label is according to the strategy σ_0. When is the turn to a process $i \neq 0$ to play (some node hv s.t. $v \in V_i$) it is labeled by "letter" $*_i$. This letter encodes the fact that Process i could do any choice in the state v.

Formally, the labeling of the nodes in the strategy tree is defined by:

(i) $t_{\sigma_0}(\epsilon) = \#$

(ii) $t_{\sigma_0}(hv) = \sigma_0(hv)$ if $v \in V_0$,

(iii) $t_{\sigma_0}(hv) = *_i$ if $v \in V_i$ for $i \neq 0$ (only the turn i is encoded)

Note that each Λ-labeled V-tree represents a partial function from V^* to Λ, which may not be a strategy, because it is not total and may not be consistent with the edge relation E of the interaction model. A *branch* in t_{σ_0} is an infinite sequence of directions $\rho \in V^\omega$. It is *compatible* with σ_0 if for all finite prefixes h of ρ whose last state is in V_0, $h.t_{\sigma_0}(h)$ is a prefix of ρ.

Characterization of solutions for NCRSP

We now want to characterize the strategy trees t_{σ_0} s.t. σ_0 is a solution to non-cooperative rational synthesis problem in an interaction model $\mathcal{M} = \langle \Omega, V, (V_i)_{i\in\Omega}, v_0, E, (\Theta_i)_{i\in\Omega}\rangle$ with either all safety, all reachability, or all tail objectives.

We already saw that a branch of a strategy tree represents an infinite sequence of states in the interaction model. It may not correspond to executions, but the tree automaton that we will define will take care to consider only the branches that are executions compatible with the strategy σ_0 encoded by the tree.

The strategy tree t_{σ_0} is a solution for the problem if for all branches π compatible with σ_0, either it is winning for Process 0, or it does not correspond to a 0-fixed Nash equilibrium.

Good deviations A branch π is not the outcome of a 0-fixed Nash equilibrium if and only if some process i loses along π ($\pi \notin \Theta_i$ for some $i \neq 0$) and there is a history h from which Process i has a winning strategy against all other processes in the environment (when Process 0 plays σ_0). We call the history h a good deviation point.

Formally, h is a *good deviation point* along a branch π (h is a prefix of π) if there is $i \in \Omega \setminus \{0\}$ s.t. $\pi \notin \Theta_i$ and there is a strategy σ_i for Process i s.t. for all strategies $(\sigma_j)_{i\in\{1,\dots,k\}\setminus\{i\}}$, holds $h.\mathrm{exec}(\mathcal{M}, \sigma_0^h, \dots, \sigma_{i-1}^h, \sigma_i^h, \sigma_{i+1}^h, \dots, \sigma_k^h) \in \Theta_i$. A branch $\pi \in V^\omega$ has a *good deviation* if some of its prefix h is a good deviation point.

Let us denote by $\mathsf{NCRSP}(\mathcal{M})$ the set of strategy trees t_{σ_0} such that σ_0 is a solution to the NCRSP in $\mathcal{M}$. Then:

Lemma 5.4.1. *For all strategies σ_0 of Process 0, $t_{\sigma_0} \in \mathsf{NCRSP}(\mathcal{M})$ iff for all branches π of t_{σ_0} compatible with σ_0, either $\pi \in \Theta_0$ or π has a good deviation.*

Proof. First, let prove the implication from left to right and consider $t_{\sigma_0} \in \mathsf{NCRSP}(\mathcal{M})$ and a branch (execution in $\mathcal{M}$) π such that $\pi \notin \Theta_0$. Then, since t_{σ_0} is a solution to $\mathsf{NCRSP}(\mathcal{M})$, for all $\sigma_1, \dots, \sigma_k$ such that $\mathrm{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \dots, \sigma_k\rangle) = \pi$, $\langle \sigma_0, \sigma_1, \dots, \sigma_k\rangle$ is not a 0-fixed Nash equilibrium.

Let us define $\sigma_1, \dots, \sigma_k$ such that all the processes follow the path π and punish the Process i that deviates from it by playing the worst strategy profile for him (they play the retaliating strategies $\mathsf{ret}_j^{v,i}$ against Process i from the state v to which he deviates). That is, each Process j plays σ_j defined as

$$\sigma_j(x) = \begin{cases} \pi[l+1] & \text{if } x = v_0v_1\dots v_l \text{ is a prefix of } \pi \\ \mathsf{ret}_j^{v,i}(vx_2) & \text{if condition (1) is satisfied} \\ \beta_j(x) & \text{otherwise} \end{cases}$$

where β_j is an arbitrary strategy of Process j and condition (1) requires that x can be decomposed into $x = x_1vx_2$ such that $x_1 \in V^*V_i$, $v \notin W_i^{\mathcal{M}[\sigma_0]}$, x_1 is a prefix of π, x_1v is not

a prefix of π. The fact that $v \notin W_i^{\mathcal{M}[\sigma_0]}$ means that Process i has deviated to a state from which he does not have a winning strategy when the strategy σ_0 of Process 0 is fixed, i.e., Processes $1, ..., j-1, j+1, ..., k$ have a strategy to make him lose under σ_0.

Clearly, $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, ..., \sigma_k \rangle) = \pi$ and therefore by hypothesis, $\langle \sigma_0, \sigma_1, ..., \sigma_k \rangle$ is not a 0-fixed Nash equilibrium. Hence, there is a process i that prefers to deviate from π and has a strategy σ_i' such that $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, ..., \sigma_{i-1}, \sigma_i', \sigma_{i+1}, ..., \sigma_k \rangle) \in \Theta_i$. From the construction of the strategy profile, Process i chooses to deviate to a state in which he has a winning strategy when Process 0 plays σ_0 (otherwise the other processes make him lose). Let $h \in V^* V_i$ be the prefix of π after which Process i deviates and v its last state. Then, for all strategies $\tilde{\sigma}_1, ...\tilde{\sigma}_{i-1}, \tilde{\sigma}_{i+1}, ..., \tilde{\sigma}_k$ for the processes $1, ..., i-1, i+1, ..., k$, we have that $h.\mathsf{exec}(\mathcal{M}[v], \langle \sigma_0^h, ..., \tilde{\sigma}_{i-1}^h, \sigma_i'^h, \tilde{\sigma}_{i+1}^h, ..., \tilde{\sigma}_k^h \rangle) \in \Theta_i$ which means that there is a good deviation for Process i and then π has a good deviation h.

In the other direction, let us take the strategy tree t_{σ_0} and π a branch of t_{σ_0} compatible with σ_0 and representing an execution in $\mathcal{M}$ s.t. $\pi \notin \Theta_0$. Then, there is a good deviation from π for a Process i that loses in π. That is, Process i has a strategy σ_i' such that he wins by deviating from π at a position j against any strategy profile that follows π. That is, for all $\sigma_1, ..., \sigma_k$ s.t. $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, ..., \sigma_k \rangle) = \pi$, $\langle \sigma_0, \sigma_1, ..., \sigma_k \rangle$ is not a 0-fixed NE since Process i can deviate and win. Therefore, $t_{\sigma_0} \in \mathsf{NCRSP}(\mathcal{M})$.

The equivalence is straightforward for the branches $\pi \in \Theta_0$. ∎

The previous lemma provides enough intuition on the structure and the properties that have to be verified along branches of the strategy trees t_{σ_0}.

Reduction to tree-automata emptiness

Based on Lemma 5.4.1, we construct a nondeterministic automaton announced in Theorem 5.4.1 that defines the set $NCRSP(\mathcal{M})$ of solutions for the non-Cooperative Rational Synthesis Problem in the interaction model $\mathcal{M} = \langle \Omega, V, (V_i)_{i \in \Omega}, v_0, E, (\Theta_i)_{i \in \Omega} \rangle$.

The nondeterministic tree automaton $\mathcal{T}_{\mathcal{M}}$ is obtained as a product of two automata. First, we construct a deterministic safety tree automaton $\mathcal{C}_{\mathcal{M}}$ that checks that a t_{σ_0} is a proper encoding of a strategy σ_0 in the turn-based interaction model $\mathcal{M}$. Then, we construct a nondeterministic tree automaton $\mathcal{U}_{\mathcal{M}}$ that is assumed to run on proper encodings of strategies and checks that it corresponds to a solution to the NCRSP. Details on the construction of the two automata are given in the following.

Automaton $\mathcal{C}_{\mathcal{M}}$ The deterministic safety automaton $\mathcal{C}_{\mathcal{M}}$ accepts trees that are proper strategy trees encoding a strategy σ_0 of Process 0. That is, he has to remember the last direction v taken and make sure that if $v \in V_0$, the current node is labeled by some $v' \in V$ s.t. $(v, v') \in E$, and otherwise by the symbol $*_i$ if $v \in V_i$. This automaton is polynomial in the size of the interaction model.

Formally, the automaton $\mathcal{C}_{\mathcal{M}}$ is defined as $\mathcal{C}_{\mathcal{M}} = \langle \Lambda, \Upsilon, Q_{\mathcal{C}}, q_0^{\mathcal{C}}, \delta_{\mathcal{C}}, \alpha_{\mathcal{C}} \rangle$ where $\Lambda = V \cup \{*_i \mid 1 \leq i \leq k\} \cup \{\#\}$ is the alphabet and $\Upsilon = V$ is the set of directions. The set of states is $Q_{\mathcal{C}} = V \cup \{\bot, q_0^{\mathcal{C}}\}$ and the transition relation is defined as

- $\delta_{\mathcal{C}}(q_0^{\mathcal{C}}, \#) = f$ where $f(v) = v$ for all $v \in V$,
- $\delta_{\mathcal{C}}(q_0^{\mathcal{C}}, \ell) = f_\bot$ where $f_\bot(v') = \bot$ for any $v' \in V$ and $\ell \neq \#$,
- $\delta_{\mathcal{C}}(q, \#) = f_\bot$ for $q \neq q_0^{\mathcal{C}}$,
- $\delta_{\mathcal{C}}(\bot, \ell) = f_\bot$ and
- $\delta_{\mathcal{C}}(v, \ell) = f_{v,\ell}$ where $f_{v,\ell}(v') = \begin{cases} v' & \text{if } (v \in V_0 \text{ and } (v, \ell) \in E) \\ & \quad \text{or } (v \in V_{i \neq 0} \text{ and } \ell = *_i) \text{ for any } v' \in V \\ \bot & \text{otherwise} \end{cases}$

The first two cases verify that the root of the accepted tree is labeled with "#". If another letter is read, the automaton goes to the rejecting state " $\bot$ " on all directions, which is a trap according to the fourth case. The third case checks that the letter "#" does not appear in the rest of the tree. Finally, the last case verifies the proper labeling the rest of the nodes of the strategy tree.

The acceptance condition on $\mathcal{C}_{\mathcal{M}}$ is $\alpha_{\mathcal{C}} = \{\eta \in (Q_{\mathcal{M}} \setminus \{\bot\})^\omega\}$. Note that by the construction of the automaton, this is a safety condition that asks to avoid " $\bot$ ". The state " $\bot$ " appears in two cases. Either is the turn of Process 0(state $v \in V_0$) to play and the letter l that is read is not a valid choice of his ($(v, l) \notin E$), or is the turn of Process $i \neq 0$ (state $v \in V_i$) and the label is different from $*_i$. Also, the automaton $\mathcal{C}_{\mathcal{M}}$ is deterministic.

In the following, we construct the automaton $\mathcal{U}_{\mathcal{M}}$ that accepts trees that are solutions for the non-Cooperative Rational Synthesis Problem. We assume that the automaton $\mathcal{U}_{\mathcal{M}}$ only runs on proper encodings t_{σ_0} of strategies σ_0 of Process 0. That is, trees that are accepted by the tree automaton $\mathcal{C}_{\mathcal{M}}$.

Automaton $\mathcal{U}_{\mathcal{M}}$ The construction of $\mathcal{U}_{\mathcal{M}}$ is based on Lemma 5.4.1. For each branch, the automaton checks that either it belongs to Θ_0, or it guesses a prefix and checks it is a good deviation. That is, in the later case, the automaton has to guess subtrees in which at least one process has a winning strategy. This information is stored in a set $W \subseteq \Omega$, with the following semantics: if $\mathcal{U}_{\mathcal{M}}$ is in some state with set W at some node $h \in V^*$ and $i \in W$, then Process i has a winning strategy in the subtree rooted at h against σ_0 and any strategy of processes in $\Omega \setminus \{0, i\}$. The set of processes for which a good deviation has been guessed is stored in a set $D \subseteq \Omega$, with the following semantics: if $\mathcal{U}_{\mathcal{M}}$ is in some state with set D and $i \in D$, at some node $h \in V^*$, then some prefix of h is a good deviation.

The information in D is monotonic. Whenever $i \in D$ in a state, $i \in D$ in all the successor states. In addition, it is updated by adding processes in D depending on the

updates of W. The information on W is maintained as follows: at some node $hv \in V^*$, if $i \in W$ and $v \in V_i$, the automaton $\mathcal{U}_\mathcal{M}$ non-deterministically chooses a strategy move for the Process i and send W to one of the successor of v (and $W \setminus \{i\}$ in the other ones). If $i \notin W$ and $v \in V_i$, there are two possibilities. First, if $i \in D$ means that a deviation was guessed before and then W is sent to all successors. Otherwise, if $i \notin D$, there was not guessed a good deviation point before. Then either the current node h (owned by Process i) is not guessed to be a good deviation point and D and W are sent to all successors, or it is guessed to be a good deviation for Process i and then $D \cup \{i\}$ and W are sent to all successors but one in which is sent D and $W \cup \{i\}$. If $v \notin V_i$, the automaton $\mathcal{U}_\mathcal{M}$ keeps $i \in W$ in all successors of v.

Formally, $\mathcal{U}_\mathcal{M} = \langle \Lambda, \Upsilon = V, Q_\mathcal{U}, q_0^\mathcal{U}, \delta_\mathcal{U}, \alpha_\mathcal{U} \rangle$ where the alphabet is $\Lambda = V \cup \{*_i \mid 1 \leq i \leq k\} \cup \{\#\}$ and the set of states is $Q_\mathcal{U} \subseteq \{q_0^\mathcal{U}, \top\} \cup 2^\Omega \times 2^\Omega \times V$. A state $q = (W, D, v)$ stores information about the set W of processes that need a winning strategy from the current node, the set D of processes that may deviate to win, and the last direction taken.

To define the transition relation, we will define functions mapping directions to states. If we do not define them for some directions d, it means that d is mapped to $\top$ and corresponds to successors that cannot be reached by a transition in the interaction model compatible with the strategy encoded by the tree t_{σ_0} to accept. The transition relation $\delta_\mathcal{U}$ is defined as:

- $\delta_\mathcal{U}(q_0^\mathcal{U}, \#) = \{g_0\}$ where $g_0(v_0) = (\varnothing, \varnothing, v_0)$ and $g_0(v) = \top$ for all $v \neq v_0$,
- if $q = (W, D, v)$ s.t. $v \in V_0$: $\delta_\mathcal{U}(q, v') = \{g_{v'}\}$ where $g_{v'}(v') = (W, D, v')$ and $g_{v'}(v'') = \top$ for $v'' \neq v'$ (the later case corresponds to directions v'' that are not compatible with the strategy),
- $\delta_\mathcal{U}(\top, \ell) = \{g_\top\}$ where $g_\top(v') = \top$, for all $v' \in V$ and $\ell \neq \#$
- For a state $q = (W, D, v)$ and a label $*_i$ ($i \neq 0$) we consider four cases:
  1. $i \in D \cap W$: Such a state is never reached by construction
  2. $i \in D \cap \overline{W}$: Just propagate the information D and W. That is,
  $$\delta_\mathcal{U}(q, *_i) = \{g\} \text{ s.t. } g(v') = (W, D, v') \text{ for all } (v, v') \in E$$
  3. $i \in \overline{D} \cap W$: One has to check that Process i has a winning strategy in some successor v' (guessed nondeterministically) and to which the W information is sent. That is,
  $$\begin{aligned}\delta_\mathcal{U}(q, *_i) = &\{g_{v'} \mid (v, v') \in E\} \text{ s.t.} \\ &g_{v'}(v') = (W, D, v') \text{ and} \\ &g_{v'}(v'') = (W \setminus \{i\}, D \cup \{i\}, v'') \text{ for all } v'' \neq v'\end{aligned}$$

4. $i \in \overline{D} \cap \overline{W}$: Either nothing is guessed, or one guesses that Process i has a good deviation, and updates the sets W and D accordingly. That is,

$$\begin{aligned}\delta_{\mathcal{U}}(q, *_i) = &\{g\} \cup \{g'_{v'} \mid (v, v') \in E\} \text{ s.t.} \\ &g'_{v'}(v') = (W \cup \{i\}, D, v') \text{ and} \\ &g'_{v'}(v'') = (W, D \cup \{i\}, v'') \text{ for all } v'' \neq v'\end{aligned}$$

Along a branch of a run of $\mathcal{U}_{\mathcal{M}}$, there are monotonicity properties for the W and D-components of the states. Indeed, by construction, $\mathcal{U}_{\mathcal{M}}$ never removes a process from D. For W, a Process i can be removed (case 3) but then it is added to D and, once a process belongs to D, it can never be added to W again. This optimization is correct since for a history h, if we guess that Process i has a winning strategy from history hv, then i is added to D for all successors hv' ($v' \neq v$) and there is no need to guess again later on a good deviation for Process i in the subtrees rooted at the nodes hv', and therefore no need to add i in W again.

A consequence of the monotonicity property is that along a branch η of a run, there is only a polynomial number of different components D and W, and they necessarily stabilize eventually, to a set that we denote by $\lim_D(\eta)$ and $\lim_W(\eta)$. This monotonic behavior is crucial for complexity.

Finally, the accepting condition $\alpha_{\mathcal{U}}$ asks that on each path of the accepting run, either it is of the form $Q_{\mathcal{U}}^*\{\top\}^\omega$ (it is not an execution in $\mathcal{M}$ compatible with the strategy encoded by the input tree), or Process 0 wins, or there is a process that loses but belongs to some D eventually (therefore in the past he could have deviate and win). For safety objectives, we also have to add the constraint that the losing process belongs to D before visiting an unsafe state. Additionally, the accepting condition also expresses constraints on the W components. It verifies that the processes that belong to W after it stabilizes, indeed win by checking that the projection on the directions belong to Θ_i. That is, each process in $\lim_W(\eta)$ wins.

Formally, if we denote by $\mathrm{IRuns}(\mathcal{U}_{\mathcal{M}})$ the set of images of branches of runs of $\mathcal{U}_{\mathcal{M}}$, and by $\eta|_V$ the V-projection of any $\eta \in (Q_{\mathcal{U}} \setminus \{\top\})^\omega$, we have:

$$\begin{aligned}\alpha_{\mathcal{U}} = Q_{\mathcal{U}}^*\{\top\}^\omega \cup \Big(\{\eta \in \mathrm{IRuns}(\mathcal{U}_{\mathcal{M}}) \cap (Q_{\mathcal{U}} \setminus \{\top\})^\omega \mid \eta|_V \in \Theta_0 \vee \bigvee_{i=1}^{k} \big(\eta|_V \notin \Theta_i \wedge \varphi_{\exists dev}(i, \eta)\big)\} \cap \\ \cap \{\eta \in \mathrm{IRuns}(\mathcal{U}_{\mathcal{M}}) \cap (Q_{\mathcal{U}} \setminus \{\top\})^\omega \mid \bigwedge_{i \in \lim_W(\eta)} \eta|_V \in \Theta_i\}\Big)\end{aligned}$$

where the formula $\varphi_{\exists dev}(i, \eta)$ says that there is a good deviation for Process i. That is, if $D_0 D_1 \dots$ is the sequence of D-components in η, then $\varphi_{\exists dev}(i, \eta) = i \in \lim_D(\eta)$ for tail or reachability objectives, and $\varphi_{\exists dev}(i, \eta) = \exists p \geq 0, i \in D_p \wedge \forall p' \leq p, \eta|_V[p] \in S_i$ for safety conditions $\mathsf{Safe}(S_i)$.

We now provide the general property of the tree automaton $\mathcal{U}_{\mathcal{M}}$ that is independent on the type of individual objectives of the processes in $\mathcal{M}$. The only assumption is that one

is able to write a formula $\varphi_{\exists dev}(i, \eta)$ that correctly expresses the fact that Process i has a deviation to win. That is, a good deviation point for Process i along η may appear only before the objective of Process i is falsified. Thanks to the construction of the automaton $\mathcal{U}$, we can write an LTL formula that refers to the position along η when Process i is added to the set D.

Lemma 5.4.2. *For all strategies σ_0 of Process 0, $t_{\sigma_0} \in \mathcal{L}_{\alpha_\mathcal{U}}(\mathcal{U}_\mathcal{M})$ iff for all branches π of t_{σ_0} compatible with σ_0, either $\pi \in \Theta_0$ or π has a good deviation.*

Proof. Let $t_{\sigma_0} \in \mathcal{L}_{\alpha_\mathcal{U}}(\mathcal{U}_\mathcal{M})$. By construction, the automaton $\mathcal{U}_\mathcal{M}$ takes transitions such that all the branches that are not compatible with the strategy σ_0 encoded by t_{σ_0} lead to the state "$\top$". On all the other branches π, the accepting run r_t on t_{σ_0} satisfies the condition that the branch η compatible with π either $\eta|_V \in \Theta_0$ (which means that $\pi \in \Theta_0$) or there is a process i that loses along $\pi = \eta|_V$ and has a good deviation. The automaton verifies that all the deviations are correctly guessed by constructing a winning strategy for Process i from the deviation point. Then, the condition $\bigwedge_{i \in lim_W(\eta)} \eta|_V \in \Theta_i$ asks that all processes for which η is compatible with their (guessed) winning strategy satisfy their objective.

On the other direction, lets take a tree t_{σ_0} such that for all branches π of t_{σ_0} compatible with σ_0, either $\pi \in \Theta_0$ or π has a good deviation. We can build an accepting run in $\mathcal{U}$ on t_{σ_0} as follows. For each branch π of the tree t_{σ_0} such that $\pi \notin \Theta_0$, guesses the deviation point of some Process i. Then, the automaton starts to build the winning strategy for the process that deviates. This strategy exists by the definition of a good deviation point. Note that the accepting condition α is satisfied along the branches of the constructed run. This is because the formula $\varphi_{\exists dev}$ correctly expresses the existence of a deviation in terms of LTL formula on the D projection of branches. Then, whenever a process is added in D, the construction of the winning strategy using the set W starts. Since the winning strategy of Process i was correctly guessed, he wins along all branches compatible with it. That is, along all branches η such that $i \in lim_W(\eta)$. ∎

Automaton $\mathcal{T}_\mathcal{M}$. Then, as mentioned before, the tree automaton $\mathcal{T}_\mathcal{M}$ with the accepting condition α such that $\mathcal{L}_\alpha(\mathcal{T}_\mathcal{M}) = \mathsf{NCRSP}(\mathcal{M})$ is defined as the product of the two automata $\mathcal{C}_\mathcal{M}$ and $\mathcal{U}_\mathcal{M}$. Formally, the automaton $\mathcal{T}_\mathcal{M} = \langle \Lambda, \Upsilon = V, Q, q_0, \delta, \alpha \rangle$ is defined over the alphabet $\Lambda = V \cup \{*_i \mid 1 \leq i \leq k\} \cup \{\#\}$, has directions $\Upsilon = V$, states in $Q \subseteq (2^\Omega \times 2^\Omega \times V) \cup \{\bot, (q_0^\mathcal{U}, q_0^\mathcal{C})\} \cup (\{\top\} \times V)$, $q_0 = (q_0^\mathcal{U}, q_0^\mathcal{C})$ is the initial state and the transition relation for $\ell \in \Lambda$ is defined by

- $\delta((q_0^\mathcal{U}, q_0^\mathcal{C}), \ell) = \begin{cases} \{f_\bot\} & \text{if } \delta_\mathcal{C}(q_0^\mathcal{C}, l) = f_\bot \\ \{g_0\} & \text{otherwise} \end{cases}$
- $\delta(\bot, \ell) = \{f_\bot\}$

- $\delta((\top, v), \ell) = \begin{cases} \{g_t\} & \text{with } g_t(v') = (\top, v') \text{ if } \delta_{\mathcal{C}}(v, v') = v' \text{ for } v' \in V \\ \{f_\perp\} & \text{if } \delta_{\mathcal{C}}(v) = f_\perp \end{cases}$

- $\delta((W, D, v), \ell) = \begin{cases} \delta_{\mathcal{U}}((W, D, v), \ell) & \text{if } \delta_{\mathcal{C}}(v, v') = v' \text{ for all } v' \in V \\ \{f_p\} & \text{otherwise} \end{cases}$

Remark 5.4.1. *Note that the automaton $\mathcal{T}_{\mathcal{M}}$ follows the transitions in $\mathcal{U}_{\mathcal{M}}$ as far as the automaton $\mathcal{C}_{\mathcal{M}}$ does not reach the state " $\perp$ ". Therefore, on each branch of a run, the number of different states in $\mathcal{T}_{\mathcal{M}}$ is still polynomial in the size of the interaction model $\mathcal{M}$. This is also because $\mathcal{T}_{\mathcal{M}}$ is the product of $\mathcal{U}_{\mathcal{M}}$ with a deterministic safety tree automaton of polynomial size.*

The acceptance condition for the automaton $\mathcal{T}_{\mathcal{M}}$ is in essence the condition $\alpha_{\mathcal{U}}$ but also asks to avoid states " $\perp$ " that are reached in $\mathcal{C}_{\mathcal{M}}$ if the tree to accept is not a proper encoding of some strategy σ_0. That is,

$$\alpha = Q^*(\{\top\} \times V)^\omega \cup \Big\{\eta \in \mathrm{IRuns}(\mathcal{T}_{\mathcal{M}}) \cap \{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega \mid \Big(\eta|_V \in \Theta_0 \vee \bigvee_{i=1}^{k} \big(\eta|_V \notin \Theta_i \wedge \varphi_{\exists dev}(i, \eta)\big)\Big) \wedge \bigwedge_{i \in \lim_W(\eta)} \eta|_V \in \Theta_i\Big\}$$

Remark 5.4.2. *Since the automaton $\mathcal{T}_{\mathcal{M}}$ is obtained by taking the product between the automata $\mathcal{U}_{\mathcal{M}}$ and $\mathcal{C}_{\mathcal{M}}$ that verifies that a tree is a proper encoding of a strategy for Process 0, and because of Lemmas 5.4.1 and 5.4.2, the automaton $\mathcal{T}_{\mathcal{M}}$ is the witness proving Theorem 5.4.1.*

The following lemma proves that on all loops that are taken by the branches of a run in the tree automaton $\mathcal{T}_{\mathcal{M}}$, the sets D and W are constant.

Lemma 5.4.3. *Let $\pi \in (Q \cap \{q_0\}(2^\Omega \times 2^\Omega \times V))^\omega$ be a path of a run in $\mathcal{T}_{\mathcal{M}}$. Then, each loop on π has only one value on states for the sets W and D.*

Proof. Let us take a path $\pi = xq'yq'z$ of a run in $\mathcal{T}_{\mathcal{M}}$. Because of the definition of $\delta_{\mathcal{U}}$, $\pi' = xq'(yq')^\omega$ is also a valid path of a run in $\mathcal{T}_{\mathcal{M}}$. Suppose that there are two consecutive states in y such that a process is removed/added from/to W in the second state compared to the previous one. Then, there are also two consecutive states in y such that it is added/removed to/from W. This contradicts the fact that π' is a valid path of a run in $\mathcal{T}_{\mathcal{M}}$ since we could do more than one addition of a process to W. Then, there is no change on W on a loop. Also, because of the monotonicity of D, we prove that the value of D remains unchanged along a cycle using the same argument. ∎

From tree automata to two-player games. As presented in Section 3.4.4, checking emptiness non-deterministic tree automata is reduced to solving a two-player zero sum game between Eve, who constructs a tree and a run on this tree, and Adam, whose goal is to prove that the run is non-accepting, by choosing directions in the tree and falsifying the acceptance condition.

Formally, remind that the alphabet is $\Lambda = V \cup \{*_i \mid 1 \leq i \leq k\} \cup \{\#\}$ and for a function $f : V \to Q$, we denote by $\mathsf{Range}(f)$ its range. The zero-sum two-player game $\mathcal{G}_\mathcal{T} = \langle V_E, V_A, E', q_0, \Theta \rangle$ is such that $V_E = Q$, $V_A = \{\mathsf{Range}(f) \mid \exists q \in Q, \ell \in \Lambda, f \in \delta(q, \ell)\}$ and the transition relations is defined for all $q \in Q$, all $Y \in V_A$, by

- $(q, Y) \in E'$ if there exists $\ell \in \Lambda$ and $f \in \delta(q, \ell)$ s.t. $Y = \mathsf{Range}(f)$,
- $(Y, q) \in E'$ if $q \in Y$.

In other words, to go from q to Y, Eve chooses a symbol ℓ and a function $f : V \to Q$ in $\delta(q, \ell)$. Then, Adam chooses a direction in V, but since he wants to construct a sequence of states not in α, one only needs to remember $\mathsf{Range}(f)$. Adam then picks a state in that set. Finally, Eve's objective is then the set $\Theta = \{\pi = q_1 Y_1 q_2 Y_2 \cdots \in (V_E V_A)^\omega \mid \pi\restriction_{V_E} = q_1 q_2 \cdots \in \alpha\}$ where $\pi\restriction_{V_E}$ stands for the restriction of the play π on Eve's states.

Proposition 5.4.1. *Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$ iff $\mathcal{L}_\alpha(\mathcal{T}_\mathcal{G}) \neq \emptyset$.*

Note that for different particular winning conditions, we may use gadgets to add more information on states of the game $\mathcal{G}_\mathcal{T}$ in order to check the satisfaction of the winning conditions Θ of the processes and therefore slightly modify the game. For example, in the case of Safety conditions, we may need a set of processes that already lost (reached an unsafe state) and ask that the deviation is made before losing. More details will be given later, when will be studied the complexities for particular objectives.

5.4.2 Upper Bounds for NCRSP

The game $\mathcal{G}_\mathcal{T}$ has linear size in the size of $\mathcal{T}_\mathcal{M}$. A precise analysis of the time complexity of solving $\mathcal{G}_\mathcal{T}$ gives upper bounds to the non-cooperative rational synthesis problem.

For safety, reachability, Büchi and coBüchi winning objectives, we exploit the monotonicity of the sets W and D (the fact that only a polynomial number in k of different sets W and D can be met along a play). We show that if Eve can win the game $\mathcal{G}_\mathcal{T}$, then she can win in a polynomial number of steps (in the size of the interaction model $\mathcal{M}$), in the sense that she wins if and only if she can enforce, in a polynomial number of steps, to visit a state q she has already visited and which forms a *good cycle*. The notion of good cycle depends on the winning condition of $\mathcal{G}_\mathcal{T}$. In other words, $\mathcal{G}_\mathcal{T}$ reduces to a finite duration game with a polynomial number of steps. This kind of reduction is known as *first-cycle game* in the literature [8]. This game is not constructed explicitly, but

solved on-the-fly by a PTIME alternating algorithm. This gives a PSPACE upper-bound for NCRSP.

For Muller conditions, the polynomial reduction to first-cycle game doesn't work. Therefore, we transform the game $\mathcal{G}_\mathcal{T}$ into a two-player zero-sum parity game with an exponential number of states but a polynomial number of priorities, which can be solved in EXPTIME (in the size of $\mathcal{M}$). This reduction is based on the *Last Appearance Record (LAR)* [43, 94], which allows us to identify states in V appearing infinitely often. More details on the exact complexity for each type of winning condition are given in the following.

Safety Objectives

Let us first consider the rational synthesis problem with safety objectives $(\mathsf{Safe}(S_i))_{i\in\Omega}$ for processes in $\mathcal{M}$. We show how to verify Eve's winning condition in the game $\mathcal{G}_\mathcal{T}$ constructed in the general case. In fact, the difficulty is to verify the winning condition $(\mathsf{Safe}(S_i))_{i\in\Omega}$ of the processes. To do this, we keep an extra set of processes $I \subseteq \Omega$ with the states of $\mathcal{G}_\mathcal{T}$ with the following semantics. Let us take some history $h = (q_1, I_1)(Y_1, I_1)(q_2, I_2)(Y_2, I_2)...(q_l, I_l)$ with $q_j \in Q$ and $Y_j \in \{\mathsf{Range}(f) \mid \exists q \in Q, \ell \in \Lambda, f \in \delta(q, \ell)\}$ for all $1 \leq j \leq l$. If $i \in I_l$, then Process i lost the play by reaching an unsafe state along h, i.e., there is a position $s \leq l$ s.t. $q_s|_V \notin S_i$.

We get a new game $\mathcal{G}'_\mathcal{T}$ that annotates plays in $\mathcal{G}_\mathcal{T}$ with sets I. Initially, $I = \varnothing$ and it is updated as follows. If a process i belongs to I, then $i \in I$ also in all successor nodes. Otherwise, whenever the game $\mathcal{G}_\mathcal{T}$ goes in a state $q = (W, D, v)$ such that $v \notin S_i$ for some $i \in \Omega$, then $i \in I$ in the associated set I to q. Then, if eventually there is a process $i \in W \cap I$, the only next state of the game is " $\perp$ "(losing state for Eve). The last situation appears when it is made a wrong guess for a good deviation for some process. Then, if the play never go to the node " $\perp$ " we are sure that all the processes from the set W win. We do not need to keep the information about the processes in I if a state in $\{\perp\} \cup (\{\top\} \times V)$ is reached.

Formally, the states of the game $\mathcal{G}'_\mathcal{T}$ has the sets of states $V'_E = (Q \setminus (\{\perp\} \cup (\{\top\} \times V))) \times 2^\Omega \cup \{\perp\} \cup (\{\top\} \times V)$ and $V'_A = \{\mathsf{Range}(f) \mid \exists q \in Q, l \in \Sigma, f \in \delta(q, l)\} \times (2^\Omega \cup \{\{\perp\}\})$ controlled by Eve and Adam respectively, the initial state is $(q_0, \varnothing)$ and the transition relation E'' deterministically updates the information I along plays as follows:

- $((q, I), (Y, I)) \in E''$ if $(q, Y) \in E'$
- $((Y, I), (q, I')) \in E''$ if $q \neq \perp$ and $(Y, q) \in E'$ and $I' = I \sup\{i \in \Omega \mid q|_V \notin S_i\}$
- $((Y, I), \perp) \in E''$ if $(Y, \perp) \in E'$ and
- $(\perp, \{\perp\}) \in E''$ and $(\{\perp\}, \perp) \in E''$

Note that the set I is also monotonous along a play. Therefore, Eve's winning condition simplifies to a Büchi condition. $\Theta = \mathsf{Buchi}(F^S)$ where

$$F^S = (\{\top\} \times V) \cup \{(W, D, I, v) \in Q \mid 0 \notin I\} \cup \{(W, D, I, v) \in Q \mid D \cap I \neq \varnothing\}$$

Intuitively, the first set corresponds to the branches of the tree t_{σ_0} that do not correspond to executions in $\mathcal{M}$ compatible with the strategy σ_0. For the executions compatible to σ_0, the Büchi condition asks that Process 0 never belongs to the set I (therefore wins) or there is a Process i for which was guessed a good deviation but loses in the current play ($i \in D \cap I$).

Now, having simplified the winning condition for Eve, we can define a finite duration game $\mathcal{G}_{\mathcal{T}}^f$ that ends after a ploynomial time. Intuitively, the finite duration game is an unfolding of the game $\mathcal{G}'_{\mathcal{T}}$ up to the first cycle.

Finite Duration Game for Safety NCRSP Given the two-players zero-sum game $\mathcal{G}'_{\mathcal{T}}$, we define the finite duration two-player zero-sum game $\mathcal{G}_{\mathcal{T}}^f$ over the same game arena as $\mathcal{G}'_{\mathcal{T}}$ where each play ends after the first cycle on Eve's states. Then, a play $\pi = xqyq$ in $\mathcal{G}_{\mathcal{T}}^f$ is winning for Eve if either $q \in \{\top\} \times V$ or $q = (W, D, I, v)$ such that either $0 \notin I$ or $D \cap I \neq \varnothing$.

The definition is correct since once a state with $D \cap I \neq \varnothing$ is reached, because D and I are both monotone, the property holds on the remaining of the play. We prove that the two games are equivalent in the following:

Proposition 5.4.2. *Eve has a winning strategy in the game $\mathcal{G}'_{\mathcal{T}}$ iff she has a winning strategy in the first cycle game $\mathcal{G}_{\mathcal{T}}^f$.*

Proof. From right to left, if Eve has a winning strategy σ_E^f in $\mathcal{G}_{\mathcal{T}}^f$, for all σ_A^f a strategy for Adam, $out(\sigma_E^f, \sigma_A^f) = xqyq$ either is such that $q \in \{\top\} \times V$ or $q = (W, D, I, v)$ s.t. ($0 \notin I$ or $I \cap D \neq \varnothing$).

We define now the strategy σ_E of Eve in $\mathcal{G}'_{\mathcal{T}}$ as $\sigma_E(hq) = \sigma_E^f(h'q)$ s.t. h' is h from which are removed all cycles and prove that σ_E is winning for Eve in $\mathcal{G}'_{\mathcal{T}}$. Let π be a play compatible with σ_E. Then, by the definition of σ_E, we can decompose π in $\pi = \pi_1\pi_2\pi_3...$ such that each π_j is a suffix of a play π'_j compatible with σ_E^f in $\mathcal{G}_{\mathcal{T}}^f$. If all π_j on π satisfy $0 \notin I$ on the last state (resp. if it belongs to $\{\top\} \times V$), then also π will satisfy $\Box(0 \notin I)$(because I is monotone)(resp. $\pi\restriction_{V_E} \in Q^*(\{\top\} \times V)^\omega$) and then Eve wins. Otherwise, if there is j such that π_j ends in a state $q = (W, D, I, v)$ s.t. $I \cap D \neq \varnothing$, because of the monotonicity of I and D(Lemma 5.4.3), all the states of Eve in the continuations of the game will satisfy $I \cap D \neq \varnothing$ and then Eve wins.

Now, if there is no winning strategy for Eve in $\mathcal{G}_{\mathcal{T}}^f$, by the determinacy of perfect information games, there is a winning strategy σ_A^f for Adam such that $\forall \sigma_E^f$ of Eve, either $\mathsf{exec}(\mathcal{G}_{\mathcal{T}}^f, \langle \sigma_E^f, \sigma_A^f \rangle)$ contains " $\perp$ " (has a suffix in $(\{\perp\})^*$) or it does not contain " $\perp$ ", but $\mathsf{exec}(\mathcal{G}_{\mathcal{T}}^f, \langle \sigma_E^f, \sigma_A^f \rangle) = xqyq$ such that $q = (W, D, I, v)$ with $0 \in I$ and $I \cap D = \varnothing$.

Let σ_A be the strategy of Adam in $\mathcal{G}'_{\mathcal{T}}$ defined as $\sigma_A(hq) = \sigma^f_A(h'q)$ where h' is obtained from h by removing all cycles. We prove that σ_A is winning for Adam in the game $\mathcal{G}'_{\mathcal{T}}$.

Let π be a play compatible with σ_A. By definition of σ_A, we can decompose π in $\pi = \pi_1\pi_2\pi_3...$ such that each π_j is a suffix of a play π'_j compatible with σ^f_A in $\mathcal{G}^f_{\mathcal{T}}$. If all π_j are such that they don't contain " $\perp$ " but they end in a state $q = (W, D, I, v)$ such that $0 \in I$ and $I \cap D = \varnothing$, because of the monotonicity of I and D(Lemma 5.4.3), $0 \in I$ in all states of Eve in $\pi_{j'>j}$ on π and since $I \cap D = \varnothing$, it means that all the states of π will satisfy $I \cap D = \varnothing$ and therefore $I \cap D \neq \varnothing$ appears a finite number of times which means that Adam wins. Otherwise, if there is a π_j that ends in " $\perp$ ", then by definition of the game arena(induced by the transition relation in $\mathcal{T}_{\mathcal{M}}$) all $\pi_{j'>j}$ have Eve's states equal to " $\perp$ " which is again winning for Adam since they visit a finite number of times states in the set F^S. ■

The fact that the plays in $\mathcal{G}^f_{\mathcal{T}}$ are stopped after the first loop, implies that they have polynomial length as follows:

Lemma 5.4.4. *All the plays of the game $\mathcal{G}^f_{\mathcal{T}}$ are of polynomial length in the size of the interaction model $\mathcal{M}$.*

Proof. Since D and I are monotone, there are at most $|\Omega| + 1$ different values that they can take on a path of $\mathcal{G}_{\mathcal{T}}$. Also, in the set W we can have at most one addition and one removal for each process $i \in \Omega$ and hence $2 \cdot |\Omega| + 1$ different values for W. Therefore, along a play π there are at most $m = 1 + (2|\Omega| + 1) \cdot (|\Omega| + 1)^2 \cdot |V|$ different states. Then, since all the plays in $\mathcal{G}^f_{\mathcal{T}}$ stop after the first cycle, the length of each play is of at most $m + 1$ states since there is only one state that appears twice. Therefore, all plays in $\mathcal{G}^f_{\mathcal{T}}$ have polynomial length in Ω and V of the interaction model $\mathcal{M}$. ■

Thanks to Lemma 5.4.4 and Proposition 5.4.2, to decide the existence of a solution for the non-cooperative synthesis in interaction model $\mathcal{M}$ with Safety objectives is equivalent to solve the two-player zero-sum finite game $\mathcal{G}^f_{\mathcal{T}}$ that has all the plays of polynomial size in the size of $\mathcal{M}$. According to Theorem 3.1.3, this can be done in PSPACE using an alternating Turing machine running in PTIME.

Theorem 5.4.2. *The perfect information non-cooperative rational synthesis problem for safety objectives is in* PSPACE.

Reachability Objectives

For the reachability objectives $(\mathsf{Reach}(R_i))_{i\in\Omega}$, we have the same approach as in the case of safety objectives but with a new meaning for the newly introduced set. We keep a set $J \subseteq \Omega$ of agents that already reached their objectives in the past.

Initially, $J = \varnothing$ and it is updated as follows. Whenever a processes belongs to the set J, this remains true for the successor nodes. Otherwise, whenever the game goes in a

state $q = (W, D, v)$ such that $v \in R_i$ for some $i \in \Omega$, then i is added to the set J that is associated to the state q. Note that along a play, the set J is monotone since there are only additions of new processes.

The formal definition of the game is the same as in the case of Safety objectives, but with the later semantics for the introduced set of processes. Then, the Eve's winning condition translates to the Büchi objective $\Theta = \mathsf{Buchi}(F^R)$ where

$$F^R = (\{\top\} \times V) \cup \{(W, D, J, v) \in Q \mid W \subseteq J \text{ and } (0 \in J \text{ or } D \setminus J \neq \varnothing)\}$$

Since the sets J and D are monotonous and also W is establishing after at most $2k$ changes, a play π satisfies the winning condition Θ iff

$$\pi\restriction_{V_E} \vDash \Diamond \Box \big(\top \vee (W \subseteq J \wedge (0 \in J \vee D \setminus J \neq \varnothing))\big)$$

where $\pi\restriction_{V_E}$ stands for the restriction of the play to Eve's states and $\top$ holds in the states from the set $\{\top\} \times V$.

Given the later property on the paths that are winning for Eve, one can define the finite duration game $\mathcal{G}^f_{\mathcal{T}}$ as follows.

Finite Duration Game for Reachability NCRSP Given a two-player zero-sum game $\mathcal{G}'_{\mathcal{T}}$, we define the finite duration two-player zero-sum game $\mathcal{G}^f_{\mathcal{T}}$ over the same game arena as $\mathcal{G}'_{\mathcal{T}}$ where each play ends after the first cycle. Then, a play $\pi = xqyq$ in $\mathcal{G}^f_{\mathcal{T}}$ is winning for Eve if either $q \in \{\top\} \times V$ or $q = (W, D, J, v)$ such that $W \subseteq J$ and either $0 \in J$ or $D \setminus J \neq \varnothing$.

Proposition 5.4.3. *Eve has a winning strategy in the game $\mathcal{G}'_{\mathcal{T}}$ iff she has a winning strategy in the first cycle game $\mathcal{G}^f_{\mathcal{T}}$.*

Lemma 5.4.5. *All the plays of $\mathcal{G}^f_{\mathcal{T}}$ are of polynomial length in the size of the interaction model $\mathcal{M}$.*

The arguments for proving Proposition 5.4.3 and Lemma 5.4.5 are very similar as in the case of Safety games since the set L is replaced by the set J in the Reachability case having the same monotonic property. The only change appears in Proposition 5.4.3 in the evaluation of the accepting condition which slightly differs.

The next complexity result is thanks to Lemma 5.4.5 and Proposition 5.4.3 and the fact that the first cycle games can be solved in Pspace using an alternating Turing machine running in Ptime.

Theorem 5.4.3. *The perfect information non-cooperative rational synthesis problem for reachability objectives is in* Pspace.

Büchi Objectives

Consider now that the objective Θ_i of Process i is given as a Büchi set $F_i \subseteq V$ for all $0 \leq i \leq k$. A sequence $v_0 v_1 v_2 ... \in V^\omega$ belongs to Θ_i if and only if it satisfies the $LTL[\mathcal{M}]$ formula $\Box \Diamond F_i$ where F_i in this case is an abbreviation for $\bigvee_{v \in F_i} v$. Then, Eve's winning condition in the game $\mathcal{G}_\mathcal{T}$ is $\Theta = \{\pi = q_1 Y_1 q_2 Y_2 \cdots \in (V_E V_A)^\omega \mid \pi \restriction_{V_E} = q_1 q_2 \cdots \in \alpha\}$ where

$$\alpha = Q^*(\{\top\} \times V)^\omega \cup \Big\{\eta \in \mathrm{IRuns}(\mathcal{T}_\mathcal{M}) \cap \{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega \mid \Big(\eta|_V \vDash \Box \Diamond F_0 \vee \bigvee_{i=1}^{k} \big(\eta|_V \nvDash \Box \Diamond F_i \wedge i \in \lim_D(\eta)\big)\Big) \wedge \bigwedge_{i \in \lim_W(\eta)} \eta|_V \vDash \Box \Diamond F_i\Big\}$$

Outline of the construction Our goal is to simplify the winning condition in the game $\mathcal{G}_\mathcal{T}$ by slightly changing the game and finally to obtain a finite duration game. To do so, we pass through tree intermediate games. First, we introduce two counters that allows us to verify the two subconditions $\phi_W = \bigwedge_{i \in \lim_W(\eta)} \eta|_V \vDash \Box \Diamond F_i$ and $\phi_D = \bigvee_{i=1}^{k} \big(\eta|_V \nvDash \Box \Diamond F_i \wedge i \in \lim_D(\eta)\big)$. Then, we need another bit for further simplifications in the accepting condition that lead to a parity game with only 6 priorities. Finally, having the parity game, we can build a finite duration game that has all the plays of polynomial length.

First simplification: the two counters In order to check the satisfaction of α along the plays of the game $\mathcal{G}_\mathcal{T}$, we introduce two counters $c_W \in \Omega \cup \{-1\}$ and $c_D \in \Omega \cup \{-1\}$ in the states of the game. The two counters help to monitor the appearance of states in F_i that make the formulas $\phi_W = \bigwedge_{i \in \lim_W(\eta)} \eta|_V \vDash \Box \Diamond F_i$ and $\phi_D = \bigvee_{i=1}^{k} \big(\eta|_V \nvDash \Box \Diamond F_i \wedge i \in \lim_D(\eta)\big)$ true. The goal in using this counters is to write the formulas ϕ_W and ϕ_D as Büchi and respectively co-Büchi conditions. Intuitively, whenever c_W or c_D equal to i means that a state belonging to F_i is expected.

In order to correctly update the counters, we also need to keep in Adam's states the last previous state belonging to Eve. Note that by doing this, the size of the game remains exponential in the size of the initial interaction game $\mathcal{M}$ and the number of different states along a play remains polynomial in the size of the initial interaction model.

Formally, given the game $\mathcal{G}_\mathcal{T} = \langle V_E, V_A, E', q_0, \Theta \rangle$, we define the new game $\tilde{\mathcal{G}}_\mathcal{T} = \langle \tilde{V}_E, \tilde{V}_A, \tilde{E}, \tilde{q}_0, \tilde{\Theta} \rangle$ with $\tilde{q}_0 = (q_0, -1, -1)$, $\tilde{V}_E = V_E \times (\Omega \cup \{-1\}) \times (\Omega \cup \{-1\})$, $\tilde{V}_A = V_A \times V_E \times (\Omega \cup \{-1\}) \times (\Omega \cup \{-1\})$ and the transition relation $\tilde{E}$ is such that

- $((q_E, c_W, c_D), (q_A, q_E, c_W, c_D)) \in \tilde{E}$ iff $(q_E, q_A) \in E'$ for $q_E \in V_E$ and $q_A = Y \in V_A$
- $((q_A, q_E, c_W, c_D), (q'_E, c'_W, c'_D)) \in \tilde{E}$ iff $(q_A, q'_E) \in E'$ where $q_E \in V_E$ and $q_A = Y \in V_A$ and

$$c'_W = \begin{cases} -1 & \text{if } q_E = \bot \text{ or } q_E \in \{\top\} \times V \\ & \quad \text{or } q'_E = (W', D', v') \text{ s.t. } W' = \varnothing \\ min\{(c_W + l) \bmod k \in W' \mid l > 0\} & \text{if } q_E = (W, D, v), q'_E = (W', D', v') \\ & \text{s.t. } W' \neq \varnothing \wedge (v \in F_{c_W} \vee c_W \in W \setminus W' \vee W = \varnothing) \\ c_W & \text{otherwise} \end{cases}$$

and

$$c'_D = \begin{cases} -1 & \text{if } q_E = \bot \text{ or } q_E \in \{\top\} \times V \\ & \quad \text{or } q'_E = (W', D', v') \text{ s.t. } D' = \varnothing \\ min\{(c_D + l) \bmod k \in D' \mid l > 0\} & \text{if } q_E = (W, D, v), q'_E = (W', D', v') \\ & \text{s.t. } D' \neq \varnothing \text{ and } (v \in F_{c_D} \text{ or } D = \varnothing) \\ c_D & \text{otherwise} \end{cases}$$

Note that a play $\pi \in (\tilde{V}_E, \tilde{V}_A)^\omega$ is in $\mathsf{exec}(\tilde{\mathcal{G}}_\mathcal{T})$ if and only if π' obtained from π by projecting away c_W and c_D (and q_E from Adam's nodes) is in $\mathsf{exec}(\mathcal{G}_\mathcal{T})$. The role of the counters c_W and c_D is to wait for the first occurrence of a state such that $v \in F_{c_W}$ and $v \in F_{c_D}$ respectively as follows.

Initially, the information in c_W is first -1, meaning that there is nothing to verify yet ($W = \varnothing$). Then, the value changes in three cases. First, if in the current state $W = \varnothing$ and one process is added in W', the value of c'_w is the index of that process. The second case is when a final state for the Process c_w is reached, the final set of the process having the next index in W' is waited. Finally, it may be the case that the process c_w for which was expected the final state F_{c_W} is removed from the state W. In this case, we stop waiting for such a final state and pass to the next process. If none of the three conditions is satisfied, the value of c_w remains unchanged. The above definition of c_w correctly simulates the formula ϕ_W defined above by verifying that infinitely often is reached a state when $v \in F_{c_w}$ and c_w takes the smallest index in W. This is since the value of W eventually establishes to some limit (say W_{lim}) and then it only waits for final states of all processes in W_{lim}.

The information in c_D is updated similarly. Intuitively, the process having the index c_D is assumed to be the one that loses along he play. Then, when one of its final states is reached, the next process becomes the candidate to lose. Therefore, if some process in D (which eventually stabilizes) loses, the value of c_D remains unchanged. Then, the formula ϕ_D defined above holds in states where $v \in F_{c_D}$ and c_D is the smallest index in D appears only a finite number of times.

Let us identify as $q = (W, D, v) \vDash F_i$ the fact that $v \in F_i$. Also, for a play π in $\tilde{\mathcal{G}}_\mathcal{T}$, let $\pi{\restriction_{V_E}}$ stand for the restriction on Eve's states. Then, the following two lemmas prove that the above intuitions are correct.

Lemma 5.4.6. *For a play π in $\tilde{\mathcal{G}}_\mathcal{T}$, if $\pi{\restriction_{V_E}} \in \{q_0\}\{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega$ and $W_{lim} = lim_W(\pi{\restriction_{V_E}})$,*

$$\pi\restriction_{V_E} \vDash \bigwedge_{i \in W_{lim}} \Box \Diamond F_i \text{ iff } \pi\restriction_{V_E} \vDash \Box \Diamond H_w$$

$$\text{where } H_w = \{(W, D, v, c_W, c_D) \in \tilde{V}_E \mid W = \varnothing \vee (v \in F_{c_W} \wedge c_W = min\{i \in W\})\}$$

Proof. Note that since we considered $\pi\restriction_{V_E} \in \{q_0\}\{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega$, states in $\{\top\} \times V$ are not reached by π. Let us first treat the case $W_{lim} = \varnothing$, then $\pi \vDash true$ and also $\pi\restriction_{V_E} \vDash \Box \Diamond H_w$ since the only set W visited infinitely often is $\varnothing$.

Now, consider $W_{lim} \neq \varnothing$. If $\pi\restriction_{V_E} \vDash \bigwedge_{i \in W_{lim}} \Box \Diamond F_i$, then $inf(\pi|_{c_W}) = W_{lim}$ by the construction of the game $\tilde{\mathcal{G}}_\mathcal{T}$(whenever a final state is reached, the counter c_W is increased to the next process in W). Then, we see an infinite number of times final states of the "smallest" process in W, i.e., states in which $v \in F_{c_W}$ and $c_W = min\{i \in W\}$ and then $\pi\restriction_{V_E} \vDash \Box \Diamond H_w$.

In the other direction, if we see an infinite number of times states with $v \in F_{c_W}$ and $c_W = min\{i \in W\}$, because of the construction of the game, once we reach a final state with $v \in F_{c_W}$ the counter c_W is increased to the next process in W and so on. Therefore, between two states having $v \in F_{c_W}$ and $c_W = min\{i \in W\}$, the projection on the direction visits all the states F_i where $i \in W$. Then, since W is stabilizes to W_{lim} and since we visit an infinite number of times final states with $v \in F_{c_W}$ where $c_W = min\{i \in W_p\}$, it means that we visit infinitely often the final states of all processes in $\lim_W(\pi)$ and therefore $\pi\restriction_{V_E} \vDash \bigwedge_{i \in W_p} \Box \Diamond F_i$. ∎

Lemma 5.4.7. *For a play π in $\tilde{\mathcal{G}}_\mathcal{T}$, if $\pi\restriction_{V_E} \in \{q_0\}\{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega$ and $D_{lim} = lim_D(\pi\restriction_{V_E})$,*

$$\pi\restriction_{V_E} \vDash \bigvee_{i \in D_{lim}} \Diamond \Box \neg F_i \text{ iff } \pi\restriction_{V_E} \vDash \Diamond \Box \neg H_d$$

$$\text{where } H_d = \{(W, D, v, c_W, c_D) \in V_E \mid D = \varnothing \vee (v \in F_{c_D} \wedge c_D = min\{i \in D\})\}$$

Proof. The proof is similar to the proof of the previous Lemma. Indeed, if $D_{lim} = \varnothing$, then by the monotonicity of D, all the states along the play are such that $D = \varnothing$ and then both $\pi\restriction_{V_E} \vDash \bigvee_{i \in D_{lim}} \Diamond \Box \neg F_i$ and $\pi\restriction_{V_E} \vDash \Diamond \Box \neg H_d$ are false.

If $D_{lim} \neq \varnothing$, and $\pi\restriction_{V_E} \vDash \bigvee_{i \in D_{lim}} \Diamond \Box \neg F_i$, then there is a process that sees finitely often F_i. Therefore, from the construction of the game $\tilde{\mathcal{G}}_\mathcal{T}$, there is a process that blocks the cycling through all the values in D for the counter c_D and for that process, there are eventually seen only non-final states. That is, there are not seen infinitely often states in which $v \in F_{c_D}$ and $c_D = min\{i \in D\}$ and therefore $\pi\restriction_{V_E} \vDash \Diamond \Box \neg H_d$.

In the other direction, if $\pi\restriction_{V_E}$ visits finitely often states in which $v \in F_{c_D}$ and $c_D = min\{i \in D\}$, from the definition of the game $\tilde{\mathcal{G}}_\mathcal{T}$, either F_{c_D} is seen a finite number of times along π, or there is a $i \in D$, $i \neq min\{i \in D\}$, that blocks the cycling of c_D through all the values in D. Therefore, since D stabilizes to D_{lim}, there is a process $i \in D_{lim}$ such that $\pi\restriction_{V_E} \vDash \Diamond \Box \neg F_i$. ∎

Using Lemmas 5.4.6 and 5.4.7, if we note $H_\top = \{q \in \tilde{V}_E \mid q|_{V_E} \in \{\top\} \times V\}$ and $H_0 = \{(W, D, v, c_W, c_D) \in V_E \mid v \in F_0\}$ Eve's winning condition in the game $\tilde{\mathcal{G}}_\mathcal{T}$ is rewritten as

$$\tilde{\Theta} = \{\pi \in (\tilde{V}_E\tilde{V}_A)^\omega \mid \pi\restriction_{V_E} \vDash \Diamond\Box H_\top \vee \big((\Box\Diamond H_0 \vee \Diamond\Box\neg H_d) \wedge \Box\Diamond H_w\big)\}$$

Note that by asking to see infinitely often H_w, there are also avoided the states containing $q = \bot$. Also, the above formula $\Diamond\Box H_\top \vee \big((\Box\Diamond H_0 \vee \Diamond\Box\neg H_d) \wedge \Box\Diamond H_w\big)$ is equivalent to $\Diamond\Box H_\top \vee (\Box\Diamond H_0 \wedge \Box\Diamond H_w) \vee (\Diamond\Box\neg H_d \wedge \Box\Diamond H_w)$, which leads to the following.

Second simplification: An extra bit Now, to be able to check if a path satisfies $\Box\Diamond H_0 \wedge \Box\Diamond H_w$, we need to introduce a counter $b \in \{0, 1\}$ in the states of the game $\tilde{\mathcal{G}}_\mathcal{T}$ as follows.

Given the game $\tilde{\mathcal{G}}_\mathcal{T}$ from above, we define a game $\hat{\mathcal{G}}_\mathcal{T} = (\hat{V}_E = \tilde{V}_E \times \{0, 1\}, \hat{V}_A = \tilde{V}_A \times \{0, 1\}, (\tilde{q}_0, 0), \hat{E}, \hat{\Theta})$ where

- $((q, b), (q', b')) \in \hat{E}$ iff $(q, q') \in \tilde{E}$ and $b' = \begin{cases} 1 - b & \text{if } q \in H_w \cup H_0 \\ b & \text{otherwise} \end{cases}$
- $\hat{\Theta} = \{\pi \in (\hat{V}_E\hat{V}_A)^\omega \mid \pi\restriction_{V_E} \vDash \Diamond\Box H'_\top \vee \Box\Diamond H'_0 \vee (\Diamond\Box\neg H'_d \wedge \Box\Diamond H'_w)\}$ where $H'_\top = H_\top \times \{0, 1\}$, $H'_0 = H_0 \times \{0\}$, $H'_d = H_d \times \{0, 1\}$ and $H'_w = H_w \times \{0, 1\}$.

The intuition is similar as in the case of counter c_w. The bit b waits alternatively for H_0 and H_w to appear. Note that the updates of the counter b are deterministic. Therefore, for each path π in $\tilde{\mathcal{G}}_\mathcal{T}$, there is an unique corresponding path π' in $\tilde{\mathcal{G}}_\mathcal{T}$ such that by projecting away the counter b from π', we obtain the path π.

The following lemma says the fact that the two games $\tilde{\mathcal{G}}_\mathcal{T}$ and $\hat{\mathcal{G}}_\mathcal{T}$ are equivalent. That is, verifying the fact that $\Box\Diamond H_0 \wedge \Box\Diamond H_w$ holds, is equivalent to verifying that the set $H_0 \times \{0\}$ is visited infinitely often.

Lemma 5.4.8. *Let $\pi' \in \mathsf{exec}(\hat{\mathcal{G}}_\mathcal{T})$ and $\pi \in \mathsf{exec}(\tilde{\mathcal{G}}_\mathcal{T})$ obtained from π' by projecting away the counter b. Then,*

$$\pi\restriction_{V_E} \vDash \Diamond\Box H_\top \vee \big((\Box\Diamond H_0 \vee \Diamond\Box\neg H_d) \wedge \Box\Diamond H_w\big) \text{ iff } \pi'\restriction_{V_E} \vDash \Diamond\Box H'_\top \vee \Box\Diamond H'_0 \vee (\Diamond\Box\neg H'_d \wedge \Box\Diamond H'_w)$$

Reduction to Parity condition Further, we express Eve's winning condition $\hat{\Theta} = \{\pi \in \mathsf{Plays}(\hat{\mathcal{G}}_\mathcal{T}) \mid \pi\restriction_{V_E} \vDash \Diamond\Box H'_\top \vee \Box\Diamond H'_0 \vee (\Diamond\Box\neg H'_d \wedge \Box\Diamond H'_w)\}$ using a parity condition where the priority function pr defined as follows:

$$pr(q \in \hat{V}_E) = \begin{cases} 0 & \text{if } q \notin H'_\top \wedge q \in H'_0 \\ 1 & \text{if } q \notin H'_\top \wedge q \notin H'_0 \wedge q \in H'_d \\ 2 & \text{if } q \notin H'_\top \wedge q \notin H'_0 \wedge q \notin H'_d \wedge q \in H'_w \\ 3 & \text{if } q \notin H'_\top \wedge q \notin H'_0 \wedge q \notin H'_d \wedge q \notin H'_w \\ 4 & \text{if } q \in H'_\top \end{cases}$$

For all states belonging to Adam, we set the priority $pr(q \in \hat{V_A}) = 6$, so they have no influence and only the sequence of states corresponding to the automaton $\mathcal{T}_{\mathcal{M}}$ are verified. The following lemma proves that the reduction is correct.

Lemma 5.4.9. *Let $\pi \in \mathsf{exec}(\hat{\mathcal{G}}_{\mathcal{T}})$. Then*

$$\pi\restriction_{V_E} \vDash \Diamond\Box H'_{\top} \vee \Box\Diamond H'_0 \vee (\Diamond\Box \neg H'_d \wedge \Box\Diamond H'_w) \text{ iff } min\{pr(q)) \mid q \in inf(\pi)\} \text{ is even}$$

Finite Duration Game for Büchi NCRSP Now, given a two-player zero-sum parity game $\hat{\mathcal{G}}_{\mathcal{T}}$ with the priority function pr defined above, we define the first cycle two-player zero-sum game $\mathcal{G}^f_{\mathcal{T}}$ over the same game arena as $\hat{\mathcal{G}}_{\mathcal{T}}$ where each play ends after the first cycle. Then, a play $\pi = xqyq$ in $\mathcal{G}^f_{\mathcal{T}}$ is winning for Eve iff $min\{pr(yq[j]) \mid 0 \leq j < |yq|\}$ is even.

Proposition 5.4.4. *Eve has a winning strategy in the game $\hat{\mathcal{G}}_{\mathcal{T}}$ iff she has a winning strategy in the first cycle game $\mathcal{G}^f_{\mathcal{T}}$.*

The proof for Proposition 5.4.4 is similar to the ones in the case of Safety. The difference in this case is that the smallest priority on each loop has to be even when Eve wins.

Lemma 5.4.10. *All the plays in $\mathcal{G}^f_{\mathcal{T}}$ are of polynomial length in the size of the interaction model $\mathcal{M}$.*

Proof. It follows from the monotonicity of D and quasi-monotonicity of W and the fact that $1 \leq c_W, c_D \leq k$ and $b \in \{0, 1\}$. ∎

The following theorem follows from Proposition 5.4.4 and Lemma 5.4.10 and the fact that the finite duration game $\mathcal{G}^f_{\mathcal{T}}$ can be solved in PSpace using an alternating Turing machine running in PTime.

Theorem 5.4.4. *The perfect information non-cooperative rational synthesis problem for Büchi objectives is in* PSpace.

coBüchi Objectives

In the case when the objectives of the processes are given as coBüchi conditions $F_i \subseteq V$, Eve's winning condition in the game $\mathcal{G}_{\mathcal{T}}$ constructed in Section 5.4.1 is $\Theta = \{\pi = q_1 Y_1 q_2 Y_2 \cdots \in (V_E V_A)^\omega \mid q_1 q_2 \cdots \in \alpha\}$ where

$$\alpha = Q^*(\{\top\} \times V)^\omega \cup \Big\{\eta \in \mathrm{IRuns}(\mathcal{T}_{\mathcal{M}}) \cap \{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega \mid \Big(\eta|_V \vDash \Diamond\Box \neg F_0 \vee \bigvee_{i=1}^{k} \big(\eta|_v \nvDash \Diamond\Box \neg F_i \wedge i \in \lim_D(\eta)\big)\Big) \wedge \bigwedge_{i \in \lim_W(\eta)} \eta|_V \vDash \Diamond\Box \neg F_i\Big)\Big\}$$

As in the case of Büchi objectives, the goal is to obtain a parity game with 5 priorities and then to reduce to a finite duration game. This is simply done by rewriting the winning condition as in the following two lemmas. We say that a state $q = (W, D, v) \vDash F_i$ if $v \in F_i$.

Lemma 5.4.11. *For a play π in $\mathcal{G}_\mathcal{T}$, if $\pi\restriction_{V_E} \in \{q_0\}\{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega$ and $W_{lim} = lim_W(\pi\restriction_{V_E})$, then*

$$\pi\restriction_{V_E} \vDash \bigwedge_{i \in W_{lim}} \Diamond \Box \neg F_i \text{ iff } \pi\restriction_{V_E} \vDash \Diamond \Box \neg H_w$$
$$\text{where } H_w = \{(W, D, v) \in V_E \mid v \in \bigcup_{i \in W} F_i\}.$$

Proof. This holds because all Eve's states that appear an infinite number of times along π have $W = W_{lim}$ (W stabilizes along a play) and visiting a finite number of sets a finite number of time is equivalent to visiting their union a finite number of times. ■

Lemma 5.4.12. *For a play π in $\mathcal{G}_\mathcal{T}$, if $\pi\restriction_{V_E} \in \{q_0\}\{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega$ and $D_{lim} = lim_D(\pi\restriction_{V_E})$, then*

$$\pi\restriction_{V_E} \vDash \bigvee_{i \in D_{lim}} \Box \Diamond F_i \text{ iff } \pi\restriction_{V_E} \vDash \Box \Diamond H_d$$
$$\text{where } H_d = \{(W, D, v) \in V_E \mid v \in \bigcup_{i \in D} F_i\}.$$

Proof. This holds because all the states that appear an infinite number of times along π have $D = D_{lim}$ (D stabilizes along a play) and visiting one set among $F_1, ..., F_r$ an infinite number of times is equivalent to visiting their union an infinite number of times. ■

Let us now define $H_0 = \{(W, D, v) \in V_E \mid v \in F_0\}$ and $H_\top = \{q \in V_E \mid q \in \{\top\} \times V\}$. Then, using Lemmas 5.4.11 and 5.4.12, we get that Eve's winning condition is equivalent to

$$\Theta = \{\pi \in (V_E V_A)^\omega \mid \pi\restriction_{V_E} \vDash \Diamond \Box H_\top \vee \big((\Diamond \Box \neg H_0 \vee \Box \Diamond H_d) \wedge \Diamond \Box \neg H_w\big)\}$$

Let $I = H_0 \cup H_w$. Then the above formula is equivalent to $\Diamond \Box H_\top \vee \Diamond \Box \neg I \vee (\Box \Diamond H_d \wedge \Diamond \Box \neg H_w)$. Further, from the construction, a play cannot alternate states in $H_\top$ and I (once in $H_\top$, all the future states are in the same set). Therefore, we can define the set $J = H_\top \cup (V_E \setminus I)$ and equivalently write $\Diamond \Box J$ instead of $\Diamond \Box H_\top \vee \Diamond \Box \neg I$.

Note that the condition Θ restricts only the sequence of Eve's states. Therefore, when transforming to a parity condition, we set the priority of Adam's states even and high enough to have not influence.

Reduction to Parity Condition Given the two player game $\mathcal{G}_\mathcal{T}$, we define Eve's winning condition as a parity condition with the priority function $pr : (V_E \cup V_A) \to \{1, .., 6\}$ with

$$pr(q \in V_E) = \begin{cases} 1 & \text{if } q \notin J \wedge q \in H_w \\ 2 & \text{if } q \notin J \wedge q \in H_d \wedge q \notin H_w \\ 3 & \text{if } q \notin J \wedge q \notin H_d \wedge q \notin H_w \\ 4 & \text{if } q \in J \end{cases}$$

For states belonging to Adam, we just put priority $pr(q \in V_A) = 6$, so that they have no influence.

The following lemma is easy to prove by considering all the cases in the definition of the priority function.

Lemma 5.4.13. *Let $\pi \in \mathsf{exec}(\mathcal{G}_\mathcal{T})$. Then,*

$$\pi\restriction_{V_E} \models \Diamond\Box H_\top \vee \left((\Diamond\Box \neg H_0 \vee \Box\Diamond H_d) \wedge \Diamond\Box \neg H_w\right) \textit{ iff } min\{pr(q) \mid q \in inf(\pi)\} \textit{ is even}$$

Now, having a parity objective for the game $\mathcal{G}_\mathcal{T}$, we define the finite duration game $\mathcal{G}^f_\mathcal{T}$ ans we did in the case of Buchi objectives for NCRSP. The plays of the finite duration game also have polynomial length in the size of the interaction model $\mathcal{M}$. Therefore, there is an alternating PTIME (PSPACE) algorithm to solve the NCRSP. We omit further proofs and definition since they are the same as in the previous section.

Theorem 5.4.5. *The perfect information non-cooperative rational synthesis problem for coBüchi objectives is in* PSPACE.

Muller Objectives

In the case of Muller objectives, we cannot apply the same technique as in the previous cases. This is because Muller condition asks that the set of states appearing infinitely often represents a valuation that makes the formula μ true. Then, the reduction to finite duration games does not work since two individual loops may satisfy the condition μ, but their alternation does not satisfy it. Therefore, we apply other techniques to solve the game $\mathcal{G}_\mathcal{T}$ as follows.

To establish the EXPTIME upper bound for the rest of the objectives we consider in this chapter, it suffices to establish it for Muller objectives. This is because Streett, Rabin and parity objectives can be polynomially expressed as Muller conditions.

Let us consider the Muller objectives $\mathsf{Muller}(\mu_i)$ for the $k+1$ processors in $\mathcal{M}$. Then, Eve's winning condition in the game $\mathcal{G}_\mathcal{T}$ is $\Theta = \{\pi = q_1Y_1q_2Y_2\cdots \in (V_EV_A)^\omega \mid \pi\restriction_{V_E} = q_1q_2\cdots \in \alpha\}$ where

$$\alpha = Q^*(\{\top\} \times V)^\omega \cup \Big\{\eta \in \mathrm{IRuns}(\mathcal{T}_\mathcal{M}) \cap \{q_0\}(2^\Omega \times 2^\Omega \times V)^\omega \mid \Big(\eta|_V \in \mathsf{Muller}(\mu_0) \vee \bigvee_{i=1}^{k} \big(\eta|_V \notin \mathsf{Muller}(\mu_i) \wedge i \in \lim_D(\eta)\big)\Big) \wedge \bigwedge_{i \in \lim_W(\eta)} \eta|_V \in \mathsf{Muller}(\mu_i)\Big)\Big\}$$

We transform $\mathcal{G}_\mathcal{T}$ into a two-player zero-sum parity game with an exponential number of states but a polynomial number of priorities, which can be solved in $\mathsf{Exptime}$ (in the size of $\mathcal{M}$). This reduction is based on the *Last Appearance Record (LAR)* [43, 94], which allows us to identify states in V appearing infinitely often.

Least Appearance Record (LAR) Let us first recall the LAR[43, 94] construction. For the given set of states V, we define the deterministic transition system LAR_V that records the most recent states in V that appeared along an execution. We let $P(V)$ the set of permutations of V, which we denote by words of length n over alphabet V such that each element of V appears exactly once. Then the deterministic finite automaton that records the last states appeared in a sequence of elements in V is defined as $LAR_V = \langle M = P(V) \times \{0, \ldots, |V|-1\}, (m_0, h_0), \rightarrow \rangle$ where M is the set of states, the initial state is such that $m_0 = v_1 \ldots v_n$ and $h_0 = 1$, and the deterministic transition relation is such that $(m, h) \xrightarrow{v} (x_1 x_2 v, |x_1|)$ with $m = x_1 v x_2$ for some $x_1, x_2 \in V^*$.

Let (m, h) be a state of LAR_V. Then, h is called the *hit*, representing the position from which the last state v is taken and moved to the back. Also, the states after position h on in m are the most recent states v seen along the path, called *recent states*. Then, let $\xi = v_0 v_1 v_2 ...$ be an infinite sequence of states in V. A path in LAR_V on ξ is a infinite sequence $r(\xi) = (m_0, h_0)(m_1, h_1)(m_2, h_2)...$ such that $(m_0, h_0) \in M$ and for all $j \geq 1$, $(m_{j-1}, h_{j-1}) \xrightarrow{v_{j-1}} (m_j, h_j)$. Let h_{min} be the smallest hit appearing infinitely often along $r(\xi)$. Then, the set of vertexes v in m situated after position h_{min} is always the same from some point on and is equal to $inf(\xi)$, i.e., the sequence of subsets $(\{m_i[l] \mid l \geq h_{min}\})_{i \geq 0}$ eventually stabilizes to $\inf(\xi)$.

Reduction to Parity Game We use the LAR construction to reduce the game $\mathcal{G}_\mathcal{T}$ (defined in Section 5.4.1) to the parity game $\tilde{\mathcal{G}}_\mathcal{T}$ obtained by taking the product of $\mathcal{G}_\mathcal{T}$ and LAR_V as follows.

Given the two-players zero-sum game $\mathcal{G}_\mathcal{T} = (V_\mathcal{T} = V_E \uplus V_A, q_0, E', \mathcal{O})$ and the deterministic transition system LAR_V defined as above, we define the parity game $\tilde{\mathcal{G}}_\mathcal{T} = (\tilde{V} = \tilde{V}_E \uplus \tilde{V}_A, \tilde{q}_0, \tilde{E}, pr)$ where $\tilde{V} = V_\mathcal{T} \times M$, $\tilde{q}_0 = (q'_0, (m_0, h_0))$ and the set E' is defined by

- $((q_E, (m, h)), (q_A, (m, h))) \in E'$ iff $(q_E, q_A) \in E$ with $q_E \in V_E$ and $q_A = Y \in V_A$
- $((q_A, (m, h)), (q_E, (m', h'))) \in E'$ iff $(q_A, q_E) \in E$ where $q_E \in V_E$ and $q_A = Y \in V_A$ and (m', h') is defined as follows:
  - $(m, h) \xrightarrow{q_E|_V} (m', h')$ if $q_E \notin \{\bot\} \cup (\{\top\} \times V)$
  - $(m', h') = (m, h)$ if $q_E \in \{\bot\} \cup (\{\top\} \times V)$

Finally, the priority function $pr : \tilde{V} \to \{0, ..., 2|V|+2\}$ is defined as follows: $pr(\bot, m, h) = 1$, $pr(q_\top, m, h) = 0$ for $q_\top \in \{\top\} \times V$ and

$$pr((W, D, v), (m, h)) = \begin{cases} 2h & \text{if } \forall i \in W \{m[l] \mid l \geq h\} \vDash \mu_i \text{ and} \\ & (\{m[l] \mid l \geq h\} \vDash \mu_0 \text{ or } \exists i \in D \text{ s.t. } \{m[l] \mid l \geq h\} \vDash \neg\mu_i) \\ 2h+1 & \text{otherwise} \end{cases}$$

For states whose first component belongs to Adam, we just put priority $2|V|+2$, so that they have no influence.

Intuitively, the priority function associates an even priority to the states having the most recent set of states $\{m[l] \mid l \geq h\}$ such that it first satisfies the winning condition of all agents in W and also either satisfies the Muller condition μ_0 of Process 0, or there is a process i that has a deviation, but loses. All the other states have off priorities.

Let π a play in $\mathcal{G}_\mathcal{T}$. According to the definition above, there is a unique play π' in $\tilde{\mathcal{G}}_\mathcal{T}$ such that by projecting away the LAR construction along π', we obtain the play π. Also, the LAR component changes only on states belonging to Eve which helps verifying the winning condition Θ. The following lemma proves the equivalence between the two games.

Lemma 5.4.14. *Let π a play in $\mathcal{G}_\mathcal{T}$ and the corresponding play π' in $\tilde{\mathcal{G}}_\mathcal{T}$. Then,*

$$\pi \in \Theta \text{ iff } \pi' \in \mathsf{Parity}(pr)$$

Proof. Let h_{min} be the smallest hit appearing infinitely often along π'. As remarked before, $\{m[l] \mid l \geq h_{min}\} = \inf(\pi' \restriction_{V_E}|_V) = \inf(\pi \restriction_{V_E}|_V)$.

Let $\pi \in \mathsf{Muller}(\mu_0)$. This is equivalent to $\{m[l] \mid l \geq h_{min}\} \vDash \mu_0$. If $D_{lim} = \lim_D(\pi \restriction_{V_E})$, the fact that $\exists i \in D_{lim}$ s.t. $\inf(\pi \restriction_{V_E}|_V) \notin \mathsf{Muller}(\mu_i)$, since $\pi \restriction_{V_E}|_D = \pi' \restriction_{\tilde{V}_E}|_D$ and $\{m[r] \mid r \geq h_{min}\} = \inf(\pi' \restriction_{\tilde{V}_E}|_V) = \inf(\pi \restriction_{V_E}|_V)$, is equivalent with $\{m[l] \mid l \geq h_{min}\} \nvDash \mu_i$.

Also, considering $W_{lim} = \lim_W(\pi \restriction_{V_E})$, the property that $\forall i \in W_{lim}, \pi \restriction_{V_E}|_V \in \mathsf{Muller}(\mu_i)$ translates to $\inf(\pi \restriction_{V_E}|_V) = \{m[l] \mid l \geq h_{min}\} \vDash \mu_i$.

From the above, $\pi \in \Theta$ if and only if the smallest priority appearing infinitely often when hitting h_{min} is $2h_{min}$ which is even and therefore $\pi' \in \mathsf{Parity}(pr)$. ∎

Theorem 5.4.6. *The perfect information non-cooperative rational synthesis problem with Muller objectives is in* Exptime.

Proof. The complexity comes from the fact that the game $\tilde{\mathcal{G}}_\mathcal{T}$ is a two-player Parity game with exponential number of states, but with polynomial number of priorities which can be solved in Exptime. This is because parity games can be solved in PTime in the number of states and exponential in the number of priorities [51, 80], so proving the theorem. ∎

5.4.3 Lower Bounds for NCRSP

In this section we provide some lower bounds to the complexity of the non-cooperative rational synthesis problem. More precisely, we prove that for each type of objective $\mathcal{X} \in \{\mathsf{Reach}, \mathsf{Safe}, \mathsf{Büchi}, \mathsf{coBüchi}, \mathsf{Street}, \mathsf{Rabin}, \mathsf{Parity}, \mathsf{Muller}\}$, a PSpace lower bound applies to the corresponding non-cooperative rational synthesis problem. The result is obtained by reduction from the quantified boolean formula (QBF) problem.

Theorem 5.4.7. *For each* $\mathcal{X} \in \{\mathsf{Reach}, \mathsf{Safe}, \mathsf{Buchi}, \mathsf{coBuchi}, \mathsf{Street}, \mathsf{Rabin}, \mathsf{Parity}, \mathsf{Muller}\}$*, the perfect information non-cooperative rational synthesis problem with objectives of type* $\mathcal{X}$ *is* PSPACE-H.

Proof. By reduction from QBF. Let $\psi = \exists x_1 \forall x_2 ... \exists x_m \gamma(x_1, x_2, ..., x_m)$ be a QBF in 3CNF with k clauses $C_1, C_2, ..., C_k$.

Given $\mathcal{X} \in \{\mathsf{Reach}, \mathsf{Safe}, \mathsf{Buchi}, \mathsf{coBuchi}, \mathsf{Street}, \mathsf{Rabin}, \mathsf{Parity}, \mathsf{Muller}\}$, we build an interaction model $\mathcal{M}_\psi$ with processes having objectives of type $\mathcal{X}$ so that ψ is true if, and only if, there is a solution to the non-cooperative rational synthesis problem in $\mathcal{M}_\psi$. The interaction model $\mathcal{M}_\psi$ involves $2m + 2$ agent $\Omega = \{A, B, P_{10}, P_{11}, P_{20}, P_{21}, \ldots, P_{m0}, P_{m1}\}$. Intuitively, Agent A (Process 0 to synthesize) controls the existential variables, while Agent B (first process of the environment) controls the universal ones. Figure 5.5 presents a sketch of the interaction model where the circle nodes are owned by Agent A, the diamond ones by Agent B, and the rectangular ones by Agents $P_{10}, P_{11} \ldots, P_{m0}, P_{m1}$ as specified below.

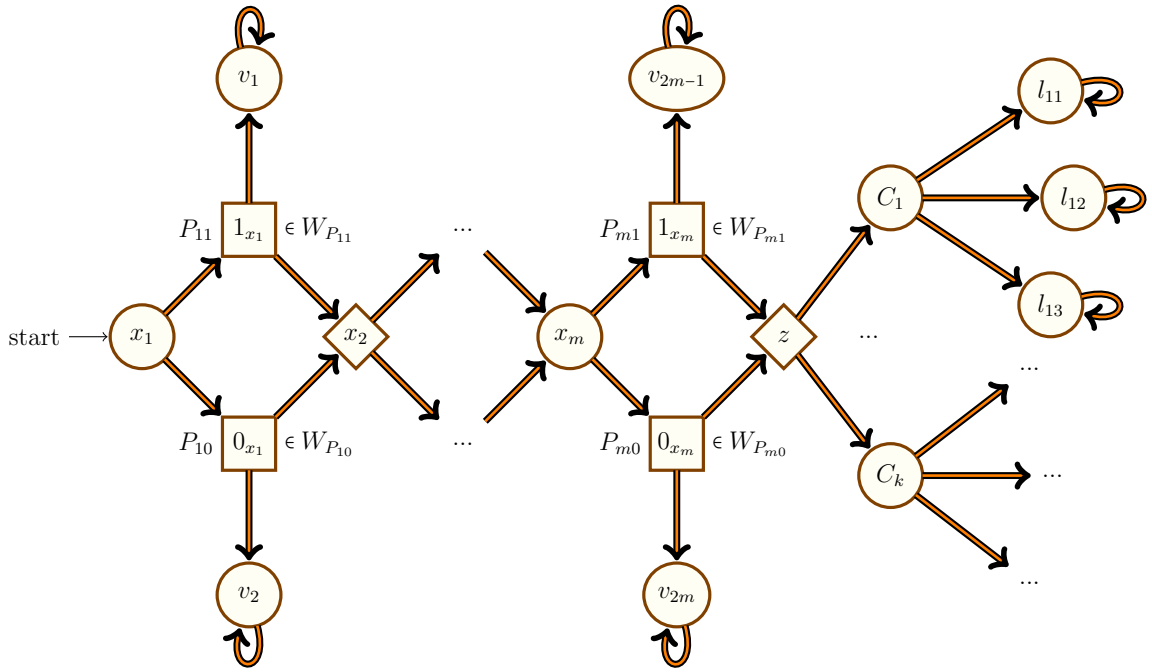


Figure 5.5: NCRSP lower bounds: Reduction from QBF

For each existential (resp. universal) variable x_i the interaction model $\mathcal{M}_\psi$ contains a node x_i controlled by the Agent A (resp. by Agent B). For each node $x_i, 1 \leq i < m$, it contains the edges $(x_i, 0_{x_i}), (x_i, 1_{x_i}), (0_{x_i}, x_{i+1}), (1_{x_i}, x_{i+1})$, where the vertex 0_{x_i} (resp. 1_{x_i}) intuitively represents the value $val(x_i) = 1$ (resp. $val(x_i) = 0$) for the variable x_i. For each $1 \leq i \leq m$, the value-node 1_{x_i} (resp. 0_{x_i}) is controlled by Agent P_{i1} (resp. P_{i0}) and has a further edge leading to the self-loop over the node v_{2i-1} (resp. v_{2i}), owned by the

Agent A. The value nodes $1_{x_m}, 0_{x_m}$ (for the last variable x_m) are then connected to a vertex z controlled by Agent B, where intuitively Agent B can choose a clause (i.e. an edge $(z, C_i), 1 \leq i \leq k$, out from z). Each clause-node C_i, controlled by the Agent A, has three outgoing edges toward the terminal nodes (with self-loops) l_{i1}, l_{i2}, l_{i3}, one for each literal in C_i.

Given the interaction model described above, the objectives of agents are properly designed so that the following conditions are satisfied:

(i) Given v_i, where $1 \leq i \leq 2m$, each lasso-path ending up into v_i is winning for each agent in the interaction model.

(ii) Given l_{ij}, where $1 \leq i \leq k$ and $1 \leq j \leq 3$, each lasso-path ending up into l_{ij} is winning for each agent in the interaction model except the process to synthesize (i.e. Agent A) and the Process P_{cb}, where:

$$(l_{ij} = x_c \wedge b = 1) \vee (l_{ij} = \neg x_c \wedge b = 0)$$

(iii) Agent B wins on all executions

Note that condition (i) implies that for each $1 \leq i \leq m$ and $b \in \{0, 1\}$, the vertex b_{x_i} belongs to the winning region $W_{P_{ib}}$ of Process P_{ib} since it is controlled by him and has can choose to go in a state winning for P_{ib}.

We claim that the formula ψ is true if, and only if, there is a solution for the non-cooperative rational synthesis problem in the interaction model $\mathcal{M}$ defined above.

Assume that ψ is true. Then, the existential player has a winning strategy in the QBF game associated to ψ. Therefore, Agent A can play in $\mathcal{M}_\psi$ according to such a strategy up to the node z, ensuring a configuration of variables such that all the clauses are satisfied. Therefore, from each clause node C_i, $1 \leq i \leq k$, Agent A can choose one literal l_{ij}, $1 \leq j \leq 3$, that makes true C_i and go to the corresponding node l_{ij}. Each execution ρ on $\mathcal{M}_\psi$ compatible with such a strategy for Agent A is either winning for him (since it does not reach z, i.e. is a lasso-path to some v_i, where $1 \leq i \leq 2m$) or it ends up into a node l_{ij} such that: either $l_{ij} = x_h$ and ρ passed trough 1_{x_h} or $l_{ij} = \neg x_h$ and ρ passed trough 0_{x_h} (i.e. either Agent P_{h1} or Agent P_{h0} doesn't play a Nash equilibrium since he loses but passed trough his winning region).

Otherwise, assume that ψ is false. Then, the universal player has a winning strategy σ in the QBF game associated to ψ. Consider a strategy profile (for the environment) where Agent B plays according to σ and each Process P_{ib}, for $1 \leq i \leq m, b \in \{0, 1\}$, plays to the next variable-node (or to z). Once in z, Agent B can choose a clause C_i that is false according to the instantiation of variables along the path followed so far. Therefore, for any choice of Agent A from C_i, the play will be losing for him and will be compatible with a 0-fixed Nash equilibrium. Indeed, let l_{ij} be the choice of Agent A from C_i. Then,

there is an index c such that $l_{ij} = x_c$ or $l_{ij} = \neg x_c$. In the first case, Agent P_{h1} loses but he could not avoid it (since the play did not pass trough 1_{x_h} and he never played) and each other agent in the environment wins. In the second case P_{h0} loses but he could not avoid it (since the play did not pass trough 0_{x_h} and he never played) and each other agent in the environment wins.

To conclude the proof, we just need to show that the objectives of type $\mathcal{X}$ of the agents in the interaction model $\mathcal{M}_\psi$ can be defined in order to satisfy the conditions (i) and (ii) above, for each $\mathcal{X} \in \{\mathsf{Safe}, \mathsf{Reach}, \mathsf{Buchi}, \mathsf{coBuchi}, \mathsf{Street}, \mathsf{Rabin}, \mathsf{Parity}, \mathsf{Muller}\}$, .

- $\mathcal{X} = \mathsf{Safe}$. The safety objective for each agent is defined as follows. $S_A = V \setminus \{(l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3\}$, $S_B = V$, and for each $c \in \{1, \ldots, m\}$: $S_{P_{c1}} = V \setminus \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = x_c\}$ and $S_{P_{c0}} = V \setminus \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = \neg x_c\}$

- $\mathcal{X} = \mathsf{Reach}$. The reachability objective for each agent is defined as follows: $R_A = \{v_i \mid 1 \leq i \leq 2m\}$, $R_B = V$, and for each $c \in \{1, \ldots, m\}$, the reachability objective of P_{c1} is $R_{P_{c1}} = \{v_i \mid 1 \leq i \leq 2m\} \cup \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} \neq x_c\}$ and the one for P_{c0} is $R_{P_{c0}} = \{v_i \mid 1 \leq i \leq 2m\} \cup \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} \neq \neg x_c\}$.

- $\mathcal{X} = \mathsf{Büchi}$. The Büchi objectives of the agents are defined as follows: $F_A = \{v_i \mid 1 \leq i \leq 2m\}$, $F_B = V$, and for each $c \in \{1, \ldots, m\}$: $F_{P_{c1}} = V \setminus \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = x_c\}$ and $F_{P_{c0}} = V \setminus \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = \neg x_c\}$

- $\mathcal{X} = \mathsf{coBüchi}$. The co-Büchi objectives of the agents are defined as follows. The coBüchi objective of Process A is $F_A = \{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3\}$. The coBüchi objective of Process B is $F_B = \varnothing$. For each $c \in \{1, \ldots, m\}$, the coBüchi objective of P_{c1} is $\{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = x_c\}$ and the objective of P_{c0} is $\{l_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3$ and $l_{ij} = \neg x_c\}$

- $\mathcal{X} \in \{\mathsf{parity}, \mathsf{Street}, \mathsf{Rabin}\}$. The PSPACE-hardness for the non-cooperative rational synthesis for parity, Streett and Rabin comes directly from the fact that we can easily express any Büchi condition as a parity, Streett or Rabin condition.

- $\mathcal{X} = \mathsf{Muller}$. The PSPACE-hardness for the non-cooperative strategy synthesis problem for Muller games follows from the fact that 0-sum two player Muller games are PSPACE-hard (we could clearly also define proper Muller objectives in $\mathcal{M}_\psi$ that satisfy conditions (i),(ii) and (iii)).

■

5.4.4 NCRSP with Fixed Number of Processes

We finally prove the upper bounds and the lower bounds to the complexity of k-fixed NCRSP, reported in the last column of Table 5.1.

Upper Bounds to k-fixed NCRSP

For $\mathcal{O} \in \{\mathsf{Safety}, \mathsf{Reachability}, \mathsf{Buchi}, \mathsf{coBuchi}\}$, a polynomial upper bound applies, as shown in the following Theorem.

Theorem 5.4.8. *The perfect information non-cooperative rational synthesis problem for a fixed number of processes with Safety, Reachability, Büchi and co-Büchi objectives can be solved in* PTIME.

Proof. In the case of a fixed number $k+1$ of processes, we obtain the polynomial size two-player zero-sum game $\mathcal{G}_{\mathcal{T}}$ and a fixed objective ϕ, where $\phi \in \{\varphi_s, \varphi_r, \varphi_c, \varphi_b\}$ are the formulas characterizing the winning objectives in the case of Safety, Reachability, Büchi and coBüchi games. Namely, one has to verify the formulas $\varphi_s = \Box \Diamond F^S$, $\varphi_r = \Diamond \Box F^R$, $\varphi_b = \Diamond\Box H_\top \vee \big((\Box\Diamond H_0 \vee \Diamond\Box\neg H_d) \wedge \Box\Diamond H_w\big)$, and $\varphi_c = \Diamond\Box H_\top \vee \big((\Diamond\Box\neg H_0 \vee \Box\Diamond H_d) \wedge \Diamond\Box\neg H_w\big)$, where the sets F^S, F^R, $H_\top$, H_0, H_d and H_w are defined in Section 5.4.2 for different types of objectives.

First, we can label in polynomial time the nodes of the game $\mathcal{G}_{\mathcal{T}}$ with atomic propositions atomic propositions F^S(for safety), F^R(for reachability), $H_\top$, H_0, H_d and H_w(for Buchi and coBuchi respectively defined according the considered condition). Each node is labeled with the atomic proposition corresponding to the set it belongs.

Then, since the formula ϕ is constant over the newly introduced atomic propositions, we get a constant size automaton $\mathcal{A}_\phi$ equivalent to the LTL formula ϕ and by taking the product $\mathcal{A}_\phi \times \mathcal{G}_{\mathcal{T}}$ we obtain a Büchi game that can be solved in polynomial time [23]. ■

The procedure outlined within the proof of Theorem 5.4.8 does not yield a polynomial upper bound for the remaining objectives considered in this thesis. However, we show that the non-cooperative rational synthesis problem with fixed number of processes having Muller objectives can be solved in PSPACE (cfr. Theorem 5.4.9). This entails a PSPACE upper bound also for NCRSP with respect to $\mathcal{X} \in \{\mathsf{Parity}, \mathsf{Streett}, \mathsf{Rabin}\}$ objectives and fixed number of processes.

Theorem 5.4.9. *The perfect information non-cooperative rational synthesis problem with a fixed number of processes having Muller objectives is in* PSPACE.

Proof. For a fixed number $k+1$ of processes , the game $\mathcal{G}_{\mathcal{T}}$ has size polynomial in the size of the interaction model $\mathcal{M}$. Moreover, Eve's objective in $\mathcal{G}_{\mathcal{T}}$ is equivalent to a Muller condition μ that is polynomial in the size of the game, as we show below. The complexity result follows then from the fact that $\mathcal{G}$ two-players zero-sum Muller games can be solved in PSPACE [48].

To conclude the proof, we show how to transform Eve's objective Θ (when each process in the interaction model $\mathcal{M}$ has an implicit Muller condition μ_i) into an unique equivalent implicit Muller objective μ. Note that we can ignore the states belonging to Adam and

define the objective μ only on Eve's states. This is correct since by doing so, we let the apparition of Adam's states unrestricted by the implicit Muller condition, which is desired.

First, for each tuple (W, D, v), we consider an atomic proposition $x_{W,D,v}$. Note that since the number of processes is fixed, the state space of the game $\mathcal{G}_{\mathcal{T}}$ is polynomial and so is the size of the set of newly introduced atomic propositions. Then, let consider the Muller condition μ_0 and $\eta \in \mathrm{IRuns}(\mathcal{T}_{\mathcal{M}}) \cap \{q_0\}(2^{\Omega} \times 2^{\Omega} \times V)^{\omega}$ such that $\eta|_V \in \mathsf{Muller}(\mu_0)$. Since the sets W and D stabilize along η, we can equivalently write that $\eta \in \mathsf{Muller}(\mu_0')$ where

$$\mu_0' = \mu_0[v \leftarrow \bigvee_{W,D} x_{W,D,v}]$$

is the boolean formula where each state v in is replaced by a disjunction for all W and D of $x_{W,D,v}$. Further, the condition $\psi_i = (\eta|_V \notin \mathsf{Muller}(\mu_i) \wedge i \in \lim_D(\eta))$ asks that the Process i belongs to D (therefore has a profitable deviation) from a position on, but his Muller condition μ_i is not satisfied. Using again the monotonicity of the sets W and D, we can rewrite the condition ψ_i as a Muller condition

$$\mu_i^D = \bigwedge_{\substack{D \subseteq \Omega \\ i \in D}} \Big(\big(\bigvee_{W,v} x_{W,D,v} \big) \rightarrow \neg \mu_i[v \leftarrow \bigvee_{W} x_{W,D,v}] \Big)$$

Intuitively, the formula says that for the set D that appears infinitely often (after stabilization) that contains i, the formula $\neg\mu_i$ holds for some W (that is also fixed after some steps). Similarly, we take the condition $\bigwedge_{i \in \lim_W(\eta)} \eta|_V \in \mathsf{Muller}(\mu_i)$ and write the equivalent Muller condition

$$\mu^W = \bigwedge_{W \subseteq \Omega} \Big(\big(\bigvee_{D,v} x_{W,D,v} \big) \rightarrow \bigwedge_{i \in W} \mu_i[v \leftarrow \bigvee_{D} x_{W,D,v}] \Big)$$

The formula says that for the set W that appears infinitely often, for all the players in this set, the Muller condition μ_i holds for some D.

Finally, the condition that $\eta \in Q^*(\{\top\} \times V)^{\omega}$ can be expressed using an atomic proposition $x_{\top}$ that is true only in the states belonging to $\{\top\} \times V$ as $\mu_{\top} = x_{\top}$ since once η goes outside $\{\top\} \times V$, it goes to $\bot$ and all the following states equal $\bot$. Therefore, the objective of Eve in the game $\mathcal{G}_{\mathcal{T}}$ is equivalent to the Muller condition $\Theta = \mathsf{Muller}(\mu)$ with

$$\mu = \mu_{\top} \vee \big((\mu_0' \vee \bigvee_{i=1}^{k} \mu_i^D) \wedge \mu^W \big)$$

■

Corollary 5.4.1. *The perfect information non-cooperative rational synthesis problem with a fixed number of processors having all Parity, Street or Rabin objectives is in* PSPACE.

Lower Bounds to k-fixed NCRSP

In this section we prove the lower bounds for the complexity of non-cooperative rational synthesis problem when the number of processors is fixed.

First, let us note that the reduction from QBF to general NCRSP provided in Theorem 5.4.7 does not apply to the case of a fixed number of processes, as it requires a number of components for the environment to be linear in the number of variables of the given QBF formula. Clearly, NCRSP with Muller conditions is PSPACE-hard by reduction from the corresponding two-player zero-sum games.

The lower bounds for Parity conditions reported in the last column of Table 5.1 have been obtained by reduction from the generalized two-player zero-sum parity games considered in [22], where the objective is a disjunction (dually, a conjunction) of parity conditions. In particular, we have proven that NCRSP is NP-hard (cfr. Theorem 5.4.10) on interaction models involving 3 players with parity objectives, and coNP-hard (cfr. Theorem 5.4.11) on 4-agents case.

Finally, as listed in Table 5.1, we provide a PSPACE lower bound also to Street and Rabin k-fixed NCRSP. This is done in two steps: First a reduction from QBF to zero-sum two players Muller games is provided (cfr. the proof of Theorem 5.4.12), similar to the one given in [48]. Then, the latter is reduced to a Street (resp. Rabin, cfr. Theorem 5.4.13) NCRSP with two players.

Parity Objectives

We first prove the NP-hardness and coNP-hardness of solving the non-cooperative rational synthesis problem with a fixed number of players by reduction from the generalized two-player zero-sum parity games [22], where the objective is a conjunction of two parity conditions.

Theorem 5.4.10. *The perfect information non-cooperative rational synthesis problem in an interaction model with 3 processes having Parity objectives is* NP*-hard.*

Proof. We prove the theorem by reduction from the two-player zero-sum game $\mathcal{G} = (V = V_A \uplus V_B, E, v_0, \Theta_A = parity(p_1) \wedge parity(p_2))$ where Player A(protagonist) has as objective an outcome satisfying a conjunction of two parity objectives p_1 and p_2. In [22] was proven that computing the winning region for the antagonist is NP-hard.

Without loss of generality, we consider that the game $\mathcal{G}$ is turn-based and that the initial state belongs to Player A. Intuitively, the interaction model $\mathcal{M}$ with 3 processes having parity objectives consists in a modified copy of $\mathcal{G}$ by duplicating the states of Player A and adding an extra sink state called $\ddot\smile$ where are players are happy with priority function equal to 0.

Formally, the interaction model is $\mathcal{M} = \langle \Omega = \{0,1,2\}, V' = V_0 \uplus V_1 \uplus V_2, E', v_0, (p'_i)_{i\in\Omega}\rangle$ where $V_0 = V_B \cup \{\ddot\smile\}$, $V_1 = \{v' \mid v \in V_A\}$, $V_2 = V_A$ and E' is defined as the smaller set such

that

- for all $(v_A, v_B) \in E$, $(v_A, v_B) \in E'$ and
- for all $(v_B, v_A) \in E$, we have $(v_B, v'_A) \in E'$ and $(v'_A, v_A) \in E'$ and
- for all $v \in V_1 \cup V_2$, $(v, \ddot\smile) \in E'$.

A sketch of the interaction model is depicted in Figure 5.6.

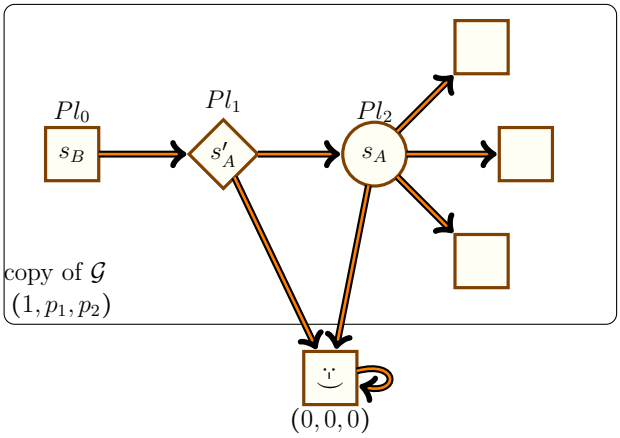


Figure 5.6: k-fixed Non-cooperative Parity: NP-hardness

Then, we define the parity functions for the tree processes as $p'_i : V' \to \mathbb{N}$ for $0 \leq i \leq 2$ such that

- $p'_0(v) = 1$ for all $v \neq \ddot\smile$ and $p'_0(\ddot\smile) = 0$;
- $p'_1(v) = p_1(v)$ for all $v \in V_B \cup V_A$, $p'_1(v') = p_1(v)$ for $v \in V_A$ and $p'_1(\ddot\smile) = 0$;
- $p'_2(v) = p_2(v)$ for all $v \in V_B \cup V_A$, $p'_2(v') = p_2(v)$ for $v \in V_A$ and $p'_2(\ddot\smile) = 0$;

We claim that Player A has a winning strategy in $\mathcal{G}$ if, and only if, there is no solution to the synthesis problem in the model $\mathcal{M}$. Indeed, if there is a strategy σ_A in $\mathcal{G}$ so that $parity(p_1) \wedge parity(p_2)$ holds on all $\rho \in \mathsf{exec}(\mathcal{G}, \sigma_A)$, in $\mathcal{M}$ Process 2 can play σ_2 defined as $\sigma_2(h) = \sigma_A(h')$ where h' is the restriction of h on the states in $V_A \cup V_B$. Then, for all σ_0, there is a 0-fixed Nash equilibrium $\langle \sigma_0, \sigma_1, \sigma_2 \rangle$ such that the execution compatible with it stays in the copy of $\mathcal{G}$. Therefore, Process 0 loses and there is no solution for the non-cooperative synthesis problem.

Otherwise, if there is no strategy σ_A to ensure $parity(p_1) \wedge parity(p_2)$ on all the paths compatible with it, it means that there is a strategy σ_B s.t. $\forall \rho \in \mathsf{exec}(\mathcal{G}, \sigma_B)$, $\rho \vDash \overline{parity(p_1)} \vee \overline{parity(p_2)}$ (one of the parity conditions p_1 and p_2 is not satisfied). That is, there is a strategy σ_0 for Process 0 s.t. at least one of the processes 1 and 2 wants to

deviate to $\ddot\smile$. Let us take $\langle \sigma_0, \sigma_1, \sigma_2 \rangle$ a strategy profile where $\sigma_1(v') = v$ and $\sigma_2(v) \neq \ddot\smile$. If $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \models \overline{parity(p_1)}$, this is not a 0-fixed Nash equilibrium because Process 1 loses and prefers to go to $\ddot\smile$. Otherwise, if $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \models \overline{parity(p_2)}$, Process 2 loses and prefers $\ddot\smile$ instead of staying in the copy of $\mathcal{G}$. Therefore, all the 0-fixed Nash equilibria are such that their outcome reaches $\ddot\smile$ and Process 0 wins. This means that σ_0 is a solution to the non-cooperative rational synthesis problem. ■

Theorem 5.4.11. *The perfect information non-cooperative rational synthesis problem in an interaction model with 4 processes having Parity objectives is coNP-hard.*

Proof. The proof is done by reducing from two-player zero-sum games $\mathcal{G}$ where the objective of the protagonist (Player A) is a conjunction of two parity objectives p_1 and p_2. For this games, in [22] is proven that the protagonist has a winning strategy from a given state is coNP-hard.

The interaction model with 4 processes is obtained from the game $\mathcal{G}$ by making two extra copies of each node of Player B (antagonist in $\mathcal{G}$) and adding two extra states $\ddot\frown_1$ and $\ddot\frown_2$. We define the interaction model $\mathcal{M} = \langle \Omega = \{0,1,2,3\}, V' = V_0 \uplus V_1 \uplus V_2 \uplus V_3, E', v_0, (p'_i)_{i\in\Omega} \rangle$ where $V_0 = V_A \cup \{\ddot\frown_1, \ddot\frown_2\}$, $V_1 = \{v'' \mid v \in V_B\}$, $V_2 = \{v' \mid v \in V_B\}$, $V_3 = V_B$ and E' is the smaller set such that

- for all $(v_A, v_B) \in E$, holds $\{(v_A, v''_B), (v''_B, v'_B), (v'_B, v_B)\} \subseteq E'$,
- for all $(v_B, v_A) \in E$, also $(v_B, v_A) \in E'$ and
- for all $v \in V_1$, $(v, \ddot\frown_1) \in E'$ and for all $v \in V_2$, $(v, \ddot\frown_2) \in E'$.

A sketch of the interaction model is depicted in Figure 5.7.

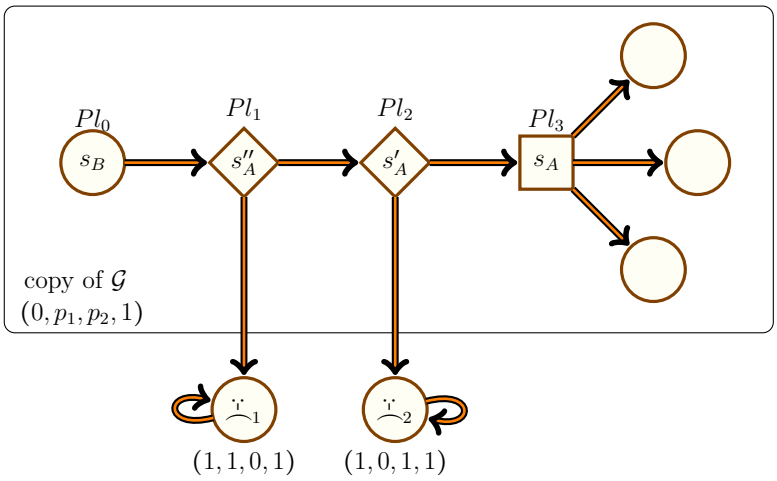


Figure 5.7: k-fixed Non-cooperative Parity: co-NP-hardness

Then, the parity functions for the four processes in $\mathcal{M}$ are defined as $p'_i : V' \to \mathbb{N}$ for $0 \leq i \leq 3$ such that

- $p'_0(v) = 0$ for all $v \notin \{\dddot{\frown}_1, \dddot{\frown}_2\}$ and $p'_0(\dddot{\frown}_1) = p'_0(\dddot{\frown}_2) = 1$
- $p'_1(v) = p_1(v)$ for all $v \in V_A$, $p'_1(v'') = p'_1(v') = p'_1(v) = p_1(v)$ for all $v \in V_B$, $p'_1(\dddot{\frown}_1) = 1$ and $p'_1(\dddot{\frown}_2) = 0$.
- $p'_2(v) = p_2(v)$ for all $v \in V_A$, $p'_2(v'') = p'_2(v') = p'_2(v) = p_2(v)$ for all $v \in V_B$, $p'_2(\dddot{\frown}_1) = 0$ and $p'_2(\dddot{\frown}_2) = 1$.
- $p'_3(v) = 1$ for all $v \in V'$.

Note that Process 3 loses on all executions in the interaction model $\mathcal{M}$ since its parity function p'_3 associates to all states the priority 1. Therefore, he acts completely antagonist when considering the non-cooperative rational synthesis problem, which is according to the behavior of Player B in the general parity game $\mathcal{G}$.

We claim that there is a winning strategy σ_A for player A in $\mathcal{G}$ if, and only if, there is a solution for NCRSP in $\mathcal{M}$. If there is a strategy σ_A to satisfy $parity(p_1) \wedge parity(p_2)$, it means that there is a strategy σ_0 for Player 0 defined as $\sigma_0(h) = \sigma_A(h')$ where h' is the restriction of h on the states in $V_A \cup V_E$ such that for any strategy σ_3 of Process 3, both Process 1 and Process 2 prefer to play in the copy of $\mathcal{G}$ since they win and in $\dddot{\frown}_i$ Process i loses, for $i \in \{1, 2\}$. Therefore, all 0-fixed Nash equilibria have as outputs executions in $\mathcal{G}$ and then Process 0 wins and σ_0 is a solution for the rational synthesis problem.

On the other way, if there is a solution for the non-cooperative rational synthesis problem in $\mathcal{M}$, all the 0-fixed Nash equilibria have outputs in the copy of $\mathcal{G}$ which means that $\langle \sigma_0, \sigma_1, \sigma_2, \sigma_3 \rangle$ where $\sigma_1(v'') = v'$ and $\sigma_2(v') = v$ are the only 0-fixed Nash equilibria. This means that both $parity(p'_1)$ and $parity(p'_2)$ are satisfied for any strategy σ_3. That is, there is a strategy σ_A defined as $\sigma_A(h') = \sigma_0(h)$ where h' is the restriction of h on the states in $V_A \cup V_E$(note there is only one such h by the definition of $\mathcal{M}$) in $\mathcal{G}$ s.t. for all σ_B, holds $\mathsf{exec}(\mathcal{G}, \langle \sigma_A, \sigma_B \rangle) \vDash parity(p_1) \wedge parity(p_2)$. ■

Rabin and Streett Objectives

In the following we prove the PSPACE-hardness of the non-cooperative rational synthesis problem in an interaction model $\mathcal{M}$ on which act two processes having both either Streett or Rabin objectives.

The proof is done in two steps. First, we reduce the validity of a quantified boolean formula to the problem of solving a two player zero-sum game where the protagonist's objective is a conjunction between an Streett and a Rabin objective. Then, we define an interaction model with two processes on which there is a solution for the non-cooperative rational synthesis problem if and only if the protagonist in the above game does not have a winning strategy.

The reduction to the two-player zero-sum game in the first part of the proof is common for the two conditions and it follows similar techniques as in [48] to proves

the PSPACE-hardness of the two-players zero-sum Muller games. Then, depending on the type (Streett or Rabin) of objectives that we want to obtain in the interaction model, we define the objectives of the processes accordingly. Let first provide the hardness result for Streett objectives.

Theorem 5.4.12. *The perfect information non-cooperative Streett synthesis problem in a 2-player interaction model is* PSPACE*-hard.*

Proof. We prove the theorem by reduction from QBF. Let $\phi = \exists x_k \forall x_{k-1} \dots \forall x_1 \exists x_0 \gamma$ be a quantified boolean formula in disjunctive normal form, where the quantifiers are strictly alternating. The proof will proceed as follows. First, we build a two-players zero-sum game $\mathcal{G}_\phi$ such that Player 0 (the protagonist) has a winning strategy in $\mathcal{G}_\phi$ if and only if ϕ is true. Then, we use $\mathcal{G}_\phi$ to build a non-cooperative Street strategy synthesis two-players interaction model $\mathcal{M}_\phi$, such that the protagonist wins if and only if ϕ is false.

Two-players zero-sum game. Let us first define the two-players zero-sum game $\mathcal{G}_\phi$. Let ϕ be a disjunction of the clauses $C_0, \dots, C_m$ over the literals $\{x_0, \neg x_0, \dots, x_k, \neg x_k\}$.

Intuitively, there is an initial state ϕ from which there are transitions to all nodes C_i controlled by Player 1 corresponding to the clauses in the formula ϕ. From each clause C_i, there are transitions to all the literals appearing in C_i and finally, from each literal there is a transition back to the initial state ϕ. Both the initial state ϕ and the states corresponding to the literals in ϕ are controlled by Player 0. A sketch of the game is depicted in Figure 5.8 where the round states are controlled by Player 0 and the square states belong to Player 1.

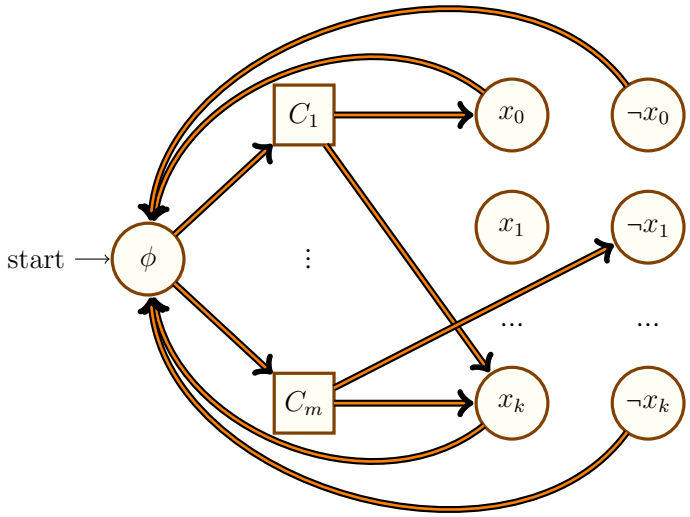


Figure 5.8: QBF to two-player zero-sum game

Formally, the two-players zero-sum Muller game $\mathcal{G}_\phi = \langle V = V_0 \uplus V_1, E, v_0, \Theta_0 \subseteq V^\omega \rangle$ is defined as follows:

- $V_0 = \{\phi\} \cup \{x, \neg x \mid x \text{ is a variable appearing in } \phi\}$
- $V_1 = \{C_0, \ldots, C_m\}$, the set of clauses in ϕ
- $v_0 = \phi$
- E is such that:
  - for each $0 \leq i \leq m$, $(\phi, C_i) \in E$
  - for each $C_i = \ell_0 \wedge \ell_1 \wedge \ell_2$, holds $(C_i, \ell_0) \in E$, $(C_i, \ell_1) \in E$, $(C_i, \ell_2) \in E$
  - for each $0 \leq i \leq k$, $(x_i, \phi) \in E$, $(\neg x_i, \phi) \in E$

In order to define the winning condition Θ_0, given a path $\rho \in V^\omega$, let $i(\rho)$ be the index $0 \leq i(\rho) \leq k$ such that:

- either $x_{i(\rho)}$ or $\neg x_{i(\rho)}$ is seen infinitely often on ρ and
- for all $i(\rho) < j \leq k$, both x_j and $\neg x_j$ are seen finitely often along ρ

Intuitively, if we refer to the set of literals $\{x_i, \neg x_i\}$ as literals of level i, then $i(\rho)$ is the index of the last level of literals (counting the levels from 0 to k) visited infinitely often in ρ. Note that $i(\rho)$ is well defined since, by definition of E, each infinite path of $\mathcal{G}_\phi$ contains at least one literal that repeats infinitely often.

The winning condition $\Theta_0 \subseteq V^\omega$ for Player 0 is defined by:

$$\begin{aligned}\Theta_0 = &\{\rho \mid i(\rho) \text{ is odd } \wedge \{x_{i(\rho)}, \neg x_{i(\rho)}\} \subseteq \inf(\rho)\} \cup \\ &\{\rho \mid i(\rho) \text{ is even } \wedge (x_{i(\rho)} \notin \inf(\rho) \vee \neg x_{i(\rho)} \notin \inf(\rho)\}\end{aligned}$$

where $\inf(\rho)$ is the set of nodes that appear infinitely often on the path ρ.

The winning condition expresses the fact that Player 0 wins the play π if and only if:

- either the index of the last level of literals visited infinitely often is odd (i.e. $i(\rho)$ is odd) and both $x_{i(\rho)}$ and $\neg x_{i(\rho)}$ are visited infinitely often, or
- the index of the last level of literals visited infinitely often is even (i.e. $i(\rho)$ is even), but only one literal in $\{x_{i(\rho)}, \neg x_{i(\rho)}\}$ appears infinitely often in ρ.

We now show that Θ_0 can be written as a combination of a Street and a Rabin condition, i.e. $\Theta_0 = S \wedge R$ where S (resp. R) is a Street (resp. Rabin) condition represented

as a conjunction (resp. disjunction) of pairs of sets of states. Given $0 \leq i \leq k$, denote by $L_{j>i}$ the set of literals $L_{j>i} = \{x_j, \neg x_j \mid j > i\}$. Then:

$$S = \bigwedge_{\text{i odd}} (\{x_i\}, \{\neg x_i\} \cup L_{j>i}) \wedge (\{\neg x_i\}, \{x_i\} \cup L_{j>i})$$
$$R = \bigvee_{\text{i odd}} (\{x_i, \neg x_i\}, L_{j>i}) \vee \bigvee_{\text{i even}} (\{x_i\}, \{\neg x_i\} \cup L_{j>i}) \vee (\{\neg x_i\}, \{x_i\} \cup L_{j>i})$$

The Streett condition S states that for each odd level i, if x_i (resp. $\neg x_i$) appears infinitely often along a path ρ, then either $i(\rho) > i$ (i.e. i is not the last level visited) or $i(\rho) = i$ (i.e. the last level visited is odd) and both literals at the odd level $i(\rho) = i$ are seen infinitely often on ρ.

The Rabin condition R instead states that either the last level visited is even and only one between $x_{i(\rho)}$ and $\neg x_{i(\rho)}$ is seen infinitely often, or otherwise the last level is odd (and the condition S takes care of its properties).

Given the above definition of the zero-sum Muller game $\mathcal{G}_\phi$, we are now ready to prove that $\phi = \exists x_k \forall x_{k-1} \dots \forall x_1 \exists x_0 \gamma$ is true if and only if Player 0 wins $\mathcal{G}_\phi$. In particular, we will proceed by induction on k. Note that if the existential quantified variable x_0 does not appear in ϕ, we can add the clause $x_0 \wedge \neg x_0$ without changing the truth value of ϕ.

Base cases: Assuming that ϕ is a closed formula, in the base case the formula ϕ is logically equivalent to one of the following forms. We also provide the winning strategy for Player A in the game $\mathcal{G}_\phi$.

1. $\phi = \exists x_0(x_0)$ or $\exists x_0(\neg x_0)$. In this case, the arena consists of four vertexes $\{\phi, C_0, x_0, \neg x_0\}$. If $\phi = \exists x_0(x_0)$, then $\neg x_0$ is isolated, otherwise x_0 is isolated. Therefore, $\mathcal{G}_\phi$ contains only one cycle winning for Player 0.

2. $\phi = \exists x_0(x_0 \vee \neg x_0)$. $\mathcal{G}_\phi$ consists of five vertexes $\{\phi, C_0, C_1, x_0, \neg x_0\}$. Player 0 wins by choosing always C_0 from ϕ.

3. $\phi = \exists x_0(x_0 \wedge \neg x_0)$. $\mathcal{G}_\phi$ consists of four vertexes $\{\phi, C_0, x_0, \neg x_0\}$. Player 0 can only play to $C_0 = x_0 \wedge \neg x_0$ from ϕ. Player 1 wins by choosing alternatively x_0 and $\neg x_0$ from C_0.

Inductive step: By inductive hypothesis, we know that if ϕ has $k-1$ quantifiers and is closed, than Player 0 has a winning strategy if and only if ϕ is true. To prove the inductive step for k quantifiers we use the following claim that shows how subgames correspond to restricted subformulas.

First, let us introduce some notation. Given $v \in V, U \subseteq V$ and $i \in \{0, 1\}$, we denote by $\mathsf{Avoid}_i(U, v)$ the subset of U from which Player i has a strategy to avoid vertex v without leaving U. Also, the formula $\gamma[x \to \mathsf{true}]$ (resp. $\gamma[x \to \mathsf{false}]$) stands for the formula γ where the variable x is replaced by true (resp. true).

Claim: If $\phi = \mathsf{Q}x.\gamma$ and $\gamma[x \to \mathsf{true}]$ does not simplify to either true or false, then $\mathsf{Avoid}_1(\mathsf{Avoid}_0(V, \neg x), x)$ induces a subgame of $\mathcal{G}_\phi$ that is isomorphic to $\mathcal{G}_{\phi[x\to\mathsf{true}]}$. Dually, if $\gamma[x \to \mathsf{false}]$ does not simplify to either true or false, then $\mathsf{Avoid}_1(\mathsf{Avoid}_0(V, x), \neg x)$ induces a subgame of $\mathcal{G}_\phi$ that is isomorphic to $\mathcal{G}_{\phi[x\to\mathsf{false}]}$.

Proof of Claim. The formula $\gamma[x \to \mathsf{true}]$ consists of the clauses of γ that do not contain $\neg x$, say $C_1, \ldots, C_p$ with all the occurrences of x replaced by true. The arena of the game $\mathcal{G}_{\gamma[x\to\mathsf{true}]}$ consists therefore of an initial vertex, one vertex for each clause $C_1, \ldots, C_p$ and one vertex for each literal different from x and $\neg x$. The edges are the same of $\mathcal{G}_\gamma$ restricted to the above set of vertexes. We show that the graph induced by $\mathsf{Avoid}_1(\mathsf{Avoid}_0(V, \neg x), x)$ is isomorphic to the arena of $\mathcal{G}_{\gamma[x\to\mathsf{true}]}$. Since we consider formulas in disjucntive normal form, the set of vertexes $U = \mathsf{Avoid}_0(V, \neg x)$ is given by V except the set Cl of clauses containing $\neg x$ and the vertex $\neg x$. Note that Cl of clauses is not empty since $\gamma[x \to \mathsf{true}]$ does not simplify to false. Now, the set of vertexes $W = \mathsf{Avoid}_1(U, x)$ is then obtained by removing from U the only vertex x because Player 1 has more than one choice from each clause since $\gamma[x \to \mathsf{true}]$ does not evaluate to true and therefore can chose another literal then x. Therefore, W precisely consists of the initial vertex, one node for each clause not containing $\neg x$ in γ and a node for each literal different from x and $\neg x$. Hence, the graph induced by $\mathsf{Avoid}_1(\mathsf{avoid}_0(V, \neg x), x)$ is isomorphic to $\mathcal{G}_{\gamma[x\to\mathsf{true}]}$. The proof of the case $\gamma[x \to \mathsf{false}]$ is symmetric.

Proof of inductive step. Given the above claim, we are now ready to deal with the inductive step. We consider two cases, depending on whether the variable x in $\phi = \mathsf{Q}x.\gamma$ is quantified universally or existentially.

1. $\phi = \exists x.\gamma$. If ϕ is true, then there is a value $v \in \{0, 1\}$ such that $\gamma[x \to v]$ is true. Assume $v = 1$ is such a value. Then, Player 0 plays in $\mathsf{Avoid}_0(V, \neg x)$ trying to reach infinitely often x. If Player 1 at some point prevents him to reach x (from that point of the game over) then the game gets restricted to $\mathsf{Avoid}_1(\mathsf{avoid}_0(V, \neg x), x)$ in which Player 0 has a strategy to win. The subcase where $v = 0$ is symmetric.

   If ϕ is false, one of $\gamma[x \to 0]$ and $\gamma[x \to 1]$ is false. If it is the case of $\gamma[x \to 0]$, Player 1 can use the following strategy to win. Indefinitely, alternatively try to reach first x (while avoiding $\neg x$), and then try to reach $\neg x$ (while avoiding x). If at any point the opponent prevents him to reach his current objective, the game gets restricted in $\mathsf{Avoid}_1(\mathsf{avoid}_0(V, \neg x), x)$ or $\mathsf{Avoid}_1(\mathsf{avoid}_0(V, x), \neg x)$ in which Player 1 has a winning strategy.

2. $\phi = \forall x.\gamma$. If ϕ is true, since x has an odd index, Player 0 can adopt the following strategy to win. He will try alternatively to reach x (while avoiding $\neg x$) and then reach $\neg x$ while avoiding x. If at any point of the game Player 1 prevents Player 0 to reach its target then the game gets restricted into $\mathsf{Avoid}_1(\mathsf{Avoid}_0(V, \neg x), x)$ or

$\mathsf{Avoid}_1(\mathsf{Avoid}_0(V, x), \neg x)$ where Player 0 has a winning strategy. The subcase where $\phi = \forall x.\gamma$ is false is symmetric to the subcase where $\phi = \exists x.\gamma$ is true seen above.

Note that, intuitively, Player 1 fixes the value of a variable each time he forbids Player 0 to reach from some point on of the play its target literal. Whenever the value of a variable (say x_i) is fixed by Player 1 in this way, the play proceeds into a inner layer of variables, i.e. into the arena of a subgame that contains only literals of levels less than i.

Reduction to NCRSP with Streett Objectives. Resuming, we have now proven that the QBF formula $\phi = Q_k x_k \dots \forall x_1 \exists x_0 \gamma$ (in DNF) is true if and only if Player 0 wins the zero-sum game $\mathcal{G}_\phi$, in which the objective $\Theta_0 = S \wedge R$ of Player 0 is a conjunction of a Street condition S and a Rabin condition R. Given $\mathcal{G}_\phi$, consider now the following interaction model $\mathcal{M}_\phi$

- the support graph of $\mathcal{M}_\phi$ is exactly the same of $\mathcal{G}_\phi$
- Player 1 is the process to synthesize
- the environment is composed by the only Player 0
- the objective of the Player 1 is the Street condition $\neg R$
- the objective of the only component in the environment is the Street condition S

We show that ϕ is false if and only if there is a solution to the non-cooperative strategy synthesis problem in $\mathcal{M}$. Note that since the process to synthesize is Process 1, the problem is to synthesize a strategy σ_1 such that for any strategy σ_0 of Player 0 (the environment), if $\langle \sigma_0, \sigma_1 \rangle$ is a 1-fixed Nash equilibrium, then $\mathrm{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1 \rangle) \vDash \neg R$.

Before proceeding in such a proof, note that $\neg R \to S$. In fact:

$$\neg R = \bigwedge_{\text{i even}} (\{x_i\}, \{\neg x_i\} \cup L_{j>i}) \wedge (\{\neg x_i\}, \{x_i\} \cup L_{j>i}) \wedge \bigwedge_{\text{i odd}} (\{x_i, \neg x_i\}, L_{j>i})$$

Hence $\neg R$ states that the last level visited infinitely often is even, that implies S.

Given the above observation, we proceed to prove that ϕ is false if and only if there is a solution to the non-cooperative strategy synthesis problem in $\mathcal{M}$ where Player 1 represents the process to synthesize.

There are two cases to consider. If ϕ is true, then the environment can ensure $S \wedge R$, i.e. he has a strategy to guarantee that he accomplishes his objective, while the process to synthesize (Player 1) does not.

In the other case, suppose that ϕ is false. Then the Player 1 has a strategy to ensure $\neg S \vee \neg R$. Since $\neg R \to S$, either the environment loses, or it holds $\neg R$ and both the players win. The environment has always the possibility to cooperate to establish $\neg R$ by always choosing the clause $x_0 \wedge \neg x_0$. Without lost of generality we can assume that the only clause containing x_0 in ϕ is $x_0 \wedge \neg x_0$. In fact, if this is not the case we can leadϕ to such a form by renaming each variable x_i to x_{i+2} and adding the clause $x_0 \wedge \neg x_0$ ■

Theorem 5.4.13. *The perfect information non-cooperative Rabin synthesis problem in a 2-player interaction model is* PSPACE*-hard.*

Proof. As in the case of Theorem 5.4.12 for Streett objectives, the proof is done by reduction from QBF. Let $\phi = \forall x_k \exists x_{k-1} \dots \forall x_1 \exists x_0 \gamma$ be a quantified boolean formula in disjunctive normal form, and consider the equivalent QBF:

$$\phi' = \forall x_k \exists x_{k-1} \dots \forall x_1 \exists x_0 \forall y_1 \exists y_0 ((y_1 \wedge \gamma) \vee (\neg y_1 \wedge \gamma))$$

where there are introduced two variables y_1 and y_0. Let ϕ'' be the formula obtained from ϕ' by first renaming each variable x_i, $i = 0 \dots k$, to x_{i+2}, and each variable y_j, $j \in \{0, 1\}$, to x_k, and then normalizing the resulting formula in DNF.

Let $\mathcal{G}_{\phi''}$ be the two-player zero-sum game such that Player 0 has a winning strategy in $\mathcal{G}_{\phi''}$ if and only if ϕ'' is true, built according to the procedure shown within the proof of Theorem 5.4.12. Given $\mathcal{G}_{\phi''}$, consider the following interaction model $\mathcal{M}$:

- the support graph of $\mathcal{M}$ is exactly the same of $\mathcal{G}_{\phi''}$
- Player 1 represents the process to be synthesized
- the environment is composed by the only Player 0
- the objective of Player 1 is the Rabin condition $\neg S$
- the objective of the only component in the environment is the Rabin condition R

We show that ϕ'' is false if and only if there is a solution to the non-cooperative strategy synthesi problem $\mathcal{G}^*_{\phi''}$.

There are two cases to consider: In the first case, assume that ϕ'' is true. Then, the environment can ensure $S \wedge R$ that is a $\{1\}$-fixed Nash equilibrium where Player 0 wins while the process to synthesize (Player 0) looses.

In the second case, suppose that ϕ'' is false. Then, the system has a strategy to ensure $\neg S \vee \neg R$. We claim that such a strategy is indeed a solution to the non-cooperative Rabin strategy synthesis problem in $\mathcal{M}$. Indeed, the environment (Player 0) can win if he cooperates with Player 1 to establish $\neg S$, i.e. if he cooperate to let the last level of variables visited to be odd. The environment can effectively force the last level visited to be odd by opposing to Player 1 a strategy that forbid him to reach its target literal, restricting the play within inner and inner layers (of literals), until the objective of Player 1 is to reach (only one) literal of level 1 (e.g. x_1). At that point, the environment simply let the system to pursue its objective by choosing only clauses with the literal x_1 (that appears in ϕ'' by construction). ∎

5.5 Conclusions

In this chapter, we have studied the complexity of rational synthesis in both the cooperative and non-cooperative settings, and depending on whether the number of players is fixed or not. For LTL objectives, the problem was proven by Kupferman et. al [38, 53] to be 2ExpTime-complete. Since the solution proposed for LTL objectives is based to reduction to satisfiability of Strategy Logic formulas, out goal was a fine understanding of the computational complexity of the rational synthesis problem and how to manipulate the underlying game algorithmically. Therefore, we considered restrictions for this problem for safety, reachability, Büchi, coBüchi, parity, Rabin, Streett and Muller objectves and rationality if the environment modeled as Nash equilibria. Our results are summarized in Table 5.1.

To solve the cooperative rational synthesis problem (CRSP), we characterize the executions compatible with Nash equilibria by means of LTL formula over the set of states in the interaction model. Then, finding a strategy profile that is a solution to CRSP is reduced to the problem of finding an execution satisfying the formula $\phi^{\mathcal{M}}_{0\text{Nash}} \wedge \varphi_0$ where $\phi^{\mathcal{M}}_{0\text{Nash}}$ is the characterization of 0-fixed Nash equilibria and φ_0 is the objective of Process 0, expressed in LTL$[\mathcal{M}]$. This approach allows us to obtain NP complexities for the majority of the objectives we consider.

The above procedure cannot be applied to the non-cooperative setting. We prove by counterexample that is wrong to reduce directly to a two-player zero-sum game where the protagonist's objective is to satisfy the formula $\phi^{\mathcal{M}}_{0\text{Nash}} \rightarrow \varphi_0$. This is because in the case of non-cooperative rational synthesis problem (NCRSP) one has first to fix the strategy of the process to synthesize (Process 0) and to consider only 0-fixed Nash equilibria with respect to this strategy.

Therefore, we propose an automata-based solution for NCRSP that first encodes the strategies of Process 0 as trees and then define a nondeterministic tree automaton that accepts exactly solutions to the considered problem. The automaton that we obtain is exponential in the number of processes and polynomial in the number of states in the interaction model. Moreover, it has monotonic properties that allow us to obtain PSpace complexity for Safety, Reachability, Büchi and coBüchi objectives and ExpTime complexity for Parity, Streett, Rabin and Muller.

We also studied the cooperative and non-cooperative rational synthesis problems when the number of processes is fixed and obtained even better complexities.

Generalization to Concurrent setting The algorithms presented in this chapter are for turn-based interaction models, where only one process is active at a time. However, the LTL characterization does not work in the case of concurrent models, as it is shown in Example 5.5.1. It is unlikely to be a simple generalization as, in general, going from turn-based to concurrent models increase the complexity of finding Nash equilibria [13].

Example 5.5.1. *Let us take the concurrent interaction model illustrated in Figure 5.9 on which act two processes by playing actions in the set* $\Sigma_0 = \Sigma_1 = \{a, b\}$.

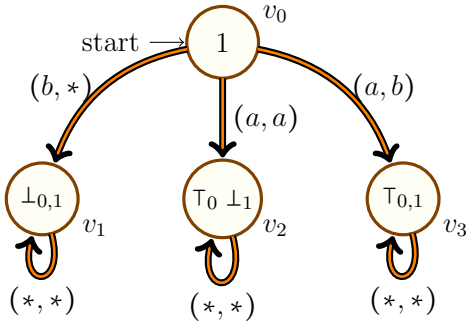


Figure 5.9: Example showing that the LTL characterization doesn't apply for CRSP in concurrent models

We consider reachability objectives for the two processes as follows. The target set for Process 0 is $R_0 = \{v_2, v_3\}$ *(states labeled with* $\top_0$*) and the target set of Process 1 is* $R_1 = \{v_3\}$ *(labeled with* $\top_1$*).*

The only state from which Process 1 has a strategy to reach the target (in the zero-sum setting) is state v_3*. Indeed, the action b of Process 0 from the initial state leads to the state* v_1 *that is losing for Process 1. Therefore,* $W_1 = \{v_3\}$.

Now, let's take the path $\rho = v_0(v_2)^\omega$*. It satisfies the LTL characterization* $\phi = \Diamond R_0 \wedge \big(\Box \neg R_1 \rightarrow \Box \neg W_1\big)$ *since* $v_2 \in R_0$ *is reached and the set* $W_1 = \{v_3\}$ *is not visited. However, the path* ρ *is not compatible with any Nash equilibrium since Process 1 may deviate and play from the initial state the action b while Process 0 keeps his strategy that asks to play a in the initial state. Then, the tuple* (a, b) *of actions leads to the state* $v_3 \in R_1$*. Therefore, Process 1 reaches its target.*

The reason for which the LTL characterization does not apply is that in the concurrent setting, deviations are not immediately visible since processes observe only the effect of the actions played and not the actions themselves. For instance, in Example 5.5.1, since both Processes 0 and 1 play at the same time, the deviation b of Process 1 is only seen by Process 0 when v_3 is reached.

The fact that processes only observing the effect of the actions played by processes induce some partial observation. Results published in [13] show that this partial observation makes decision of the existence of a (constrained) Nash equilibrium in concurrent models possibly more difficult than in the turn-based models. For instance, for Streett objectives, the complexity of finding a constrained Nash equilibrium increases from NP-complete[87] in the turn-based case to PSPACE and P^{NP}-hard in the concurrent models[13].

However, in the case of Safety, Reachability, Büchi, co-Büchi, Rabin and Muller objectives, we can apply the algorithms from [13] to obtain the same complexities for

CRSP as for the turn-based models when the number of processes is not fixed. More precisely, in [13] is studied the problem of finding constraints Nash equilibria in concurrent games. It asks for the existence of Nash equilibria whose outcome is between two thresholds. Then, by choosing the lower thresholds to be such that only Process 0 satisfies his objective and the upper thresholds such that all processes win, we reduce to the cooperative rational synthesis problem. It is shown in [13] that the existence of constrained Nash equilibria in concurrent games can be solved in PTIME for Büchi objectives, NP for Safety, Reachability and coBüchi objectives, in P^{NP} for Rabin objectives and PSPACE for Muller objectives. All hardness results are inferred directly from the hardness results in Section 5.3. For Streett objectives, by reducing to [13] we only obtain PSPACE-easiness and the NP-hardness comes from the turn-based setting. We leave this problem for further work.

The construction presented in Section 5.4 does not work from similar reasons as in the case of CRSP. We leave open the extension to the concurrent setting, but it seems alternation would be needed in a naive attempt, as a path the model may be compatible with more than one sequence of actions of processes. Therefore, it is unlikely toto improve the complexity results from [53] obtained for LTL objectives.

Future work A first question that has to be answered is, as discussed above, what are the tight complexities for the rational synthesis problem in concurrent interaction models.

Then, as already mentioned above, in this thesis the rationality of the environment is modeled by assuming that the players composing it play a Nash equilibrium. Interesting directions for future work would be to assume other notions of rationality, e.g. secure equilibria [21], doomsday equilibria [19], subgame perfect equilibria [86, 87], or admissible strategies [10, 34]. Choosing secure equilibria to model rationality involves two processes and finally the rational synthesis problem reduces to the case when Nash equilibria are considered (the case studied in this thesis). For other solution concepts, the technique to follow is not straightforward.

6. Rational Synthesis with Imperfect Information

In this chapter we study the rational synthesis problem under imperfect information. We consider several capabilities (perfect/imperfect information) for both processes in $P \subseteq \Omega$ to be synthesized and the environment. All results presented here represent ongoing work that is not published.

We recall that, given an interaction model $\mathcal{M}$ where the $k+1$ processes have the objectives $(\Theta_i)_{i \in \Omega}$ and a set $P \subseteq \Omega$ of processes to be synthesize, the rational synthesis problem asks the following questions according to the two settings:

cooperative: Is there a P-fixed Nash equilibrium $\bar{\sigma}$ such that $pay_i(\bar{\sigma}) = 1$ for all $i \in P$?

non-cooperative: Is there a strategy profile σ_P for processes in P such that for any P-fixed Nash equilibrium $\bar{\sigma} = (\sigma_P, \sigma_{\overline{P}})$, we have $pay_i(\bar{\sigma}) = 1$ for all $i \in P$?

Setting We consider turn-based interaction models in which the successor state is deterministically given by the action of the process controlling the current state. Formally, the interaction model is defined as $\mathcal{M} = \langle \Omega, (\Sigma_i)_{i \in \Omega}, V, (V)_{i \in \Omega}, v_0, E, \mathcal{O}_0 \rangle$ where Σ_i is the set of actions of Process i and the transition relation is such that for each state $v \in V$ there is exactly one successor determined by each action of the process controlling v, i.e., $\forall i \in \Omega,\ \forall v \in V_i,\ \forall a_i \in \Sigma_i$, holds $|E(v, a_i)| = 1$. We introduce the actions for the processes since Process 0 has imperfect information and is not realistic to ask him to name the next state. We can assume that the actions of the other processes consist in choosing the next state (i.e., $\Sigma_i = V$ for $1 \leq i \leq k$) since they have perfect information.

In the following we prove that the rational synthesis problem is undecidable in general if we consider imperfect information for all agents in the interaction model. However, we gain decidability if we restrict to synthesize the strategy for one process that acts against a omniscient, rational environment.

6.1 Imperfect Information for All Processes

In this section we prove that the rational synthesis problem with all processes in the interaction model having imperfect information is undecidable for both settings (cooperative and non-cooperative).

Indeed, if it is asked to be synthesized two or more processes against some environment (i.e. $|P| \geq 2$), the problem coincides with the distributed synthesis problem (defined in Section 2.3.1) against an omniscient environment which, according to [75], is undecidable. We recall that the question in the case of distributed synthesis problem is if there exist strategies for the processes in P such that for any strategy of the environment, the joint objective φ holds. Now, to reduce to the rational synthesis problem, we simply set the formula φ as individual objective of the processes in P and associate to all the processes in the environment the objective $\neg\varphi$.

Theorem 6.1.1. *The LTL cooperative and non-cooperative rational synthesis problem are undecidable when asked to synthesize two or more processes having imperfect information. It holds already if the environment consists of one perfectly informed process.*

Proof. The proof results directly from the distributed synthesis problem (defined in Section 2.3.1) undecidability.

Let's take the interaction model $\mathcal{M} = \langle \Omega = \{0, 1, 2\}, V, (\Sigma_i)_{i\in\Omega}, \Delta, (\mathcal{O}_i)_{i\in\Omega} \rangle$ where Process 2 is the only one having perfect information, i.e., $\mathcal{O}_2 = V$. We set the objectives of the three processes as follows. Process 0 and 1 have the same objective $\varphi_0 = \varphi_1 = \varphi$ and the objective for Process 2 is $\varphi_2 = \neg\varphi$.

We claim that there is a solution for the distributed synthesis problem of processes 0 and 1 with the joint objective φ if and only if there is a solution for the (non-)cooperative synthesis problem where Process 2 represents the rational environment.

Indeed, if there is a solution for the distributed synthesis problem, there are strategies σ_0 and σ_1 for Processes 0 and 1 such that for any strategy σ_2 of Process 2, holds $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \vDash \varphi$. Then, the tuple (σ_0, σ_1) is also a solution for the non-cooperative synthesis problem since all the executions compatible with this two strategies φ (the objective of both Process 0 and 1), which makes Process 2 lose. Moreover, any strategy profile $\langle \sigma_0, \sigma_1, \sigma_2 \rangle$ is a solution of the cooperative rational synthesis problem since Process 2 cannot deviate and win.

On the other hand, if there is a solution for the non-cooperative synthesis problem, the there is a tuple (σ_0, σ_1) of strategies for Processes 0 and 1 such that for any strategy σ_2 of Process 2, if the strategy profile $\langle \sigma_0, \sigma_1, \sigma_2 \rangle$ is a Nash equilibrium, then it satisfies φ. It results that the only $\{0, 1\}$-fixed Nash equilibria consistent with σ_0 and σ_1 are such that Process 2 loses. Therefore, since he cannot deviate to improve his payoff, all the executions compatible with σ_0 and σ_1 are losing for him (satisfy φ). This means that for

any strategy σ_2 of Process 2, $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \vDash \varphi$ and therefore (σ_0, σ_1) is a solution for the distributed synthesis problem.

If there is a solution $\bar{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2 \rangle$ for the cooperative synthesis problem, it holds that $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \vDash \varphi$ and it is a $\{0, 1\}$-fixed Nash equilibrium. Therefore, the Process 2 (that loses under $\bar{\sigma}$) cannot deviate to improve its payoff. This means that for any strategy σ_2' of Process 2, $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2' \rangle) \vDash \varphi$ and therefore (σ_0, σ_1) is a solution for the distributed synthesis problem. ■

Moreover, the rational synthesis problem is easily proven undecidable for cooperative setting even if there is only one process to synthesize (i.e. $|P| = 1$) against an imperfectly informed rational environment. The proof is by reduction from the distributed synthesis problem against one perfectly informed antagonist environment.

Theorem 6.1.2. *The Cooperative Rational Synthesis Problem of one process is undecidable for three-processes imperfect information interaction model and LTL objectives.*

Proof. Let $\mathcal{M} = \langle \Omega = \{0, 1, 2\}, V, (\Sigma_i)_{i \in \Omega}, \Delta, (\mathcal{O}_i)_{i \in \Omega} \rangle$ be an interaction model with three processes and φ be an LTL formula and let consider in the following Process 2 having perfect information. That is, the set $\mathcal{O}_2$ contains only singletons and we can abuse and write $\mathcal{O}_2 = V$.

We claim that there is a solution for CRSP in $\mathcal{M}$ with objectives $\varphi_0 = \varphi_1 = \varphi$ and $\varphi_2 = \neg\varphi$ if and only if there is a solution for the distributed synthesis problem (defined in Section 2.3.1) in $\mathcal{M}$ with antagonist environment and the objective φ.

From left to right, consider there is a solution for CRSP in $\mathcal{M}$ and objectives $\varphi_0 = \varphi_1 = \varphi$ and $\varphi_2 = \neg\varphi$. It means that there is a strategy profile $\bar{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2 \rangle$ that is a 0-fixed NE and $\text{exec}(\mathcal{M}, ()\bar{\sigma}) \vDash \varphi_0$. From the definition of the objectives, $pay_0(\bar{\sigma}) = pay_1(\bar{\sigma}) = 1$ and $pay_2(\bar{\sigma}) = 0$, but since $\bar{\sigma}$ is a 0-fixed NE, Process 2 cannot deviate and improve his payoff. Therefore, for all strategies σ_2' of Process 2, $out(\sigma_0, \sigma_1, \sigma_2') \vDash \varphi$ and therefore the tuple (σ_0, σ_1) is a solution for the distributed synthesis problem.

From right to left, consider there is a solution for distributed synthesis problem. Then, there is a tuple (σ_0, σ_1) of observation-based strategies such that for any strategy σ_2, $\text{exec}(\mathcal{M}, \langle \sigma_0, \sigma_1, \sigma_2 \rangle) \vDash \varphi$. Consider such a strategy profile $\bar{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2 \rangle$ where processes 0 and 1 play the solutions of distributed synthesis problem. Considering objectives $\varphi_0 = \varphi_1 = \varphi$ and φ_2 in the cooperative rational synthesis problem, $pay_0(\bar{\sigma}) = pay_1(\bar{\sigma}) = 1$ and $pay_2(\bar{\sigma}) = 0$ but on any deviation of Process 2, he is not improving his payoff since σ_0 and σ_1 ensure φ against any strategy of Process 2. Therefore, $\bar{\sigma} = \langle \sigma_0, \sigma_1, \sigma_2 \rangle$ is a 0-fixed Nash equilibrium s.t. $\text{exec}(\mathcal{M}, \bar{\sigma}) \vDash \varphi_0$ and therefore is a solution for the cooperative rational synthesis problem. ■

As already mentioned, the rational synthesis problem is undecidable when asked to

synthesize two processes against an omniscient environment due to the undecidability of distributed synthesis problem.

Since the cooperative rational synthesis problem is undecidable when is asked to synthesize one process in a three-process interaction model, it is very likely to be the case of non-cooperative setting too. However, we leave this question open because the existence of Nash equilibria in imperfect information games is essential in constructing a reduction from distributed synthesis problem as in the cooperative case. Unfortunately, in imperfect information games, Nash equilibria does not always exist. Therefore, one needs another reduction to prove the undecidability of non-cooperative synthesis problem when only one process is asked to be synthesized.

6.2 One Process Against Omniscient Multi-component Environment

We already saw that the problem is undecidable when there are two or more partially informed processes to synthesize against an omniscient environment.

In this chapter we consider imperfect information only for the process to be synthesized to gain the decidability of the rational synthesis problem. Moreover, a positive answer to the setting when only the system to be synthesized has imperfect information, is also a positive answer for the more restricted environment that has to play observational strategies. Indeed, if Process 0 has a strategy to win against the omniscient rational environment, then it can apply the same strategy to ensure its objective when the environment's observation is more restricted. Therefore, a solution for this setting represents a *semi-procedure* for the more general case when there is asked to synthesize one process against a partially informed multi-component rational environment.

6.2.1 Imperfect Information Cooperative Rational Synthesis (ICRSP)

We discuss in this section the cooperative rational synthesis problem under the assumption of imperfect information for Process 0. We stress the difficulties that have to be faced when solving the problem and give a possible direction towards a solution.

In the perfect information case (Section 5.3), we define a characterization of the equilibrium in terms of $\mathsf{LTL}[\mathcal{M}]$ properties on executions in the interaction model. However, it does not apply to the case of imperfect information for Process 0.

First, one needs to compute the winning regions for all processes $i \in \Omega \setminus \{0\}$. Once the winning regions computed (if possible), it is not clear how the $\mathsf{LTL}[\mathcal{M}]$ characterization from Section 5.3 may apply. Indeed, assume that we computed the winning regions and that there is an execution ρ in $\mathcal{M}$ that satisfies the formula $\phi^{\mathcal{M}}_{0\text{Nash}}$ defined in Section 5.2.

We have to define a strategy profile whose outcome is the execution ρ and in which the strategy for Process 0 is observation-based. When a Process deviates from a state which he does not have a winning strategy, the other agents should punish him by ensuring he is not reaching his objective. But, since the we are in the imperfect information (for Process 0) setting, it is not clear if such retaliating strategy exists. Moreover, if it exists, Process 0 may not observe that there is a deviation and the retaliating strategy may ask a different action then the one that follows ρ.

Having said this, for now we leave open the cooperative rational synthesis problem and show in the next section how to solve the non-cooperative case.

6.2.2 Imperfect Information Non-Cooperative Rational Synthesis (INCRSP)

In this section we show how the general algorithm for perfect information can be used to solve the case where it is asked to synthesize a partially informed process against a non-cooperative omniscient rational environment.

In this case we do not have the difficulties discussed in the cooperative setting. This is because in the non-cooperative setting, we fix the strategy of Process 0 and consider the Nash equilibria with respect to the fixed strategy. Moreover, in the procedure we describe in the following, we don't need to compute the winning regions of the agents.

Goal The idea is to build an interaction model $\mathcal{M}'$ containing three processes A, B, C having hierarchical observations such that there is a solution for the distributed synthesis problem with antagonist environment if and only if there is a solution for INCRSP. They are as follows. There is a Process A that has imperfect information and takes the decisions of Process 0 in $\mathcal{M}$, a second Process B has perfect information and decides which agents in the environment have deviations along executions and a third Process C that plays for the processes in the environment. Then, the imperfect information non-cooperative rational synthesis is reduced to the problem of distributed synthesis (defined in Section 2.3) for the two processes A and B against the antagonist environment that consists of Process C. Note that due to Theorem 2.3.3 in Chapter 2, the later problem is decidable since the two Agents A and B have hierarchical observations on the interaction model. We further call "agents" the processes in the new interaction model, so that is easier for the reader to follow.

Motivation of reduction The construction follows the idea in the definition of the nondeterministic tree automaton $\mathcal{T}_{\mathcal{M}}$ in Section 5.4.1 built to solve the non-cooperative rational synthesis problem in the case of perfect information, and therefore is similar to the game $\mathcal{G}_{\mathcal{T}}$ induced by the automaton. In the game built for the perfect information case,

Eve's actions correspond to first choosing the actions of Process 0 in the interaction model and then decide possible deviations for the other processes consisting the environment.

This is not possible in the case when Process 0 has imperfect information in the interaction model $\mathcal{M}$. This is because Eve's decisions should be taken as follows. First, she should decide observational strategies for Process 0 and therefore should not make difference between indistinguishable states for Process 0 (has to decide depending on observations on states of $\mathcal{M}$). Then, she has to know the current state (and "concrete" history) in the model $\mathcal{M}$ to be able to guess the possible deviations of the components of the environment. Therefore, one cannot define an observation relation for Eve such that she has in the same time perfect and partial observation on the states of the interaction model $\mathcal{M}$.

To solve the problem in the case of incomplete information, we use a dedicated agents for the two decisions that have to be made. First, Agent A corresponds to Process 0 and has the same observation as him, and then Agent B is perfectly informed and decides the deviation points. Finally, there is a third Agent C that plays the same actions as Adam in the game $\mathcal{G}_{\mathcal{T}}$, corresponding to the choices of the components of the environment.

Intuition on construction The game arena is as follows: it contains all the nodes in the interaction model $\mathcal{M}$ and for each node v controlled by a process in the environment, there is a state (v, ℓ) for each $\ell \in E(v) \cup \{\star\}$. Agent A controls all the nodes belonging to Process 0 in $\mathcal{M}$, Agent B controls all the nodes belonging to one of the processes of the environment and Agent C controls the newly introduced states. Along the states, we also keep the information W and D of the players that have a winning strategy from the current node and the set of players that passed through a good deviation point.

Intuitively, the interaction is as follows. At each state $(v, W, D) \in V_A$, Agent A makes the choice of Process 0 in the interaction model $\mathcal{M}$ and the game moves to the next state v' dictated by the transition function in $\mathcal{M}$ and transfers the information regarding the two sets W and D. Then, if the current state v' is controlled by some process $i \neq 0$ in the environment in $\mathcal{M}$, it is the turn of Agent B to play from (v', W, D). By his action, the winning strategy of Process i (if any) is guessed. Agent B plays the action "$\star$" to indicate that it does not guess a strategy for Process i controlling the state v'. However, Agent B cannot play "$\star$" if $i \in W$ and is forced to play it if $i \in D$. This is because in the first case was guessed that Process i has a winning strategy and he has to continue guessing it. If $i \in D$, a good deviation has already been guessed and Agent B does not need to guess another one. The next state after Agent B played $\ell \in E(v) \cup \{\star\}$, is (v', ℓ, W, D).

Finally, from the state (v', ℓ, W, D) is the turn Player C to play for Process i controlling the state v in the model $\mathcal{M}$. He can choose any successor of v' in $\mathcal{M}$. The letter ℓ chosen by Player B serves only to update the sets W and D and it does not influence the choice of Player C. The sets W and D are updated as follows. If $\ell = \star$, all the successors are

of type (v'', W, D), where v'' is a successor of v' in $\mathcal{M}$. If $\ell \in E(v')$ (Player B guessed a winning strategy for Process i), Process i is added in the set W' associated to the next state (ℓ, W', D') and on all other successors Process i is removed from W (if contained) and added to the set D, i.e., for all successors $v'' \neq \ell$ of v' in $\mathcal{M}$, $(v'', W \setminus \{i\}, D \cup \{i\})$ is successor of (v', ℓ, W, D) in $\mathcal{M}'$. The update of the two sets encodes the fact that it is continued to be guessed a winning strategy from the successor ℓ chose by Agent B and on the other successors is kept the information that there is a deviation.

Formal Construction Let us formally define the three-agent interaction model. Given the interaction model $\mathcal{M} = \langle \Omega = \{0, ..., k\}, (\Sigma_i)_{i\in\Omega}, V, (V_i)_{i\in\Omega}, v_0, E, \mathcal{O}_0 \rangle$, we define the three-agent interaction model as $\mathcal{M}' = \langle \Omega', \Sigma_A, \Sigma_B, \Sigma_C, V' = V_A \uplus V_B \uplus V_C, v_0, E', \mathcal{O}_A \rangle$ where

- $\Omega' = \{A, B, C\}$ where Process C is the environment,
- $\Sigma_A = \Sigma_0$, $\Sigma_B = V \cup \{\star\}$ and $\Sigma_C = V$
- $V_A = V_0 \times 2^\Omega \times 2^\Omega \cup \{\bot, \top\}$
- $V_B = \bigcup_{1\leq i\leq k} V_i \times 2^\Omega \times 2^\Omega$
- $V_C = V \times (V \cup \{\star\}) \times 2^\Omega \times 2^\Omega$
- $v_0' = v_0 \times \varnothing \times \varnothing$
- the observation of Agent A is such that
  - $o_A(v, W, D) = o_A(v', W', D')$ iff $o_0(v) = o_0(v')$ for all $(v, W, D), (v', W', D') \in V \times 2^\Omega \times 2^\Omega$
  - $o_A(v, \ell, W, D) = o_A(v', \ell', W', D')$ for all $(v, \ell, W, D), (v', \ell', W', D') \in V_C$
- the transition relation E' is defined by
  - for $v \in V_0$, $E'((v, W, D), a) = (E(v, a), W, D))$
  - for $(v, W, D) \in V_B$, $v \in V_i$, $i \neq 0$ and $\ell \in V \cup \{\star\}$,
    1. if $i \in W$, Agent B has to choose an action according to the winning strategy of Process i in v:
    $$E'((v, W, D), \ell) = \begin{cases} (v, v', W, D) & \text{if } \ell = v' \in E(v) \\ \bot & \text{otherwise} \end{cases}$$
    2. if $i \in \overline{W} \cap \overline{D}$, Agent B decides if the current history is a good deviation point or not:
    $$E'((v, W, D), \ell) = \begin{cases} (v, \ell, W, D) & \text{if } \ell \in E(v) \cup \{\star\} \\ \bot & \text{otherwise} \end{cases}$$

3. if $i \in \overline{W} \cap D$, Agent B does not guess any deviation for Process i since it was already guessed in the past:

$$E'((v,W,D),\ell) = \begin{cases} (v,\star,W,D) & \text{if } \ell = \{\star\} \\ \bot & \text{otherwise} \end{cases}$$

– for $(v,\ell,W,D) \in V_C$ and $v' \in V$,

$$E'((v,\ell,W,D),v') = \begin{cases} (v',W,D) & \text{if } \ell = \{\star\} \text{ and } v' \in E(v) \\ (v',W \cup \{i\},D) & \text{if } \ell = v' \text{ and } v' \in E(v) \\ (v',W \setminus \{i\},D \cup \{i\}) & \text{if } \ell, v' \in E(v) \text{ and } \ell \neq v' \\ \top & \text{otherwise} \end{cases}$$

Note that the structure of the interaction model $\mathcal{M}'$ is very similar to the one of the tree automaton built in Section 5.4.1 for the case of perfect information NCRSP. Moreover, we keep monotonicity properties for the sets D and W along the executions in the model $\mathcal{M}'$. That is, once a process is added in the set D, it is never removed and in the set W each process i may be added and removed at most once.

Since Agent A simulates Process 0, it has the same observation. That is, it observes only the V-component of each state (v,W,D) and makes no difference between states of Agent C. That is, Agent A cannot observe the actions of the agent that guesses the deviation points.

Finally, for $\eta \in \mathsf{exec}(\mathcal{M}') \cap (V' \setminus \{\top,\bot\})^\omega$, let $\eta_{\restriction(V_A \cup V_B)}$ stand for the restriction of η to the states belonging to Agents A and B. Note that if we further project on the component V each state, we obtain an execution in the interaction model $\mathcal{M}$. Then, the joint objective of Agent A and Agent B is defined as following. They try to propose actions such that all the executions that are compatible with their decisions either satisfy the objective of Process 0 in $\mathcal{M}$, or there is one process i that loses, but Process B correctly guessed a deviation for him. The correct deviations are verified by checking that all the agents that are in the set W (after stabilization) and for which was guessed a winning strategy, win. Formally,

$$\Theta = (V')^*(\{\top\})^\omega \cup \Big\{\eta \in \mathsf{exec}(\mathcal{M}') \cap (V' \setminus \{\top,\bot\})^\omega \mid \Big(\eta_{\restriction(V_A \cup V_B)} \vDash \varphi_0 \vee \bigvee_{i=1}^{k} (\eta_{\restriction(V_A \cup V_B)} \nvDash \varphi_i \wedge \varphi_{\exists dev}(i,\eta))\Big) \wedge \bigwedge_{i \in lim_W(\eta)} \eta_{\restriction(V_A \cup V_B)} \vDash \varphi_i\Big\}$$

where the formula $\varphi_{\exists dev}(i,\eta)$ expresses the fact that Process i has a good deviation along η and is formally defined depending on the type of objective we consider for the processes in $\mathcal{M}$. For tail objectives, $\varphi_{\exists dev}(i,\eta) = i \in lim_D(\eta)$. For other objectives, we can define the formula such that it specifies where deviation points should appear. For example, in

the case of Safety objectives, a deviation point can appear only before reaching an unsafe state for Process i.

The following Proposition holds because the interaction model $\mathcal{M}'$ corresponds to $\mathcal{M}$ in which states are annotated with the extra information W and D and there is an intermediary state (controlled by Process B) before any decision of a Process $i \neq 0$ that is used to update the information in W and D. Then, the accepting condition corresponds exactly to the definition of non-cooperative rational synthesis problem. It asks that on an execution, either Process 0 wins in $\mathcal{M}$, or there is a process in the environment that looses, but could deviate. As in the case of perfect information for all processes, the formula $\bigwedge_{i \in lim_W(\eta)} \eta_{\restriction (V_A \cup V_B)} \vDash \varphi_i$ asks that all the players for which it was guessed that they play their winning strategy, indeed win.

Proposition 6.2.1. *There is a solution for the INCRSP in $\mathcal{M}$ iff there exists σ_A (observational) and σ_B such that $\forall \sigma_C, \mathsf{exec}(\mathcal{M}', \langle \sigma_A, \sigma_B, \sigma_C \rangle) \in \Theta$.*

Proof. Let us take a strategy σ_0 for Process 0 which is a solution to INCRSP problem in $\mathcal{M}$. Then, on all executions compatible with σ_0, either φ_0 is satisfied, or there is a Process i that looses and would rather prefer to deviate. Let us consider the same strategy for Agent A in $\mathcal{M}'$ defined formally by $\sigma_A(h) = \sigma_0(h')$ where $h' = (h \restriction (V_A \cup V_B))|_V$ is the projection on the first component of the restriction of h on the states in V_A and V_B. It is a valid observational strategy for Agent A since he has the same observation as Process 0. Then, the strategy of Agent B is as follows. On each execution, he guesses which are the processes that lose and may deviate to another strategy. It is possible to do this because σ_0 is a solution to INCRSP. Then, since Agent B guesses good deviations for the processes from the environment that lose, it is also able to guess their winning strategy. Therefore, on each execution η in $\mathcal{M}'$, all processes $i \in lim_W(\eta)$ win since Agent B guessed the winning strategy for him.

On the other direction, let us consider a tuple (σ_A, σ_B) of strategies for Agents A and B to ensure Θ on all executions of $\mathcal{M}'$. We define the strategy of Process 0 as $\sigma_0(h) = \sigma_A(h')$ for some $h' = (h \restriction (V_A \cup V_B))|_V$. Note that it is a correct definition since the annotation with W and D is not visible for Agent A. Since all executions compatible with (σ_A, σ_B) satisfy Θ, and since to each execution in $\mathcal{M}'$ corresponds an execution in $\mathcal{M}$ by projecting away the sets W and D, σ_0 is a solution to INCRSP. Indeed, on all executions that satisfy $\bigvee_{i=1}^{k} (\eta_{\restriction (V_A \cup V_B)} \nvDash \varphi_i \wedge \varphi_{\exists dev}(i, \eta))$, Agent B guessed a deviation for at least one process in the environment that looses. Therefore, the execution is not compatible with a 0-fixed Nash equilibrium. ∎

The interaction model $\mathcal{M}'$ that we constructed is hierarchical since only Agent A has imperfect information. Therefore, as a consequence of Proposition 6.2.1 and Theorem 2.3.3, we obtain the following result:

Theorem 6.2.1. *The synthesis problem of one partially informed process against an omniscient, non-cooperative, but rational multi-component environment is decidable.*

6.3 Conclusions

In this chapter we proved that generally the rational synthesis problem is undecidable under the assumption of imperfect information for the participating processes. This is obtained very easily from the undecidability of distributed synthesis problem.

We also considered some restrictions and prove that the cooperative rational synthesis problem remains undecidable if there is one process to synthesize and multicomponent environment, both with imperfect information. It is very likely that the undecidability result holds for the non-cooperative setting too, but we leave open this question.

We gain decidability for the non-cooperative rational synthesis problem in the case there is only one partially informed process to be synthesized and its environment is omniscient. The algorithm we provide is inspired by the construction in Section 5.4.1 and reduces the rational synthesis problem to the distributed synthesis problem in three-processes interaction model where processes have hierarchical observations.

7. Conclusions

7.1 Main Results

In this thesis we study the synthesis problem of reactive interactive systems. We consider multi-component interaction models in which processes may have partial observation of the current state of the model and study the synthesis of some components when the environment (the other components) is either antagonist or rational. We study the problem considering different observation capabilities of the components to be synthesized and their environment. A summary of the decidability results is contained in Table 2.1.

The goal of the thesis is that, in case the problem is decidable under the considered observation capabilities for the processes, to provide Safraless decision procedures for the synthesis problem that may lead to efficient implementation.

In Chapter 2, we give a general definition of the synthesis problem that encapsulates the two settings in which we study the problem and also present existing results in the domain of synthesis and in Chapter 3, we present some tools and techniques that are used by the procedures solving the synthesis problem. We also provide complete proofs for testing the emptiness of universal coBüchi tree automata, that are used in Chapter 4.

The synthesis problem from KLTL specifications with an antagonist environment is undecidable in general [75]. In Chapter 4, we propose a Safraless decision procedure to synthesize one partially informed process against an antagonist one-component omniscient environment from specifications expressed in the positive fragment of KLTL (KLTL$^+$). It is based on the idea of [36] to define some universal coBüchi tree automata that accepts exactly solutions for the synthesis problem and for which the non-emptiness can be tested efficiently by reduction to safety games. Our implementation[1] is based on the tool *Acacia+*[12] and, to the best of our knowledge, it is the first implementation of a synthesis procedure for epistemic temporal specifications considering perfect recall for processes.

In Chapter 5, we study the synthesis problem with a rational environment under the assumption of perfect information. That is, the components of the environment are rational in the sense that they first target the satisfaction of their own objectives and then

try to harm the others. We use the notion of Nash equilibria to model the rationality. We provide general solutions for both cooperative and non-cooperative settings. Then, we apply them for particular classes of objectives and provide tight complexity results. Table 5.1 gives an overview of the complexity results for the rational synthesis problem.

In this thesis we study the complexities considering the same type of objectives for all player. However, we can also consider that the agents have different types of objectives. In this case, the complexity to solve the problem is the maximum of the complexities of the for the restrictions that we consider for the used classes of objectives.

Finally, in Chapter 6, we study the rational synthesis problem under the assumption of partial information of components. We show that the problem is undecidable in the general case when all processes have partial information. However, we gain decidability of the non-cooperative synthesis problem if we consider one partially informed process to synthesize against a multi-component rational omniscient environment. The procedure we propose reduces our problem to the distributed synthesis problem of two components with hierarchical observations against an omniscient antagonist environment.

It is important to note that the procedure that solves the rational synthesis problem against an omniscient environment represents a semi-procedure for the more general (and more realistic) case where the processes consisting the environment have partial observation.

We only discuss the difficulties that have to be faced when studying the cooperative rational synthesis problem when only one process (to be synthesized) has partial information and its environment is omniscient. However, we conjecture that the problem is also decidable, guided by the results in perfect information setting where, for the different types of objectives that we consider, the rational synthesis problem in the cooperative setting has lower complexities than the non-cooperative setting.

7.2 Perspectives

Several perspectives have already been given for each chapter of the thesis: In Section 4.4 for the KLTL synthesis with antagonist environment and in Sections 5.5 and 6.3 for the synthesis problem with rational multi-component environment.

The general goal of the synthesis problem is to obtain "Safraless" procedures that deal with a large enough class of formulas to prove themselves useful in practice. That is, our goal is to obtain some tools that help companies develop safety-critical programs that are correct by construction, and therefore streamline the process of reactive systems design.

Safraless Procedure for KLTL Synthesis

For KLTL synthesis with rational environment, we only provided a "Safraless" procedure from the positive fragment of KLTL (KLTL$^+$). However, results obtained by testing the implementation on several examples are an encouraging step towards developing implementable procedures for the entire KLTL logic.

The difficulty appears when a formula contains some knowledge operator K under an odd number of negations, or the formula in negative normal form contains the operator P (the dual of K). The knowledge operator P expresses the fact that some property is possible, meaning that there is an execution compatible with the sequence of observations of the process to be synthesized that satisfies the desired property.

As already mentioned in Section 4.4, a first step to cope with this operator would be to consider formulas of the type $K\varphi \rightarrow \psi$ where φ is an LTL formula and ψ is a KLTL$^+$formula. This means, the formula contains only one operator P that may be easier to cope with. However, it is not obvious how this problem can be solved using automata-based procedures that avoid alternation.

Procedures for Rational Synthesis

Tight complexities for the rational synthesis problem were studied only when all components of the interaction model have perfect information.

In this thesis we proved the decidability of the non-cooperative synthesis problem when is asked to synthesize a partially informed process against a multi-component omniscient rational environment.

Our target is to develop (automata-based) procedure that efficiently solves the rational synthesis problem for the process to be synthesized having partial information. The final goal is to implement them in a tool. Already, an ongoing work is to define a nondeterminist tree automaton that accepts solutions for the problem when all the processes have either reachability or safety objectives. Our hope is that they will lead to procedures that can be efficiently implemented.

Until now, we studied efficient procedures for the rational synthesis problem when the rationality of agents is modeled by Nash equilibria. For the case of imperfect information, the difficulty is that such equilibria may not exist. Another line of research would be to consider other solution concepts to model rationality of processes. An interesting direction would be to consider Doomsday equilibria that extends secure equilibria to multi-component setting. In [19] has been proven that such equilibria always exist even in the case of imperfect information. Moreover, the cooperative setting for the rational synthesis is directly solved by finding such an equilibria.

REFERENCES

[1] Acacia-k. `http://lacl.fr/~rbozianu/Acacia-K/`. Year: 2014.

[2] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, ICALP '89, pages 1–17, London, UK, UK, 1989. Springer-Verlag.

[3] Thomas Ågotnes, Valentin Goranko, and Wojciech Jamroga. Alternating-time temporal logics with irrevocable strategies. In *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2007), Brussels, Belgium, June 25-27, 2007*, pages 15–24, 2007.

[4] Christoph Schulte Althoff, Wolfgang Thomas, and Nico Wallmeier. Observations on determinization of büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006.

[5] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, September 2002.

[6] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for ltl fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, January 2004.

[7] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2003.

[8] Benjamin Aminof and Sasha Rubin. First cycle games. In *Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014.*, pages 83–90, 2014.

[9] Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to büchi automata translation: Fast and more deterministic. *CoRR*, abs/1201.0682, 2012.

[10] Dietmar Berwanger. Admissibility in infinite games. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February*

22-24, 2007, Proceedings, volume 4393 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2007.

[11] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. Ratsy - a new requirements analysis tool with synthesis. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 425–429. Springer Berlin Heidelberg, 2010.

[12] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for ltl synthesis. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 652–657. Springer, 2012.

[13] Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure nash equilibria in concurrent deterministic games. *Logical Methods in Computer Science*, 11(2), 2015.

[14] Rodica Bozianu, Catalin Dima, and Emmanuel Filiot. Safraless synthesis for epistemic temporal specifications. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2014.

[15] Thomas Brihaye, Arnaud Da Costa, François Laroussinie, and Nicolas Markey. ATL with strategy contexts and bounded memory. In Sergei N. Artemov and Anil Nerode, editors, *Proceedings of the Symposium on Logical Foundations of Computer Science (LFCS'09)*, volume 5407 of *Lecture Notes in Computer Science*, pages 92–106, Deerfield Beach, Florida, USA, January 2009. Springer.

[16] Julius R. Büchi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[17] Petr Cermák, Alessio Lomuscio, Fabio Mogavero, and Aniello Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 525–532. Springer, 2014.

[18] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *J. Comput. Syst. Sci.*, 79(5):640–657, 2013.

[19] Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. In *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, volume 8318 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2014.

[20] Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. The complexity of request-response games. In *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, volume 6638 of *Lecture Notes in Computer Science*, pages 227–237. Springer, 2011.

[21] Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Games with secure equilibria. *Theor. Comput. Sci.*, 365(1-2):67–82, 2006.

[22] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In *Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'07, pages 153–167, Berlin, Heidelberg, 2007. Springer-Verlag.

[23] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Algorithms for büchi games. *CoRR*, abs/0805.2620, 2008.

[24] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

[25] Krishnendu Chatterjee, Rupak Majumdar, and Marcin Jurdzinski. On nash equilibria in stochastic games. In *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2004.

[26] Alonzo Church. Logic, arithmetic, and automata. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*, pages 23–35. Inst. Mittag-Leffler, Djursholm, 1963.

[27] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, 2009.

[28] Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[29] Arnaud Da Costa, François Laroussinie, and Nicolas Markey. ATL with strategy contexts: Expressiveness and model checking. In Kamal Lodaya and Meena Mahajan, editors, *Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'10)*, volume 8 of *Leibniz International Proceedings in Informatics*, pages 120–132, Chennai, India, December 2010. Leibniz-Zentrum für Informatik.

[30] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 1999.

[31] Catalin Dima and Ferucio Laurentiu Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.

[32] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32Nd Annual Symposium on Foundations of Computer Science*, SFCS '91, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society.

[33] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[34] Marco Faella. Admissible strategies in infinite games over graphs. In *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, Novy Smokovec, High Tatras, Slovakia, August 24-28, 2009. Proceedings*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009.

[35] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for ltl realizability. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2009.

[36] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for ltl synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

[37] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. *CoRR*, abs/0907.3019, 2009.

[38] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010.*

Proceedings, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.

[39] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.

[40] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.

[41] Barbara Di Giampaolo, Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Safraless procedures for timed specifications. In *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010.

[42] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[43] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 60–65, New York, NY, USA, 1982. ACM.

[44] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *PODC*, pages 50–61. ACM, 1984.

[45] Joseph Y. Halpern and Kevin R. O'Neill. Anonymity and information hiding in multiagent systems. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003), 30 June - 2 July 2003, Pacific Grove, CA, USA*, pages 75–88, 2003.

[46] Joseph Y. Halpern and Rafael Pass. Sequential equilibrium in games of imperfect recall. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 278–287. AAAI Press, 2016.

[47] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[48] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science, pages 495–506, Berlin, Heidelberg, 2005. Springer-Verlag.

[49] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 117–124. IEEE Computer Society, 2006.

[50] Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262. Springer, 2007.

[51] Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 117–123, 2006.

[52] Yonit Kesten, Zohar Manna, Hugh McGuire, and Amir Pnueli. A decision algorithm for full propositional temporal logic. In *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1993.

[53] Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. In *Multi-Agent Systems - 12th European Conference, EUMAS 2014, Prague, Czech Republic, December 18-19, 2014, Revised Selected Papers*, pages 219–235, 2014.

[54] Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete information. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[55] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE Computer Society, 2005.

[56] François Laroussinie and Nicolas Markey. Augmenting ATL with strategy contexts. *Information and Computation*, 245:98–123, December 2015.

[57] Wanwei Liu and Ji Wang. A tighter analysis of piterman's büchi determinization. *Inf. Process. Lett.*, 109(16):941–945, 2009.

[58] Christof Löding. Automata on infinite trees. 2011.

[59] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, Lecture Notes in Computer Science, pages 682–688. Springer, 2009.

[60] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[61] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[62] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521 – 530, 1966.

[63] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[64] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *CoRR*, abs/1112.6275, 2011.

[65] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. What makes atl* decidable? A decidable fragment of strategy logic. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2012.

[66] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 133–144. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[67] Andrzej Wlodzimierz Mostowski. Regular expressions for infinite trees and a standard form of automata. In *5th Symp. Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 157 – 168. Springer, 1984.

[68] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the 1963 Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, SWCT '63, pages 3–16, Washington, DC, USA, 1963. IEEE Computer Society.

[69] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987.

[70] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[71] Marc Pauly. A modal logic for coalitional power in games. *J. Log. Comput.*, 12(1):149–166, 2002.

[72] Gary Peterson, John Reif, and Salman Azhar. Decision algorithms for multiplayer non-cooperative games of incomplete information. *Journal of Computers and Mathematics with Applications*, 43:179–206, 2002.

[73] Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.

[74] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

[75] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990.

[76] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Bulletin of the American Mathematical Society*, 74(5):1025–1029, 09 1968.

[77] John H. Reif. Universal games of incomplete information. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 288–308. ACM, 1979.

[78] John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.

[79] Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327. IEEE Computer Society, 1988.

[80] Sven Schewe. Solving parity games in big steps. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer, 2007.

[81] Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 474–488. Springer, 2007.

[82] Pierre-Yves Schobbens. Alternating-time logic with imperfect recall. *Electr. Notes Theor. Comput. Sci.*, 85(2):82–93, 2004.

[83] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985.

[84] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1):121 – 141, 1982.

[85] Serdar Tasiran, Ramin Hojati, and Robert K. Brayton. Language containment of non-deterministic *omega*-automata. In *Correct Hardware Design and Verification Methods, IFIP WG 10.5 Advanced Research Working Conference, CHARME '95, Frankfurt/Main, Germany, October 2-4, 1995, Proceedings*, volume 987 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 1995.

[86] Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006.

[87] Michael Ummels. The complexity of nash equilibria in infinite multiplayer games. In *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 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*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2008.

[88] Ron van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 280, 2004.

[89] Ron van der Meyden and MosheY. Vardi. Synthesis from knowledge-based specifications. In Davide Sangiorgi and Robert Simone, editors, *CONCUR'98 Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 1998.

[90] Ron van der Meyden and Thomas Wilke. Synthesis of distributed systems from knowledge-based specifications. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 562–576. Springer, 2005.

[91] Moshe Y. Vardi. Alternating automata: Unifying truth and validity checking for temporal logics. In *Automated Deduction - CADE-14, 14th International Conference on Automated Deduction, Townsville, North Queensland, Australia, July 13-17, 1997, Proceedings*, volume 1249 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 1997.

[92] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

[93] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths (extended abstract). In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 185–194. IEEE Computer Society, 1983.

[94] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Index

List of Figures

List of Tables

Summary

We study the problem of automatic synthesis of programs in multi-component architectures such that they fulfill the specifications by construction. The main goal of the thesis is to develop procedures to solve the synthesis problem that may lead to efficient implementations.

Each component has partial observation on the global state of the multi-component system. The synthesis problem is then to provide observation-based protocols for the components that have to be synthesized that ensure that specifications hold on all interactions with their environment.

The environment may be antagonist, or may have its own objectives and behave rationally. We first study the synthesis problem when the environment is presumed to be completely antagonist. For this setting, we propose a "Safraless" procedure for the synthesis of one partially informed component and an omniscient environment from KLTL$^+$specifications. It is implemented in the tool Acacia-K.

Secondly, we study the synthesis problem when the components in the environment have their own objectives and are rational. For the more relaxed setting of perfect information, we provide tight complexities for particular ω-regular objectives. Then, for the case of imperfect information, we prove that the rational synthesis problem is undecidable in general, but we gain decidability if is asked to synthesize one component against a rational omniscient environment.

Key words: synthesis, reactive systems, games, KLTL

Résumé

Nous étudions le problème de la synthèse automatique de programmes dans des architectures multi-composants teles qu'elles respectent les spécifications par construction. Le principal objectif de cette thèse est de développer des procédures pour résoudre le problème de synthèse qui peut conduire à des implémentations efficaces.

Chaque composant a une observation partielle sur l'état global du système multi-composants. Le problème est alors de fournir des protocoles basés sur les observations afin que les composants synthétisés assurent les spécifications pour tout le comportement de leur environnement.

L'environnement peut être antagoniste, ou peut avoir ses propres objectifs et se comporter de façon rationnelle. Nous étudions d'abord le problème de synthèse lorsque l'environnement est présumé antagoniste. Pour ce contexte, nous proposons une procédure "Safraless" pour la synthèse d'un composant partiellement informé et un environnement omniscient à partir de spécifications KLTL$^+$. Elle est implémentée dans l'outil Acacia-K.

Ensuite, nous étudions le problème de synthèse lorsque les composants de l'environnement ont leurs propres objectifs et sont rationnels. Pour le cadre plus simple de l'information parfaite, nous fournissons des complexités serrées pour des objectifs ω-réguliers particuliers. Pour le cas de l'information imparfaite, nous prouvons que le problème de la synthèse rationnelle est indécidable en général, mais nous regagnons la décidabilité si on demande à synthétiser un composant avec observation partielle contre un environnement multi-composante, omniscient et rationnel.

Mots clés: synthèse, systèmes réactifs, jeux, logiques