A contribution to formal methods : games, logic and automata

David Janin

N° d'ordre: 333

UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES et INFORMATIQUE
Laboratoire Bordelais Recherche en Informatique

Rapport de recherche de

David Janin

présenté pour l'obtention d'une

HABILITATION À DIRIGER LES RECHERCHES

SPÉCIALITÉ : INFORMATIQUE

Contributions aux fondements des méthodes formelles : jeux, logique et automates

Contributions to formal methods: games, logic and automata

Soutenance le : 5 décembre 2005
Devant le jury composé de :

Président	Bruno Courcelle	Professeur (Université Bordeaux 1)
Rapporteurs	Erich Grädel	Professeur (RWTH Aachen)
	Anca Muscholl	Professeur (Université Paris VII)
	Damian Niwiński	Professeur (Universytet Warszawski)
Examinateurs	Hubert Comon-Lundh	Professeur (ENS Cachan)
	Yves Métivier	Professeur (ENSEIRB)
Invité	André Arnold	Professeur (Université de Bordeaux I)

– 2005 –

A mes filles, Louise, Margaux, Sarah et Julie !

Avant-propos

«Dis papa : quand est-ce que tu passes ton habilitation ?»
Anonyme(s), XXIième siècle

Je présente ici une thèse d'habilitation à diriger les recherches. Ce document est, non seulement, une synthèse de mes travaux de recherche en informatique théorique, mais aussi, car ils recouvrent largement le domaine, un parcours plutôt complet de la logique, la théorie des automates (ou la théorie des calculs de points fixes) et de la théorie des jeux appliquées à la spécification, la vérification ou la synthèse de systèmes discrets finis.

Domaine de recherche

La logique, les automates, les calculs de points-fixes et les jeux offrent un fondement théorique aux méthodes formelles de la façon suivante.

La *logique* offre un cadre *descriptif* à ces méthodes en proposant un langage et des outils qui permettent de définir formellement, c'est à dire sans ambiguïté, les systèmes à concevoir à travers la description des propriétés qu'ils devront satisfaire. Les notions de modèles, formules, théories, satisfactions, cohérences, etc..., font partie des notions couramment utilisées. Les concepts, résultats et techniques issues de la logique s'appliquent donc naturellement aux problèmes liés à la *spécification* des systèmes.

La *théorie des automates* offre le cadre *opérationnel* naturellement associé à ces outils logiques de spécifications. Les formules, compactes, construites avec des opérateurs très expressifs ont en effets des algorithmes de traitement généralement complexes, même non élémentaires. Lorsqu'elles peuvent se traduire en automates équivalents, dont la sémantique est plus opérationnelle, on dispose alors d'algorithmes beaucoup plus efficaces, PTIME ou NPTIME, de *vérification* ou de *synthèse* des systèmes spécifiés en logique.

Les *calculs de points-fixes*, étroitement reliés à la théorie des automates, offrent une description inductive (et co-inductive) des langages définis à l'aide d'automates. Ils donnent ainsi une description plus *mathématique* des concepts mis en oeuvre dans les automates d'arbres infinis. Par exemple, le critère de parité apparaît comme une traduction opérationnelle, dans les automates, de l'alternance d'expressions de plus grands et plus petit points-fixes.

La *théorie des jeux* offre, quant à elle, le cadre *combinatoire* sous-jacent à l'utilisation de ces techniques algorithmiques. Elle permet une modélisation précise, et uniforme, des interactions qu'il peut exister entre les systèmes conçus et les procédures de vérification mises en oeuvre. En particulier, la théorie des jeux capture la *complexité* des méthodes liés à l'utilisation des automates. Les algorithmes de synthèse ou de vérification de systèmes se réduisent polynomialement sinon linéairement à la résolution de jeux formels de même taille.

Bien sûr, cette répartition entre logique, automates et jeux n'est pas aussi stricte qu'il y paraît. Les frontières entre ces trois domaines sont parfois poreuses.

Par exemple, une fois étudié et compris le pouvoir d'expression des automates dans le cadre de référence qu'offre la logique mathématique on peut, tout aussi bien, manipuler des langages de spécification plus spécialisés tels que les logiques temporelles ou les calculs de points fixes. Ces langages se situent quelques part entre la logique mathématique et les la théorie des automates puisqu'ils s'appuient sur des concepts et des outils qui viennent, selon le cas, de l'un ou de l'autre.

Autre exemple, on peut comprendre les jeux comme un cas particulier d'automates d'arbres qui définissent, sur leur propre structure, des ensembles d'arbres : les stratégies gagnantes. Ce point de vue pourra même être appliqué de façon pertinente en appliquant, sur les jeux eux-mêmes, des résultats classiques de théorie des automates.

Les jeux eux-mêmes empruntent beaucoup à la logique puisque qu'ils peuvent être vus comme des sortes de formules booléennes infinies - on traite ici de jeux infinis - dont on souhaite évaluer la satisfaisabilité. Construire des systèmes formels de raisonnement sur les jeux, par exemple construire des stratégies gagnantes à partir d'autres stratégies existantes, peut ainsi s'appuyer sur les outils et notions déjà développés en théorie de la démonstration.

Principaux résultats

Mon activité de recherche s'inscrit, précisément, dans l'étude et le développement des liens étroits qui existent entre logique, automates, calculs de points-fixes et jeux. Plus précisément, mes principaux résultats sont les suivants :

1. Etude du non déterminisme tel qu'il est manipulé en logique modale et obtention d'un théorème de normalisation ; application à la caractérisation du pouvoir d'expression du fragment invariant par bisimulation de la logique monadique du second ordre (MSO) sur les structures amorphes (voir [92] et [93]),
2. Etude du mu-calcul modal et de ses liens avec la théorie des automates alternants ; normalisation - par théorème de simulation - des formules du mu-calcul modal en formules dites disjonctives - analogue logique (et modal) de la notion d'automates non déterministe sur les arbres binaires ; applications à la décidabilité et à la propriété dites du modèle fini (voir [103] et [93] - résultats obtenus en collaboration avec Igor Walukiewicz),
3. Etude des liens entre mu-calcul modal et logique monadique du second ordre via la notion d'invariance par bisimulation ; caractérisation du pouvoir d'expression du mu-calcul modal comme fragment invariant par bisimulation de MSO ; application

à l'étude du mu-calcul quantifié ; démonstration simple de l'interpolation uniforme pour le mu-calcul (voir [93], [104] et section 4.3 dans ce rapport - résultats obtenus en collaboration avec Igor Walukiewicz),

4. Définition et étude d'un calcul de points-fixes abstrait ; généralisation de la notion de formules disjonctives ; obtention, sous hypothèses de commutation locale adéquates, d'un théorème de simulation des formules quelconques par des formules disjonctives ; applications à la décidabilité (voir [94] et [95]),
5. Extension des résultats d'invariance par bisimulation aux mu-calcul avec «counting modalities» et la notion d'invariance par «counting bisimulation» ; nouvelle [1] caractérisation du pouvoir d'expression de MSO sur les arbres relationnels à l'aide d'expressions de point-fixes (voir [100] et chapitre 4.2.1 dans ce rapport - résultats obtenus en collaboration avec Giacomo Lenzi),
6. Etude des liens entre les niveaux de la hiérarchie d'alternance de quantificateurs (en logique monadique du second ordre) et les niveaux de la hiérarchie d'alternance de points-fixes (en mu-calcul) ; établissement d'une correspondance entre les premiers niveaux (voir [98] et [99]) ; plus précisément : caractérisation logique du pouvoir d'expression des premiers niveaux de la hiérarchie du mu-calcul en établissant une correspondance entre :
   (a) le niveau Σ_1 de la logique monadique et le niveau N_1 du mu-calcul - les langages reconnaissables et fermés au sens de la topologie préfixe - (voir [100]),
   (b) le niveau Σ_2 de la logique monadique et le niveau N_2 du mu-calcul - les langages définissables à l'aide d'automates de Büchi - (voir [101]),

   en établissant aussi que cette correspondance ne peut aller au delà (voir [99] - résultats obtenus en collaboration avec Giacomo Lenzi),
7. Etude de la restriction du mu-calcul aux modèles finis ; étude, dans le niveau monadique Σ_1 et étude des «graphs acceptors» invariant par bisimulation ; caractérisation partielle de ces derniers par normalisation en accepteurs de graphes avec spécifications arborescentes des voisinages ; étude et caractérisation de la logique monadique du second ordre sur les graphes unaires ; caractérisation de MSO sur les représentations finies de mots infinis (voir [50] et section 6.1 dans ce rapport - résultats obtenus en collaboration avec Anuj Dawar),
8. Etude des clôtures par booléens ou par quantificateurs du premier ordre des niveaux de la hiérarchie monadique sur les graphes finis ; résultats de séparations (voir [102] - résultats obtenus en collaboration avec Jerzy Marcinkowski),
9. Etude des jeux de parité ; définition de la notion de stratégies permissives dans ces jeux ; mise au point d'un algorithme de calcul de ces stratégies ; comparaison avec les algorithmes existants de résolution des jeux de parités (voir [26] et Thèse de Julien Bernet - résultats obtenus en collaboration avec Igor Walukiewicz et Julien Bernet),
10. Etude des relations de simulations entre jeux de parité ; définition de plusieurs niveaux de simulation (asynchrones, synchrones, généralisées) ; résultats de corrections pour l'utilisation de ces simulations comme outils de démonstrations sur les jeux (voir section 2.3 dans ce rapport),

1. une première caractérisation est obtenu par Walukiewicz [181]

11. Définition et étude d'un modèle de jeux étendus, les jeux distribués, pour le traitement uniforme des problèmes de synthèse de programmes distribués ; mise en évidence de l'applicabilité - et application - de la théorie classique des automates infinis à la résolution des jeux distribués linéaires ; démonstration de l'équivalence entre jeux synchrones et jeux asynchrones dans le cas des stratégies distribuées à mémoire finie (voir [25] et thèse de Julien Bernet - résultats obtenus en collaboration avec Julien Bernet).

Ces résultats, sont présentés, en anglais, dans le document qui suit.

L'étude systématique à laquelle je me suis livrée des liens qui existent entre logique, théorie des automates, calculs de points-fixes et théorie des jeux - dans le cadre des modèles discrets - fait que l'ensemble de ces résultats offre une bonne couverture du domaine. Le document qui suit, qui est ma *thèse d'habilitation*, peut aussi constituer la base d'un cours de niveau Maîtrise.

Perspectives

Je présente ici les problèmes ouverts et les perspectives de recherches qui, volontairement, sont étroitement liés à mes travaux.

Jeux à deux joueurs

Le problème ouvert majeur dans le domaine des jeux de parité est le suivants :

Problème 1 *La résolution des jeux de parité est-elle PTIME ?*

Une meilleur compréhension des propriétés mathématiques des jeux de parités - et des stratégies gagnantes qu'ils définissent - contribuera, sans doute, à la résolution de ce problème.

Expressions de points-fixes relationnels pour la définition (et le calcul) des positions gagnantes. A l'exception, notable, de l'algorithme de Vöge et Jurdzinski dont la complexité est, à ce jour, inconnue [179], la plupart des algorithmes de résolution des jeux de parité tend a s'appuyer, implicitement, sur une description monadique de l'ensemble des positions gagnantes. Le théorème d'invariance par bisimulation sur MSO et l'infinitude de la hiérarchie du mu-calcul semble alors indiquer que ces algorithmes se ramèneront, peu ou prou, à l'évaluation de formules du mu-calcul arbitrairement complexes, c'est à dire à la résolution de jeux de parité quelconques !

On peut éviter ce point de vue circulaire en s'attachant à définir les positions gagnantes dans les jeux de parités à l'aide d'expressions de points-fixes sur des relations binaires (ou plus). Bien sûr, la caractérisation partielle de PTIME par la logique du premier ordre étendu par points-fixes inductifs tend à suggérer que si la résolution de ces jeux est effectivement polynomiale, ce programme de recherche à toutes les chances d'aboutir ..., a posteriori.

Dans une démarche a priori, on peut aussi, par exemple, tenter de résoudre le problème suivant :

Problème 2 *Existe t'il, pour chaque entier strictement positif n, un jeu canonique $\mathcal{G}_n$ de taille polynomiale en n tel que, pour tous jeux de parité $\mathcal{G}$ de taille au plus n, il existe*

une relation de simulation asynchrone (ou étendue) permettant de simuler toute position gagnante de $\mathcal{G}$ par une position gagnante de $\mathcal{G}_n$.

Dans l'affirmative, si le calcul d'une telle simulation est, de plus, PTIME, le problème de la complexité des jeux de parité est résolu.

C'est une tentative dans ce sens, avec une simulation synchrone, qui nous a conduit à l'algorithme des stratégies permissives [26].

Système de preuves complet pour les jeux de parité. Les diverses notions de relation de simulation entre jeux définies dans ce rapport s'apparentent à des systèmes de démonstrations. En effet, les positions des joueurs peuvent être vue comme des conjonctions et des disjonctions. Il y a une analogie certaine entre les notions de simulation présentées dans ce rapport et les règles de raisonnement du calcul des séquents de Gentzen.

Nous avons démontré la correction de ces simulations (voir Lemme 2.3.3.5). On peut se poser la question de leur éventuelle complétude.

Plus précisément : sur les jeux de parité finis, étendus à l'aide de variables libres, toute position peut-être vue comme une formule propositionnelle sur ces variables. On peut alors dire qu'un séquent de positions $\overline{x} \vdash \overline{y}$ est valide lorsque sa traduction à l'aide de formules booléennes est valide.

Il vient :

Problème 3 *La notion de simulation généralisée (étendue aux variables libres) est-elle complète, i.e. tout séquent valide de positions de jeux de parité est-il prouvable à l'aide d'une relation de simulation généralisée ?*

Remarquons cependant que la définition actuelle s'appuie sur une condition globale de transfert des critères de gains infinitaires, i.e. cette condition doit être vérifiée une fois la relation de simulation construite. Il vient :

Problème 4 *Comment étendre la notion de simulation généralisée pour garantir localement la condition de transfert, des hypothèses vers les conclusions, des conditions infinitaires de gains ?*

Normalisation des indices de parité dans les jeux. Pour chercher une axiomatisation complète de la validité des séquents dans les jeux de parité avec règles de preuves purement locales, on peut aussi s'appuyer sur l'axiomatisation de Kozen du mu-calcul dont Walukiewicz a démontré la complétude [180]. Tout jeu de parité peut, en effet, être traduit en mu-calcul booléen (ou même en système d'équations de points-fixes [16]) et les conditions de parité du jeu se retrouvent codées par l'alternance de plus grand et plus petit points-fixes dans sa traduction.

Cependant, dans cette modélisation par formule de points-fixes, (1) le jeu est - implicitement - développé en arbres avec arcs retour, et (2) la traduction des indices de parité en alternance de points-fixes induit une normalisation des conditions de gains que le développement du jeu rend difficile à suivre.

Il vient :

Problème 5 *Pour un jeu de parité donné, quels sont les changements d'indices de parité des positions qui préservent l'ensemble des stratégies (ou des parties) gagnantes ? Peut-on définir un coloriage canonique ?*

Les ω-semigroupes [146], qui offrent une représentation canonique des langages ω-rationnels, pourraient être utilisés pour cela. Plus encore, ils pourraient même permettre de traiter tout jeux régulier, à condition, sans doute, de comprendre au sein des ω-semigroupes, comment est implicitement codée la mémoire de ces stratégies régulières.

Automates et graphes finis

Dans le cas des graphes finis, le problème ouvert majeur lié aux travaux présentés ici, est un problème énoncé par Anuj Dawar et Martin Otto au milieu des années 90.

Problème 6 *Est-il vrai, comme dans le cas de graphes quelconques, que le fragment de la logique monadique du second ordre invariant par bisimulation* sur les modèles finis *est égal au mu-calcul modal ?*

Malgré quelques réponses partielles [144, 50], ce problème semble toujours difficile. Même restreint au niveau monadique Σ_1, le problème reste ouvert. L'invariance par bisimulation autorisant la définition, sur les graphes finis, de problèmes difficiles pour le niveau $N_2 \cap M_2$, on sait seulement que les correspondances établies dans le cas général ne tiennent plus dans le cas fini.

Les travaux et résultats obtenus dans le cas de monadique Σ_1 soulèvent par ailleurs les questions suivantes.

Accepteurs de graphes étendus par coloriage d'arc. La notion d'accepteur de graphes, définie par Thomas [173] comme extension des automates d'états finis sur les mots ou arbres finis, capture le pouvoir d'expression de monadique Σ_1.

Il apparaît cependant que, sur les (codages des) mots ou arbres finis, chaque arc est complètement déterminé par son sommet cible. On peut donc, indifféremment, définir le calcul d'un automate par coloriage des sommets ou des arcs. Sur les graphes quelconques, cela n'est plus vrai. Un coloriage des seuls sommets induit une extension de la notion d'automate qui peut sembler artificiellement limitée. On constate par exemple une correspondance irrégulière entre le niveau monadique Σ_1 et le mu-calcul ; ce premier ne capture que partiellement le niveau $N_2 \cap M_2$ de ce dernier. Pour remédier à cela, on peut se placer dans le cadre de la logique MSO_2 définie par Courcelle [42, 44].

Il vient :

Problème 7 *Quel est le pouvoir d'expression des accepteurs de graphes* étendus *par coloriage des arcs ?*

Ce problème est actuellement à l'étude en collaboration avec Dietmar Berwanger.

Accepteur de graphes étendus avec conditions infinitaires. De la même façon que les automates de mots et d'arbres sont étendus aux mots et arbres infinis par ajout de conditions infinitaires sur les chemins infinis, on peut se demander que devient la notion d'accepteurs de graphes étendue de même.

L'étude en cours de cette piste de recherche, en collaboration avec Dietmar Berwanger, montre que : (1) ajouter des conditions infinitaires régulières aux accepteurs de graphes échoue à capturer le pouvoir d'expression du mu-calcul, (2) ajouter des conditions infinitaires de parité aux accepteurs de graphes étendus par coloriage des arcs échoue, aussi, à

capturer le pouvoir d'expression du mu-calcul, et, par contre, (3) ajouter des conditions infinitaires régulières aux accepteurs de graphes étendus par coloriage des arcs donne, sur les graphes finis, un langage au moins aussi expressif que le mu-calcul.

Il vient :

Problème 8 *Quel est le pouvoir d'expression des accepteurs de graphes* étendu *par coloriage des arcs et conditions infinitaires régulières ? Les propriétés ainsi définies sont-elle nécessairement PTIME ?*

Jeux distribués

La définition, puis l'étude des jeux distribués, en collaboration avec Julien Bernet, Swarup Mohalik et Igor Walukiewicz, vise à offrir un cadre uniforme et minimaliste aux traitements des problèmes de synthèse de programmes d'états finis, dont, en particulier, les programmes distribués.

Il convient maintenant, puisqu'il s'agit là d'un modèle récent, inventé au LaBRI, de poursuivre encore, par des travaux de recherche adéquats, la défense de sa pertinence théorique et de son applicabilité pratique. On peut distinguer, en particulier, les trois axes de recherche suivants.

Universalité des jeux distribués. Quels modèles de synthèses de programmes peuvent encore être encodés et résolubles dans le cadre des jeux distribués ?

Liens avec d'autres théories. Quels outils, provenant d'autres théories telles que la théorie des automates, ou la théorie de la démonstration, sont applicables aux jeux distribués ?

Applicabilité. Quels problèmes concrets, logiciels ou matériels, peuvent être modélisés et résolus dans le cadre des jeux distribués ? Quelles limites impose l'utilisation des jeux distribués dans de telles modélisations ?

Répondre à ces questions conduira sans doute à enrichir de concepts ou de structures la notion de jeux distribués. La manipulation de données arbitraires, l'étude des flux d'informations possibles entre les joueurs, la résolution d'autre problèmes théoriques encore ouverts, etc. . ., sont autant de pistes qui restent à explorer !

Remerciement

Je souhaiterais remercier ici André Arnold qui fut, jusqu'en 1996, mon directeur de thèse, et, depuis, un interlocuteur privilégié ; Bruno Courcelle dont la précision de la pensée et les travaux qui en découlent, restent un exemple pour moi ; Igor Walukiewicz qui fut et est encore le collaborateur le plus efficace et le plus constant que j'ai la chance d'avoir ; Giacomo Lenzi qui, à l'occasion d'un long postdoc qu'il effectua au LaBRI, fut une source constante d'inspirations tant comme chercheur que comme enseignant ; Julien Bernet qui a le privilège ambigu d'être mon premier étudiant thèse ; Pascal Weil, responsable de l'équipe L3A, qui s'est fait un devoir de remplir ce rôle. A tous, merci !

Bien sur, j'omets de remercier beaucoup de monde, la liste serait longue. De nombreux autres chercheurs à Bordeaux, mais aussi à Paris, à Rennes, à Varsovie, à Aachen, à Cambridge, à Wrocław, ..., souvent du réseaux de recherche européen GAMES, m'ont largement aidé, à travers leur culture informatique et mathématique, leur talent pédagogique, leur courage scientifique, à mieux comprendre et mieux apprécier le métier de chercheur en informatique : son histoire, ses enjeux, ses méthodes, ses outils, ses plaisirs, ses difficultés et ses contraintes. Merci à tous !

J'ai eu aussi l'occasion de participer, puis de diriger quelques projets de transferts technologiques. Pour cela, j'ai largement bénéficié de l'expérience de Richard Castanet qui m'en a ouvert les portes ainsi que du soutien et des conseils de Véronique Bogati à la cellule LaBRI-transfert. Je les en remercie vivement.

Enfin, je souhaite remercier tout particulièrement Anca Muscholl, Erich Graedel et Damian Niwinski qui ont accepté d'être les rapporteurs de cette habilitation, ainsi que André Arnold, Hubert Comon-Lundh, Bruno Courcelle, Yves Métivier qui ont accepté d'en compléter le jury.

David Janin,
Bordeaux, le 21 novembre 2005.

Introduction

The Way that can be told of is not an Unvarying Way;
The names that can be named are not unvarying names.

Lao-Tseu, Tao Te Ching (Tr. A. Waley)

A brief historical overview

In the sixties, Büchi [35] and Rabin [152], prove the decidability of the monadic second order theory of the infinite word (S1S) and the infinite binary tree (S2S). These two difficult results are obtained by means of an automata characterization of the expressive power of monadic second order logic (MSO). In mathematical logic, they open the way to many other decidability results. The use of automata theory as an algorithmical and combinatorial tools then spreads in the research community. Following these works, many other results [134, 109, 38, 133, 153, 81, 145] have completed, developed and enriched this theory that is now central in many other fields.

In computer science, for instance, the increasing complexity of computerized applications requires, since the seventies and more and more, formal methods that help designers to guarantee their reliability. These methods must, as much as possible, be grounded upon a solid and well-defined theory. Automata theory on infinite objects probably finds here one of its most spectacular application field.

In fact, words or trees, whether they are finite or infinite, can be seen as simple models of systems behaviors [89, 127, 52, 7, 88, 85]. It follows that logic offers an adequate language for the formal modeling and specification of these behaviors. Such observations leads to an increase of interest for Büchi and Rabin works. Since then, a significant number of important results have been proved. They form, altogether, the mathematical basis for modeling, specifying, analyzing and synthesizing interactive discrete computerized systems [78].

The work presented in this report lays in this quite recent and still evolving fundationnal research field.

Word language theory and automata

It is first infinite words language theory that is developed towards applications to formal methods in computer science [38, 158, 80, 63].

In fact, a process behavior can simply be modeled by the sequence of the elementary actions it executes. These actions being seen as letters of an alphabet, the process behavior is, from this point of view, a finite or - even more often - an infinite word. It follows that the specification of the behavior expected from a process can be made by describing the set - also called language - of the words modeling the many possible correct behaviors. In turn, this specification can be formally described in monadic second order logic. The works of Büchi and others can be applied.

For instance, deciding the existence of a process satisfying the behavioral specifications amounts in deciding the satisfiability of the corresponding logical specification. It follows that building an automaton equivalent to a specification leads to the definition of an effective and efficient *verification* procedure to check that a given model of this process satisfies this specification [177]. From this automaton one can also *synthesize* the model of a process that does satisfy this specification [154, 39].

Although conceptually attracting, this approach may fall on the quite complex syntax and associated algorithms used with MSO. Many specialized languages - called temporal or program logics [109, 149, 57, 145, 63] - have been defined to remedy to this fact. Easier to use, all these languages still have the property of being translatable in MSO. It follows that the theory and tools developed from Büchi's works can still be applied efficiently [177].

Since then, many software tools implement this language theory and associated tools. After adequate modeling of computerized systems, these tools allow formal verification of the correctness of these systems. They have already been efficiently used. Among others, the software SPIN/PROMELA has been successfully used for certifying the correctness of many communication protocols [91].

Tree language theory and branching time logic

From the early eighties, the development of concurrency theory, that aims at defining the behavior of the process interacting with its environment, leads to model process behaviors as trees instead of words [84, 127, 86, 85, 88].

In fact, with the word models alone, especially in presence of systems inputs, one may fail to describe the way the expected behaviors of a process can be conditioned, at every moment, by the environment in which it evolves. Conceptually, there is a shift from a linear notion of time to a branching notion of time[70, 71].

But here again, automata theory, with Rabin's theorem as a cornerstone, offers a conceptual, combinatorial and algorithmic tool case applicable to this richer approach [171]. Here again, specialized languages are developed and proposed, in place of MSO, for the specification of tree shaped models of process behaviors such as the branching time temporal logics CTL, CTL*[149, 56, 60] or logics based on fixed point such as the mu-calculus [57, 151, 110, 18, 138, 169].

While much richer from a theoretical point of view, this branching time approach is, so far, much less developed towards application than the linear time approach.

The high complexity of the various concepts it handles, compared to the much simpler setting of infinite words, probably explain this. Application fields - and both conceptual and technical tools - probably still need to be developed in order to penetrate more the world of industrial applications.

From a didactic point of view, it also seems that available textbooks are still oriented towards researchers or PhD students [16, 78]. There is probably a need for more elementary books, say for average master students, which would slowly open the way into this rich and difficult theory.

Two player games in formal methods

The necessity of achieving a better understanding of the underlying theory was probably one of the main motivation for introducing, in the seventies, two player discrete game theory into tree automata theory [81]. It occurs that game theory offers an ideal setting for the uniform treatment and understanding of many theoretical problems encountered in this field. In fact, deciding the emptiness problem for a non deterministic tree automaton amounts to solving an infinite two players game: a *satisfiability game*. Deciding if an alternating automaton accepts a given model can also be made - or even defined - by means of solving a two player game: a *model-checking game* [167].

In these two reductions to game, the roles played by the two player AND and OR already differs. In a satisfiability game: the player OR builds a model and proves it is accepted while, at the same time, the AND player (arbitrarily) chooses branches from which this construction/proof have to be continued. In a model-checking game: the roles of the player AND and the player OR are to handle, moreover, the arbitrary conjunctions and disjunctions that occur in automata transition specifications.

Other roles can also be assigned to players. For instance, defining the player OR as a process that evolved in an environment governed by the player AND, solving a game amounts to proving that there exists a program for the process (encoded by a strategy) that fulfilled a desired task (encoded in the rules of the game and its winning conditions) in an environment that may have various behavior. In other words, games can also be seen as versatile models of interacting systems.

On a conceptual point of view, game theory may enable clear, abstract and unambiguous definitions of various phenomena that commonly occur in computerized systems. In a more application oriented perspective, game theory may offer concrete models of system specifications: games, and algorithms to solve these specifications: winning strategies. Increasing the number of players having partial information about the play globally performed may even describe more complex phenomena that occur in distributed systems.

Objective of the report

This report first aims at describing the contribution of the author to these fields: this document is a research habilitation thesis. for this reason, it first describes in detail the author's own results obtained during the last decade [25, 50, 101, 26, 100, 99, 102, 98, 94, 95, 93, 104, 103, 92].

However, these results were achieved while aiming at studying, developing and finding new relationships between logic, automata, fixed points calculus and games. It occurs that

these results, altogether, offer quite a good coverage of the field. Writing a monograph that could also form the basis of a one semester graduate lecture became a secondary goal. It follows that a particular attention has also been paid, in writing this document, to give as much as possible simple and uniform proofs of the many results that are presented.

Observe that, in this text, some results were, to some extent, already known in the field.

For instance, in Chapter 4, we do provide an original characterization of MSO on unranked and unordered trees by means of the counting mu-calculus and give a complete proof of it. No doubt however that this result would have been considerably more difficult to achieve - not to say more - without, especially, Rabin's original result on the binary tree [152], Gurevich and Harrington game based approach [81], Muller and Schupp definition of alternating automata and simulation theorem [136, 137], and, more specifically for the author's own understanding, many other works, e.g. [139, 171, 27, 10, 181].

This is especially true concerning the relationship between monadic second order logic and tree automata [152, 153, 168, 81, 171, 27, 174, 137, 10, 181, 182], and the relationship between fixed point calculus and automata [138, 139, 58, 167, 167, 56, 59, 54, 60, 16] that have been also discovered, understood, analyzed, and detailed for many years by many authors in various settings: from binary trees to tree-like structures, via unranked trees with ordered or unordered successors, possibly modulo bisimulation, etc...

In some sense, these various concepts, techniques and notions were already between the lines of Rabin's original article. However, observing the long list of research publications - and the number of years - that have occurred after Rabin published his seminal result, there is no doubt that they deserved to be better understood, specifically explained, related to other fields or techniques, better distinguished one from the other, cut into smaller and simpler concepts and tools, and, moreover, extended to more general settings requiring new proofs and, sometimes, even new technicalities and concepts.

The present document follows this quite long research tradition.

Structure and contribution of the report

The present report is organized as follows.

In the first chapter, we review some standard notations and concepts that are used all through this report. Words and word automata, transition systems, bisimulation relations, trees, MSO and the mu-calculus are presented there. Some original technicalities such as graph κ-expansions also defined and studied.

Two player games are presented in the second chapter. We first review some classical definitions and results on discrete two player games from regular, to parity and safety games.

An algorithm that solves parity games while computing what we call *permissive strategies* is presented. To some extent, this algorithm is very close to Jurdzinski's small progress measure algorithm [108]. However, the emphasis made on permissiveness not only shed a new light on this algorithm but also gives new results and raises several new open problems. A presentation of this algorithm has already been published in a joint work with Bernet and Walukiewicz [26]

We conclude this chapter by describing several notions of game simulations. These simulations are used in the sequel to compare and prove game (or automata) equivalence in a more uniform way.

A generalization of Muller and Schupp alternating tree automata [136], with transition specifications expressed by means of arbitrary fixed point formulas, is defined in the third chapter. Automata runs are defined in terms of strategies in model checking games. Several operations and automata transformations such as boolean compositions and general substitutions are presented and semantically characterized. This leads to relate the expressive power of alternating automata with the expressive power of the mu-calculus in an inductive way. The mu-calculus invariance properties under bisimulation are obtained as corollaries.

This chapter is a detailed presentation and a generalization of techniques and concepts that have been first used during a collaboration with Walukiewicz [103] and latter extended and partially published by the author himself [94, 95].

The relationship between the fixed point calculus and MSO is studied in the fourth chapter. An original semantical notion of non alternating tree automata is defined and characterized syntactically by means of - an extension of - non deterministic automata. A simulation theorem, *à la* Muller and Schupp [137], is then established to prove expressiveness equivalence between alternating and non alternating automata. Further proving that non deterministic tree languages are closed under projection, this series of results shows that on trees (resp. on κ-expanded trees) MSO is as expressive as the counting mu-calculus (resp. the modal mu-calculus).

On arbitrary graphs, these results provide a characterization of the expressive power of the counting bisimulation invariant (resp. bisimulation invariant) of MSO by the counting mu-calculus (resp. the modal mu-calculus). The bisimulation invariance result has been established in a collaboration with Walukiewicz [104].

Several applications are given at the end of this chapter.

The fifth chapter shows that there is even a finer correspondence between the first levels of the quantifier alternation depth hierarchy of monadic second order logic and the fixed point alternation depth of the mu-calculus. More precisely, up to the monadic Σ_2 level, the bisimulation invariance correspondence still holds level by level.

Classical model theoretical notions such as ultraproducts are reviewed and then used in order to achieve these finer characterizations. The case of levels above the monadic Σ_2 level are considered at the end of the chapter.

These results have been established in a joint work with Lenzi [101, 100, 99, 98]

The finite model case, that is quite distinct from the general case, is investigated in the sixth chapter.

A Büchi like characterization theorem of MSO on finite representations of infinite (ultimately periodic) words, obtained in a joint work with Dawar [50], is established. Extended to arbitrary finite graphs, this result provides counter examples to properties of bisimulation invariance one could have expected to hold in the finite.

Graph acceptors, which generalizes to arbitrary finite graphs the notion of finite state automata on finite words or trees, are reviewed. We study bisimulation invariant properties which are definable by graph acceptors.

Various separation results within the first order closure of levels of the monadic hierarchy are also proved. These results were obtained with Marcinkowski [102].

System modeling by means of game is proposed in the seventh chapter. For this purpose, a notion of distributed games is defined and studied. In particular, we show that many distributed synthesis problems can be encoded and solved in this setting. Complexity issues are also addressed. This work is partially the result of a collaboration with Bernet, Mohalik and Walukiewicz [25, 128]. It is also the core of Bernet's PhD thesis under the author's supervision.

Precise open problems, or more general research directions to be developed are presented in the last chapter.

Acknowledgment to co-authors

This report is signed by the author (alone) who claims the following text - and possible mistakes - to be his own. However, as mentioned above, several parts of this text are extracted and adapted from published material resulting from collaborations with others. So one shall keep in mind that, whenever needed, the related original concepts, definitions and results must be also accredited to, in historical order, Igor Walukiewicz, Giacomo Lenzi, Jerzy Marcinkowski, Xavier Briand [2], Anuj Dawar, and Julien Bernet [3].

2. in a master thesis in 2002 under the author's supervision
3. in a master and a PhD thesis under the author's supervision from 2001 to 2006

Contents

Chapter 1

Preliminaries

Dessine moi un mouton !
Le petit prince, Antoine de Saint Exupery

In this chapter, we first review standard notations, definitions of labeled oriented graphs (also called transition systems) and bisimulation equivalences. We also review first-order logic (FO) and monadic second order logic (MSO) over transition systems. We define the modal and the counting mu-calculus as fragments of monadic second-order logic. An we also review some classical properties of word languages and automata theory.

1.1 Notations and standard concepts

An *alphabet* is a finite set Σ. The elements of alphabet Σ are called *letters* and are written a, b, c, etc. A *finite word* on alphabet Σ is a finite sequence w of letters of Σ. We write $w = a_0.\cdots.a_{n_1}$ with a_0, a_1, ..., $a_{n-1} \in \Sigma$, such a word w, the *length* of word w, written $|w|$, defined to be n. Equivalently, a word can also be seen as a partial function from $\mathbb{N}$ to Σ with domain $dom(w) = [0, |w| - 1]$ and $w(i) = a_i$ for each $0 \leq i < |w|$. The empty word (the unique word with empty domain) is written ϵ.

The set of all finite words on alphabet Σ is written Σ^*, and the set of all words but the empty word is written Σ^+. Set Σ^* is equipped with the catenation operation written multiplicatively, i.e. the catenation of two words v and $w \in \Sigma^*$ is written $v.w$. The structure $\langle \Sigma^*, . \rangle$ is a monoïd.

A *language of words* on alphabet Σ is any subset L of Σ^*. Given any two languages L and $M \subseteq \Sigma^*$, we write $L + M$ the union of the languages and $L.M$ the element-wise extension of concatenation to languages, i.e. $L.M = \{u.v \in Sigma^* : u \in L, v \in M\}$. For every language $L \subseteq \Sigma^*$, we put $L^0 = \{\epsilon\}$ and, for every $n \in \mathbb{N}$, $L^{n+1} = L.L^n$. We write then $L^* = \bigcup_{0 \leq n} L_n$ and $L^+ = \bigcup_{0 < n} L^n$. Language L^* is called the *Kleene star* of language L. A language $L \subseteq \Sigma^*$ is a *regular language* when it can be defined from finite languages and sum, product or star operation.

In the forthcoming text, we use ordinals and cardinals. In particular, we write ω the first infinite ordinal. This set is isomorphic to the set $\mathbb{N}$ of natural numbers. We may also use the notation $\mathbb{N}$ for this ordinal. We do not distinguish finite ordinals and their corresponding cardinals that are just seen as natural numbers. We also use the symbol ∞ for an infinite cardinal big enough. Most of the time, it will stand for $\aleph_0$ (the cardinals of ω). In particular, we will often (and implicitly) consider the extension of the ordered set of naturals by adding an extra maximal element written ∞.

1.2 Infinite word languages

We review in this section some standard notations and results in (infinite words) language theory.

1.2.1 Infinite words and languages

Infinite words on alphabet Σ are defined as (total) function from $\mathbb{N}$ to Σ. An infinite words w will be written similarly $w = a_0.a_1.a_2.\cdots$ with $w(i) = a_i$ for each $i \in \mathbb{N}$. The length of an infinite word w is still written $|w|$ but now equals ∞.

The set of infinite words on the alphabet Σ is written Σ^ω. For every finite word $v \in \Sigma^*$ and every infinite word $w \in \Sigma^\omega$, we write $v.w$ for the *mixed product* of v and w, that is the infinite word defined, for every $i \in [0, |u| - 1]$ by $v.w(i) = u(i)$ and, for every $i \geq |u|$ by $u.v(i) = v(i - |u|)$. For every infinite sequence $(v_i)_{i\in\omega}$ of finite non empty words, we also write $v_0.v_1.\cdots$ the *infinite product* of all words of the sequence. More precisely, writing, for every $k \in \mathbb{N}$, $s_k = \Sigma_{j\in[0,k-1]}|v_j|$, the infinite product is defined to be the infinite word w defined for every $k \in \mathbb{N}$ and every $i \in \mathbb{N}$ with $s_k \leq i < s_{k+1}$, $w(i) = v_k(i - s_k)$. As a particular case, for every word $v \in \Sigma^+$, we write w^ω for the infinite words obtained by making the infinite product of the sequence $(v_i = v)_{i\in\omega}$.

A language of infinite words is any subset L of Σ^ω. Given a language of non empty finite words $L \subseteq \Sigma^+$ we write $L^\omega \subseteq \Sigma^\omega$ for the set of infinite words one can built by infinite products of words of L. Given also a language of infinite words $M \subseteq \Sigma^\omega$, we write $L.M \subseteq \Sigma^\omega$ for the set of infinite words one can built by mixed products of words of L followed by words of M.

A language $L \subseteq \Sigma^\omega$ is called *ω-regular* when it can be defined as a finite union of languages of the form $L.(U)^\omega$ with regular languages L and $U \subseteq L^+$.

The set of finite and infinite words on alphabet Σ is written Σ^∞. Set Σ^∞ can be equipped with a distance $d(w_1, w_2)$ defined to be 0 when $w_1 = w_2$ or $1/2^n$ otherwise, where n is the length of the longest common prefix of w_1 and w_2. This distance is ultrametric and induces a compact topology called the *word prefix topology*.

Given two alphabets A et B, we write π_A or π_1 (resp. π_B or π_2) the projection from $A\times B$ to A (resp. from $A\times B$ to B). By extension, for every word $w = (a_0, b_0).(a_1, b_1).\cdots \in (A \times B)^\infty$, we write $\pi_A(w) = \pi_1(w) = a_0.a_1.\cdots$ or $\pi_B(w) = \pi_2(w) = a_0.a_1.\cdots$ the projections of word w on A^∞ or B^∞.

Given n numbered sets $A_1, \ldots, A_n$, given $A = A_1 \times \ldots \times A_n$, given any set of indices $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ with $i_1 < \ldots < i_k$, we write $A[I]$ for the set $A[I] = A_{i_1} \times \ldots A_{i_k}$, for every $x = (a_1, \ldots, a_n) \in A$, we write $x[I]$ for the elements $x[I] = (x_{i_1}, \ldots, x_{i_k}) \in A[I]$,

and, for every $P \subseteq A$, we write $P[I]$ for the set $P[I] = \{x[I] \in A[I] : \ x \in P\}$. In case $I = \{i, i+1, \ldots, j\}$ (where $1 \leq i \leq j \leq n$), these notations simplify to $A[i,j]$, $x[i,j]$ and $P[i,j]$ respectively (and even simplify to $A[i]$, $x[i]$ and $P[i]$ when $i = j$).

These notations also extend to words as follows: for every word $w = a_1.a_2.\ldots \in A^\infty$, for every $I \subseteq \{1, \ldots, n\}$, $w[I] = a_1[I].a_2[I].a_3[I]\ldots$. For instance, given $w = (a_0, b_0).(a_1, b_1).(a_2, b_2) \in (A \times B)^*$ there shall be no confusion between $w[2] = b_0.b_1.b_2$ (equivalently $\pi_B(w)$ or $\pi_2(w)$), i.e. the second projection of w on alphabet B, and $w(2) = (a_2, b_2)$ the third letter of w.

These notations also extend to relations: for every relation $R \subseteq A \times B$, we write $R[I] \subseteq A[I] \times B[I]$ defined by $R[I] = \{(x[I], y[I]) \in A[I] \times B[I] : \ (x, y) \in R\}$.

We also need in the sequel to restrict to words where no sequence of identical letters occurs, i.e. words where only changes of letters are relevant. For this purpose we define the *function view* that delete these repetitions.

More precisely, for every word $w \in \Sigma^\infty$, we define function $view(w)$ by induction on the length of w as follows. For every a and $b \in \Sigma$, for every finite words $w \in \Sigma^*$, $view(\epsilon) = \epsilon$, $view(a) = a$, $view(w.a.b) = view(w.a).b$ when $a \neq b$ or $view(w.a)$ when $a = b$. Observe that for every infinite words $w \in \Sigma^\omega$, $view(w)$ equals the limit, in the prefix topology, of the converging sequence $\{view(w_n)\}_{n \in I\!N}$ of the images of the prefix w_n of length n of w by function $view$.

1.2.2 Infinite word automata

A *ω-word automaton* is a tuple $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, T \rangle$ with finite set of states Q, finite alphabet Σ, initial state $q_0 \in Q$, transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ and infinite acceptance table $T \subseteq \mathcal{P}(Q)$ called *Muller acceptance* criteria.

The Automaton $\mathcal{A}$ is a *deterministic* when, for every $q \in Q$, every $a \in \Sigma$, $|\delta(q, a)| \leq 1$.

The Muller acceptance criterion is called a *parity condition* when there is a mapping $\Omega : Q \to I\!N$ such that $T = \{Q' \subseteq Q | \min(\Omega(Q')) \text{ is even}\}$. In this case, we write Ω instead of T in the automaton definition.

A parity condition is a *Büchi condition* (resp. co-Büchi condition) when $\Omega(Q) \subseteq \{0, 1\}$ (resp. $\Omega(Q) \subseteq \{1, 2\}$).

A parity condition is a *closed condition* when $\Omega(Q) = \{0\}$.

Given an infinite word $w \in \Sigma^\omega$, a *run* of the automaton $\mathcal{A}$ on the word w is an infinite word $\rho \in Q^\omega$ such that: $\rho(0) = q_0$ and, for each $i \in \omega$, $\rho(i+1) \in \delta(\rho(i), w(i))$.

One may also use ω-automata with ϵ-transition, i.e. automata with transition function δ extended to $Q \times \{\epsilon\}$.

Given an infinite word $w \in \Sigma^\omega$, a *run* of the automaton $\mathcal{A}$ with ϵ-transition on the word w is an infinite word $\rho \in Q^\omega$ such that $\rho(0) = q_0$ and there is an increasing sequence $\{k_i\}_{i \in I\!N}$ such that $k_0 = 0$, for every $i \in I\!N$:

1. either $k_{i+1} = k_i + 1$ and $\rho(i+1) \in \delta(\rho(i), w(k_i))$,
2. or $k_{i+1} = k_i$ and $\rho(i+1) \in \delta(\rho(i), \epsilon)$.

Given a run $\rho \in Q^\omega$, given the set $Inf(\rho) \subseteq Q$ of states that occurs infinitely often in ρ, we say that ρ is an *accepting run* when $Inf(\rho) \in T$.

Word $w \in \Sigma^\omega$ is *accepted* by the automaton $\mathcal{A}$ when there is an accepting run of $\mathcal{A}$ on w. The set of words accepted by the automaton $\mathcal{A}$ is written $L^\omega(\mathcal{A})$ and is called the language recognized by the automaton $\mathcal{A}$.

Remark. A language recognized by a closed automaton is closed in the prefix topology. Conversely, one can check that a recognizable closed language is recognized by a closed automaton.

The following theorem is a condensed version of several results that make infinite word automata theory so rich and useful [34, 35, 134, 126].

1.2.2.1 Theorem (Büchi, Muller, MacNaughton). *For every language $L \subseteq \Sigma^\omega$, the following properties are equivalent:*

1. *L is ω-regular,*
2. *L is recognizable by a Muller automaton,*
3. *L is recognizable by a deterministic Muller automaton,*
4. *L is recognizable by a Büchi automaton,*
5. *L is recognizable by a deterministic parity automaton.*

In particular:

1.2.2.2 Theorem (Safra[157, 158]). *Every Büchi non deterministic automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, T\rangle$, with or without ϵ-transition, is equivalent to a deterministic Muller automaton with $\mathcal{A}_d = \langle Q_d, \Sigma, q_{0,d}, \delta_d, T_d\rangle$ with $|Q_d| =!|Q|$.*

Languages recognizable by deterministic parity automata are obviously closed by complement and projection, it follows:

1.2.2.3 Corollary. *Ω-regular languages are closed under union,complement and projection.*

1.3 Transition systems and bisimulation

1.3.1 Models

Let D and Σ be two alphabet. A *transition system* is a rooted directed labeled graph

$$M = \langle V^M, r^M, D, \{T_d^M\}_{d\in D}, \Sigma, \lambda^M\rangle$$

with a set V^M of *vertices*, a *root vertex* $r^M \in V^M$, a *direction* (or *action*) alphabet D, for each direction $d \in D$, a binary *d-edge relation* $T_d^M \subseteq V^M \times V^M$, a labeling alphabet Σ and a labeling function $\lambda^M : V^M \to \Sigma$.

In the sequel, we shall write $T^M = \bigcup_{d\in D} T_d^M$ for the *edge relation* regardless the direction taken into account. Observe that when D is a singleton alphabet, there is only a single d-edge relation so that the direction alphabet D can also be omitted.

When Σ is a singleton, the labeling function is uniquely determined and thus can be omitted as well. The graph $Sk(M)$ obtained from a transition system M by removing the labeling function is called the *skeleton graph* of M.

In the sequel, a transition system is simply called a a *D, Σ-graph*, a *Σ-graph*, a *D-graph* or even simply a *graph* when alphabets are either singletons or clear from the context.

Let $d \in D$. We say that a vertex $v \in V^M$ is a *d-successor* (or just *successor*) of u when $(u, v) \in T_d^M$. The set of all *d-successors* of u is written $Succ_d^M(u)$.

Graph M is called *D-deterministic graph* when, for every direction $d \in D$, any vertex $u \in V^M$ has at most one d-successor, i.e. $|Succ_d^M(u)| \leq 1$.

Graph M is called *D-complete graph* when, for every direction $d \in D$, any vertex $u \in V^M$ has at least one d-successor, i.e. $1 \leq |Succ_d^M(u)|$.

The set of all *successors* of vertex u is written $Succ^M(u)$. Again, when D is a singleton, directions are omitted so that successors and d-successors are the same. The *branching degree* of vertex u is defined to be the cardinal $|Succ^M(u)|$. We say that graph M is *finitely branching* when, for every vertex $u \in V^M$, the branching degree of u is finite.

Given two D, Σ-graphs M and N, a *graph morphism* from M to N is a mapping f from V^M to V^N such that: $f(r^M) = r^N$, for every vertex $u \in V^M$, $\lambda^M(u) = \lambda^N(f(u))$, and, for every $d \in D$, for every vertex $v \in V^M$, if $(u, v) \in T_d^M$ then $(f(u), f(v)) \in T_d^N$. A *graph embedding* from M into N if a one to one morphism from M to N such that, moreover, for every direction $d \in D$, every vertices u and $v \in V^M$, $(u, v) \in T^M$ if and only if $(f(u), f(v)) \in T^N$. An onto embedding is called a *graph isomorphism*. Two graphs are *isomorphic* when there is an isomorphism from one to the other. In every theoretical situation, isomorphic graphs are considered to be just the same.

Given a graph M and a subset of vertices $X \subseteq V^M$ with $r^M \in X$, the *subgraph induced* by set X, written $M|X$, is defined to be the graph

$$M|_X = \langle X, r^M, D, \{T_d^M \cap X \times X\}_{d \in D}, \Sigma, \lambda^M|_X \rangle$$

One can check that the canonical inclusion mapping from X to V^M is a graph embedding from $M|X$ into M.

Given two alphabets A and B with $\Sigma = A \times B$, and a Σ-graph M, we define the *graph projection* $\pi_A(M)$ (resp. $\pi_B(M)$) on the alphabet A (resp. B) to be the graph obtained from M by keeping every components identical but the labeling function λ that is composed with the projection Π_A of Σ into A (resp. π_B of Σ into B).

1.3.2 Paths and trees

Let $\overline{D}$ be is a disjoint set of copy of directions of D (an element of $\overline{D}$ is called a *reverse direction*) and let $\overline{()} : D \to \overline{D}$ be a canonical bijection from D into $\overline{D}$.

A *undirected path* is in a graph M is a non empty finite sequence of vertices and directions $p = v_1.d_1.v_2.d_2.\cdots.v_n \in (V^M.(D \uplus \overline{D}))^*.V^M$ such that, for every $i \in \{1, \cdots, n-1\}$, $(v_i, v_{i+1}) \in T_d$ when $d_i = d \in\in D$ or $(v_{i+1}, v_i) \in T_d$ when $d_i = \overline{d} \in \overline{D}$. The *length of path* p, written $|p|$, is defined to be $n-1$ and we say that its source (resp. its target) is the vertex v_1 (resp. the vertex v_n).

A *directed path* (or simply called *path* in the sequel) is a undirected path p as above with the extra requirement that only directions (and not their reverse) are taken.

The *undirected distance* $d(u, v)$ (resp. the *directed distance* $\vec{d}(u, v)$) between any two vertices u and $v \in V^M$ is defined to be the length of the smallest undirected path (resp. directed path) from u to v if such a path exists and ∞ otherwise.

1.3.2.1 Definition (Unraveling and trees). The *unraveling* of a graph M is defined to be the D, Σ-labeled graph

$$T(M) = \langle V^{T(M)}, r^{T(M)}, D, \{T_d^{T(M)}\}_{d \in D}, \Sigma, \lambda^{T(M)} \rangle$$

defined as follows: the set of vertices $V^{T(M)}$ is defined to be the set of paths in M with source r^M, the root $r^{T(M)}$ is the one-length path r^M, for each $d \in D$, the d-successor relation $T_d^T(M)$ is the set of all pairs of paths of the form $(p.u, p.u.v) \in V^{T(M)} \times V^{T(M)}$ that $(u, v) \in T_d^M$, and the labeling function $\lambda^{T(M)}$ is defined, for every path $p \in V^{T(M)}$, to be $\lambda^{T(M)}(p) = \lambda^M(v)$ where v is the target of p.

A *tree* is a graph M isomorphic to its unraveling $T(M)$.

Strictly speaking, such a tree shall be called a D, Σ-tree. However, in the sequel, with a little abuse of language, we shall often only called D, Σ-*trees* those tree-shaped D, Σ-graphs that are (at least) deterministic.

Remark (On the representation of deterministic trees). In D-deterministic graphs, a directed path

$$p = v_1.d_1.v_2.d_2.\cdots.v_{n-1}.d_{n-1}.v_n$$

from a known vertex v_1 is uniquely determined by the sequence

$$\pi_D(p) = d_1.d_2.\ldots.d_{n-1}$$

of directions followed along this path. It follows that the above "encoding" is, indeed, a faithful encoding.

It follows that every deterministic D, Σ-trees M can be seen as a partial function

$$t_M : D^* \to \Sigma$$

with a non empty and prefix closed domain $dom(t_M) \subseteq D^*$ defined by $t_M(\epsilon) = \lambda^M(r^M)$ and, for every non empty word $p \in D^+$, $t_M(p) = \lambda(r^M.p)$ where $r^M.p$ is the unique vertex (if it exists) reached from the root following the sequence of directions p.

For instance with $D = \{l, r\}$, Σ-colored binary infinite trees are represented by mappings $t : D^* \to \Sigma$.

Remark (Representing D, Σ-Graphs by $D \times \Sigma$-graphs). Observe also that any D, Σ-tree M - or more generally any D, Σ-graph - can be encoded as a $D \times \Sigma$-tree M' - or $D \times \Sigma$-graph M' -, i.e. with a single edge relation, obtained from M by "moving" any edge label to the label of its target vertices.

On trees, this can be done keeping the same domain, with an arbitrary edge label put at the root, since every vertex but the root is the target of a single edge. On graphs, this can be done by taking $V^{M'} = D \times V^M$ with, for every $v \in V^M$, $\lambda^{M'}(d, v) = (d, \lambda^M(v))$. In fact, since several edges with distinct label may reach the same vertex.

In the sequel, in order to simplify notations and proofs, we shall often use this encoding considering thus graphs with single edge relation.

1.3.3 Bisimulation and counting bisimulation

1.3.3.1 Definition (Van Benthem [20], Park [145]). Given two Σ-labeled graphs M and N, we say that $R \subseteq V^M \times V^N$ is a *bisimulation relation*, when, for every $(u, v) \in R$, $\lambda^M(u) = \lambda^N(v)$ and, for every direction $d \in D$:

1. for every $u' \in V^M$ such that $(u, u') \in T_d^M$ there exists v' such that $(v, v') \in T_d^N$ and $(u', v') \in R$,

2. for every $v' \in V^N$ such that $(v, v') \in T_d^N$ there exists u' such that $(u, u') \in T_d^M$ and $(u', v') \in R$.

The relation R is a *counting bisimulation* [99] if, in addition, for each $(u, v) \in R$, each $d \in D$, there is a bijection $f_{u,v} : Succ_d^M(u) \to Succ_d^N(v)$ such that, for each $u' \in Succ^M(u)$ one has $(u', f_{u,v}(u')) \in R$.

We say that the graphs M and N are *bisimilar* (resp. *counting bisimilar*) when there is a bisimulation relation (resp. a counting bisimulation relation) $R \subseteq V^M \times V^N$ such that $(r^M, r^N) \in R$.

Since, in a given graph M, identity is a (counting) bisimulation relation and the class of (counting) bisimulation relation is closed under symmetry and composition, both counting bisimulation and bisimulation are equivalence relations in the class of graphs.

These equivalences admit algebraic characterizations by means of adequate notions of graph saturating morphisms.

1.3.3.2 Definition (Castellani [40], Arnold and Dicky [12]). Given two Σ, D-graphs M and N, a mapping $f : V^M \to V^N$ is a *saturating morphism* (resp. *strictly saturating morphism*) from M to N when $f(r^M) = r^N$ and, for every $u \in V^M$, $\lambda^M(u) = \lambda^N(f(u))$ and, for every $d \in D$, $f(Succ_d^M(u)) = Succ_d^N(f(u))$ (resp. $f(Succ_d^M(u)) = Succ_d^N(f(u))$ and the restriction of f to $Succ_d^M(u)$ is one to one).

In this case, we say that graph M is a partial expansion (resp. partial unraveling) of graph N.

Remark. To a certain extend, saturating morphisms can be seen as sort of homomorphism in the usual algebraic sense on the algebraic signature $\{\lambda\} \cup \{Succ_d\}_{d \in D}$ relating sets with or without cardinality constraints.

Saturating morphisms capture bisimulation and counting bisimulation in the following sense.

1.3.3.3 Theorem (Castellani [40], Arnold et al. [12], J. and Lenzi [100]).
Two D, Σ-graphs M_1 and M_2 are bisimilar (resp. counting bisimilar) if and only if there exists a third graph N and two saturating morphisms (resp. strictly saturating morphisms) $f_1 : N \to M_1$ and $f_2 : N \to M_2$.

Proof. In the bisimulation case, graph N can be built out of the pairs of vertices that belong to a bisimulation relation between M_1 and N_1. The saturating morphisms f_1 and f_2 are then just defined as the first and the second projection. In the counting bisimulation case the construction is analogous although slightly more technical since local bijections need to be extracted from the counting bisimulation.

The converse is immediate since (strictly) saturating homomorphisms do define (counting) bisimulation relation. A complete proof is given in [100].

□

Going further in this algebraic characterization, one may ask if there are canonical graphs (initial objects in the underlying category) that would characterize bisimulation or counting bisimulation classes of graphs.

The answer for counting bisimulation is positive.

1.3.3.4 Theorem (J. and Lenzi [99, 100]). *Every graph M is counting bisimilar to its unraveling $T(M)$. Moreover, two graphs M_1 and M_2 are counting bisimilar if and only if their unravelings $T(M_1)$ and $T(M_2)$ are isomorphic.*

Proof. The function from $V^{T(M)}$ to V^M that maps each finite path to its target is a strictly saturating morphism from $T(M)$ and M. Then we conclude the proof applying Lemma 1.3.3.3 on $T(M_1)$ and $T(M_2)$ observing, moreover, that, on trees, strictly saturating morphisms are just isomorphisms.

□

The purpose of the κ-expansion defined below is to provide, up to some cardinal, similar representatives for bisimulation equivalence classes.

1.3.3.5 Definition (J. and Walukiewicz [93, 104]). Let κ be a non zero cardinal. A *κ-indexed path* in M is a non empty finite word

$$p = u_1.d_1.k_1.u_2.\cdots.u_{n-1}.d_{n-1}.k_{n-1}.u_n \in V^M.(\kappa.V^M)^*$$

with $u_1, \ldots, u_n \in V^M$ and $k_2, \ldots, k_n \in \kappa$ such that $u_1.d_1.u_2.\cdots.d_{n-1}.u_n$ is a (directed) path in M. As for a path, we say that u_1 (resp. u_n) is the source (resp. the target) of the κ-indexed path p. The length of p is also defined to be $|p| = n - 1$.

The *κ-expansion* of a D, Σ-labeled graph M is defined to be the D, Σ-labeled graph

$$M^\kappa = \langle V^{M^\kappa}, r^{M^\kappa}, D, \{T_d^{M^\kappa}\}_{d\in D}, \Sigma, \lambda^{M^\kappa}\rangle$$

where the set of vertices V^{M^κ} is the set of all finite κ-indexed paths of M with source r^M, the root r^{M^κ} is r^M, for every $d \in D$, the relation $T_d^{M^\kappa}$ is the set of all pairs of the form $(w.u, w.u.d.k.v) \in V^{M^\kappa} \times V^{M^\kappa}$ such that $(u, v) \in T_d^M$ and $k \in \kappa$ is an arbitrary element of κ. Finally, given any κ-indexed path $w \in V^{M^\kappa}$ with target $v \in V^M$, the labeling function is defined to be $\lambda^{M^\kappa}(w) = \lambda^M(v)$.

Remark. Observe that when $\kappa = 1$, the κ-expansion M^κ of M is isomorphic to its unraveling $T(M)$. Observe also that for every cardinal κ and for every graph M, the κ-expansion M^κ of M is a tree.

1.3.3.6 Theorem (J. and Walukiewicz [93, 104]). *For every infinite cardinal κ, two graphs M and N of out-degree bounded by κ are bisimilar if and only if their κ-expansions M^κ and N^κ are isomorphic.*

Proof. Let R be a bisimulation relation between M and N and let a cardinal κ be as above. Let R' be the relation between vertices of M^κ and N^κ that relates any two κ-indexed paths of the same length whose targets are related in R. Relation R' is a bisimulation relation. Moreover, provided κ is infinite (so that, $\kappa.\kappa = \kappa$) and big enough (actually as big as the branching degree of M and N), one can observe that the relation R' is a counting bisimulation.

The converse is immediate since every graph M is bisimilar with its expansion M^κ.

□

Example. For the reader who is not familiar with bisimulation equivalence, let us remind that the two finite trees M and N defined by a single edge (resp. two edges only) from the root, with all vertices labeled identically, are bisimilar while not counting bisimilar. Moreover, for every finite and non empty cardinal κ, M^κ and N^κ are not isomorphic, i.e. in Theorem 1.3.3.6, the hypothesis that κ is infinite is essential.

1.3.4 The prefix topology over finitely branching trees

We consider a topology on the set of all *finitely branching* trees. This topology is a straightforward generalization of the classical prefix topology on words, binary trees, or, more generally, k-ary trees, where k is a fixed integer.

Let $FBT(Pred)$ be the class of all finitely branching trees over a finite set $Pred$ of predicate symbols. We define the prefix topology on $FBT(Pred)$ by taking as the basic open sets, the sets of the form:

$$\{M \in FBT(Pred) \mid P_h(M) \cong F\},$$

where h is a positive integer and F is a finite tree. This topology is Hausdorff's, as shown by the following lemma.

1.3.4.1 Lemma. *Let M and N be two finitely branching trees. The trees M and N are isomorphic if and only if for infinitely many h, $P_h(M)$ and $P_h(N)$ are isomorphic.*

Proof. The only non-trivial argument which we need to make is to show that if M and N are infinite and $P_h(M) \cong P_h(N)$ for infinitely many h (hence for every h) then $M \cong N$. In order to do so, let $I_{M,N}$ be the set of isomorphisms from $P_h(M)$ to $P_h(N)$, for h ranging over positive integers. The set $I_{M,N}$ ordered by inclusion forms an infinite finitely branching trees. Hence, by Koenig's Lemma, it has an infinite branch which defines an isomorphism between M and N.

□

As a curiosity, note that the lemma does *not* extend to arbitrary trees M and N. As an example consider for every n, a unary tree M_n with n vertices, and a unary, infinite tree M_∞. Let M be the graph obtained from the disjoint union of the M_n's by adding a new root on top of them (hence the root of M has countably many successors: one per graph M_n). And let N be the graph obtained in a similar way from the disjoint union of the M_n's and M_∞. For every $h \in \omega$, both $P_h(M)$ and $P_h(N)$ are trees formed by a copy of M_k for each $k < h$, plus countably many copies of M_h. But M and N are not isomorphic, because N has an infinite branch and M has none.

Observe that the prefix topology can be defined by a metric. In fact, given two trees M and N, let $d(M, N) = 0$ when M and N are isomorphic and $d(M, N) = 2^{-k}$ otherwise where k is the biggest integer such that $P_k(M)$ and $P_k(N)$ are isomorphic (which exists by the gluing Lemma). The function d is obviously a metric that defines the prefix topology.

The prefix topology satisfies a weak form of compactness. More precisely, we define the *skeleton* of a tree M to be the tree (over zero predicates)

$$Sk(M) = \langle V^M, r^M, T^M \rangle$$

i.e. $Sk(M)$ is the structure obtained from M by forgetting all unary predicates. Then,

1.3.4.2 Theorem (J. and Lenzi [100]). *Let $(M_n)_{n\in\mathbb{N}}$ be a sequence in $FBT(Pred)$. If the sequence*
$\{Sk(M_n)\}_{n\in\mathbb{N}}$ has a converging subsequence so does have $\{M_n\}_{n\in\mathbb{N}}$.

Proof. Let M_n be such a sequence. Assuming the sequence $\{Sk(M_n)\}_{n\in\mathbb{N}}$ has a converging subsequence, let $I \subseteq \mathbb{N}$ be an infinite set such that $\{Sk(M_n)\}_{n\in I}$ converges.

By induction, we build a strictly decreasing sequence $\{I_h\}_{h\in\mathbb{N}}$ of infinite sets of positive integers such that $I_0 = I$ and, for every $h > 0$, m and $n \in I_h$, $P_h(M_n) = P_h(M_m)$. This enables us to conclude as it implies that the sequence $\{M_{\min(I_h)}\}_{h\in\mathbb{N}}$ converges.

More precisely, assume that, for some $h \in \mathbb{N}$, the finite sequence of infinite sets $I = I_0 \supset I_1 \supset \cdots \supset I_h$ has already been built.

Since the sequence $\{Sk(M_n)\}_{n\in I}$ converges, there exists $m_h \in I$ such that, for every $n \in I$ with $n \geq m_h$, $P_{h+1}(Sk(M_{m_h})) = P_{h+1}(Sk(M_n))$. Now, as there are finitely many trees in $FBT(Pred)$ with skeleton $P_{h+1}(Sk(M_{m_h}))$ and the set I_h is infinite, there exists an infinite subset I_{h+1} of I_h (which can be chosen distinct from I_h) such that, for every n and $m \in I_{h+1}$, $P_{h+1}(M_n) = P_{h+1}(M_m)$.

□

The prefix topology is not compact as shown, for instance, by any sequence of trees M_n where the root has degree n and which has no converging subsequence.

1.4 Logic and mu-calculi

1.4.1 First order and monadic second order logic

In this section, we review the first order and the monadic second order logic over graphs.

Let $Pred$ be a finite set of unary predicate symbols and let Σ be $\mathcal{P}(Pred)$. Every D, Σ-graph M can be represented by a *first order structure*, still denoted by M, on the vocabulary $\{r\} \cup \{T_d\}_{d\in D} \cup Pred$, with *structure domain* $dom(M) = V^M$, the constant symbol r interpreted by r^M, each binary relation symbols T_d interpreted by T_d^M, and each unary predicate symbol $p \in Pred$ interpreted by the set $p^M = \{v \in V^M : p \in \lambda^M(v)\}$.

Let $var = \{x, y, \cdots\}$ and $Var = \{X, Y, \cdots\}$ be respectively some disjoint sets of first order and monadic second order variable symbols. First order (FO) and monadic second order (MSO) formulas over the vocabulary $\{r\} \cup \{T\} \cup Pred$ can be defined as follows.

The set of FO formulas is the smallest set containing formulas $p(t)$, $t = t'$, $T_d(t, t')$, $X(t)$ for $p \in Pred$, $d \in D$, $X \in Var$ and $t \in var \cup \{r\}$ and closed under negation $\neg$, disjunction $\vee$, conjunction $\wedge$, implication $\Rightarrow$ and existential $\exists$ and universal $\forall$ quantification over FO variables.

The set of MSO formulas is the smallest set containing all FO formulas and closed, moreover, under existential $\exists$ and universal $\forall$ quantification over set variables.

We use the notation $\varphi(x_1, \cdots, x_m, X_1, \cdots, X_n)$ for an MSO formula φ with free FO-variables among $\{x_1, \cdots, x_m\}$ and free set variables among $\{X_1, \cdots, X_n\}$. A sentence is a formula with no free variable.

Given a model M, if $v_1, \ldots, v_n \in V^M$ and if $V_1, \ldots, V_n \subseteq V^M$, we write $M \models \varphi(v_1, \cdots, v_n, V_1, \cdots, V_n)$, or simply $M \models \varphi$ when there is no ambiguity, to say that the formula φ is true in M (or M satisfies φ) when the interpretation x_i^M (resp. X_j^M) of each FO-variable x_i (resp. set variable X_j) is defined as the element v_i (resp. the set V_j). This satisfaction relation is classical and not redefined here; see for instance [164, 41]. In the sequel, we also use the notation $\bot$ (resp. $\top$) for any formula false (resp. true) in all models.

A class $\mathcal{C}$ of graphs is said *MSO definable* when there exists a sentence $\varphi \in MSO$ such that $\mathcal{C}$ is the class of all models of φ.

A class $\mathcal{C}$ of graphs is *bisimulation closed* (resp. *counting bisimulation closed*) if whenever $M \in \mathcal{C}$ and M' is bisimilar (resp. counting bisimilar) to M then $M' \in \mathcal{C}$.

Then, a sentence φ is *bisimulation invariant* (resp. *counting bisimulation invariant*) if the class of transition systems it defines is bisimulation closed (resp. counting bisimulation closed).

The notion of bisimulation invariance (or counting bisimulation invariance) is extended to arbitrary formulas $\varphi(X_1, \cdots, X_n)$ by considering graphs over the set of predicate symbols $Pred' = Pred \cup \{X_1, \cdots, X_n\}$. In the sequel, fixed point formulas may have free set variables and we will often implicitly consider this extension of graphs to $Pred'$ whenever there is no ambiguity.

Finally, the monadic quantifier alternation depth hierarchy is defined as follows. The first (or zeroth) level $\Sigma_0 = \Pi_0$ is defined as the set of all formulas of first order logic. Then, for each integer k, the level Σ_{k+1} (resp. level Π_{k+1}) is defined as the set of all formulas of the form $\exists X_1 \cdots \exists X_n \varphi$ with $\varphi \in \Pi_k$ (resp. $\forall X_1 \cdots \forall X_n \varphi$ with $\varphi \in \Sigma_k$). The bisimulation invariant (resp. unwinding invariant) fragment of the level Σ_k of MSO sentences is defined as the set of all bisimulation invariant (resp. unwinding invariant) sentences of Σ_k.

The level monadic Σ_1 of the monadic hierarchy is also called existential monadic second-order logic (EMSO). Aside the monadic hierarchy, an extension of the level monadic Σ_1 has also been defined and is of some interest in this report. Closed existential monadic second-order logic (CEMSO) is defined as the set of all formulas of the form $\theta_1 \overline{x}_1 \exists X_1 \cdots \theta_n \overline{x}_n \exists X_n \varphi$ where φ is an FO formula, the $(\theta_i x_i)$s are finite sequences of FO quantifications. Ajtai et al. proved that, over finite models, CEMSO is strictly more expressive than EMSO [4] Arnold et al. [13] show that the same holds over infinite trees.

1.4.2 Modal and counting mu-calculus

In this section we review the definition of the counting and modal propositional mu-calculus on Σ-graphs. The modal mu-calculus was introduced by Kozen [110]. The counting mu-calculus is just the extension of the modal mu-calculus with counting modalities.

1.4.2.1 Definition (Kozen [110]). The set of modal mu-calculus formulas is the smallest set containing $Pred \cup Var$ and closed under negation, disjunction, conjunction, and the following formation rules, if α is a formula then $\Diamond\alpha$ and $\Box\alpha$ are mu-calculus formulas, and, provided X occurs only positively (i.e. under an even number of negations) in α then $\mu X.\alpha$ and $\nu X.\alpha$ are modal mu-calculus formulas.

The set of counting μ-calculus formulas is defined as above replacing standard modalities $\Diamond$ and $\Box$ by counting modalities $\Diamond_k$ and $\Box_k$ for any integer k.

Remark. Strictly speaking, the counting mu-calculus is an extension of the mu-calculus defined by the author and Lenzi in [99].

We use the same convention as for MSO with free set variables, i.e. we denote by $\alpha(X_1, \cdots, X_n)$ a formula with free variables among $\{X_1, \cdots, X_n\}$. For convenience, we may also omit these free set variables in the formula α considering then implicitly that graphs have been built over the set of unary predicate symbols $Pred' = Pred \cup \{X_1, \cdots, X_n\}$. In the sequel, we call *fixed point formula* any formula of the modal or counting μ-calculus.

We also write $\alpha[\beta/X]$ for the formula obtained from the formula α by replacing any free occurrence of X by the formula β.

The semantics of fixed point formulas can be defined formally by translating fixed point formulas into MSO. An equivalent fixpoint semantics discussed in the next section.

1.4.2.2 Definition (Fixed point formulas semantics). Let $p \in Pred$, α and β be fixed point formulas, X be a set variable, x and z be FO variables, and $\overline{z} = (z_1, \cdots, z_k)$ be a k-tuple of FO variables. The semantics of the fixed point formulas is defined inductively according to the following induction rules:

- Atomic formulas :
  $\varphi_p(x) = p(x)$ and $\varphi_X(x) = X(x)$,
- Boolean connectives :
  $\varphi_{\alpha\wedge\beta}(x) = \varphi_\alpha(x) \wedge \varphi_\beta(x)$, $\varphi_{\alpha\vee\beta}(x) = \varphi_\alpha(x) \vee \varphi_\beta(x)$
  and $\varphi_{\neg\alpha}(x) = \neg\varphi_\alpha(x)$
- Modalities :
  $\varphi_{\Diamond\alpha}(x) = \exists z\,(T(x,z) \wedge \varphi_\alpha(z))$,
  $\varphi_{\Box\alpha}(x) = \forall z\,(T(x,z) \Rightarrow \varphi_\alpha(z))$
- Counting modalities :
  $\varphi_{\Box_k\alpha}(x) = \forall\overline{z}\,\Big((\mathit{diff}(\overline{z}) \wedge \bigwedge_{i\in[1,k]} T(x,z_i)) \Rightarrow \bigvee_{i\in[1,k]} \varphi_\alpha(z_i)\Big)$,
  $\varphi_{\Diamond_k\alpha}(x) = \exists\overline{z}\,\Big(\mathit{diff}(\overline{z}) \wedge \bigwedge_{i\in[1,k]} T(x,z_i) \wedge \varphi_\alpha(z_i)\Big)$,
- Fixed points :
  $\varphi_{\mu X.\alpha(X)}(x) = \forall X\,\big((\varphi_{\alpha(X)} \subseteq X) \Rightarrow X(x)\big)$,
  $\varphi_{\nu X.\alpha(X)}(x) = \exists X\,\big((X \subseteq \varphi_{\alpha(X)}) \wedge X(x)\big)$.

There, $\mathit{diff}(\overline{z})$ is the quantifier-free FO formula stating that $z_i \neq z_j$ for every $i \neq j$, $\varphi_{\alpha(X)} \subseteq X$ is the MSO formula $\forall z\varphi_{\alpha(X)}(z) \Rightarrow X(z)$, and, similarly, $X \subseteq \varphi_{\alpha(X)}$ is the MSO formula $\forall z X(z) \Rightarrow \varphi_{\alpha(X)}(z)$. For every fixed point formula α, every model M, we write $M \models \alpha$ when $M \models \varphi_\alpha(r)$.

Remark. In the sequel, we also use *backward modalities* $\Diamond^{-1}$ and $\Box^{-1}$, and *backward counting modalities* $\Diamond_i^{-1}$ and $\Box_i^{-1}$ that are defined like the ordinary modalities but with respect to the inverse edge relation $(T^M)^{-1}$ in place of T^M. In the presence of backward modalities, the standard modalities are referred to as *forward modalities*.

1.4.3 Fixed points in mu-calculus

The above definition of fixed point formulas semantics does not give much intuition on the meaning of these formulas. We review here also a standard (and somehow more intuitive) point of view on the semantics of fixed point formulas.

1.4.3.1 Definition (Set mappings defined by fixed point formulas). In a model M, a fixed point formula $\alpha(X_1, \ldots, X_n)$ induces an n-ary mapping α^M from $(\mathcal{P}(V^M))^n$ to $\mathcal{P}(V^M)$ defined, for every sets $V_1, \ldots, V_n \subseteq V^M$ by $\alpha^M(V_1, \ldots, V_n) = \{v \in V^M : M \models \varphi_\alpha(v, V_1, \cdots, V_n)\}$.

Remark. The meaning of a formula with no free variable is a set. For instance, rephrasing the definition of modalities semantics, the sentence $\Diamond\alpha$ (resp. the sentence $\Box\alpha$) defines the set of vertices which have at least one successor (resp. all successors) satisfying predicate α.

Likewise, for counting modalities, the sentence $\Diamond_k\alpha$ (resp. $\Box_k\alpha$) defines the set of vertices which have at least k successors (resp. all but at most $k-1$ successors) satisfying predicate α.

In particular, modalities $\Diamond$ and $\Diamond_1$ on the one hand, and modalities $\Box$ and $\Box_1$ on the other hand, have the same meaning. For this reason, we always consider *the modal mu-calculus to be a sub-language of the counting mu-calculus.*

If X occurs only positively in a fixed point formula $\alpha(X)$ then the mapping α^M from $\mathcal{P}(V^M)$ to $\mathcal{P}(V^M)$ is monotonic increasing w.r.t. the inclusion order. By Knaster and Tarski's Theorem [170], it has a least and greatest fixed point. The purpose of the μ and the ν connectives is to define these fixed points.

1.4.3.2 Lemma (Least and greatest fixed points). *The meaning $(\mu X.\alpha(X))^M$ of the formulas $\mu X.\alpha(X)$ and, respectively, the meaning $(\nu X.\alpha(X))^M$ the formula $\nu X.\alpha(X)$ in M are the least and, respectively, the greatest solution of the set equation $X = \alpha^M(X)$. They are given by*

$$\mu X.\alpha^M(X) = \bigcap\{X \subseteq V^M : \alpha^M(X) \subseteq X\}$$

and

$$\nu X.\alpha^M(X) = \bigcup\{X \subseteq V^M : X \subseteq \alpha^M(X)\}$$

In particular, we do have:

1.4.3.3 Corollary. *For every fixed point formula α, the formula $\mu X.\alpha$ (resp. $\nu X.\alpha$) is equivalent to the formula $\alpha[\mu X.\alpha/X]$ (resp. $\alpha[\nu X.\alpha/X]$).*

1.4.4 The fixed-point alternation depth hierarchy

We review, in the rest of the section, some properties of the syntax of fixed point formulas. This leads to the definition of the alternation depth of fixed point formulas.

1.4.4.1 Definition (Well-named formula). We call a fixed point formula *well named* if negation only applies to unary predicates, every variable is bound at most once, and, free variables are distinct from bound variables.

For a variable X bound in a well-named formula α there is a unique subformula of α, of the form $\sigma X.\beta$, from now on called the *binding definition* of X in α. We call X a ν-variable when $\sigma = \nu$, we call X a μ-variable when $\sigma = \mu$.

By applying de Morgan laws and by consistent renaming of bound variables in a fixed formula α, one can always built an equivalent well-named formula β.

1.4.4.2 Definition (Bound variable dependency ordering). We define the *dependency order relation* $\leq_\alpha$ over the set of bound variables in a fixed point formula α as the least order relation $\leq_\alpha$ such that: for any two bound variables X and Y, if the variable X occurs free in the binding definition $\sigma Y.\beta$ of the variable Y then $X \leq_\alpha Y$.

Observe that because α is well-named, the order $\leq_\alpha$ is well defined. Moreover, every non minimal variable has a unique predecessor. In fact, the dependency order $\leq_\alpha$ over the bound variables in α inherits the (tree-shaped) syntactic structure of the formula α, i.e. the predecessors of each variable are totally ordered.

1.4.4.3 Definition (Bound variable alternation depth). Let α be a well named formula. The *nu-depth* $N_\alpha(X)$ (resp. the *mu-depth* $M_\alpha(X)$) of any bound variable X in the formula α, is defined by induction on the depth of X is the dependency order $\leq_\alpha$ as follows:

1. when variable X is minimal in $\leq_\alpha$: if X is a ν-variable then $N_\alpha(X) = 0$ (resp. $M_\alpha(X) = 1$) and if X is a μ-variable then $N_\alpha(X) = 1$ (resp. $M_\alpha(X) = 0$);
2. otherwise, given Y the (unique) immediate predecessor of X in the dependency order $N_\alpha(X) = N_\alpha(Y)$ (resp. $M_\alpha(X) = M_\alpha(Y)$): X and Y are both at the same time μ-variables or ν-variables, or $N_\alpha(X) = N_\alpha(Y) + 1$ (resp. $M_\alpha(X) = M_\alpha(Y) + 1$) otherwise.

Observe that, with such a definition, ν-variables have even nu-depth (resp. odd mu-depth) while μ-variables have odd nu-depth (resp. even nu-depth).

1.4.4.4 Definition (Niwinski [138, 139]). Levels of the modal (resp. counting) propositional fixpoint alternation depth hierarchy $\{N_k\}_{k\in I\!N}$ and $\{M_k\}_{k\in I\!N}$ (resp. $\{NC_k\}_{k\in I\!N}$ and $\{MC_k\}_{k\in I\!N}$) are defined as follows: for every $k \in I\!N$: N_k (resp. NC_k) is the set of modal fixed point formulas (resp. counting fixed point formulas) with bound variable of nu-depth at most $k-1$ and, similarly, M_k (resp. MC_k) is the set of modal fixed point formula (resp. counting fixed point formula) with bound variable of mu-depth at most $k-1$.

Remark. The level N_0 (or M_0) is the set of fixed point free modal formula. Levels N_1 and M_1 are formulas built with greatest or least fixpoint only. Level N_2 corresponds to formulas with greatest fixpoint nested by least fixpoint. etc...

An interested level is the level $N_2 \cap M_2$ of formulas where no bound variables of distinct type are dependent one with the other. This level is often called the alternation free mu-calculus. It can be equivalently defined as the closure of levels N_1 and M_1 under boolean connectives and (well-behaving) substitutions.

Aside such a syntactical hierarchy: formulas that belong to such or such level, there is a semantical hierarchy: properties that are definable by means of formulas in such or such level.

1.4.4.5 Theorem (Bradfield [32] and Arnold [11]). *The modal (resp. counting) fixed point alternation depth hierarchy is strict, i.e. for every $k > 0$, there is a property α_k in N_k (resp. NC_k) which does not belong to N_{k-1} (resp. NC_{k-1}).*

Proof. Strictly speaking, the counting case is rather a corollary of Arnold's result. In fact, he proved that the modal mu-calculus hierarchy remains strict over the binary tree. Then, the strictness of the counting fixed point alternation depth hierarchy follows from the facts that: first, the binary tree is definable in the counting mu-calculus, and, next, on the binary tree, the counting and the modal mu-calculus are, level by level, equal.

□

Remark. An interesting and independent related result has also been obtained by Lenzi [113, 114].

1.4.5 MS0-definable languages of words and automata

With a singleton (direction) alphabet D, one can encode words as deterministic Σ-graph. More precisely, given any finite or infinite word $w \in \Sigma^\infty$, there is a unique deterministic Σ-tree M_w such that the sequence of labels of vertices along the (unique) path emanating from the root equals w. Then it make sense to speak about *MSO-definable* language as languages of words which set of models is definable is MSO.

1.4.5.1 Theorem (Büchi [34], Trakhtenbrot [175, 176]). *A language $L \subseteq \Sigma^\infty$ is MS-definable if and only if it is recognizable by a non deterministic Büchi automaton.*

1.5 Notes

Bisimulation equivalence appears already in logic [20] under the name of π-equivalence. In computer science, it was rediscovered by Park [145]. Observe that in this work, Park is comparing word automaton. It follows that he defines there bisimulation where double simulation would have suffice. Bisimulation really becomes popular with Milner et al. [85, 86, 127] studying process algebras [88, 89, 127, 23, 84, 7, 9]. The relevance of bisimulation equivalence as behavioral equivalence for processes have been studied a lot. It is generally consider to be the finest. A quite exhaustive overview and classification of other behavioral equivalences can be found in [70, 71].

The algebraic point of view presented here appears in the works of Castellani [40] and Arnold and Dicky [12]. Other algebrico-categorical approaches have been proposed and studied later [1, 106]. The notion of κ-unraveling, strongly related with the above, is an original notion developed by the author [93]. It has been used in several work studying bisimulation invariant fragment of various logical formalisms [104, 129, 2, 90, 99, 100, 101].

The prefix topology, presented here on relational graphs, is quite a straightforward generalization of the prefix topology on binary trees - equivalently the standard metric topology on sets of reals in $[0, 1]$ - . One may observe, however, that the restriction to finitely branching trees make statements and proofs slightly more delicate than in the binary case.

In Kozen's original definition [110], the mu-calculus is *multi-modal* in the sense that, for each direction $d \in D$, there is a box $[d]$ and a diamond $\langle d \rangle$ modality. It essentially means that the resulting fixed point calculus is interpreted on D, Σ-graphs instead of Σ-graphs. More precisely, for each $d \in D$, the pair of modalities $[d]$ and $\langle d \rangle$ are interpreted on relation T_d in D, Σ-graphs while $\Diamond$ and $\Box$ are interpreted on the unique edge relation T on Σ-graphs. It shall be clear that Kozen's original multi-modal mu-calculus essentially have the same property than the mu-calculus as shown by the (counting bisimulation preserving) encoding of D, Σ-graphs into $D \times \Sigma$-graphs given in page 6 of this document.

Chapter 2

Two player games

Two player games bear an important role in computer science. Verifying that a given model of program satisfies a given specification often equivalently amounts to checking that in some *model-checking game* - a game built from the model and the specification - one of the player has a winning strategy. Synthesizing a program from a specification also often equivalently amounts to finding a winning strategy in some *satisfiability game*, a game built from the specification formula. In both these cases, the notion of two player games captures, under adequate assumption, the combinatorial properties of these problems. More generally, two player games can also be seen as a versatile model of potential interactions between processes and their environment. They allow a firm and precise definition of many fundamental concepts that are commonly used in program design and validation.

How two player games can effectively be used and related with model-checking or program synthesis is illustrated in next chapters. In this chapter, we are mainly concerns with two player games themselves.

In the first section, we are reviewing classical definitions and properties of two player games. In the second section, we provide an algorithm to compute winning position in parity games. More precisely, this algorithm computes a winning strategy that is called permissive strategy in the sense that it allows every move that is allowed by a positional winning strategy. Various notions of two player games simulations are then presented and illustrated in the last section. The purpose of these simulations is to be used later in the text as uniform proof techniques to show the existence (or transfer) of winning strategies.

2.1 Classical definitions and results

2.1.1 General background and notations

The games we consider are discrete two players turn based games. The two players are called *Process* (or the player P) and *Environement* (or the player E).

2.1.1.1 Definition (Game). A *game arena* or *game* for short is a bipartite graph

$$\mathcal{G} = \langle V^{\mathcal{G}} = V_P^{\mathcal{G}} \uplus V_E^{\mathcal{G}}, T^{\mathcal{G}} = T_P^{\mathcal{G}} \uplus T_E^{\mathcal{G}}, Acc^{\mathcal{G}} \rangle$$

with a partition $V^{\mathcal{G}} = V_P^{\mathcal{G}} \uplus V_E^{\mathcal{G}}$ of the set of vertices called *game positions*, a partition $T^{\mathcal{G}} = T_P^{\mathcal{G}} \uplus T_E^{\mathcal{G}} \subseteq V^{\mathcal{G}} \times V^{\mathcal{G}}$ of the set of edges called *game moves* with $T_P^{\mathcal{G}} \subseteq V_P^{\mathcal{G}} \times V_E^{\mathcal{G}}$ and

$T_E^{\mathcal{G}} \subseteq V_E^{\mathcal{G}} \times V_P^{\mathcal{G}}$, equipped with a distinguished set of infinite paths $Acc^{\mathcal{G}} \subseteq (V^{\mathcal{G}})^\omega$ called the (infinite) acceptance condition. A game $\mathcal{G}$ may also have an *game initial position* that is a distinguished element of $V^{\mathcal{G}}$ written $r^{\mathcal{G}}$.

The Game $\mathcal{G}$ may be written $\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}}, Acc^{\mathcal{G}} \rangle$, superscript $\mathcal{G}$ possibly omitted when there is no ambiguity, sets $V^{\mathcal{G}}$ and $T^{\mathcal{G}}$ being implicitly defined.

The dual game $\overline{\mathcal{G}}$ of the game $\mathcal{G}$ is defined by taking $V_P^{\overline{\mathcal{G}}} = V_E^{\mathcal{G}}$, $V_E^{\overline{\mathcal{G}}} = V_P^{\mathcal{G}}$, $T_P^{\overline{\mathcal{G}}} = T_E^{\mathcal{G}}$, $T_E^{\overline{\mathcal{G}}} = T_P^{\mathcal{G}}$, $Acc^{\overline{\mathcal{G}}} = (V^{\mathcal{G}})^\omega - Acc^{\mathcal{G}}$.

We say that the game $\mathcal{G}$ is *P-deterministic* (resp. *E-deterministic*) when relation $T_P^{\mathcal{G}}$ (resp. relation $T_E^{\mathcal{G}}$) is functional.

2.1.1.2 Definition (Play). A *partial play* or just *play* in a game arena $\mathcal{G}$ is a non empty path in the underlying game graph $\langle V_P^{\mathcal{G}} \cup V_E^{\mathcal{G}}, T_P^{\mathcal{G}} \cup T_E^{G} g \rangle$. A play is *winning for Process* either when it is finite and ends in a *Environement*'s position or when it is infinite and belongs to $Acc^{\mathcal{G}}$. Dually, a play is *winning for Environement* either when it is finite and ends in a *Process*'s position or when it is infinite and belongs to $\overline{Acc}^{\mathcal{G}}$. A *complete play* is a play maximal with respect to the prefix ordering.

2.1.1.3 Definition (Strategy). A (non deterministic) *strategy* for *Process* (resp. for *Environement*) is a function

$$\sigma : (V^{\mathcal{G}})^*.V_P^{\mathcal{G}} \to \mathcal{P}(V_E^{\mathcal{G}}) \quad (\text{resp. } \tau : (V^{\mathcal{G}})^* \to \mathcal{P}(V_E^{\mathcal{G}}))$$

such that, for every play $p.v \in (V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$ (resp. $p.v \in (V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$), $\{v\} \times \sigma(p.v) \subseteq T_P^{\mathcal{G}}$ (resp. $\{v\} \times \tau(p.v) \subseteq T_P^{\mathcal{G}}$).

In the sequel, especially when used with quantifiers, letter σ will always denote a strategy for the player P and letter τ will always denote a strategy for the player E.

From a position $v \in V^{\mathcal{G}}$, the *plays induced by strategies* σ and τ, written

$$\sigma * \tau(v) \subseteq (V^{\mathcal{G}})^\infty$$

is defined to be the set of play p starting in v and such that for every prefix of p of the form $p'.v'$ either $v' = \sigma(p')$ when p' ends in $V_P^{\mathcal{G}}$ or $v' = \tau(p')$ when p' ends in $V_E^{\mathcal{G}}$.

Observe that all plays induced by strategies are path the the game graphs.

Strategy σ (resp. strategy τ) is a *deterministic strategy* when, for every $p \in (V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$, $|\sigma(p)| \leq 1$ (resp. for every $p \in (V^{\mathcal{G}})^*.V_E^{\mathcal{G}}$, $|\tau(p)| \leq 1$. In this case strategy σ (resp. strategy τ) is often seen as a partial function from $(V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$ to $V_E^{\mathcal{G}}$ (resp. from $(V^{\mathcal{G}})^*.V_E^{\mathcal{G}}$ to $V_P^{\mathcal{G}}$).

Strategy σ is a *non blocking strategy* from position $v \in V^{\mathcal{G}}$ when, for every counter strategy τ, every position $v \in V^{\mathcal{G}}$, there is no maximal play $w \in \sigma * \tau(v)$ that ends in a player P position.

2.1.1.4 Lemma. *For every strategy σ, every counter strategy τ and every position v, the set $\sigma * \tau(v)$ is prefix-closed . Moreover, an infinite play belongs to $\sigma * \tau(v)$ if and only if all its finite prefixes belong to $\sigma * \tau(v)$.*

In particular, the set $Beh(\mathcal{G}, v, \sigma) = \bigcup_\tau \sigma * \tau(\sigma)$ called the behavior *of strategy σ is prefix-closed and closed in the prefix topology.*

Proof. Observe that if all the prefixes of a path are allowed by strategy σ (strategy τ) then the whole path is allowed by strategy σ (strategy τ).

□

For arbitrary strategies, it makes sense to compare strategies by comparing the plays they allow.

2.1.1.5 Definition (Strategy order). A strategy σ is *subsumed* by a strategy σ', which is written $\sigma \sqsubseteq \sigma'$, if, for every position v, every counter strategy τ, $\sigma * \tau(v) \subseteq \sigma' * \tau(v)$. Strategies σ and σ' are *equivalent* when $\sigma \sqsubseteq \sigma'$ and $\sigma' \sqsubseteq \sigma$.

2.1.1.6 Definition (Winning strategy, winning position). Strategy σ for the player P (resp. strategy τ for player E) is a *winning strategy* from position $v \in V^{\mathcal{G}}$ when, for every strategy τ for the player E (resp. every strategy σ for the player P), every *maximal* play $p \in \sigma * \tau(v)$ is winning for the player P (resp. winning for the player E).

A position in the game arena is a *winning position* for a player when there is a winning strategy for this player from this position.

Remark. Observe that if there is a non deterministic winning strategy for some player then there is a deterministic one.

Observe also that there might be two reasons for a strategy for the player P to fail to be winning from a given position v. The first reason, local, is that strategy σ allows a play from position v that stops in a player P position (either because σ is no longer defined or this position has no successor). The second global reason is that strategy σ allows an infinite play from position v that does not belong to *Acc*.

This observation justifies somehow the definition of non blocking strategies given above. In fact, a strategy that is winning from position v is non blocking, and a non blocking strategy that fails to be winning always fails to be winning for the second reason.

Since the main issues and difficulties when dealing with two player infinite games arise with infinitary conditions, it is often assumed that strategies are always non blocking, and even games themselves does not have positions for the player P without successors. For reasons that shall become clear when dealing with tree automata, this is *not* the point of view followed in this presentation.

2.1.1.7 Definition (Game determinacy). A game is *determined* when every position of the game arena is winning for one of the player.

2.1.1.8 Theorem (Gale and Stewart [65]). *There are games (even on finite arena) that are not determined.*

2.1.1.9 Theorem (Martin [123]). *Games with Borel infinitary conditions are determined.*

2.1.2 Strategy trees vs strategies

2.1.2.1 Definition (Strategy tree). Given a game $\mathcal{G} = \langle V_P, V_E, T_P, T_E, Acc \rangle$, a *strategy tree* for player P in game $\mathcal{G}$ is defined to be a function $f : (V_P)^+ \to V_E$. Dually, a strategy tree for player E is defined to be a function $g : (V_E)^+ \to V_P$).

From an initial position v, the *plays induced by strategy trees* f and g, written $f * g(v)$ is defined to be the set of (finite or infinite) word $p \in (V_E^{\mathcal{G}} + V_P^{\mathcal{G}})^\infty$ such that $p(0) = v$

and, for every prefix of p of the form $p'.v_1.v_2$, if $v_1 \in V_P$ then $v_2 = f(\pi_{V_P^{\mathcal{G}}}(p'.v_1))$ with $(v_1, v_2) \in T_P$ or, if $v_1 \in V_E$ then $v_2 = g(\pi_{V_E^{\mathcal{G}}}(p'.v_1))$ with $(v_1, v_2) \in T_E$.

Observe that, as for deterministic strategies, the set of play $f * g(v)$ has a unique maximal elements and is closed under prefix.

As for standard strategies, one can define the *behavior* of a strategy tree f from a position $v \in V^{\mathcal{G}}$ to be the set $Beh(\mathcal{G}, v, f) = \bigcup_g f * g(v)$.

The following lemma says that the notion of strategy tree is essentially equivalent with the notion of strategy.

2.1.2.2 Lemma. *For every game $\mathcal{G}$, for every position $v \in V^{\mathcal{G}}$, for every strategy tree f there exists a deterministic strategy $\sigma_{f,v}$ such that $Beh(\mathcal{G}, v, f) = Beh(\mathcal{G}, v, \sigma_{f,v})$ and, conversely, for every deterministic strategy σ there exists a strategy tree $f_{\sigma,v}$ such that $Beh(\mathcal{G}, v, \sigma) = Beh(\mathcal{G}, v, f_{\sigma,v})$.*

In particular, a position v is winning for player P in game $\mathcal{G}$ if and only if there is a strategy tree f for player P such that, for every strategy tree for player E, the maximal play in $f * g(v)$ is winning for player P.

Proof. We show here the translation from strategy to strategy tree. The other direction is essentially the same.

Observe first that for every deterministic winning strategy σ in a game $\mathcal{G}$, for every position $v \in V^{\mathcal{G}}$, the projection mapping

$$\pi_{V_P^{\mathcal{G}}} : Beh(\mathcal{G}, \sigma, v) \cap (V^{\mathcal{G}})^*.V_P^{\mathcal{G}} \to (V_P^{\mathcal{G}})^+$$

is one to one, i.e. for every $p \in (V_P^{\mathcal{G}})^+$, there is at most one play from now on written $h_{\sigma,v}(p) \in (V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$ such that $h_{\sigma,v}(p).\sigma(h_{\sigma,v}(p)) \in Beh(\mathcal{G}, \sigma, v)$ and $\pi_{V_P^{\mathcal{G}}}(h_{\sigma,v}(p)) = p$.

It follows that we can define the strategy tree $f_{\sigma,v}$ from the strategy σ and the position $v \in V^{\mathcal{G}}$ as follows: for every $p \in (V_P^{\mathcal{G}})^+$, we put $f_{\sigma,v}(p) = \sigma(h_{\sigma,v}(p))$ whenever defined. One then can easily check that $Beh(\mathcal{G}, v, \sigma) = Beh(\mathcal{G}, v, f_{\sigma,v})$.

□

2.1.3 Regular and parity games

Computability is not an issue in the definitions above. Regular games are quickly reviewed here. They are essentially games with ω-regular winning conditions. In the finite case, regular games are *solvable*.

2.1.3.1 Definition (Regular and Muller game). A game $\mathcal{G} = \langle V^{\mathcal{G}} = V_P^{\mathcal{G}} \uplus V_E^{\mathcal{G}}, T^{\mathcal{G}} = T_P^{\mathcal{G}} \uplus T_E^{\mathcal{G}}, Acc^{\mathcal{G}} \rangle$ is a *regular game* when there is a finite alphabet C, an ω-regular language $L \subseteq C^\omega$ and a labeling function $\lambda : V^{\mathcal{G}} \to C$ such that $Acc^{\mathcal{G}} = \lambda^{-1}(L)$.

A *Muller game* is a regular game such that there is a set $T \subseteq \mathcal{P}(C)$ with $L = \bigcup_{P \in T} \bigcap_{a \in P} (C^*.a)^\omega$, i.e. $p \in L$ if and only if the set $Inf(p)$ of letters occurring infinitely often in p belongs to T.

Observe that, as a particular case, when $V^{\mathcal{G}}$ is finite, $Acc^{\mathcal{G}}$ is itself a regular language on the alphabet $V^{\mathcal{G}}$ and there is no longer any need to distinguish (even in Muller games) the alphabet C from the set of position $V^{\mathcal{G}}$. In the sequel, in order to simplify statement

and notation, we essentially consider finite games with $C = V^{\mathcal{G}}$, and, except when stated otherwise or just not applicable, all statements and proofs are easily generalizable to infinite regular or Muller games.

The size $|\mathcal{G}|$ of a finite regular game is defined to be the size of the game graph $|V^{\mathcal{G}}| + |T^{\mathcal{G}}|$. It follows that the complexity of computing winning positions may also take into account the size of (a representation) of the winning condition $Acc^{\mathcal{G}}$.

2.1.3.2 Definition (Memory of a strategy, regular strategy). A strategy σ for the player P (or for the player E accordingly) is a *strategy with memory* when there is a triple

$$c : M \times V_P^{\mathcal{G}} \to \mathcal{P}(V^{\mathcal{G}}), \quad \delta_u : M \times V^{\mathcal{G}} \to M, \quad m_0 \in M$$

such that, defining inductively, $\delta_u^* M \times (V^{\mathcal{G}})^* \to M$ by

$$\delta_u^*(m, \epsilon) = m \quad \text{and} \quad \delta_u^*(m, p.v) = \delta_u^*(\delta_u(m, p), v)$$

for every play $p.v \in (V^{\mathcal{G}})^*.V_P^{\mathcal{G}}$

$$\sigma(p.v) = c(\delta_u^*(m_0, \lambda(p)), v)$$

i.e. the strategy at a partial play $p.v$ is defined up to some memory $\delta_u^*(p)$ of the past of the play. When δ_u and m_0 will be clear from the context, we will sometimes use σ to denote the function c.

With a slight abuse of language, M is called the memory of the strategy. A *regular strategy* is a strategy with finite memory. A *memoryless strategy* or a *positional strategy* is a strategy with memory M which is a singleton set. Alternatively one can see it as a function $\sigma : V_P^{\mathcal{G}} \to \mathcal{P}(V_E^{\mathcal{G}})$.

2.1.3.3 Theorem (Büchi et al. [36, 37, 38, 81]). *In finite regular games a position is winning for a player if and only if this player has a regular winning strategy from this position. Moreover, winning positions (and winning strategies) are computable.*

2.1.3.4 Definition (Parity games). A game $\mathcal{G}$ is a *parity game* when the winning condition $Acc^{\mathcal{G}}$ is specified by means of a priority function $\Omega : V^{\mathcal{G}} \to I\!N$ (with $\Omega(V^{\mathcal{G}})$ finite when $\mathcal{G}$ is infinite) such that $Acc^{\mathcal{G}} = \{w \in (V^{\mathcal{G}})^\omega : \lim_{inf} \Omega(w) \text{ even}\}$. In the sequel, a parity game is written $\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}}, \Omega^{\mathcal{G}} \rangle$ with priority function $\Omega^{\mathcal{G}}$ given in place of *Acc*.

The *parity index* of a parity game $\mathcal{G}$ is defined to be $d = \max(\Omega(V^{\mathcal{G}}))$.

Observe that parity games are a special case of Muller games. In fact, one can define the Muller table $T = \{P \subseteq V^{\mathcal{G}} : \min(\Omega(P)) \text{ even } \}$ that define the same winning condition.

2.1.3.5 Theorem (Emerson et al. [59], Mostowski [131], Jurdzinski [107]). *In parity games a position is winning for a player if and only if this player has a memoryless winning strategy from this position. The problem of solving a parity game is in $UP \cap co\text{-}UP$ and winning positions (and strategies) can be computed in time $O(|\mathcal{G}|^{[d+1/2]})$.*

Remark. An algorithm for solving finite parity games is presented in section 2.2 below. It is a variant of Jurdzinski's small progress measure algorithm [108]. A basic algorithm to solve parity games with priorities $[0, 2k]$ interpreted by predicate symbols $P_{ii \in [0,2k]}$ and

partition of states described by predicate symbols P_E and P_P, amounts to evaluate the following mu-calculus formula:

$$\nu X_0 \mu X_1 \cdots \nu X_{2k}. \left(P_P \to \left(\bigwedge_{i \in [0,2k]} P_i \to \Diamond X_i \right) \wedge P_E \to \left(\bigwedge_{i \in [0,2k]} P_i \to \Box X_i \right) \right)$$

that computes [181, 11] the set of winning positions for the player P. A naive evaluation leads, for a game $\mathcal{G}$, to an algorithm in $O(|\mathcal{G}\|^{2k+1})$. A more clever evaluation of the mu-calculus formula (see for instance [162, 16]) gives an algorithm in $O(|\mathcal{G}|^{k+1})$.

2.1.4 Safety games

A safety game is a special kind of parity game where the player P wins a play if it never enters any of forbidden positions.

2.1.4.1 Definition. A *safety game* is a parity game

$$\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}}, \Omega^{\mathcal{G}} \rangle$$

with $\Omega^{\mathcal{G}}(V^{\mathcal{G}}) = \{0, 1\}$ and the property that for every vertex v:
- if $v \in V_P$ and all successors of v have priority 1 then $\Omega^{\mathcal{G}}(v) = 1$,
- if $v \in V_E$ and $\Omega^{\mathcal{G}}(v) = 1$ then there must exist a successor of v with priority 1,

The definition may at first seem too complicated, but we want it to be general with the property that a play is winning for the player P if and only if it never enters a position of priority 1. Observe in particular that, with this definition, player P positions with priority 0 always have successors.

2.1.4.2 Lemma. *In a safety game the player P has a $\sqsubseteq$-biggest winning strategy. This (memoryless) strategy is to stay in winning positions.*

Proof. To calculate the set of winning positions for the player P in a game G, one can proceed as follows. First, one sets $W_1 = \{v : \Omega(v) = 1\}$. Then, repeatedly one tries to find a vertex v such that either:
- $v \in V_P$ and all successors of v are in W_1, or
- $v \in V_E$ and there is a successor of v in W_1.

One adds v to W_1 and repeats the process. The loop ends when there are no more vertices to add.

It is not difficult to show that from every in W_1, the player E can force the play to reach (and to stay in) the set of vertices of priority 1 and thus the player E has winning strategy.

On the other hand, from every vertex in $W_0 = V^{\mathcal{G}} - W_1$, the player P has a strategy to stay in W_0. This strategy is given by $\sigma(v) = \{v' \in W_0 : T_P^{\mathcal{G}}(v, v')\}$. The strategy is maximal as no winning strategy can allow a play to get outside W_0.

□

Remark. In general, there is not $\sqsubseteq$-maximum winning strategies. Moreover, the following Theorem shows that, to some extent, existence of $\sqsubseteq$-maximum strategy characterizes safety games.

2.1.4.3 Theorem (Bernet, J. and Walukiewicz [26]). *If a game $\mathcal{G}$ with suffix closed acceptance condition has a $\sqsubseteq$-maximum winning strategy σ, then one can assign to each vertex of $\mathcal{G}$ a priority in $\{0,1\}$ in such a way that the result $\mathcal{G}'$ is a safety game and σ is also the $\sqsubseteq$-maximum winning strategy in $\mathcal{G}'$.*

Proof. Let σ be the $\sqsubseteq$-maximum winning strategy for player P, and let W be the set of winning positions in $\mathcal{G}$ for player P.

If every path through W is winning for the player P then we are done. We put $\Omega(v) = 0$ for every vertex in W and $\Omega(v) = 1$ for every other vertex.

Otherwise, suppose that there is a maximal path p in W which is not winning for the player P. Play p cannot be a finite path as every vertex in $W \cap V_P$ has a successor in W. We conclude the proof by showing that p cannot either be an infinite path.

In fact, for every finite prefix p' of p we define a strategy $\sigma_{p'}$ that extends σ by allowing moves along p'. When the play goes out of p', or p' finishes, the strategy becomes the same as σ. Formally, for every play q:

$$\sigma_{p'}(qv) = \begin{cases} \sigma(qv) \cup \{v'\} & \text{if } qvv' \text{ is a prefix of } p' \\ \sigma(q_2 v) & \text{if } q = q_1 q_2 \text{ and } q_1 \text{ is the longest common prefix} \\ & \text{of } p' \text{ and } q. \end{cases}$$

Every play respecting $\sigma_{p'}$ has a suffix which is a play starting from some $v \in W$ and respecting σ. Since $Acc^{\mathcal{G}}$ is closed under suffix every play respecting $\sigma_{p'}$ is winning.

Observe now that, because σ is the $\sqsubseteq$-maximum winning strategy, we also have $\sigma_{p'} \sqsubseteq \sigma$. It follows there is some counter strategy τ such that for every finite prefix p' of p, $p' \in \sigma * \tau(v)$ hence, by closure under limit (Lemma 2.1.1.4), $p \in \sigma * \tau(v)$ which is impossible since p is loosing for the player P while σ is winning.

□

Remark. In the above Theorem we consider relabeling that preserves the $\sqsubseteq$-maximum winning strategy (henceforth arbitrary winning strategy). We have not considered the relabeling that only preserves winning vertices because, according to such a weaker requirement, every game can be relabeled to a safety game. In fact, one can just put $\Omega(v) = 0$ for every vertex winning for the player P and $\Omega(v) = 1$ for all the vertices winning for the player E. After this relabeling the sets of winning vertices do not change as the player P has a strategy to stay in his winning vertices, and the player E has a strategy to stay in his winning vertices.

2.2 Permissive strategies in parity game

We have already seen that, in general, there is no $\sqsubseteq$-maximal winning strategy in a parity game. In this section, it is shown, however, that in finite parity games there always exists a regular strategy, called permissive strategy, that encompasses all the behaviors of all memoryless strategies. An algorithm for finding such a permissive strategy is presented. Its complexity matches currently known upper bounds for the simpler problem of finding the set of winning positions in a parity game. The algorithm can be seen as a reduction of a parity to a safety game and computation of the set of winning positions in the resulting game.

This material is extracted from paper [26]

2.2.1 Finding permissive strategies

2.2.1.1 Definition. A strategy σ in a parity game $\mathcal{G}$ is *permissive* when it is winning and if $\sigma' \sqsubseteq \sigma$ for every winning memoryless strategy σ'.

In this section we will show that there are finite memory permissive strategies. It can be shown that there cannot be a memory size that is sufficient for a permissive strategy in every game. Still we can hope to have one uniform memory for all the games of fixed size. There is a similar situation in the case of games with Muller's conditions [172, 184, 53]. There, the size of memory also cannot in general be bounded, but there is a finite memory sufficient for all games with conditions over some fixed set of elements.

Remark. One can define M-permissive strategies, which would be the strategies subsuming all strategies with memory of size M. The approach presented here extends to this setting, but we have chosen not to consider such strategies due to a substantial notational overhead.

For the rest of this section let us fix a set $I = \{0, \ldots, d+1\}$ of priorities and a number n_p for each odd priority $p \in I$. For convenience let us assume that d is odd. Let $\vec{n} = (n_1 n_3 \ldots n_d)$. This vector will be used to bound the size of considered games.

2.2.1.2 Definition. A parity game is $\vec{n}$ bounded if its set of priorities is included in $\{0, \ldots, d+1\}$ and there are at most n_p vertices of priority p, for each odd p.

In this section we will show a uniform construction of permissive strategies in $\vec{n}$-bounded games. For this we define a memory set $M(\vec{n})$ that will be used by our strategies.

$$M(\vec{n}) = \prod_{1 \leq p \leq d,\ p \text{ odd}} \{0, \ldots, n_p\}$$

An element $\vec{m} \in M(\vec{n})$ is a tuple of numbers $(m_1, m_3, \ldots, m_d)$ with $0 \leq m_i \leq n_i$. We can consider such a tuple as a counter representing the number

$$\sum_{i=1,3,\ldots,d} m_i \left(\prod_{j=i+2,i+4,\ldots,d} (n_j + 1) \right)$$

. So the most significant digit is the first one and each position p is in base n_p. For example, in the simple case when $n_p = 1$ for every p, we get a binary encoding of numbers up to $2^{(d+1)/2} - 1$.

The plan for finding a permissive strategy is the following. First, we will take $M_\top(\vec{n})$ which is an extension of $M(\vec{n})$ with an element $\top$ standing for overflow. Then, we will define a uniform memory update function $\delta_u : M_\top(\vec{n}) \times I \to M_\top(\vec{n})$. We call it uniform because it does not depend on vertices of a particular game but only on the priorities (and these are the same for all the games in question). Memory $M_\top(\vec{n})$ will allow to reduce a game $\mathcal{G}$ to a safety game $\mathcal{G}^\otimes$. The biggest strategy in this game will in turn be used to get a permissive strategy in $\mathcal{G}$.

To define the memory update function we need to define two kinds of auxiliary functions on memories: $\vec{m}|_p$ and $inc_p(\vec{m})$ for every $p \in I$. The first is just resetting to 0 all the positions bigger than p:

$$(m_1, m_3, \ldots, m_d)|_p = (m_1, \ldots, m_p, 0, \ldots, 0)$$

This operation is also defined for even p in an obvious way.

The other operation is like adding 1 to position p when considering $\vec{m}$ as a counter; if the value on this position is already n_p then we try recursively to add 1 to the position $p-2$:

$$inc_p((m_1,\ldots,m_d)) = \begin{cases} (m_1,\ldots,m_p+1,\ldots,m_d) & \text{if } m_p < n_p \\ inc_{p-2}((m_1,\ldots,m_d)) & \text{if } m_p = n_p \text{ and } p \geq 3 \\ \top & \text{otherwise} \end{cases}$$

The intuition for the last case of the above definition is that if the value of the counter on first p positions is $n_1 n_2 \ldots n_p$ then adding 1 is impossible and the value is $\top$ which denotes an overflow.

Now, we can define a generic update function $\delta_u : M_\top(\vec{n}) \times I \to M_\top(\vec{n})$,

$$\delta_u(m,p) = \begin{cases} m|_p & \text{for } p \text{ even} \\ inc_p(m) & \text{for } p \text{ odd} \end{cases}$$

Of course we also have $\delta_u(\top,p) = \top$ which means that there is no possibility to recover from the overflow. Observe that in the above we have stopped to write vectors over m. We will do it often for clarity.

Using the memory $M_\top(\vec{n})$ and the function δ_u we can reduce any $\vec{n}$ bounded game $\mathcal{G}$ to a safety game. Let us take an $\vec{n}$-bounded game $\mathcal{G} = \langle V, V_P, V_E, T, I, \Omega\rangle$. Define a safety game $\mathcal{G}^\otimes = \langle V^\otimes, V_P^\otimes, V_E^\otimes, T^\otimes, \{0,1\}, \Omega^\otimes\rangle$, where:

- $V_i^\otimes = V_P \times M_\top(\vec{n})$, for $i = 0, 1$;
- $T^\otimes((v,m),(v',m'))$ if $T(v,v')$ and $m' = \delta_u(m, \Omega(v))$;
- $\Omega^\otimes((v,m)) = 0$ if $m \neq \top$ and $\Omega^\otimes((v,\top)) = 1$.

So the player P wins in $\mathcal{G}^\otimes$ from a position (v,m) if he has a strategy to avoid vertices with $\top$ in the second component. By Lemma 2.1.4.2, in such a game there is always a maximal memoryless winning strategy.

A memoryless strategy $\sigma^\otimes$ in $\mathcal{G}^\otimes$ gives a strategy σ with memory $M(\vec{n})$ in $\mathcal{G}$. The strategy is defined by $\sigma(m,v) = \sigma^\otimes((v,m))$, the initial memory element is $m_0 = (0,\ldots,0)$ and the memory update function is $\delta_u(m,v) = \delta_u(m,\Omega(v))$.

2.2.1.3 Lemma. *For every $\vec{n}$ bounded game $\mathcal{G}$. If $\sigma^\otimes$ is a memoryless strategy winning from (v,m) in $\mathcal{G}^\otimes$ then σ is a winning strategy from v with initial memory m.*

Proof. The main observation is that if we have an infinite play $(v_1,m_1)(v_2,m_2)\ldots$ and $\top$ does not appear in the sequence, then the sequence $v_1 v_2 \ldots$ satisfies the parity condition. Suppose the contrary; then some odd priority p would be the smallest one appearing infinitely often in $v_1 v_2 \ldots$ But then, by the definition of δ_u function, we will get $\top$ after meeting $(n_1 \cdot n_3 \cdots n_p + 1)$ times a vertex of priority p and not meeting any vertex of smaller priority in between.

To see that σ is winning from v with initial memory m it is enough to note that for every play $vv_1v_2\ldots$ from v respecting σ there is a sequence of memories $mm_1m_2\ldots$ such that $(v,m)(v_1,m_1)(v_2,m_2)\ldots$ is a play from (v,m) respecting $\sigma^\otimes$.

□

There is also a construction in the opposite direction. A memoryless strategy σ in $\mathcal{G}$ defines a memoryless strategy $\sigma^\otimes$ in $\mathcal{G}^\otimes$ by:

$$\sigma^\otimes(v,m) = \{(v', \delta_u(m,\Omega(v))) : v' \in \sigma(v)\}$$

2.2.1.4 Lemma. *For every $\vec{n}$ bounded game $\mathcal{G}$ and every memoryless strategy σ for the player P. If σ is a winning strategy from v then $\sigma^{\otimes}$ is winning from $(v, (0, \ldots, 0))$ in $\mathcal{G}^{\otimes}$.*

Proof. Suppose that $\sigma^{\otimes}$ is not winning from (v, m_0) where $\vec{m}_0 = (0, \ldots, 0)$. Then there is a finite path $(v, \vec{m}_0)(v_1, \vec{m}_1)(v_2, \vec{m}_2) \ldots (v_{k+1}, \vec{m}_{k+1})$ such that $\vec{m}_{k+1} = \top$. This can happen only because $\vec{m}_k = (n_1, n_3, \ldots, n_q, \ldots)$ and $\Omega(v_k) = q$, i.e., the counter $\vec{m}_{k+1}$ overflows.

Let i be the smallest integer such that $m_{i,p} = n_p$, where $p = \Omega(v_i)$ and $\vec{m}_i = (m_{i,1}, m_{i,3}, \ldots)$. So we take the first vertex where the counter reaches the maximal value on the position corresponding to the priority of the vertex. Unlike in the paragraph above we do not require that all smaller positions have maximal values. So p may be different from q. Take the largest $j < i$ s.t. $\Omega(v_j)$ is both even and less than p (or take $j = -1$ if there is no such vertex). By definition of δ_u function we have $m_{j+1,p} = 0$. By the choice of i, in all memories up to i no position reaches its maximal allowed value. So by the definition of δ_u function, the value on position p can increase only when we see a vertex of priority p. Hence, there must exist $n_p + 1$ occurrences of vertices of priority p between v_j and v_i. As the game $\mathcal{G}$ is $\vec{n}$ bounded, some vertex must occur twice. This is a contradiction with the fact that $vv_1v_2 \ldots v_k$ is a play respecting σ. On such a play there cannot be a loop through a vertex of odd priority p without a vertex of smaller priority on this loop since σ is winning.

□

2.2.1.5 Theorem (Bernet, J. and Walukiewicz [26]). *For every $\vec{n} = (n_1, n_3, \ldots, n_d)$ and for every $\vec{n}$-bounded game $\mathcal{G}$ there is a permissive strategy on $\mathcal{G}$ using memory $M_{\top}(\vec{n})$.*

Proof. Let $\sigma^{\otimes}$ be the maximal winning strategy in the game $\mathcal{G}^{\otimes}$. This defines in $\mathcal{G}$ a strategy σ with memory $M_{\top}(\vec{n})$. The strategy is winning by Lemma 2.2.1.4. We want to show that it is a permissive strategy. For this we take some memoryless winning strategy σ_1 in $\mathcal{G}$ and show that $Beh(\mathcal{G}, v_0, \sigma_1) \subseteq Beh(\mathcal{G}, v_0, \sigma)$ for every v_0.

Take $v_0v_1 \ldots \in Beh(\mathcal{G}, v, \sigma_1)$. By Lemma 2.2.1.4, there are memories such that

$$(v_0, m_0)(v_1, m_1) \ldots \in Beh(\mathcal{G}^{\otimes}, (v, m), {\sigma_1}^{\otimes})$$

Next, by the maximality of $\sigma^{\otimes}$, we have $Beh(\mathcal{G}^{\otimes}, (v, m), {\sigma_1}^{\otimes}) \subseteq Beh(\mathcal{G}^{\otimes}, (v, m), \sigma^{\otimes})$ for every (v, m). So, $(v_0, m_0)(v_1, m_1) \ldots \in Beh(\mathcal{G}^{\otimes}, (v_0, m_0), \sigma^{\otimes})$. Finally, by the definition of σ we have that $v_1v_2 \ldots \in Beh(\mathcal{G}, v, \sigma)$

□

Remark. The memory as defined above is essentially nothing more than a deterministic automaton accepting sequences satisfying a parity condition. The important point is that this automaton is a safety automaton. It is well known that deterministic safety automata cannot recognize the language of all the sequences satisfying a parity condition [171]. We overcome this problem by limiting the number of odd priorities that can appear in the sequence without a smaller even priority in between. Other solutions are also possible with other memories and other permissive strategies.

2.2.2 Small representations of permissive strategies

In the previous section we have seen that for every game $\mathcal{G}$ there is a permissive strategy that can be represented as the biggest strategy in $\mathcal{G}^{\otimes}$. The size of $\mathcal{G}^{\otimes}$ is $(|\mathcal{G}| \cdot n_1 \cdot n_3 \cdots n_d)$,

hence it is exponential in the size of $\mathcal{G}$. So at first glance it may seem that we need this much space to describe a permissive strategy. Fortunately it is not the case. Here we will show that a permissive strategy can be determined by a function $Max : V \to M(\vec{n})$, i.e., a function assigning one memory value to each node.

The key observation is that the lexicographic ordering on memories is also a "permissiveness" ordering. We say that $\vec{m}' = (m'_1, m'_3, \ldots, m'_d)$ is *lexicographically smaller* than $\vec{m} = (m_1, m_3, \ldots, m_d)$, denoted $\vec{m}' <_L \vec{m}$, if there is a p such that $m'_p \neq m_p$, and $m'_p < m_p$ for the smallest such p. We extend this ordering by two new elements $\bot$ and $\top$ with $\bot <_L \vec{m} <_L \top$ for every $\vec{m} \in M(\vec{n})$. These two elements signify undefined and overflow respectively. Element $\top$ was already introduced in the previous section.

2.2.2.1 Lemma. *For every game $\mathcal{G}^{\otimes}$: if the player P has a winning strategy from a position $(v, \vec{m})$ then he has a winning strategy from position $(v, \vec{m}')$ for every $\vec{m}' <_L \vec{m}$.*

Proof. For the proof it is enough to observe that δ_u function is monotonic, i.e., for every priority p: $\delta_u(\vec{m}', p) \leq_L \delta_u(\vec{m}, p)$ if $\vec{m}' \leq_L \vec{m}$. In particular for overflow it means that: if $\delta_u(\vec{m}', p) = \top$ and $\vec{m}' <_L \vec{m}$ then $\delta_u(\vec{m}, p) = \top$.

□

For each vertex v, let $Max(v)$ be the $<_L$-supremum of all the memories m such that (v, m) is winning for the player P in $\mathcal{G}^{\otimes}$. So, if there is no such memory then $Max(v) = \bot$. By Lemma 2.2.1.4, $Max(v) = \bot$ if and only if v is not winning for the player P in $\mathcal{G}$. By definition, $Max(v)$ can never be $\top$.

We can use $Max(v)$ to get a permissive strategy. It is defined by telling for every v for which memories m the position (v, m) is winning in $\mathcal{G}^{\otimes}$. As $Max(v)$ gives the biggest such m, we know that (v, m) is winning for exactly those m that are lexicographically not bigger than $Max(v)$. So in a vertex v with memory $m \leq_L Max(v)$ the strategy is $\sigma(m, v) = \{v' : \delta_u(m, \Omega(v)) \leq_L Max(v')\}$.

2.2.3 Algorithmic issues

Here we will describe how to use the reduction from $\mathcal{G}$ to $\mathcal{G}^{\otimes}$ in algorithms for solving parity games, i.e., algorithms that find the set of vertices from which the player P has a winning strategy.

A simple algorithm for solving a $\vec{n}$ bounded game $\mathcal{G}$ is to construct $\mathcal{G}^{\otimes}$ and solve this safety game. This can be done by any alternating reachability algorithm. The size of $\mathcal{G}^{\otimes}$ is $(|\mathcal{G}| \cdot n_1 \cdot n_3 \cdots n_d)$, where n_p is the number of vertices of priority p in $\mathcal{G}$. Hence, the time complexity of this algorithm is as good as the best known upper bounds for solving parity games. The weakness of this approach, however, is that a memory needed for alternating reachability algorithm is proportional to the size of the game, and hence exponential in the number of priorities.

Yet, a better approach is available. The idea is to calculate Max function in a bottom-up way. Before presenting the algorithm we need to define a function $down$. For a memory m and a priority p, we put

$$down(m, p) = \max\{m' : \delta_u(m', p) \leq m\}$$

Hence, the value of $down(m, p)$ can be $\bot$ if $m = (0, \ldots, 0)$. It is easy to check that

$down(m,p)$ can be defined in a similar way to $\delta_u(m,p)$:

$$down(m,p) = \begin{cases} m|^p & \text{if } p \text{ even} \\ dec_p(m) & \text{if } p \text{ odd} \end{cases}$$

where

$$(m_1,\ldots,m_p)|^p = \quad (m_1,\ldots,m_p,n_{p+2},\ldots,n_d)$$

$$dec_p(m_1,\ldots,m_d) = \begin{cases} (m_1,\ldots,m_p-1,\ldots,m_d) & \text{if } m_d > 0 \\ dec_{p-2}(m_1,\ldots,m_d) & \text{if } m_p = 0 \text{ and } p \geq 3 \\ \bot & \text{otherwise} \end{cases}$$

The algorithm calculating function Max will work with the auxiliary assignment $F : V \to (M(\vec{n}) \cup \{\bot\})$. Initially we put $F(v) = \vec{n}$ for each v; recall that $\vec{n} = (n_1, n_3, \ldots, n_d)$. Afterwards, we start a loop were we find a vertex v such that

$$F(v) >_L down(m', \Omega(v))$$

where

$$m' = \begin{cases} \max\{F(v') : v' \text{ successor of } v\} & \text{if } v \in V_P \\ \min\{F(v') : v' \text{ successor of } v\} & \text{if } v \in V_E \end{cases}$$

For such v we set $F(v) = down(m', \Omega(v))$ and repeat the loop. We stop when we cannot find a vertex with the above property. We show below that at the end $F(v) = Max(v)$ for all vertices v.

Remark. The algorithm is just a computation of the greatest fixpoint of some operator on $V \to (M(\vec{n}) \cup \{\bot\})$. The lemmas below make it more explicit.

2.2.3.1 Lemma. *If $F : V \to (M(\vec{n}) \cup \{\bot\})$ is such that the value of no vertex can be decreased then $F(v) \leq_L Max(v)$ for all vertices v.*

Proof. It is enough to show that for every v with $F(v) \neq \bot$ the position $(v, F(v))$ in $\mathcal{G}^{\otimes}$ is winning for the player P. The observation we need is that if F is as in the assumption of the lemma then for every v s.t. $F(v) \neq \bot$ we have:

– if $v \in V_P$ then there must be a successor v' with $\delta_u(F(v), \Omega(v)) \leq_L F(v')$;
– if $v \in V_E$ then for all successors v' of v we have $\delta_u(F(v), \Omega(v)) \leq_L F(v')$.

Now the strategy for the player P is to choose in every $v \in V_P$ a successor v' such that $\delta_u(F(v), \Omega(v)) \leq_L F(v')$. By the above this is possible for every vertex with $F(v) \neq \bot$. To see that this strategy is winning take a play $(v_1, m_1)(v_2, m_2)\ldots$ respecting the strategy where $m_1 = F(v_1) \neq \bot$. Using the property above we get by induction on i that $m_i \leq_L F(v_i)$. Hence, $m_i \neq \top$ for every i, which means that the play is winning.

□

2.2.3.2 Lemma. *After each iteration of the above loop we have $F(v) \geq_L Max(v)$ for all vertices v.*

Proof. The proof is by induction on the number of iterations. The statement is true at the beginning when $F(v) = \vec{n}$ for every v. For the induction step we assume that $F(v) \geq_L Max(v)$ holds for every v and we choose one v for which $F(v)$ can be decreased.

Suppose that we have chosen $v \in V_P$ and it is to be decreased. We need to show that the new value of $F(v)$ is still not smaller than $Max(v)$. If $Max(v) = \bot$ then we are done. Otherwise, as $Max(v)$ is a memory that still guarantees a win for the player P, we know that v has a successor v' with $\delta_u(Max(v), \Omega(v)) \leq_L Max(v')$. Applying $down$ function to both sides we get:

$$Max(v) \leq_L down(\delta_u(Max(v), \Omega(v)), \Omega(v)) \leq_L down(Max(v'), \Omega(v))$$

The first inequality follows by the property: $m \leq_L down(\delta_u(m, p), p)$ for every $m \in M(\vec{n})$. The second inequality follows from the monotonicity of $down$. The new value of $F(v)$ is not smaller than $down(F(v'), \Omega(v))$. So we are done as

$$down(F(v'), \Omega(v)) \geq_L down(Max(v'), \Omega(v)) \geq_L Max(v)$$

The case for $v \in V_E$ is similar.

□

2.2.3.3 Corollary. *At the end of the algorithm $F(v) = Max(v)$.*

Let us calculate the complexity of the algorithm. It cannot do more than than $(|\mathcal{G}| \cdot n_1 \cdot n_3 \cdots n_d)$ steps. This is because at each step the F value of some node is decreased and the value of a node cannot be decreased more than $n_1 \cdot n_3 \cdots n_d$ times. The algorithm uses linear memory, as it needs to store just the current values of F assignment. This matches the best known upper bounds for solving parity games [108]. The known upper bound presently known for the strategy improvement algorithm [179] is actually worse: $(n/d)^d$ instead of $(n/d)^{\lceil d/2 \rceil}$.

2.2.4 Related open problems

Learning from the experience of discrete control synthesis theory, it seems to be a good idea to compare strategies by comparing the sets of behaviors they allow. As we presented above, there are parity games where there is no winning strategy that allows all the behaviors of all possible winning strategies in the game. Given this, we propose a more lax notion of permissive strategy which is a strategy that allows all the behaviors of all memoryless strategies. We show that a permissive strategy exists for every game and that the algorithm finding it has not worse complexity than currently known algorithms for a simpler problem of deciding if there is any winning strategy from a given vertex. Actually, the algorithm we obtain is exactly the signature improvement algorithm presented in [108]. Hence, we show that this algorithm computes more than just a set of winning vertices (and some winning strategy).

There are at least two interesting open problems. The first concerns the size of permissive strategy. We have shown that for an $\vec{n} = (n_1, \ldots, n_d)$ bounded game there is a strategy with memory of size $n_1 \cdot n_2 \cdots n_d$. We don't known whether there can be a memory of smaller size. Actually if there were a memory of size polynomial in $n_1 + n_2 + \cdots + n_d$ then it would give a PTIME algorithm for solving parity games. Our reduction to safety games shows that the question about minimal memory is equivalent to the question about automata on infinite words. The goal is to find a minimal automaton accepting all paths that are admitted by some memoryless strategy in some $\vec{n}$-bounded game.

The other problem also concerns complexity. We have shown that a permissive strategy in a game is defined by a function $Max : V \to (M(\vec{n}) \cup \bot)$. This function is unique for a given game. Hence, if we were able to check in PTIME that a given function $F : V \to (M(\vec{n}) \cup \bot)$ is exactly the *Max* function then we would show that solving parity games is in UP∩co-UP. This would be interesting as the known arguments for UP∩co-UP bound are indirect and go through discounted payoff games [107, 185].

2.3 Reasoning with games

Games are used in the following chapters to give semantics to automata. It follows that many proofs require construction of winning strategies in some games under the hypothesis that winning strategies exist in other games. In this section, we develop several tools that help us doing so in a more uniform way. In other words, we aim at defining, within game theory, a general vocabulary and proof techniques that can be used later when games are used as a semantical tool in automata (and fixed point) theory.

More specifically, in this section, we study several notions of simulation relation between games that somehow extend in a back and forth way the standard notion of graph simulations. They give sufficient condition to ensure the existence of winning strategies in the simulated games from existence of winning strategies in the simulating games.

The simplest definition, called asynchronous simulation, extends a similar notion defined in [31, 105] to infinite plays. Such an extension have also been considered in [159].

Later in this section, we also define a notion of synchronous simulation; being more simple to understand although weaker in its capacity to relate games, it is the most commonly used proof techniques in the remainder of the text.

Last, we also define a most general notion called generalized simulation. It is strictly more powerful than asynchronous simulation.

2.3.1 Asynchronous game simulations

Asynchronous game simulations are defined most conveniently by means of winning strategies in what we called a simulation game product.

2.3.1.1 Definition (Simulation product). Given two games

$$\mathcal{G}_1 = \langle V_P^{\mathcal{G}_1}, V_E^{\mathcal{G}_1}, T_P^{\mathcal{G}_1}, T_E^{\mathcal{G}_1}, Acc^{\mathcal{G}_1} \rangle \text{ and } \mathcal{G}_2 = \langle V_P^{\mathcal{G}_2}, V_E^{\mathcal{G}_2}, T_P^{\mathcal{G}_2}, T_E^{\mathcal{G}_2}, Acc^{\mathcal{G}_2} \rangle$$

the *simulation product* $\mathcal{G}_1 \; {}_E{\rightleftharpoons}^P \; \mathcal{G}_2$ is defined to be the game

$$\mathcal{G}_1 \; {}_E{\rightleftharpoons}^P \; \mathcal{G}_2 = \langle V_P, V_E, T_P, T_E, Acc \rangle$$

with set of P-positions $V_P^{\mathcal{G}} = V_E^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2}$, set of E-positions $V_E^{\mathcal{G}} = (V_P^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2}) \cup (V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2})$, sets of moves defined as follows:

1. T_P is the set of all pairs $((u_1, u_2), (v_1, v_2)) \in V_P \times V_E$ such that either:
   (a) $(E, P) \to (P, P)$ moves: $(u_1, v_1) \in T_E^{\mathcal{G}_1}$ and $u_2 = v_2$,
   (b) or $(E, P) \to (E, E)$ moves: $u_1 = v_1$ and $(u_2, v_2) \in T_P^{\mathcal{G}_2}$,
2. T_E is the set of all pairs $((u_1, u_2), (v_1, v_2)) \in V_E \times V_P$ such that either:

(a) $(P,P) \to (E,P)$ moves: $(u_1, v_1) \in T_P^{\mathcal{G}_1}$ and $u_2 = v_2$,

(b) or $(E,E) \to (E,P)$ moves: $u_1 = v_1$ and $(u_2, v_2) \in T_E^{\mathcal{G}_2}$,

and winning condition *Acc* as the set of all infinite words $p \in V_1^{\mathcal{G}}$ such that, given $p_1 = view(\pi_1(p))$ and $p_2 = view(\pi_2(p))$ if p_1 is winning for P then p_2 is winning for P.

An *asynchronous game simulation* from the game $\mathcal{G}_1$ in position v_1 to the game $\mathcal{G}_2$ in position v_2 is a winning strategy in game $\mathcal{G}_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2$ from position (v_1, v_2). We write $\mathcal{G}_1, v_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2, v_2$ when there is such a game simulation.

2.3.1.2 Lemma (Soundness). *If $\mathcal{G}_1, v_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2, v_2$ and if v_1 is winning for player P in the game $\mathcal{G}_1$ (resp. v_2 is winning for the player E in $\mathcal{G}_2$) then v_2 is winning for the player P in the game $\mathcal{G}_2$ (resp. v_1 is winning for the player E in the game $\mathcal{G}_1$).*

Proof. Assume $\mathcal{G}_1, v_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2, v_2$. By duality and game determinacy, it is sufficient to consider only the case when v_1 is winning for the player P in $\mathcal{G}_1$.

Let $\mathcal{G}_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2 = \langle V_P, V_E, T_P, T_E, Acc\rangle$ be the simulation game product, let

$$\varphi : V^*.V_P \to V_E$$

with V standing for $V_P \cup V_E$, be a (deterministic) strategy for the player P in $\mathcal{G}_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2$ winning from position (v_1, v_2).

Let σ_1 be a (deterministic) winning strategy for the player in the game $\mathcal{G}_1$ from position v_1.

For every (deterministic) counter strategy τ_2 for the player E in the game $\mathcal{G}_2$ we define counter strategy $\sigma_1 \otimes \tau_2$ for the player E in the game $\mathcal{G}_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2$ as follows: for every $p \in V^*$ and every $(u_1, u_2) \in V_E$:

1. $(P,P) \to (E,P)$ move:
   if $(u_1, u_2) \in V_P^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2}$ then $\sigma_1 \otimes \tau_2(p.(u_1, u_2)) = (\sigma_1(view(\pi_1(p).u_1)), u_2)$
2. $(E,E) \to (E,P)$ move:
   if $(u_1, u_2) \in V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2}$ then $\sigma_1 \otimes \tau_2(p.(u_1, u_2)) = (u_1, \tau_2(view(\pi_2(p).u_2)))$

Observe that strategy $\sigma_1 \otimes \tau_2$ is deterministic.

We define then a (deterministic) strategy with (possibly infinite) memory σ_2 for the player P in the game $\mathcal{G}_2$ as follows. Memory M is defined to be $\mathcal{P}(V^+)$, i.e. finite plays in game $\mathcal{G}_1 \ {}_E\rightleftharpoons^P \mathcal{G}_2$. Update function $\delta_u : M \times V^{\mathcal{G}_2} \to M$ and choice function $c : M \times V_P^{\mathcal{G}_2} \to V_E^{\mathcal{G}_2}$ are defined below, by induction over the length of plays the counter strategy allows, in such a way that the following invariant is satisfied:

> (I) for every counter strategy τ_2 in the game $\mathcal{G}_2$, for every finite plays $p_2 \in \sigma_2 * \tau_2(v_2)$, given $p = \delta_u^*(p_2)$ (the memory value after playing p_2 following strategy σ_2), $p \in \varphi * (\sigma_1 \otimes \tau_2)$ and $p_2 = view(\pi_2(p))$.

This is done as follows. For initial memory m_0, we put $m_0 = (v_1, v_2)$. Obviously, invariant (I) is satisfied with $p_2 = v_2$.

Assume that, for a given counter strategy τ_2 in the game $\mathcal{G}_2$, we have reached a play of the form $p_2.u_2 \in \sigma_2 * \tau_2(v_2)$ with $p_2 \in (V^{\mathcal{G}_2})^*$ and $u_2 \in V^{\mathcal{G}_2}$, such that (I) is satisfied, i.e. given $\delta_u^*(p_2.u_2)$ of the form $p.(u_1, u_2)$ with $p \in V^*$ and $(u_1, u_2) \in V$, we have $p.(u_1, u_2) \in \varphi * (\sigma_1 \otimes \tau_2)(v_1, v_2)$ and $p_2.u_2 = view(\pi_2(p.(u_1, u_2)))$. Now, two cases have to be considered.

The first case, when $u_2 \in V_E^{\mathcal{G}_2}$, is easy. By definition of $\mathcal{G}_1 \,{}_E{\rightleftharpoons}^P \mathcal{G}_2$, this implies that $u_1 \in V_E^{\mathcal{G}_1}$ (henceforth $(u_1, u_2) \in V_E$. If $\tau_2(p_2.u_2)$ is undefined, the induction is over (the player E loses). Otherwise, given $u_2' = \tau_2(p_2.u_2)$, by definition $\sigma_1 \otimes \tau_2(p.(u_1, u_2)) = (u_1, u_2')$ and we put $\delta_u(p.(u_1, u_2), u_2') = p.(u_1, u_2).(u_1, u_2')$. Invariant I is still satisfied.

The remaining case, when $u_2 \in V_P^{\mathcal{G}_2}$, is more technical and detailed below.

Let p' be a maximal play such that $p.(u_1, u_2).p' \in \varphi * (\sigma_1 \otimes \tau_2)(v_1, v_2)$ and such that $view(\pi_2(p')) = u_2$, i.e. a maximal among the plays such that, from position (u_1, u_2), while playing p', no move is made on the $\mathcal{G}_2$ side of the game $\mathcal{G}_1 \,{}_E{\rightleftharpoons}^P \mathcal{G}_2$.

We first claim that p' is finite. Otherwise $p.(u_1, u_2).p'$ is infinite and thus maximal in $\varphi * (\sigma_1 \otimes \tau_2)(v_1, v_2)$. It follows that $p.(u_1, u_2).p'$ is winning for the player P (since φ is winning from position (v_1, v_2)) which contradicts the fact that $view(\pi_2(p.(u_1, u_2).p'))$ is finite and thus loosing for the player P (since $u_1 \in V_P^{\mathcal{G}_1}$) while $view(\pi_1(p.(u_1, u_2).p'))$ is infinite and thus winning for the player P (since σ_1 is winning from position v_1).

We claim next that p' ends in a position in V_P. Otherwise, it ends in a position in $V_P^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2}$ while $\sigma_1(view(\pi_1(p.(u_1, u_2).p'))$ is defined (since σ_1 is winning) henceforth $\sigma_1 \otimes \tau_2(p.(u, v).p')$ is also defined which contradict the maximality hypothesis.

It follows that $\varphi(p.(u_1, u_2).p')$ is defined (since φ is winning) and, moreover, given $u_2' = \pi_2(\varphi(p.(u_1, u_2).p')$ we claim that $(u_2, u_2') \in T_P^{\mathcal{G}_1}$, i.e. φ defines a move in the $\mathcal{G}_2$ side of the simulation game. In fact, if this is not the case, it means that φ defines a move in the $\mathcal{G}_1$ side of the simulation game which, again, contradicts the maximality hypothesis. Now, we put $c(p.(u_1, u_2), u_2) = u_2'$ and $\delta_u(p.(u_1, u_2), u_2') = p.(u_1, u_2).p'.\varphi(p.(u_1, u_2).p')$. By construction, this is a well defined move, and invariant I is still satisfied.

It remains to prove that σ_2 as defined by m_0, δ_u and c above is winning for the player P from position v_2. Let τ_2 be a counter strategy and let p_2 be a maximal play $\sigma_2 * \tau_2(v_2)$.

When p_2 is finite, we claim that p_2 ends in $V_E^{\mathcal{G}_2}$. Otherwise, p_2 ends in $V_P^{\mathcal{G}_2}$ with $\sigma_2(p_2)$ undefined. Since both σ_1 and φ are winning for the player P this is impossible as shown by the arguments above defining σ_2 in this case.

When p_2 is infinite, the invariant properties (I) tells us that there is a unique infinite play $p \in \varphi * (\sigma_1 \otimes \tau_2)(v_1, v_2)$ such that, for each finite prefix p_2' of p_2, given $p' = view(\pi_2(\delta_u(p_2')))$, p' is a prefix of p with $p_2 = view(\pi_2(p))$. This tells us that $p_2 = view(\pi_2(p))$. Now, given $p_1 = view(\pi_1(p))$ we know that both p_1 is winning (by definition of strategy $\sigma_1 \otimes \tau_2$ with σ_1 winning) and p is winning (since φ is winning) henceforth, by definition of the winning condition in $\mathcal{G}_1 \,{}_E{\rightleftharpoons}^P \mathcal{G}_2$, p_2 is also winning.

□

Remark. In the construction of strategy σ_2 above, (some subgame of) the game $\mathcal{G}_1$ acts like a automaton like device that keep all information needed for the player P to define/use and win with strategy σ_2. This implies, in particular, that even if σ_1 is a memoryless strategy, strategy σ_2 may still require some memory.

Observe also that σ_2 is not uniquely (nor even canonically) defined. In fact, in presence of a simulation relation from $\mathcal{G}_1$ to $\mathcal{G}_2$, there are many ways to simulate plays in $\mathcal{G}_2$ by means of plays in $\mathcal{G}_1$.

2.3.2 Synchronous game morphisms and simulations

In this section, somehow, we extend the notion of graph morphisms to games. Doing so, we recover a restricted notion of asynchronous game simulation called synchronous simulation [5].

2.3.2.1 Definition (Synchronous game morphisms). Given two games

$$\mathcal{G}_1 = \langle V_P^{\mathcal{G}_1}, V_E^{\mathcal{G}_1}, T_P^{\mathcal{G}_1}, T_E^{\mathcal{G}_1}, Acc^{\mathcal{G}_1} \rangle$$

and

$$\mathcal{G}_2 = \langle V_P^{\mathcal{G}_2}, V_E^{\mathcal{G}_2}, T_P^{\mathcal{G}_2}, T_E^{\mathcal{G}_2}, Acc^{\mathcal{G}_2} \rangle$$

a *P-morphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ is a relation $R \subseteq V^{\mathcal{G}_1} \times V^{\mathcal{G}_2}$ such that:

1. $R(V_P^{\mathcal{G}_1}) \subseteq V_P^{\mathcal{G}_2}$,
2. $R(V_E^{\mathcal{G}_1}) \subseteq V_E^{\mathcal{G}_2}$,
3. $T_E^{\mathcal{G}_2} \circ R \subseteq R^{-1} \circ T_E^{\mathcal{G}_1}$, i.e. for every $(u_1, u_2) \in R \cap (V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2})$, for every $v_2 \in V_P^{\mathcal{G}_2}$ such that $(u_2, v_2) \in T_E^{\mathcal{G}_2}$ there exists $v_1 \in V_P^{\mathcal{G}_1}$ such that $(u_1, v_1) \in T_E^{\mathcal{G}_1}$ and $(v_1, v_2) \in R$,
4. $T_P^{\mathcal{G}_1} \circ R^{-1} \subseteq R \circ T_P^{\mathcal{G}_2}$, i.e. for every $(u_1, u_2) \in R \cap (V_P^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2})$, for every $v_1 \in V_E^{\mathcal{G}_2}$ such that $(u_1, v_1) \in T_P^{\mathcal{G}_1}$ there exists $v_2 \in V^{\mathcal{G}_2}$ such that $(u_2, v_2) \in T_P^{\mathcal{G}_2}$ and $(v_1, v_2) \in R$,
5. $R(Acc^{\mathcal{G}_1}) \subseteq Acc^{\mathcal{G}_2}$.

A game *E-morphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ is a P-morphism from $\overline{\mathcal{G}_1}$ to $\overline{\mathcal{G}_2}$.
A game *PE-morphism* is a P-morphism that is also an E-morphism.

Remark. Observe that a PE-morphism is a bisimulation relation.

Observe also that on E-deterministic games (resp. on P-deterministic games) P-morphisms (resp. E-morphisms) are just standard graph simulations between the underlying game graphs.

2.3.2.2 Lemma. *If there is a P-morphism (resp. a E-morphism) $R : \mathcal{G}_1 \to \mathcal{G}_2$ then, for every position $(v_1, v_2) \in R$, $\mathcal{G}_2, v_2 \; {}_E\!\rightleftharpoons^P \mathcal{G}_1, v_1$ (resp. $\mathcal{G}_1, v_1 \; {}_E\!\rightleftharpoons^P \mathcal{G}_2, v_2$). In particular, if R is a PE-morphism, v_1 is winning for the player P in game $\mathcal{G}_1$ if and only if v_2 is winning for the player P in game $\mathcal{G}_2$.*

Proof. By symmetry, it is sufficient to prove the P-morphism case. Let $R : \mathcal{G}_1 \to \mathcal{G}_2$ be a P-morphism. We define strategy φ in $\mathcal{G}_1 \; {}_E\!\rightleftharpoons^P \mathcal{G}_2$ as follows. The initial positions we consider are in R. Strategy φ is then defined almost positional in the sense that it only depends on the last two positions reached in a finite play ending in a position P.

More precisely, for every position $u = (u_1, u_2) \in R$, we define φ as follows:

1. if $(u_1, u_2) \in V_P^{\mathcal{G}_1} \times V_P^{\mathcal{G}_2}$, for every $u_1' \in T_P^{\mathcal{G}_1}(u_1)$, we have $(u_2, u_1') \in T_P^{\mathcal{G}_1} \circ R^{-1}$ hence $(u_2, u_1') \in R \circ T_P^{\mathcal{G}_2}$; in other words, there exists $u_2' \in T_P^{\mathcal{G}_2}(u_2)$ such that $(u_1', u_2') \in R$, so, after the $(P, P) \to (E, P)$ move $(u_1, u_2) \to (u_1', u_2)$, we can define $\varphi((u_1, u_2).(u_1', u_2))$ to be (u_1', u_2') and this is a well-defined $(E, P) \to (E, E)$ move,
2. if $(u_1, u_2) \in V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2}$, for every $u_2' \in T_P^{\mathcal{G}_2}(u_2)$, we have $(u_1, u_2') \in T_P^{\mathcal{G}_2} \circ R$ hence $(u_1, u_2') \in R^{-1} \circ T_P^{\mathcal{G}_1}$; in other words, there exists $u_1' \in T_P^{\mathcal{G}_1}(u_1)$ such that $(u_1', u_2') \in R$, so, after the $(E, E) \to (E, P)$ move $(u_1, u_2) \to (u_1, u_2')$, we can define $\varphi((u_1, u_2).(u_1, u_2'))$ to be (u_1', u_2') and this is a well-defined $(E, P) \to (P, P)$ move.

One can easily check that φ defined in such a way is a winning strategy in $\mathcal{G}_1 \,{}_E\!\rightleftharpoons^P \mathcal{G}_2$. In fact, from a position $(v_1, v_2) \in R$, by construction, a maximal finite play can only ends in player E position and thus is winning for the player P. Otherwise, p is infinite, and, by construction, given $p_1 = view(\pi_1(p))$ and $p_2 = view(\pi_2(p))$ we have $p_1 \in R(p_2)$ hence if $p_1 \in Acc_1$, $p_2 \in R(Acc_1)$ hence $p_1 \in Acc_2$ hence p is winning for the player P.

□

The above construction suggest the following definition:

2.3.2.3 Definition (Synchr. game simulations). In the simulation game $\mathcal{G}_1 \,{}_E\!\rightleftharpoons^P \mathcal{G}_2$ a game simulation φ is a *synchronous game simulation* when every $(P, P) \to (E, P)$ move (resp. every $(E, E) \to (E, P)$ move) made by the player E is necessarily followed by a $(E, P) \to (E, E)$ move (resp. a $(E, P) \to (P, P)$ move made by the player P.

We write $\mathcal{G}_1, v_1 \,{}_P\!\rightleftharpoons^E_s \mathcal{G}_2, v_2$ the existence of a synchronous game simulation from position (v_1, v_2).

In other words, a synchronous game simulation from the game $\mathcal{G}_1$ to the game $\mathcal{G}_2$, is a simulation defined by the player P in the game $\mathcal{G}_1 \,{}_E\!\rightleftharpoons^P \mathcal{G}_2$ where the player P always answer to a player E move in a one of the component game by a move in the other component game.

Notions of P-morphism and E-morphism capture synchronous game simulation in the following sense.

2.3.2.4 Theorem. *There is a synchronous game simulation*

$$\mathcal{G}_1, v_1 \,{}_P\!\rightleftharpoons^E_s \mathcal{G}_2, v_2$$

if and only if there is a game G and a position $v \in G$ such that there is a P-morphism φ_P from G_1, v_1 to $\mathcal{G}, v$ and a E-morphism φ_E from G_2, v_2 to $\mathcal{G}, v$.

Proof. Take for the game G the subgame of $\mathcal{G}_1 \,{}_E\!\rightleftharpoons^P \mathcal{G}_2$ induces by the synchronous simulation relation relating v_1 to v_2. Then morphisms are defined by projections.

□

A simple application of PE-morphism is to solve regular games by means of a reduction, described below, to parity games.

Let $\mathcal{G} = \langle V_P, V_E, T, Acc \rangle$ be a finite regular game. Since Acc is ω-regular, there exists a finite deterministic parity automaton $\mathcal{A} = \langle Q, V^{\mathcal{G}}, q_0, \delta, \Omega \rangle$ such that $L(\mathcal{A}) = Acc$.

Let define parity game $\mathcal{A} \circ \mathcal{G} = \langle V'_P, V'_E, T', \Omega' \rangle$ by taking $V'_P = Q \times V_P$, $V'_E = Q \times V_E$, $T' = \{((q, u), (p, v)) \in (V'_P \times V'_E) \cup (V'_E \times V'_P) : (u, v) \in T, p = \delta(q, u)\}$, and $\Omega'(q, v) = \Omega(q)$.

2.3.2.5 Theorem (Regular to parity cond.). *A position v is winning for the player P in the game $\mathcal{G}$ if and only if the position (q_0, v) is winning for the player P in the game $\mathcal{A} \circ \mathcal{G}$.*

Proof. Let $R \subseteq V^{\mathcal{A} \circ \mathcal{G}} \times V^{\mathcal{G}}$ defined as the set of pair $\{((q, v), v) : q \in Q, v \in V^{\mathcal{G}}\}$. One can easily check that this is a PE-morphism hence Lemma 2.3.2.2 applies.

□

2.3.3 Generalized game simulations

In this section, we want to generalize the notion of simulation so that it may relate more positions in games. Though definable in the most general case of arbitrary games, this notion is more easily definable on parity games where the possible restriction to positional strategies considerably simplifies definitions and proofs.

2.3.3.1 Definition (Delayed game). Given a parity game $\mathcal{G} = \langle V_P, V_E, T_P, T_E, Acc\rangle$ we define the *E-delayed game* $E(\mathcal{G}) = \langle V'_P, V'_E, T'_P, T'_E, Acc'\rangle$ as follows:

1. $V'_P = \mathcal{P}(V_E.V_P \cup V_P.V_E)/V'_E$,
2. $V'_E = \mathcal{P}(V_P.V_E)$,
3. T'_P is the set of moves of the form $(U, V) \in V'_P \times V'_E$ such that there is a mapping $c : U \cap V_E.V_P \to V$ such that, for every $u_1.v_1 \in U \cap V_E.V_P$, $(v_1, c(u_1.v_1)) \in T_P$ and $V = \{v_1.c(u_1.v_1) \in V_P.V_E : u_1.v_1 \in U \cap V_E.V_P\} \cup (U \cap V_P.V_E)$,
4. T'_E is the of moves of the form $(U, V) \in V'_E \times V'_P$ such that $U = U_1 \cup U_2$, $V = V_1 \cup U_2$ with $V_1 \subseteq \{v_1.u_2 \in V_E.V_P : u_1.v_1 \in U_1, (v_1, u_2) \in T_E\}$,
5. and Acc' is the the set of all infinite paths of the form $p' = U_1.\cdots.U_n.\ldots \in (V^{E(\mathcal{G})})^\omega$ such that, given the set $tr(p')$ of traces of p' in $\mathcal{G}$ defined to be the set of (finite or infinite) words of the form $view(u_1.\cdots.u_n.\cdots)$ such that there is some $u_0 \in V^{\mathcal{G}}$ such that for every i such that $0 \leq i < |p'|$, $u_i.u_{i+1} \in U_{i+1}$, for every $p \in tr(p')$, either p is finite and ends in a player E position, or p is infinite with $p \in Acc$.

The *P-delayed game* $P(\mathcal{G})$ is defined by duality by taking $P(\mathcal{G}) = \overline{E(\overline{\mathcal{G}})}$.

2.3.3.2 Lemma. *A position v is winning for the player P in a parity game G if and only if for any/all $u \in V^{\mathcal{G}}$ the position $\{u.v\}$ is winning for the player P in the game $E(G)$ (equivalently the game $D(G)$).*

Proof. By duality, we can only prove the result for E-delayed game.

First, one can check that relation R defined to be the set of pairs $\{(\{v'.v\}, v) \in V^{E(\mathcal{G})} \times V^{\mathcal{G}} : v', v \in V^{\mathcal{G}}\}$ is a P-morphism from the game $E(\mathcal{G})$ to the game $\mathcal{G}$. It follows that if $\{v\}$ is winning for the player P in the game $E(\mathcal{G})$ then it is also winning for the player P from position v.

In fact, from a position $\{v'.v\} \in V_P^{E(\mathcal{G})}$ with $v \in V_P^{\mathcal{G}}$, the player P can only move in the game $E(\mathcal{G})$ to position of the form $\{v.u\} \in V_E^{E(\mathcal{G})}$ with $(v, u) \in T_P$. Therefore, from position $v \in V_P^{\mathcal{G}}$, this move can be simulated in the game $\mathcal{G}$ by moving to position u. And, similarly, from position $v \in V_E^{\mathcal{G}}$ player E can only move in the game $\mathcal{G}$ to position of the form $u \in V_E^{\mathcal{G}}$ with $(v, u) \in T_E^{\mathcal{G}}$. Therefore, from position $\{v'.v\} \in V_E^{E(\mathcal{G})}$, this move can be simulated in the game $E(\mathcal{G})$ by moving to position $\{v.u\}$.

Conversely, assume there is a winning strategy σ for the player P from vertex v in the game $\mathcal{G}$. Since $\mathcal{G}$ is memoryless, we may assume that σ is positional. We define then positional strategy σ' in the game $E(\mathcal{G})$ for the player P as follows: for every $U \in V'_P$, given $U_1 = U \cap V_E.V_P$, we define $\sigma'(U)$ to be the set $V = \{v_1.\sigma(v_1) \in V_P.V_E : u_1.v_1 \in U \cap V_E.V_P\} \cup (U \cap V_P.V_E)$.

We claim that σ' is winning from position $\{u.v\}$ for any $u \in V^{\mathcal{G}}$. In fact, let τ' be a counter strategy for the player E.

We first show, by induction on the length n of $p' \in \sigma' * \tau'(\{u.v\})$ that

(I) for every finite play $p' \in \sigma' * \tau'(\{u.v\}$, there is a (non deterministic) counter strategy $\tau_{p'}$ in the game $\mathcal{G}$ such that, for every $p \in tr(p')$:
– $p \in \sigma * \tau_{p'}(v)$,
– if p' is maximal in $\sigma' * \tau'(\{u.v\})$ then p ends in a player E position,
– for every prefix p'_1 of p', for every $p_1 \in tr(p'_1)$, $\tau_{p'_1}(p_1) \subseteq \tau_{p'}(p_1)$.

This is true for $n = 1$ since then $p' = U_1 = \{u.v\}$ and $p = v$ by taking $\tau_{p'}$ equal to $\emptyset$ everywhere. If p' is maximal, this means that $U_1 \in V_E^{E(\mathcal{G})}$ hence $v \in V_E^{\mathcal{G}}$.

Let then $p' = U_1.\cdots.U_n \in \sigma' * \tau'(\{u.v\})$ and $\tau_{p'}$ satisfying the induction hypothesis.

If there is a $p \in tr(p')$ that ends in a player P position, it means that, for every p that ends in a player P position u_n, we know, by induction hypothesis, that $p \in \sigma * \tau_{p'}(v)$, hence $\sigma(u_n)$ is defined since it is winning. It follows, by definition of $E(\mathcal{G})$, that $\sigma'(p') = U_{n+1} = \sigma'(U_n)$. The induction hypothesis is satisfied with $p'_1 = p'.U_{n+1}$ with $\tau_{p'_1} = \tau_{p'}$. In fact, since $U_{n+1} \in V_P^{E(\mathcal{G})}$ every trace $p \in tr(p'_1)$ ends in a player E position.

Otherwise, for every $p \in tr(p')$, play p ends in a player E position, hence $U_n \in V_E^{E(\mathcal{G})}$. By induction hypothesis, let $\tau_{p'}$ be the counter strategy in the game $\mathcal{G}$ such that for every $p \in tr(p')$, $p \in \sigma * \tau_{p'}(v)$. If $\tau'(p')$ is undefined, we are done. Otherwise, let $U_{n+1} = \tau'(p')$ and let $p'_1 = p'.U_{n+1}$. We define then $\tau_{p'_1}$ as follows: for every $p \in (V^{\mathcal{G}})^+$ we take $\tau_{p'_1}(p) = \tau_{p'}(p)$ except when $p \in tr(p')$ where we take

$$\tau_{p'_1}(p) = \tau_{p'}(p) \cup \{u' \in V_P^{\mathcal{G}} : \exists v' \in V_E^{\mathcal{G}}, p = p_1.v', v'.u' \in \tau'(p') \cap V_E.V_P\}$$

One can check that $\tau_{p'_1}$ is well defined and satisfies the induction hypothesis. In particular, for every $p \in tr(p'.U_{n+1})$, either $p \in \sigma * \tau(v) \subseteq \sigma_1 * \tau(v)$, or $p = p_1.v'.u'$ with $p_1.v' \in tr(p')$ and $v'.u' \in \tau'(p') \cap V_E.V_P$ with $(v', u') \in T_E^{\mathcal{G}}$ and thus, $p \in \sigma * \tau_{p'_1}(v)$.

We define then the counter strategy τ in the game $\mathcal{G}$ as the union of all $\tau_{p'}$ for finite $p' \in \sigma' * \tau'(\{u.v\})$. It occurs that, for every maximal play $p \in \sigma' * \tau'(p')$, for every $p \in tr(p')$:

1. if p is finite then, by definition of τ, $p \in \sigma * \tau_{p'_1}(v)$ for some finite prefix p'_1 of p' and, thus, by construction, ends in a player E position,
2. otherwise, p is infinite and, since every finite prefix of p belongs to $\sigma * \tau(v)$ we also have $p \in \sigma * \tau(v)$, hence $p \in Acc$ since σ is winning for the player P.

It other words, strategy σ' is winning from position $\{u.v\}$ in the game $E(\mathcal{G})$.

□

2.3.3.3 Corollary. *In a E-delayed game $E(\mathcal{G})$ (resp. in a P-delayed game $P(\mathcal{G})$) the player P (resp. the player E) has a winning strategy from a position if and only if it has a positional strategy.*

Proof. Direct consequence of the construction above. This can also be explained by the fact that in an E-delayed game (resp. P-delayed game) winning conditions are conjunction (resp. disjunction) of parity conditions henceforth Street conditions (resp. a Rabin conditions).

□

2.3.3.4 Definition (Generalized simulation product). Given two games $\mathcal{G}_1$ and $\mathcal{G}_2$ the *generalized simulation product* $\mathcal{G}_1 \,{}_E\!\vdash^P \mathcal{G}_2$ is defined to be the game $E(\mathcal{G}_1) \,{}_E\!\rightleftharpoons^P P(\mathcal{G}_2)$.

A *generalized simulation* from the game $\mathcal{G}_1$ in position v_1 to the game $\mathcal{G}_2$ in position v_2 is a winning strategy in game $\mathcal{G}_1 \ {}_E\vdash^P \mathcal{G}_2$ from position $(\{u_1.v_1\}, \{u_1.v_2\})$ for some positions $u_1 \in V^{\mathcal{G}_1}$ and $u_2 \in V^{\mathcal{G}_2}$.

We write $\mathcal{G}_1, v_1 \ {}_E\vdash^P \mathcal{G}_2, v_2$ when there is such a game simulation.

2.3.3.5 Lemma (Soundness). *If $\mathcal{G}_1, v_1 \ {}_E\vdash^P \mathcal{G}_2, v_2$ and position v_1 is winning for the player P in the game $\mathcal{G}_1$ then position v_2 is winning for player P in the game $\mathcal{G}_2$.*

Proof. Immediate consequence of definitions, Lemma 2.3.3.2 and Lemma 2.3.1.2 □

Remark. In the remainder of the text, when building generalized simulation, we may omit brackets on singletons, and, if there is no ambiguity, we may just keep target positions p_2 in pairs of the form $p_1.p_2$.

2.4 References and notes

A lot more can be said about two player discrete games. This chapter does not intend to cover the topic. For a more detailed presentation, one can see, in particular, Zielonka's paper[184], or Grädel, Thomas and Wilke tutorial volume [78].

We shall mention in particular that many algorithms have been proposed for solving parity games [59, 131, 6, 162, 118, 108, 179, 142, 16, 30, 29, 76]. Zielonka's presentation [184] not only provides one of the most elegant algorithm but also offers a very good survey. Jurdzinski's algorithm [108] is the best known algorithm. Vöge and Jurdzinski's proposal [179] seems better, but its theoretical complexity is unknown. It is a variant of Jurdzinski's algorithm [108] that is presented in section 2.2.

Simulation relations are considered not only in game, but also in the context of alternating or non deterministic word or tree automata [5].

There is no doubt that extending game simulations could be made, following Gentzen propositional sequent calculus (this is the purpose of the notion of generalized simulation relation) and Kozen's axiomatization of the mu-calculus proved complete by Walukiewicz [180]. These issues, which are open research directions, are discussed in Chapter 8.

Chapter 3

Alternating automata and fixpoint calculus

In this chapter, we define a general notion of alternating tree automata adapted from Muller and Schupp's definition [136]. On arbitrary graphs, automata runs are defined in terms of strategies in model checking games. Several operations and automata transformations or compositions are presented and semantically characterized. Last, the expressive power of alternating automata is related with the expressive power of the mu-calculus. Various invariance properties of the mu-calculus are obtained as corollaries.

Though this equivalence between fixed point calculus and alternating automata is known for some time in various settings (see e.g. [139, 167, 58, 182, 27, 181]), the main novelty here is, in the general setting of unranked and unordered tree, to give an inductive proof of the equivalence between alternating automata and fixed point calculus. This presentation extends and adapts the following publications [92, 93, 103, 94, 95]

3.1 Finite state alternating automata

In this section, we define a notion of finite state alternating automata on labeled graphs. It generalizes Muller and Schupp's notion of alternating automata [136, 137] on the binary tree.

We define alternating automata and the class of graphs they recognized. The notion of run of an alternating automaton on a graph is defined by means of strategies in some parity game.

We explicit some simple properties of alternating automata. We give a simple equivalence criteria and we prove closure under complementation, positive projection or (some notion of) composition.

3.1.1 Alternating automata definition

3.1.1.1 Definition. A finite *(parity) alternating automaton* is a tuple

$$\mathcal{A} = \langle Q, \Sigma, q_0, \delta, Acc \rangle$$

with Q a finite set of states, $\Sigma = \mathcal{P}(Pred)$ the finite alphabet, $q_0 \in Q$ the initial state, $Acc \subseteq Q^\infty$ a (regular) acceptance condition, and δ the transition specification function that maps every state $q \in Q$ to a fixed point formula $\delta(q)$ over the vocabulary $Pred \cup Q$ such that states, now seen as unary predicate symbols, only occur positively in the formula $\delta(q)$.

We say that the automaton $\mathcal{A}$ is a *modal automaton* (resp. a *counting automaton*) when only modal formulas (resp. counting formulas) are used for transition specifications.

The Automaton $\mathcal{A}$ is also called a *flat automaton* when, for every state $q \in Q$, transition specification $\delta(q)$ is a formula of the form q_1, $q_1 \wedge q_2$, $q_1 \vee q_2$, $\Diamond_k q_1$, $\Box_k q_1$, p or $\neg p$ for some state q_1 and $q_2 \in Q$ or any constant predicate $p \in Pred$.

Let $\mathcal{A}$ be an alternating automaton and let M be a graph. For every $v \in V^M$ and $m : Q \to \mathcal{P}(V^M)$, let $\langle v, m\rangle_M$ denote the $\mathcal{P}(Pred \cup Q)$-labeled graph obtained from graph M just by changing root r^M to be vertex v and interpreting any state q (seen as a unary predicate) to be $m(q)$.

3.1.1.2 Definition. The *model-checking game* $\mathcal{G}(\mathcal{A}, M)$ of the automaton $\mathcal{A}$ reading graph M is defined as follows:

- (Process positions) $V_P^{\mathcal{G}(\mathcal{A},M)} = Q \times V^M$,
- (Environment positions) $V_E^{\mathcal{G}(\mathcal{A},M)} = Q \to \mathcal{P}(V^M)$,
- (Process moves) $T_P^{\mathcal{G}(\mathcal{A},M)}$ the set of all pairs $((q, v), m) \in V_P^{\mathcal{G}(\mathcal{A},M)} \times V_E^{\mathcal{G}(\mathcal{A},M)}$ such that $\langle v, m\rangle_M \models \delta(q)$
- (Environment moves) $T_E^{\mathcal{G}(\mathcal{A},M)}$ the set of all pairs $(m, (q, v)) \in V_E^{\mathcal{G}(\mathcal{A},M)} \times V_P^{\mathcal{G}(\mathcal{A},M)}$ such that $v \in m(q)$,
- (Initial position) $r^{\mathcal{G}(\mathcal{A},M)} = (q_0, r^M)$,
- (Acceptance condition) $Acc^{\mathcal{G}(\mathcal{A},M)}$ the set of all infinite path w in $\mathcal{G}(\mathcal{A}, M)$ such that $\pi_Q(\pi_{V_P}(w)) \in Acc$.

Observe that, in the game $\mathcal{G}(\mathcal{A}, M)$, it is only the priority of the positions of the player P (the priority of states) that does matter on infinite plays.

3.1.1.3 Definition. We say that graph M is *accepted* or *recognized* by an automaton $\mathcal{A}$ when the player P has a winning strategy in game $\mathcal{G}(\mathcal{A}, M)$. The class of graphs M accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

We say that the automaton $\mathcal{A}$ is equivalent to an automaton $\mathcal{A}'$ (resp. equivalent to a fixed point formula α) when $L(\mathcal{A}) = L(\mathcal{A}')$ (resp. when $L(\mathcal{A}) = \{M : M \models \alpha\}$).

Remark (Runs and accepting runs). With this definition of automata semantics, the (more common) notions of runs and accepting runs of an automaton $\mathcal{A}$ on an input structure M can be recovered as follows:

- a *run of the automaton* $\mathcal{A}$ *on structure* M is a consistent strategy s for the player P on the game $\mathcal{G}(\mathcal{A}, M)$,
- an *accepting run* is a winning strategy.

3.1.2 Automata traces and unravelings

From every position $(q, v) \in (Q \times V^M)$, the player P can always choose to move to a mapping $m : Q \to \mathcal{P}(V^M)$ that is minimal w.r.t. inclusion. In fact, if the player P does

so, he even increase its possibility to have a winning strategy since this just decreases the possible answers from player E.

Considering the states that may occur during a play, this shows in particular that, from a position (q, v), the set of states that do really matter are those which occurs in the transition specification $\delta(q)$.

This observation leads to the definition of automata trace and automata unraveling.

3.1.2.1 Definition. A *trace* for an alternating automaton $\mathcal{A}$ is any finite or infinite sequence of state $t = q_1.\cdots.q_i.\cdots$ such that, for every index i, state q_{i+1} occurs in transition specification $\delta(q_i)$. An infinite trace t is called *accepting trace* (resp. *refusing trace*) when $t \in Acc$ (resp. when $t \notin Acc$).

3.1.2.2 Definition. Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, Acc\rangle$ and $\mathcal{A}' = \langle Q', \Sigma, q'_0, \delta', Acc'\rangle$ be two automata. We say that the automaton $\mathcal{A}$ is a *partial unraveling* of the automaton $\mathcal{A}'$ when there is a mapping $f : Q \to Q'$ such that:

1. $f(q_0) = q'_0$,
2. for every $q \in Q$, $\delta'(f(q))$ is equivalent to the formula obtained from the formula $\delta(q)$ by replacing every state q_1 occurring in $\delta(q)$ by its image $f(q_1)$,
3. for every infinite trace $t = q_0.q_1.q_2.\cdots$ in the automaton $\mathcal{A}$, the sequence $f(t)$ defined to be $f(t) = f(q_0).f(q_1).f(q_2).\cdots$ is a trace in the automaton $\mathcal{A}'$ and it is an accepting trace if and only if the trace t in $\mathcal{A}$ is accepting.

3.1.2.3 Lemma. *If an automaton $\mathcal{A}$ is a partial unraveling of an automaton $\mathcal{A}'$, then both automata are equivalent.*

Proof. Let $f : Q \to Q'$ be a mapping witnessing the partial unraveling of the automaton $\mathcal{A}$ into the automaton $\mathcal{A}'$ and let M be a model. Mapping f is extended to marking as follows. For every mapping $m : Q \to \mathcal{P}(V^M)$, let $f(m) : Q' \to \mathcal{P}(V^M)$ be the mapping defined, for every $q' \in Q'$, by $f(m)(q') = \bigcup_{q \in f^{-1}(q')} m(q)$. By applying condition 2 in the definition of an unraveling, for every $v \in V^M$ and $q \in Q$, one has $\langle v, m\rangle_M \models \delta(q)$ if and only if $\langle v, f(m)\rangle_M \models \delta'(f(q))$.

It is an easy exercise to check that it induces a (functional) PE-morphism from the game $\mathcal{G}(\mathcal{A}, M)$ to the game $\mathcal{G}(\mathcal{A}', M)$ that relates positions (q_0, r^M) and (q'_0, r^M).

□

3.1.2.4 Lemma (Regular vs parity automata). *For every alternating automaton*

$$\mathcal{A} = \langle Q, \Sigma, q_0, \delta, Acc\rangle$$

there exists an equivalent alternating parity automaton $\mathcal{A}_p = \langle Q_p, \Sigma, q_{p,0}, \delta_p, Acc_p\rangle$ equivalent to the automaton $\mathcal{A}$.

Proof. Since Acc is ω-regular, there is a deterministic ω-word automaton $\mathcal{A}' = \langle Q', Q, q'_0, \Omega'\rangle$ with parity condition that recognizes Acc. Let then define $\mathcal{A}_p = \langle Q_p, \Sigma, q_{p,0}, \delta_p, \Omega_p\rangle$ to be the (parity) alternating automaton defined by $Q_p = Q \times Q'$, $q_{p,0} = (q_0, q'_0)$, and, for every $(q, q') \in Q_p$, $\delta_p(q, q') = \delta(q)[(q_1, \delta'(q', q))/q_1 : q_1 \in Q]$ and $\Omega_p(q, q') = \Omega'(q')$.

By construction, projection f from Q_p to Q is a partial unraveling of the automaton $\mathcal{A}_p$ into the automaton $\mathcal{A}$ hence, applying Lemma 3.1.2.3, the result.

□

In the sequel, we often assume that alternating automata are parity automata. Moreover, we may assume, for every parity automaton $\mathcal{A}$, possibly by decreasing $\Omega^{\mathcal{A}}$ by some even number, that $\min \Omega^{\mathcal{A}}(Q^{\mathcal{A}}) \in \{0, 1\}$.

3.1.3 Closure under boolean operations

The class of languages recognizable by means of (finite) alternating automata is closed under boolean operators as stated in the next Lemmas.

3.1.3.1 Lemma (Union and intersection). *The union and intersection of any two recognizable languages of graphs is recognizable.*

Proof. Let $\mathcal{A}_1 = \langle Q_1, \Sigma, q_{0,1}, \delta_1, \Omega_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma, q_{0,2}, \delta_2, \Omega_2 \rangle$ be two alternating automata. One can easily define the automata that recognizes $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ and $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ from the disjoint union of automata $\mathcal{A}_1$ and $\mathcal{A}_2$ just by adding a new initial state q_0 with $\delta(q_0) = q_{0,1} \vee q_{0,2}$ for union or $\delta(q_0) = q_{0,1} \wedge q_{0,2}$ for intersection.

□

A very interesting property of alternating automata is that closure under complementation is also quite easy to show.

3.1.3.2 Definition (Dual automata). Given an automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \Omega \rangle$ we define the dual automaton $\mathcal{A}^d = \langle Q, \Sigma, q_0, \delta^d, W^d \rangle$ by defining, for every state $q \in Q$, $\Omega^d(q)$ to be $\Omega(q) + 1$ and, given $Q = \{q_0, q_1, \cdots, q_n\}$, the transition specification $\delta^d(q)$ to be the formula the formula obtained from the formula $\neg\delta(q)$ by replacing every occurrence of every state of Q in the formula by its negation, i.e. to be the formula $\neg\delta(q)[\neg q_0/q_0, \neg q_1/q_1, \cdots, \neg q_n/q_n]$.

Observe that, by construction, states still occur positively in the transition specifications of the dual automaton.

3.1.3.3 Lemma (Complementation). *For every automaton $\mathcal{A}$ on Σ-labeled graphs, $\overline{L(\mathcal{A})} = L(\mathcal{A}^d)$ where $\overline{L(\mathcal{A})}$ is the class of Σ-labeled graphs that are not accepted by the automaton $\mathcal{A}$.*

Proof. Let $\mathcal{A}$ be an alternating automaton, let M be a model.

Assume first $M \in L(\mathcal{A})$ we want to prove that $M \notin L(\mathcal{A}^d)$.

The key point of this part of the proof is the following fact: for every $(q, v) \in Q \times V^M$, $m : Q \to \mathcal{P}(V^M)$ such that $\langle v, m \rangle_M \models \delta(q)$ and $m^d : Q \to \mathcal{P}(V^M)$ such that $\langle v, m^d \rangle_M \models \delta^d(q)$ there exists $(q', v') \in Q \times V^M$ such that $v' \in m(q') \cap m^d(q')$.

In fact, if this is not true, then, for every state $q' \in Q$, one have $m(q') \subseteq \overline{m}^d(q')$. Since $\langle v, m \rangle_M \models \delta(q)$ and $\delta(q)$ is positive (and thus monotonic) in the states of Q, this implies that $\langle v, \overline{m}^d \rangle_M \models \delta(q)$ hence, by definition of $\delta^d(q)$, $\langle v, m^d \rangle_M \models \neg\delta^d(q)$ which contradicts the hypothesis on m^d.

We then show that: in simulation the game $\mathcal{G}(\mathcal{A}, M) \,{}_E{\rightleftharpoons}^P \overline{\mathcal{G}(\mathcal{A}^d, M)}$, for every pair $(q, v) \in Q \times V^M$, for every marking $m^d : Q \to \mathcal{P}(V^M)$ such that $\langle v, m^d \rangle_M \models \delta^d(q)$, the (P, P)-position $((q, v), m^d)$ is winning for player P.

The strategy for the player P is defined as follows: from a (P, P) position $((q, v), m^d)$ with $m^d : Q \to \mathcal{P}(V^M)$ and (invariant) property $\langle v, m^d \rangle_M \models \delta^d(q)$, the player E makes a move to any position (m, m^d) with $m : Q \to \mathcal{P}(V^M)$ such that $\langle v, m \rangle_M \models \delta(q)$. Then,

applying the previous fact, from the (E,P)-position (m, m^d) the player P can move to a (E,E)-position $(m,(q',v'))$ (with $v' \in m^d(q')$) such that, after the player E answer by playing to any (E,P)-position (m, m'_d) with $m'^d : Q \to \mathcal{P}(V^M)$ such that $\langle v', m'^d\rangle_M \models \delta^d(q')$ the player P can still answer by moving to the (P,P)-position $((q',v'), m'^d)$ (with $q' \in m(v')$). Property $\langle v', m'^d\rangle_M \models \delta^d(q')$ is satisfied so the construction can be repeated.

This strategy is winning because the player P can always play and, in every infinite plays, the sequences of states (hence sequences of priorities) encountered on both side of the play in the game $\mathcal{G}(\mathcal{A},M)$ $_E\rightleftharpoons^P$ $\overline{\mathcal{G}(\mathcal{A}^d,M)}$ are the same (actually priorities on the right side are increased by two which makes no difference). It follows that the left projection of the play is winning for the player P if and only if the right projection is winning for the player P.

Now, since the player P has a winning strategy from position (q_0, r^M) in $\mathcal{G}(\mathcal{A},M)$, this fact implies, by Lemma 2.3.1.2, that $M \notin L(\mathcal{A})$ since it implies that the player E has a winning strategy from position (q_0, r^M) in $\mathcal{G}(\mathcal{A}^d,M)$ (equivalently, for every m^d such that $\langle r^M, m^d\rangle_M \models \delta^d(q_0)$ player P has a winning strategy in $\overline{\mathcal{G}(\mathcal{A}^d,M)}$)

Conversely, let $M \notin L(\mathcal{A}^d)$. We want to prove that $M \in L(\mathcal{A})$. We could try to prove that there is a simulation relation from (q_0,v) in $\overline{\mathcal{G}(\mathcal{A}^d,M)}$ (from which the player P win) to (q_0,v) in $\mathcal{G}(\mathcal{A},M)$ (henceforth from which the player P will also wins). But it seems that there is no such a simulation and we need to shift to generalized simulation and apply Lemma 2.3.3.5.

Let $\mathcal{G}$ be the generalized simulation the game $\overline{\mathcal{G}(\mathcal{A}^d,M)}$ $_E\vdash^P \mathcal{G}(\mathcal{A},M)$. We claim that for every pair $(q,v) \in Q \times V^M$, there is a winning strategy for the player P from the (E,P)-position $((q,v),(q,v))$.

The winning strategy is defined as follows: from the (E,P)-position $((q,v),(q,v))$ the player P move to the $(\mathcal{P}(P),P)$-position $(P_{q,v},(q,v))$ where

$$P_{q,v} = \{m^d : Q \to \mathcal{P}(V^M) : \langle v, m^d\rangle_M \models \delta^d(q)\}$$

Then - if possible, otherwise the player E looses and we are done - the player E moves to any the $(\mathcal{P}(E),P)$-position $(s(P_{q,v}),(q,v))$ for some mapping $s : P_{q,v} \to Q \times V^M$ such that, for every $m^d \in P_{q,v}$, given $(q',v') \in s(m^d)$, $v' \in m^d(q')$. From this $(\mathcal{P}(E),P)$-position, the player P moves to the $(\mathcal{P}(E),E)$-position $(s(P_{q,v}),m)$ with marking m defined, for every state $q' \in Q$, by $m(q') = \{v' \in V^M : (q',v') \in s(P_{q,v})\}$.

Here, we have to check that this is a valid move. More precisely, we have to check that $\langle v, m\rangle_M \models \delta(q)$. If this is not the case, one has $\langle v, m\rangle_M \models \neg\delta(q)$ or, equivalently, $\langle v, \overline{m}\rangle_M \models \delta^d(q)$ hence, by definition of $P_{q,v}$, $\overline{m} \in P_{q,v}$. Let then $(q',v') = s(\overline{m})$. By definition of s, we have $v' \in \overline{m}(q')$. By definition of m, we also have $v' \in m(q')$ which is absurd.

Now, from the $(\mathcal{P}(E),E)$-position $(s(P_{q,v}),m)$ the player E move to any $(\mathcal{P}(E),P)$-position $(s(P_{q,v}),(q',v'))$ such that $v' \in m(q')$. By definition of marking m, $(q',v') \in s(P_{q,v})$ hence, from the $(\mathcal{P}(E),P)$-position $(s(P_{q,v}),(q',v'))$ the player P can answer by moving to the (E,P)-position $((q',v'),(q',v'))$ which is a legal move and the contruction can be repeated.

This is a winning strategy since, from every position $((q,v),(q,v))$, either the construction stops before a player E move, or the construction built an infinite play such that the sequence of states encountered on both side of the play are (almost) equal (as above).

□

3.1.4 Automata contractions and expansions

We prove here a contraction/expansion lemma which, roughly speaking, tell us that, under adequate technical conditions, any sub-automaton can be *contracted* into an equivalent transition specification, and, vice versa, any fixed-point formula that occurs in the transition specification of an automaton can be *expanded* into an equivalent alternating (sub-)automaton.

The first step is to define a way to convert states into predicates and, vice versa, predicates into states. This is achieved via the notion of F-equivalence and positive projection.

3.1.4.1 Definition (F-equivalence). Let $\mathcal{A} = \langle Q, \mathcal{P}(Pred), q_0, \delta, \Omega\rangle$ be an alternating automaton and let $F \subseteq Q$ be a subset $\{f_1, \cdots, f_n\}$ of the set of states such that, for every $f \in F$, $\delta(f) = \top$, i.e. the states of F are terminal. Let also $\alpha(f_1, \cdots, f_n)$ be a fixed point formula where states f_1, ..., f_n, seen as unary predicates, only occurs positively.

The Automaton $\mathcal{A}$ is called *F-equivalent* with the formula $\alpha(f_1, \cdots, f_n)$ if for every model M, for every V_1, ..., $V_n \subseteq V^M$, the following properties are equivalent:

1. $M \models \alpha(V_1, \cdots, V_n)$,
2. there is a winning strategy for the player P in the game $\mathcal{G}(\mathcal{A}, M)$ such that, given $\mathcal{M}$ the set of moves $m : Q \to \mathcal{P}(V^M)$ the player P may play following this strategy from position (q_0, r^M), one has, for each $i \in [1, n]$, $V_i = \bigcup\{m(f_i) : m \in \mathcal{M}\}$,

In other words, the automaton $\mathcal{A}$ is F-equivalent to the formula $\alpha(f_1, \ldots, f_n)$ when it is equivalent to the formula $\exists f_1 \cdots f_n.\alpha(f_1, \ldots, f_n)$, and, moreover, any accepting run does define an adequate valuation for the sets f_is.

The following lemma says that one can build an automaton F-equivalent to a formula $\alpha(f_1, \cdots, f_n)$ as soon as one has an automaton equivalent, in the usual sense, to the formula $\alpha(f_1, \cdots, f_n)$.

3.1.4.2 Lemma (Positive projection). *Let $\alpha(f_1, \cdots, f_n)$ be a formula where symbols f_1, ..., f_n only occur positively. Let $\mathcal{A}$ be an automaton over the alphabet $\mathcal{P}(Pred \cup F)$ equivalent to $\alpha(f_1, \cdots, f_n)$ such that predicates of F only occurs positively in transition specification. One can build an automaton $\mathcal{A}'$ over alphabet $\mathcal{P}(Pred)$ which is F-equivalent to the formula $\alpha(f_1, \cdots, f_n)$.*

Proof. Let $\mathcal{A} = \langle Q, \mathcal{P}(Pred \cup F), q_0, \delta, \Omega\rangle$ be an automaton (with $Q \cap F = \emptyset$ just to avoid confusion) equivalent to the formula $\alpha(f_1, \cdots, f_n)$.

We define the automaton $\mathcal{A}' = \langle Q', \mathcal{P}(Pred), q'_0, \delta', \Omega'\rangle$ over alphabet $\mathcal{P}(Pred)$ from the automaton $\mathcal{A}$ as follows. We put $Q' = Q \uplus F$, $q'_0 = q_0$, for every state $q \in Q$, $\delta'(q) = \delta(q)$ and $\Omega'(q) = \Omega(q)$ and, for every state $q \in F$, $\delta'(q) = \top$ and $\Omega'(q) = \max\{\Omega(q) : q \in Q\}$. In other words, the automaton $\mathcal{A}'$ is built from the automaton $\mathcal{A}$ just changing by the meaning of the symbols of F from unary predicates (in the automaton $\mathcal{A}$) to states (in the automaton $\mathcal{A}'$).

Then it is easy to check that the automaton $\mathcal{A}$ is F-equivalent to the formula $\alpha(f_1, \cdots, f_n)$.

□

We are now ready to define the contraction of an automaton.

3.1.4.3 Definition (F-contraction). Let $\mathcal{A} = \langle Q, \mathcal{P}(Pred), q_0, \delta, \Omega\rangle$ be an alternating automaton and let $F = \{f_1, \cdots, f_n\} \subseteq Q$ be a subset of the set of states with $q_0 \in F$.

For every state $f \in F$, let define the *F-local automaton* $\mathcal{A}[f, F]$, from the state f to the states of F, to be (equivalent to) the sub automaton of $\mathcal{A}$ induced by the states that are in between the state f and the states of F. More precisely, $\mathcal{A}[f, F] = \langle Q', \mathcal{P}(Pred), q_{0,f}, \delta', \Omega'\rangle$ where $Q' = Q \uplus \{q_{0,f}\}$ with $q_{0,f}$ a new state, for every $q \in Q - F$, $\delta'(q) = \delta(q)$, for every $q \in F$, $\delta'(q) = \top$, and $\delta'(q_{0,f}) = \delta(f)$ with Ω' equals to Ω on all states of Q and equals to $\Omega(f)$ on the initial state $q_{0,f}$.

Assuming that for every $f \in F$, there is a fixed point formula $\alpha_f(f_1, \cdots, f_n)$ F-equivalent to the F-local automaton $\mathcal{A}[f, F]$, the *F-contraction* of the automaton $\mathcal{A}$ is defined to be the automaton

$$C(\mathcal{A}, F) = \langle Q^c, \mathcal{P}(Pred), q_0^c, \delta^c, \Omega^c\rangle$$

defined by $Q^c = F$, $q_0^c = q_0$, and, for each $f \in Q^c$, $\delta(f) = \alpha_f(f_1, \cdots, f_n)$ and $\Omega^c(f) = \Omega(f)$.

Remark. Strictly speaking, there might be several F-contraction of the automaton $\mathcal{A}$ since there might be many formulas F-equivalent to local automata $\mathcal{A}[q, F]$ for $q \in F$. However, all these F-contractions of $\mathcal{A}$ are obviously equivalent (even PE-isomorphic).

In the contraction construction, all states that do not belong to F are removed. This means in particular that the priorities of those states has been lost. For this reason, in general, the infinitary criterion is not preserved by this contraction process. The notion of F-normed automata is a way to cope with this fact.

3.1.4.4 Definition (F-normal automata). An automaton $\mathcal{A}$ is *F-normal* when for every trace $q_1.q_2.\cdots.q_n$ such that, for every $i \in \{1, n-1\}$, $q_i \notin F$ and $q_n \in F$, one has $\Omega(q_n) = \min(\Omega(q_1), \cdots, \Omega(q_n))$.

With this definition:

3.1.4.5 Lemma. *If an automaton $\mathcal{A}$ is F-normal then the automaton $\mathcal{A}$ and any F-contraction $C(\mathcal{A}, F)$ of the automaton $\mathcal{A}$ are equivalent.*

Proof. We observe first that if the automaton $\mathcal{A}$ is F-normal, then the dual automaton $\mathcal{A}^d$ is also F-normal. Moreover, for every $q \in F$, for every formula α_q with free variable in F, F-equivalent to $\mathcal{A}[q, F]$, the formula α_q^d defined to be $\neg\alpha[\neg q/q : q \in F]$ is F-equivalent to $\mathcal{A}^d[q, F]$. It follows that the dual automaton of the F-contraction of $(C(\mathcal{A}, F))^d$ of the automaton $\mathcal{A}$ is equivalent to the F-contraction $C(\mathcal{A}^d, F)$ of the dual automaton $\mathcal{A}^d$.

In other words, to prove Lemma 3.1.4.5, it suffices, by duality, to show that for very model M, every F-normal automaton $\mathcal{A}$, if $M \in L(\mathcal{A})$ then $M \in L(C(\mathcal{A}, F))$.

In fact, this will also imply that if $M \notin L(\mathcal{A})$ then, by duality Lemma 3.1.3.3, $M \in L(\mathcal{A}^d)$, hence $M \in L(C(\mathcal{A}^d, F)) = L((C(\mathcal{A}, F))^d)$, and thus, again by duality Lemma, $M \notin L(C(\mathcal{A}, F))$.

Given a model M, and an F-normal automaton $\mathcal{A}$, we prove that if $M \in L(\mathcal{A})$ then $M \in L(C(\mathcal{A}, F))$ by showing that the player P has a winning strategy φ in the simulation game $\mathcal{G}(C(\mathcal{A}, F), M)$ ${}_E\!\rightleftharpoons^P \mathcal{G}(\mathcal{A}, M)$ from position $((q_0, r^M), (q_0, r^M))$.

More precisely, we show that the player P has a winning strategy from every (P, P)-position of the form $((f, v), (f, v))$ with $f \in F$.

The player P moves to an (E,P)-position of the form $(m,(f,v))$ with $\langle v,m\rangle_M \models \alpha_f$ with transition specification $\delta^{C(\mathcal{A},F)}(f) = \alpha_f$ F-equivalent to the local automaton $\mathcal{A}[f,F]$. Since all state predicates occurring in α_f belongs to F we may assume that for every state $q \notin F$, $m(q) = \emptyset$.

By F-equivalence Lemma 3.1.4.2, since $\langle v,m\rangle_M \models \alpha_f$, there is a strategy σ_2 for the player P from position (f,v) in the game $\mathcal{G}(\mathcal{A}[f,F],\langle v,m\rangle_M)$ such that, for every counter strategy τ_2, every partial play $p \in \sigma_2 * \tau_2(f,v)$, either p is winning for the player P or eventually reach a position of the form (f',v') with $f' \in F$ and $v' \in m(f')$.

From position $(m,(f,v))$ in the game $\mathcal{G}(C(\mathcal{A},F),M)\ {}_E\!\rightleftharpoons^P \mathcal{G}(\mathcal{A},M)$, we define strategy φ to follow strategy σ_2 in (the sub game in $\mathcal{G}(\mathcal{A},M)$ from position (f,v) isomorphic to) local game $G(\mathcal{A}[f,F],\langle m,v\rangle_M)$ stopping (if ever) only at positions of the form (f',v') with $f' \in F$. Since no move is made by the player P on the left side of the simulation game, the player E always answer on the right side so this (partial) strategy for the player P on game $\mathcal{G}(C(\mathcal{A},F),M)\ {}_E\!\rightleftharpoons^P \mathcal{G}(\mathcal{A},M)$ is well-defined.

In the case this strategy stops, it stops to a position $(m,(f',v'))$ with $v' \in m(f')$ so the player P can answer moving to position $((f',v'),(f',v')$. Since $f' \in F$ the construction can be repeated.

The strategy φ defined in such a way is winning since every maximal play compatible with φ is either winning on its right side, or it is infinite (on both side) and the sequence of states occurring on the left side equals the sub-sequence of states of F that occurs on the left side. Since the automaton $\mathcal{A}$ is F-normal, this guarantees that the two side of the play are equivalently winning or loosing for the player P.

□

A construction dual to the contraction, called expansion, can be defined as follows.

3.1.4.6 Definition (S-expansion). Let $\mathcal{A} = \langle Q,\mathcal{P}(Pred),q_0,\delta,\Omega\rangle$ be an alternating automaton and let $S = \{s_1,\cdots,s_n\} \subseteq Q$ be a subset of the set of states. Assume that, for every state $s \in S$, there is a (local) automaton

$$\mathcal{A}_s = \langle Q_s,\mathcal{P}(Pred),q_{s,0},\delta_s,\Omega_s\rangle$$

with $Q \subseteq Q_s$, Q-normal and Q-equivalent to $\delta(s)$, and such that, for every $q \in Q$, $\delta_s(q) = \top$ and $\Omega_s(q) = \Omega(q)$. Assume also that these automata are compatible one with the other in the sense that, for every s_1 and $s_2 \in S$, every $q \in Q_{s_1} \cap Q_{s_2}$, the formulas $\delta_{s_1}(q)$ and $\delta_{s_2}(q)$ are equivalent, and $\Omega_{s_1}(q) = \Omega_{s_2}(q)$.

The *S-expansion* of $\mathcal{A}$ by $\{\mathcal{A}_s\}_{s\in S}$ is defined to be the automaton

$$E(\mathcal{A},\{\mathcal{A}_s\}_{s\in S}) = \langle Q^e,\mathcal{P}(Pred),q_0^e,\delta^e,\Omega^e\rangle$$

defined by $Q^e = \bigcup_{s\in S} Q_s$, $q_0^e = q_0$, for every $q \in Q - S$, $\delta^e(q) = \delta(q)$ and $\Omega^e(q) = \Omega(q)$, for every $q \in S$, $\delta^e(q) = \delta_q(q_{q,0})$ and $\Omega^e(q) = \Omega(q)$, and, for every $q \in Q^e - Q$, given any s such that $q \in Q_s$, $\delta^e(q) = \delta_s(q)$ and $\Omega^e(q) = \Omega_s(q)$.

3.1.4.7 Lemma. *The S-expansion $E(\mathcal{A},\{\mathcal{A}_s\}_{s\in S})$ of an automaton $\mathcal{A}$ is equivalent to the automaton $\mathcal{A}$.*

Proof. Observe that, for every state state $s \in S$, the automaton $\mathcal{A}[s,S]$ is isomorphic to the automaton $\mathcal{A}_s$, and the automaton $E(\mathcal{A},\{\mathcal{A}_s\}_{s\in S})$ is S-normal. It follows that

the automaton $\mathcal{A}$ is an S-contraction of the automaton $E(\mathcal{A}, \{\mathcal{A}_s\}_{s\in S})$ so Lemma 3.1.4.5 applies.

□

3.2 Alternating automata and fixpoint formula

On a conceptual point of view, alternating automata can be seen as systems of fixed-point equations [16]. In this section, we illustrate this fact by proving that both the classes of graphs definable by means of counting (resp. modal) flat alternating automata and the classes of graphs definable by means of counting (resp. modal) fixed-point formulas are the same.

3.2.1 Basic alternating automata

Building an alternating automaton $\mathcal{A}_\alpha$ equivalent to a fixed point formula α can be done in a trivial way : the one state automaton $\mathcal{A}^t_\alpha$ defined by $\mathcal{A}^t_\alpha = \langle\{\{q_0\}, \Sigma, q_0, \delta, \Omega\rangle$ with $\delta(q) = \alpha$ and $\Omega(q) = 0$ is obviously equivalent to the formula α. But this automaton is not flat.

In this section, we go further by defining a notion of basic alternating automata. These automata are called basic because there is one such automaton per connective of the fixed point calculus. Moreover, they can be composed in order to build flat automata equivalent to arbitrary fixed point formulas.

3.2.1.1 Definition (Basic alternating automata). Given any fixed formula α, we define the *basic automaton* $\mathcal{A}^b_\alpha$ associated to the formula α to be the automaton

$$\mathcal{A}^b_\alpha = \langle Q^b, \mathcal{P}(Prop \cup Var_\alpha), q^b_\alpha, \delta^b, \Omega^b\rangle$$

defined as follows:

1. if $\alpha \equiv p$ (resp. $\alpha \equiv \neg p$) then $\mathcal{A}^b_\alpha$ has a single state q^b_α with $\delta(q^b_\alpha) = p$ (resp. $\delta(q^b_\alpha) = \neg p$),
2. if $\alpha \equiv \Diamond_k\alpha_1$ (resp. $\alpha \equiv \Box_k\alpha_1$) then $\mathcal{A}^b_\alpha$ has two states q^b_α and $q^b_{\alpha_1}$ with $\delta^b(q^b_\alpha) = \Diamond_k q^b_{\alpha_1}$ (resp. $\delta^b(q^b_\alpha) = \Box_k q^b_{\alpha_1}$) and $\delta^b(q^b_{\alpha_1}) = \alpha_1$,
3. if $\alpha \equiv \alpha_1 \vee \alpha_2$ (resp. $\alpha \equiv \alpha_1 \wedge \alpha_2$) then $\mathcal{A}^b_\alpha$ has tree states q^ϵ_α, $q^b_{\alpha_1}$ and $q^b_{\alpha_2}$ with $\delta^b(q^b_\alpha) = q^b_{\alpha_1} \vee q^b_{\alpha_2}$ (resp. $\delta^b(q^b_\alpha) = q^b_{\alpha_1} \wedge q^b_{\alpha_2}$), $\delta^b(q^b_{\alpha_1}) = \alpha_1$ and $\delta^b(q^b_{\alpha_2}) = \alpha_2$,
4. if $\alpha \equiv \sigma X.\alpha$ with $\sigma = \nu$ (resp. $\sigma = \mu$) then then $\mathcal{A}^b_\alpha$ has tree states q^ϵ_α, $q^b_{\alpha_1}$ and q^b_X with $\delta^b(q^b_\alpha) = q^b_{\alpha_1}$, $\delta^b(q^b_{\alpha_1}) = \alpha_1[q^b_X/X]$, $\delta^b(q^b_X) = q^b_{\alpha_1}$ with, for every state $q \in Q^b$, $\Omega^b(q) = 0$ (resp. $\Omega^b(q) = 1$),

with, for every state $q \in Q^b$, $\Omega^b(q) = 0$ unless specified differently.

3.2.1.2 Lemma. *For every fixed point formula α, the formula α and the automaton $\mathcal{A}^b_\alpha$ are equivalent, i.e. for every model M, $M \models \alpha$ if and only if $M \in L(\mathcal{A}^b_\alpha)$.*

Proof. Let M be a Σ-labeled graph. We have to prove that $M \models \alpha$ if and only if $M \in L(\mathcal{A}^b_\alpha)$. All cases but the fixed point cases are immediate consequences of the definitions. We only

detail here the fixed point case. Moreover, by duality, it suffice to prove it for greatest fixed point.

Assume $\alpha \equiv \nu X.\alpha_1$ and assume $M \models \alpha$. We want to show that $M \in L(\mathcal{A}^b_\alpha)$. In order to do so, let $V = \nu X.\alpha_1^M(X)$. By assumption, one has, in particular, $r^M \in V$. We define then a winning strategy for the player P in the game $\mathcal{A}^b \times M$ as follows.

From initial position (q_α, r^M) (resp. any position of the form (q_X, v) with $v \in V$) the player P moves to mapping m_{r^M} (resp. m_v) define, for every state $q \in Q^b$ by $m(q) = \emptyset$ if $q \neq q_{\alpha_1}$ and $m(q_{\alpha_1}) = \{r^M\}$ (resp. $m(q_{\alpha_1}) = \{v\}$). We do have $\langle r^M, m_{r^M}\rangle_M \models \delta(q_\alpha)$ (resp. $\langle v, m_v\rangle_M \models \delta(q_\alpha)$). After this move, the player E can only move to position (q_{α_1}, r^M) (resp. (q_{α_1}, v)) and we have $\langle r^M\rangle_M \models \alpha_1(V)$ (resp. $\langle v\rangle_M \models \alpha_1(V)$).

From any position of the form (q_{α_1}, v) with $v \in V$, player P moves to mapping m defined, for every state $q \in Q^b$ by $m(q) = \emptyset$ if $q \neq q_X$ and $m(q_X) = V$. We do have $\langle v, m\rangle_M \models \delta(q_{\alpha_1})$, because $v \in V = \alpha_1^M(V)$. Then the player E can only move to a position of the form (q_X, v) with $v \in V$ and the strategy is repeated as above.

Because $\Omega(q_X) = 0$. The strategy defined here for the player P is obviously winning.

Conversely, assume that the player P has a winning strategy from the position (q_α, r^M) in the game $\mathcal{A}^b \times M$, and let W_0 be the set of (winning) the player P position that are reached when player P plays according to this strategy.

Because one has $(q_\alpha, r^M) \in W_0$ and $\delta^b(q_\alpha) = q_{\alpha_1}$ then one also has $(q_{\alpha_1}, r^M) \in W_0$. Defining then set V to be the set of vertices $v \in V^M$ such that $(q_X, v) \in W_0$, and given mapping m where the player P moves to, from position (q_{α_1}, r^M), one has $m(q_X) \subseteq V$ hence, because α_1^M is monotonic, one also has $r^M \in \alpha_1^M(V)$.

Similarly, for every vertex $v \in V^M$ such that $(q_X, v) \in W_0$, because $\delta^b(q_x) = q_{\alpha_1}$ one has $(q_{\alpha_1}, v) \in W_0$, and then, following the player P next moves, one also have $v \in \alpha_1^M(V)$. This shows that $V \subseteq \alpha_1^M(V)$ hence $V \subseteq \nu X.\alpha_1^M(X)$.

From the fact that $r^M \in \alpha_1^M(V)$ and by monotonicity of α_1^M, this shows that $r^M \in \nu X.\alpha_1^M(X)$, that is $M \models \nu X.\alpha_1(X)$.

□

3.2.2 From fixed point formulas to alternating automata

In this section, we built, for every fixed point sentence α, an equivalent alternating automaton $\mathcal{A}_\alpha$ that is *flat*: no fixed point occurs in transition specifications. This is done by induction on the structure of the formula α, applying expansion lemma proved in previous section.

3.2.2.1 Theorem. *For every counting fixed point formula α with free variables in Var_α, there exists a flat alternating automaton $\mathcal{A}_\alpha$ with same index such that, for every $\mathcal{P}(Pred \cup Var_\alpha)$-labeled graphs M: $M \models \alpha$ if and only if $M \in L(\mathcal{A}_\alpha)$.*

Proof. We define the automaton $\mathcal{A}_\alpha$ by induction on the syntactic complexity of the formula α.

If α is an atomic formula, we define $\mathcal{A}_\alpha$ to be the automaton $\mathcal{A}^b_\alpha$.

If α is a modality of the form $\Diamond_k\alpha_1$ or $\Box_k\alpha_1$, we define $\mathcal{A}_\alpha$ to be the expansion $E(\mathcal{A}^b_\alpha, \{\mathcal{A}_{q^b_{\alpha_1}}\})$ where $\mathcal{A}_{q^b_{\alpha_1}}$ is the automaton $\mathcal{A}_{\alpha_1}$.

If α is a conjunction or a disjunction of the form $\alpha_1 \wedge \alpha_2$ or $\alpha_1 \vee \alpha_2$, we define $\mathcal{A}_\alpha$ to be the expansion $E(\mathcal{A}^b_\alpha, \{\mathcal{A}_{q^b_{\alpha_1}}, \mathcal{A}_{q^b_{\alpha_2}}\})$ where $\mathcal{A}_{q^b_{\alpha_1}}$ and $\mathcal{A}_{q^b_{\alpha_2}}$ are automata $\mathcal{A}_{\alpha_1}$ and $\mathcal{A}_{\alpha_2}$.

If α is a greatest or a least fixed point of the form $\sigma X.\alpha_1(X)$, we define $\mathcal{A}_\alpha$ to be $E(\mathcal{A}^b_\alpha, \{\mathcal{A}_{q_{\alpha_1}}\})$ where $\mathcal{A}_{q_{\alpha_1}}$ is an automaton q^b_X-equivalent with the formula $\alpha_1[q^b_X/X]$. Since $\mathcal{A}_{\alpha_1}$ is equivalent to α_1 the positive projection lemma (Lemma 3.1.4.2) ensures that such an automaton can easily be built.

It remain to prove that, for every formula α, the automaton $\mathcal{A}_\alpha$ is equivalent to α.

The case where α is atomic is proved by Lemma 3.2.1.2. The induction step is proved by applying the Lemma 3.2.1.2 on the automaton $\mathcal{A}^b_\alpha$ and the Expansion Lemma 3.1.4.7 that built the automaton $\mathcal{A}_\alpha$ from the automaton $\mathcal{A}^b_\alpha$ and the automata translating the immediate subformula(s) of the formula α.

In order to check that Lemma 3.1.4.7 applies, we inductively check that the automaton $\mathcal{A}_\alpha$ has the following properties: for every state $q \in \mathcal{A}_\alpha$, the initial state q_α does not occur in $\delta(q)$, for every variable X that occur free and positively in the formula α, if X occurs in $\delta(q)$ then it occurs positively.

In all cases but the case of least fixed point, automata $\mathcal{A}_{\alpha_1}$ and $\mathcal{A}_{\alpha_2}$ are Q^b_α-normal.

In the case of the least fixed point $\mu X.\alpha_1(X)$, if ever X occurs in α_1 in the scope of a greatest fixed point construction $\nu Y.\alpha_2(Y)$ with priority $\Omega(q_Y) = 0$, the automaton $\mathcal{A}_{\alpha_1}$ is not Q^b_α-normal. Observe however that, in this case, this problem can be avoided just by increasing by two all priorities of the states in the automaton $\mathcal{A}_{\alpha_1}$. Then, also in this case, the Lemma 3.1.4.7 applies.

Observe that priority are increased only when necessary so the automaton $\mathcal{A}_\alpha$ and the formula α do have same index.

□

3.2.2.2 Corollary (Flattening). *Every alternating automaton $\mathcal{A}$ is equivalent to a flat alternating automaton $\mathcal{A}'$.*

Proof. One can first applies Theorem 3.2.2.1 to the (non flat) transition specification occurring in the automaton $\mathcal{A}$. One can then applies Lemma 3.1.4.7 to replace these transition specifications by equivalent alternating flat automata.

□

Remark. The previous corollary is the starting point of many model-checking algorithms for the modal mu-calculus. In fact, solving parity games or model checking mu-calculus formulas are inter-reducible problems.

Observe that the automaton $\mathcal{A}_\alpha$, defined in the proof of Theorem 3.2.2.1 in an inductive way by composing basic automata, can also be defined directly. Such a direct construction may have some interest for algorithmic purpose.

The *syntactic closure* $cl(\alpha)$ of the formula α, i.e. $cl(\alpha)$ is the least set of fixed point formulas that contains the formula α and closed under the following rules: if $\Box_k\beta \in cl(\alpha)$ or $\Diamond_k\beta \in cl(\alpha)$ then $\beta \in cl(\alpha)$, if $\beta \vee \gamma \in cl(\alpha)$ or $\beta \wedge \gamma \in cl(\alpha)$ then $\beta \in cl(\alpha)$ and $\gamma \in cl(\alpha)$, and, if $\sigma X.\beta \in cl(\alpha)$ for $\sigma = \nu$ or μ then $\beta \in cl(\alpha)$.

The Automaton $\mathcal{A}_\alpha = \langle Q, \Sigma, q_0, \delta, \Omega\rangle$ can then be defined as follows:

1. the set of states Q is $\{q_\beta : \beta \in cl(\alpha)\}$, i.e. one state per formulas of the closure $cl(\alpha)$ of α,

2. initial state q_0 is q_α,
3. transition function δ is defined, for every state $q_\beta \in Q$ according to the structure of the formula β:
   (a) if $\beta \equiv p$ (resp. $\beta \equiv \neg p$) then $\delta(q_\beta) = p$ (resp. $\delta(q_\beta) = \neg p$),
   (b) if $\beta \equiv \Diamond_k \beta_1$ (resp. $\beta \equiv \Box_k \beta_1$) then $\delta(q_\beta) = \Diamond_k q_{\beta_1}$ (resp. $\delta(q_\beta) = \Box_k q_{\beta_1}$),
   (c) if $\beta \equiv \beta_1 \wedge \beta_2$ (resp. $\beta \equiv \beta_1 \vee \beta_2$) then $\delta(q_\beta) = q_{\beta_1} \wedge q_{\beta_2}$ (resp. $\delta(q_\beta) = q_{\beta_1} \vee q_{\beta_2}$),
   (d) if $\beta \equiv \sigma X.\beta$ with $\sigma = \mu$ or ν then $\delta(q_\beta) = q_{\beta_1}$,
   (e) and if $\beta \equiv X$ then $\delta(q_\beta) = q_{\beta_1}$ where $\sigma X.\beta_1$ for $\sigma = \mu$ or ν is the binding definition of variable X,
4. and, priority function Ω is defined, for every state $q_\beta \in Q$, by $\Omega(q_\beta) = N_\alpha(X)$ when $\beta \equiv X$ and $\Omega(q_\beta)$ is the maximum of $N_\alpha(Y)$ where Y ranges over the set of formula's variable.

In the sequel, we shall write $G(\alpha, M)$ the model-checking game of an automaton $\mathcal{A}_\alpha$ on a model M.

3.2.2.3 Corollary (From mu-calculus to MSO). *For every $n = 0$, any formula $\alpha \in NC_n$ can be translated into an equivalent formula in monadic Σ_n.*

Proof. For $n = 0$, NC_0 is just counting modal logic so nothing have to be done. Let n be a strictly positive integer and let $\alpha \in NC_n$.

Observe that the formula φ_α defined in section 1.4 is not, in general, in monadic Σ_n. In fact, fixed point constructions that generate monadic quantifiers in the translation may occur in the scope of modalities that generate FO quantifiers. It follows that this theorem is by no mean a trivial consequence of the mu-calculus semantic definitions.

The MSO formula we are looking for in monadic Σ_n follows from the observation that, for arbitrary graph, the model-checking game $G(\alpha, M)$ is definable by means of an existential monadic transduction [43] within graph M itself. In fact, graph $G(\alpha, M)$ can be defined in FO out of $|\alpha|$ disjoint union of graph M. And these $|\alpha|$ copies can be defined by means of $|\alpha|$ existential set quantifiers. Then, on the resulting monadic Σ_1 defined model checking game, the existence of a winning strategy can be defined by means of a N_n mu-calculus formula α_n in prenex normal form [33] which, in turn, can be translated into a monadic Σ_n. Factorizing all monadic quantifiers in these two formulas build one on top of the other, the resulting formula belongs to monadic Σ_n and is equivalent to the formula α. □

3.2.3 From alternating automata to fixed point formulas

Now we want to translate back every alternating automaton to an equivalent formula. More precisely, we prove the following theorem.

3.2.3.1 Theorem. *For any counting (resp. modal) alternating automaton $\mathcal{A}$ there exists an equivalent counting (resp. modal) fixed formula $\alpha_\mathcal{A}$ with same index.*

Proof. We say that an automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \Omega \rangle$ is a *tree-shaped automaton with back-edges* when there exists a partial order $\leq_\mathcal{A}$ over Q such that:

1. state q_0 is the least element for order $\leq_\mathcal{A}$,

2. given $\prec_{\mathcal{A}}$ the successor relation induced by the order $\leq_{\mathcal{A}}$, for every state $q \in Q$, every state $q_1 \in Q$, if the state q_1 occurs in $\delta(q)$ then
   (a) (forward edge) $q \prec q_1$,
   (b) or (backward edge) $q_1 \leq_{\mathcal{A}} q$, and, in this case, for every state $q_2 \in Q$, if $q_1 \leq_{\mathcal{A}} q_2 \leq_{\mathcal{A}} q$ then $\Omega(q_1) \leq \Omega(q_2)$ (i.e. state q_1 has minimum priority on the "loop" induced by the backyard-edge from q to q_1.

Remark. Strictly speaking, the above definition is rather a definition of DAG-shaped automata. It does suffice for translating alternating automata into formulas.

3.2.3.2 Lemma. *Every finite alternating automaton is equivalent to a finite alternating* tree-shaped *automaton.*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \Omega \rangle$ be an automaton. We define an equivalent tree-shaped automaton $\mathcal{A}' = \langle Q', \Sigma, q'_0, \delta', \Omega' \rangle$ as follows.

Set of states Q' is defined to be the set of automata traces t of the form $q_0.q_1.\cdots.q_n$ such that, for every integer i and j such that $0 \leq i < j \leq n$, if $q_i = q_j$ then there is some integer k such that $i < k < j$ and $\Omega(q_k) < \Omega(q_i) = \Omega(q_j)$.

One can check that the set of traces Q' defined in such a way is closed under prefix and is finite with a cardinal at most doubly exponential in the cardinal of Q.

For every trace $t = q_0.q_1.\cdots.q_n \in Q'$, every state $q \in Q$, we define $update(t, q)$ to be the shortest prefix $q_0.\cdots.q_i$ of $t.q$ such that $q_i = q$ and, for every k such that $i < k \leq n$, $\Omega(q_i) \leq \Omega(q_k)$, i.e. the minimum priority encountered in the cycle from $q_i.q_{i+1}.\cdots.q_n.q_i$ is the priority of q_i. By definition, $update(t, q) \in Q'$.

For trace t, we define then $\delta'(t)$ to be the formula obtained from $\delta(q_n)$ by replacing every state q by trace $update(t, q)$.

We also define $\Omega'(t)$ to be $\Omega(q_n)$.

The Automaton $\mathcal{A}'$ is finite and, by construction, tree-shaped. Moreover, mapping $f : Q' \to Q$, that maps every trace of Q' to its target witnesses the fact that the automaton $\mathcal{A}'$ is an unraveling of the automaton $\mathcal{A}$. By applying Lemma 3.1.2.3, we conclude that $L(\mathcal{A}) = L(\mathcal{A}')$.

□

Assume thus $\mathcal{A}$ is a finite tree-shaped automaton. We define, for every state q, by induction on levels of states of $\mathcal{A}$ in order $\leq_{\mathcal{A}}$, from leaves to root, the formula α_q as follows: the formula α_q is the formula obtained from formula $\delta(q)$ by:

1. replacing every state q_1 occurring in $\delta(q)$ by formula α_{q_1} if $q \prec_{\mathcal{A}} q_1$, or by the formula X_{q_1} if $q_1 \leq_{\mathcal{A}} q$,
2. and if q appears in $\delta(q')$ for some state q' with $q <_{\mathcal{A}} q'$, i.e. if q is the target of a back-edge from some state q', taking the least (resp. the greatest) fixed point on the variable X_q of the obtained formula when $\Omega(q)$ is odd (resp. even).

3.2.3.3 Lemma. *The Formula α_{q_0} is equivalent to $\mathcal{A}$.*

Proof. The Automaton $\mathcal{A}$ and the automaton $\mathcal{A}_{\alpha_{q_0}}$ (as defined for Theorem 3.2.2.1) are isomorphic up to obvious contraction.

□

This conclude the proof of Theorem 3.2.3.1

□

3.2.4 Invariance properties for the mu-calculus

In this section, we prove the invariance of modal (resp. counting) fixed point formulas under bisimulation (resp. counting bisimulation).

The following lemma essentially says that the truth of modal or counting fixed point formulas only depends on the underlying tree structures obtained by unraveling.

3.2.4.1 Lemma (Counting bisimulation invariance). *Every fixed point formula α is counting bisimulation invariant.*

Proof. Let α be a modal or counting fixed point formula.

Applying Theorem 1.3.3.4 it suffice to show that for every model M, $M \models \alpha$ if an only if $T(M) \models \alpha$. Let M be model. Applying Theorem 3.2.2.1, there is a flat automaton $\mathcal{A}_\alpha = \langle Q, \Sigma, q_0, \delta, \Omega\rangle$ equivalent to the formula α so it suffices to prove that $M \in L(\mathcal{A})$ if and only if $T(M) \in L(\mathcal{A})$.

Let M be a model and let $t : V^{T(M)} \to V^M$ the mapping that maps any path of $V^{T(M)}$ to its target in V^M. We extend t to markings by composition, i.e. for every marking m in the game $\mathcal{G}(\mathcal{A}, T(M))$ we define $t(m)$ to be marking $t \circ m$. Then one can check that mapping t, induces a (functional) PE-morphism from game $\mathcal{G}(\mathcal{A}, T(M))$ to the game $\mathcal{G}(\mathcal{A}, M)$ that maps initial position in $\mathcal{G}(\mathcal{A}, T(M))$ to initial position in $\mathcal{G}(\mathcal{A}, M)$.

In fact, we essentially have to check that for every $v \in V^{T(M)}$, every state $q \in Q^{\mathcal{A}}$, every marking $m : Q \to \mathcal{P}(V^{T(M)})$, $\langle v, m\rangle_{T(M)} \models \delta(q)$ iff $\langle t(v), t \circ m\rangle_M \models \delta(q)$. This immediately follows from the fact that: $\langle v, m\rangle_{T(M)}$ and $\langle t(v), t \circ m\rangle_M$ are counting bisimilar and $\delta(q)$ (in a flat automaton) is counting bisimulation invariant.

□

3.2.4.2 Lemma (Bisimulation invariance). *Every modal fixed point formula is bisimulation invariant.*

Proof. The argument is essentially the same as above. Applying Theorem 1.3.3.6 it suffices to show that, for arbitrary model M, $M \models \alpha$ if and only if $M^\kappa \models \alpha$. Applying Theorem 3.2.2.1, there is a flat modal automaton $\mathcal{A}_\alpha = \langle Q, \Sigma, q_0, \delta, \Omega\rangle$ equivalent to formula α so it suffices to prove that $M \in L(\mathcal{A})$ if and only if $\in^\kappa L(\mathcal{A})$.

Given the mapping $t : V^{M^\kappa} \to V^M$ be the mapping from M^κ to M that maps every expanded path p in M^κ to its target in M extended to markings by composition, one can check that it induces an EF-morphism from the game $\mathcal{G}(\mathcal{A}, M^\kappa)$ to the game $\mathcal{G}(\mathcal{A}, M)$ that maps initial position in $\mathcal{G}(\mathcal{A}, M^\kappa)$ to initial position in $\mathcal{G}(\mathcal{A}, M)$.

This essentially amount to check that for every $v \in V^{M^\kappa}$, every state $q \in Q^{\mathcal{A}}$, every marking $m : Q \to \mathcal{P}(V^{M^\kappa})$, $\langle v, m\rangle_{M^\kappa} \models \delta(q)$ iff $\langle t(v), t \circ m\rangle_M \models \delta(q)$. Again, this immediately follows from the fact that: $\langle v, m\rangle_{M^\kappa}$ and $\langle t(v), t \circ m\rangle_M$ are now bisimilar and $\delta(q)$ is bisimulation invariant.

□

3.3 References and notes

Relationship between automata and fixed point calculus are known for long. Park defines for instance a unary mu-calculus [145] and proved it equivalent to Büchi word automaton. A relationship between Rabin tree automata and mu-calculus formulas is first established by Niwinski [139] and Emerson et al. [167, 58]. This correspondence is further studied

and extended a lot more [171, 56, 14, 166, 59, 54, 103, 94, 181, 60, 140, 141, 183, 16, 28].

In this subsequent approaches, one may distinguish two approaches.

The first one, which goes back to Emerson et al. works [60, 54, 59, 56, 167, 58], aims at proving, somehow by brute force, that fixed point formulas behave like alternating parity automata. For this purpose, the notion of transfinite fixed point signatures, defined by Street and Emerson [167] and used by many others [103, 181, 94, 108, 182], guarantees the correctness of the translation of alternation of least and greatest nested fixed point expression into increasing alternation of odd and even priorities.

One draw back of this approach is that it is powerful enough to prove, at the same time, memoryless game determinacy of parity games, and even simulation theorem [59, 103, 94]. It follows that the underlying numerous concepts may increase the intrinsic conceptual difficulty of these proofs.

The second approach, developed by Arnold and Niwinski [10, 140, 16], aims at proving the equivalence in an inductive way. It leads to a proof that is somehow easier to follow as it proceeds much more gradually. This approach has been followed by others like, in particular, Wilke[183].

Aiming at presenting a modular version of this result, it is also the approach that has been chosen here. Observe that the existence of memoryless strategy in parity game is, strictly speaking, not necessary in the proof presented here since game determinacy suffices to prove the correctness of complementation.

Chapter 4

Bisimulation invariance in MSO

Classical logical systems such as monadic second-order logic (MSO) often play, in computer science, the role of basic (assembly-like) languages into which programs - or rather program specifications - can be described or translated. In concurrency, where programs are often modeled as state/transition systems [85, 7, 127, 8, 9], monadic second-order logic is generally considered as a sufficiently expressive logic. In particular, it subsumes most specification languages such as LTL, CTL* [18, 56, 60] or fixed point languages [57, 110].

However, it is not full monadic second-order logic which is needed. In fact, when specifying properties of programs, one is generally interested in the behavior of programs rather than in the programs themselves. As program behaviors can be modeled by infinite trees - trees obtained by unraveling program models - it appears that specifying program behaviors amounts to specifying languages of finite and infinite trees.

Moreover, in application, when programs are modeled by means of finite state systems, it is important to check, on the finite models of programs, that their potential infinite behaviors - their unravelings - are correct w.r.t. a given specification. In other words, a program specification must define a class of graphs which is - at least - invariant under unraveling [45]. The peculiar status of non-determinism even encourages to consider classes of graphs which are invariant under bisimulation equivalence [85, 86, 127].

This leads to the study of the bisimulation (or counting bisimulation) invariant fragment of monadic second-order logic, i.e. the set of MSO sentences whose classes of models are closed under bisimulation. A first easy observation is that all specification languages mentioned above are part of this fragment and, among them, the mu-calculus is the most expressive language. A result of Walukiewicz and the author [104] shows that the mu-calculus is even maximal in this respect, i.e. *the bisimulation (resp. counting bisimulation) invariant fragment of monadic second-order logic equals the modal (resp. the counting) mu-calculus.*

The purpose of this chapter is to prove this result and to give an overview of its consequences.

4.1 Normalizing automata

In this section, we essentially prove that alternating automata are equivalent to non alternating automata (which are defined semantically) and thus can be normalized to non

deterministic automata (which are defined syntactically).

Applying this normalization results, we obtain as corollaries a solution to the emptiness problem and the finite model property. They are studied in the last part of this section.

4.1.1 Functional runs and non alternating automata

The intuitive idea behind the notion of alternation in a given alternating automaton is that, at some time, in a given vertex, one may have to run several copies of this automaton in distinct states on the same subtree. In terms of model checking game, it means that at least two positions of the form (q_1, v) and (q_2, v) with distinct states q_1 and q_2 are reached in an accepting run. Preventing alternation to occurs amounts intuitively to saying that every vertex is visited at most with one state. This is the idea of functional runs.

4.1.1.1 Definition (functional run). Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, Acc\rangle$ be an alternating automaton. Let M be a graph, and let ρ be a mapping from V^M to Q.

We say that the mapping ρ is an *functional run* of the automaton $\mathcal{A}$ over the graph M when the following conditions are satisfied:

– *Initial condition:* $\rho(v) = q_0$,

– *Local condition:* for every $v \in V^M$, one has $\langle v, \rho^{-1} \cap Succ(v)\rangle_M \models \delta(\rho(q))$.

where $\rho^{-1} \cap Succ(v)$ is the mapping $m : Q \to \mathcal{P}(V^M)$ defined, for each $q \in Q$, by $m(q) = \rho^{-1}(q) \cap Succ(v)$.

A functional run is *accepting* when no vertex is labeled by $\bot$ and, moreover the following condition is satisfied:

– *Global condition:* for every infinite path $p \in (V^M)^\omega$ of M, $\rho(p) \in Acc$.

A functional accepting run can be seen as a particular case of accepting run as stated below.

4.1.1.2 Lemma. *For every graph M, if there is a functional accepting run $\rho : V^M \to Q$ of a parity automaton $\mathcal{A}$ on M then $M \in L(\mathcal{A})$.*

Proof. Let $\rho : V^M \to Q$ be an accepting functional run of the automaton $\mathcal{A}$ over graph M. Let s_ρ be the memoryless strategy that maps every pair $(q, v) \in Q \times V^M$ to the mapping $\sigma_\rho(q, v)$ defined, for every $q' \in Q$, by $s_\rho(q, v)(q') = \rho^{-1}(q') \cap Succ(v)$.

The fact that the strategy s_ρ is a winning strategy for player 0 follows easily from the definition of functional accepting run.

□

Observe that, in general, the converse is false, i.e. there are automata for which the existence of an accepting run does not implies the existence of a functional run. Take for instance the alternating automaton $\mathcal{A}_\alpha$ defined from the formula $\alpha \equiv \Diamond p \wedge \Diamond q$.

Automata that satisfy, on trees, the converse of Lemma 4.1.1.2 are defined to be non alternating.

4.1.1.3 Definition (Non alternating automata). An automaton $\mathcal{A}$ is *non alternating* when, for every tree M, $M \in L(\mathcal{A})$ if and only if there exists an accepting functional run of $\mathcal{A}$ over M.

This definition of non alternation is semantic. We provide below an equivalent syntactic definition.

4.1.2 Non deterministic automata

Let $CNT(Q)$ be the set of finite disjunctions of FO-formulas of the form

$$\exists \overline{x}\, \mathit{diff}(\overline{x}) \wedge \bigwedge_{i\in\{1,\cdots,k\}} T(r,x_i) \wedge q_i(x_i) \wedge \left(\forall z.\mathit{diff}(z,\overline{x}) \Rightarrow \bigvee_{q\in Q_z} q(z) \right)$$

where $\overline{x} = x_1, \cdots, x_k$, possibly empty with $k = 0$, is a vector of distinct variables, for each $i \in q_i$, $q_i \in Q$ and $Q_z \subseteq Q$, and $\mathit{diff}(\overline{x})$ is the predicate stating that all elements denoted by variables in $\overline{x}$ are distinct, e.g. $\mathit{diff}(\overline{x}) = \bigwedge_{1\leq i<j\leq k} x_i \neq x_j$.

4.1.2.1 Definition (Non deterministic automata). An automaton $\mathcal{A}$ *non deterministic* when for every state $q \in Q$, transition specification $\delta(q)$ is (equivalent to) a formula of the form

$$\delta(q) \equiv \bigwedge_{a\in\Sigma} a(r) \to \varphi_{q,a}$$

with for every $a \in \Sigma$, $\varphi_{q,a} \in CNT(Q)$ and $a(x)$ is the formula

$$a(x) \equiv \bigwedge_{p\in a} p(x) \wedge \bigwedge_{p\notin a} \neg p(x)$$

Remark. Although not written with modality, formulas in $CNT(Q)$ are equivalent to counting modal formulas positive in predicate symbols of Q. This can be seen as a classical exercise in logic. It is also a consequence of Courcelle's results [42]. This implies in particular that non deterministic automata are particular case of alternating automata.

4.1.2.2 Lemma. *Counting non deterministic automata are non alternating.*

Proof. Let $\mathcal{A}$ be an non deterministic automaton. We want to prove that $\mathcal{A}$ is non alternating. Let M be a Σ-tree and let σ be a winning strategy in $\mathcal{G}(\mathcal{A}, M)$. We define a mapping $\rho : V^M \to Q$ first by taking $\rho(r^M) = q_0$. We proceed then by induction on the depth of vertex $v \in V^M$ with the following invariant property:

> (I) There is a counter strategy τ in the game $G(\mathcal{A}, M)$ and a play $p \in \sigma * \tau(q_0, r^M)$ that ends in position $(\rho(v), v)$.

The invariant is satisfied at the root r^M of M. Assume $\rho(v) = q$ for some $v \in V^M$. By invariance property, and since σ is winning, the player E can move to marking $m : Q \to \mathcal{P}(V^M)$ such that $\langle v, m\rangle_M \models \delta(q)$. By definition of non deterministic automata transition specification (formulas of $CNT(Q)$), this means that there is a function $f : \mathit{Succ}(v) \to Q$ such that $f^{-1} \subseteq m$ and $T_{v,f} \models \delta(q)$. For each successor v' of v we define then $\rho(v') = f(v')$. Since f is a mapping and $f^{-1} \subseteq m$, the player E can move to any pair $(\rho(v'), v')$ with $v' \in \mathit{Succ}(v)$ which conclude the induction step.

The functional run is accepting since, for every infinite path from the root $p \in (V^M)^\omega$, $\rho(p)$ is, by invariance property, the sequence of states that occurs in an infinite play in $\sigma * \tau(q_0, r^M)$ hence, because σ is winning, it belongs to Acc.

□

Conversely:

4.1.2.3 Lemma. *Every non alternating automaton is equivalent to a non deterministic automaton.*

Proof. Last, let $\mathcal{A}$ be a non alternating automaton. For every state q of $\mathcal{A}$, for every $a \in \Sigma$, it is an easy observation that there is a formula $\varphi_{q,a}$ of $CNT(Q)$ such that, for every depth one Σ tree M and every mapping $f : Succ^T(r^M) \to Q$, one has $\langle r^M, f^{-1}\rangle_M \models \delta(q) \wedge a(r)$ if and only if $\langle r^M, f^{-1}\rangle_M \models \varphi_{q,a}$.

It follows that the automaton $\mathcal{A}'$ built from $\mathcal{A}$ just by changing the transition specification δ to δ' defined, for each $q \in Q$, by $\delta'(q) \equiv \bigwedge_{a\in\Sigma} a(r) \to \varphi_{q,a}$, is equivalent to $\mathcal{A}$ as far as functional runs are concerned. But then, since the automaton $\mathcal{A}'$ it non deterministic, it is also non alternating (by Lemma 4.1.2.2) and thus (by definition of non alternation) equivalent to $\mathcal{A}$.

□

4.1.3 The simulation theorem

4.1.3.1 Theorem (Muller and Schupp [136, 137]). *Every alternating automaton $\mathcal{A}$ is equivalent (can be simulated by) a non alternating automaton $\mathcal{A}^n$.*

Moreover, if $\mathcal{A}$ is a parity automaton with, respectively, closed, open or Büchi infinitary conditions, then we can choose $\mathcal{A}^n$ to be a parity automaton with, respectively, closed, open or Büchi infinitary conditions, i.e. if $\Omega^{\mathcal{A}}(Q^{\mathcal{A}})$ equals $\{0,\}$, $\{1\}$ or respectively $\{0,1\}$ then we can choose $\mathcal{A}^n$ such that $\Omega^{\mathcal{A}^n}(Q^{\mathcal{A}^n}) = \Omega^{\mathcal{A}}(Q^{\mathcal{A}})$.

Proof. Let $\mathcal{A} = \langle Q, q_0, \Sigma, \delta, \Omega\rangle$ be an alternating parity automaton. We write $Acc = \{w \in Q^\omega : \liminf \Omega(q) \text{ is even}\}$. By applying Corollary 3.2.2.2, we assume that the automaton $\mathcal{A}$ is flat.

The Automaton $\mathcal{A}^n = \langle Q^n, Q_0^n, \Sigma, \delta^n, Acc^n\rangle$ is defined as follows:

1. *set of state* Q^n is defined to be $Q^n = \mathcal{P}(Q \times Q \times \{0,1\}) - F^n$ with F^n defined below,
2. *set of initial state* $Q_0^n = \{R : (q_0, q_0, 1) \in R\}$,
3. *transition specification* δ^n defined, for every $R \in Q^n$, by $\delta^n(R)$ defined to be the formula obtained from the formula $\bigwedge_{(q_1,q_2,x)\in R} \delta(q_2)$ after:
   (a) replacing every subformula of the form $\Box_k q_3$ (or resp. $\Diamond_k q_3$) for some $q_3 \in Q$, by the predicate $\Box_k D_{(q_2,q_3,1)}$ (or resp. $\Diamond_k D_{(q,R),(q_2,q_3,1)}$), where $D_{(q_2,q_3,1)} \equiv \bigvee_{(q_2,q_3,1)\in R'} R'$,
   (b) and, after the previous replacement have been performed (so that every remaining state are not in the scope of modalities), replacing every remaining state predicate $q_3 \in Q$ by $\top$ if $(q_2, q_3, 0) \in R$ and $\bot$ if $(q_2, q_3, 0) \notin R$.
4. Acc^n is defined to be the set of all infinite path $p \in (Q^n)^\omega$ such that $tr(p) \subseteq Acc$,

where for every finite or infinite word $p \in (Q^n)^\infty$, $tr(p)$ is defined to be the set of all infinite words $t \in Q^\omega$, called *trace* of $\mathcal{A}$ on p, such that there is a sequence of integer $\{j_i\}_{i\in\mathbb{N}}$ such that, for every $i \in \mathbb{N}$:

1. $0 \leq j_{i+1} - j_i \leq 1$,
2. $(t[i], t[i+1], (j_{i+1} - j_i)) \in p[j_{i+1}]$,

and $F^n \subseteq Q^n$, the set of *forbidden states*, is defined to be the set of all states $R \in Q^n$ such that $tr(R) \cap \overline{Acc} \neq \emptyset$.

Observe that, for every $q^n \in Q^n$, $\delta^n(q^n)$ is a (counting) modal formula of modal depth one so the automaton $\mathcal{A}^n$ is non deterministic.

4.1.3.2 Lemma (Validity). *The Automaton $\mathcal{A}^n$ is equivalent to the automaton $\mathcal{A}$.*

Proof. Let M be a tree. Assume $M \in L(\mathcal{A})$ and let σ be a positional winning strategy in the game $G(\mathcal{A}, M)$ from the position (q, r^M).

We define $\rho : V^M \to Q^n$ as follows the smallest mapping (ordered by inclusion) such that:

1. $(q_0, q_0, 1) \in \rho(r^M)$,
2. for every $q \in Q$ and $v \in V^M$, if there is a counter strategy τ such that there is a play $p \in \sigma * \tau(q_0, sr^M)$ that ends in position (q, v) then, given $m = \sigma(q, v)$ such that $\langle v, m\rangle_M \models \delta(q)$, for every $q' \in Q$, for every $v' \in Succ(v)$:
   (a) if $v \in m(q')$ then $(q, q', 0) \in \rho(v)$,
   (b) if $v' \in m(q')$ then $(q, q', 1) \in \rho(v')$.

One can check that ρ is an accepting functional run of $\mathcal{A}^n$ on M.

Conversely, assume there is a functional accepting run $\rho : V^M \to Q^n$ of the automaton $\mathcal{A}^n$ on M, we define positional strategy σ for the player P as follows.

For every $(q, v) \in Q \times V^M$, provided, $(q_1, q, x) \in \tau(v)$ for some $q_1 \in Q$ and $x \in \{0, 1\}$ - otherwise $\sigma(q, v)$ is left undefined - we define $m : Q \to \mathcal{P}(V)$ to be the smallest (w.r.t. inclusion) mapping such that, for every $q' \in Q$, for every $v' \in Succ(v)$:

1. if $(q, q', 0) \in \tau(v)$ then $v \in m(q')$,
2. if $(q, q', 1) \in \tau(v')$ then $v' \in m(q')$.

One can check that σ is a winning strategy for the player P in the game $G(\mathcal{A}, M)$ from position (q_0, r^M).

□

Proof of Theorem 4.1.3.1 (continued). It remain to prove that the automaton $\mathcal{A}^n$ is regular.

4.1.3.3 Lemma (Regularity). *There exists a finite word automaton $\mathcal{A}^c$ on the alphabet Q^n, called the path automaton of $\mathcal{A}$, such that $L(\mathcal{A}_c) = Acc^n$.*

Moreover, if the set $\Omega(Q)$ of priorities used in the automaton $\mathcal{A}$ equals to $\{1\}$ or $\{0, 1\}$ respectively then one can build $\mathcal{A}^c$ in such a way that the set $\Omega^c(Q^c)$ of priorities used in the automaton $\mathcal{A}^c$ equals to $\{1\}$ or $\{0, 1\}$ respectively.

Proof. Let A be the alphabet Q^n.

We first define a parity automaton $\mathcal{A}' = \langle Q', q'_0, \delta', \Omega'\rangle$ that accepts all infinite words of A^∞ that contains a trace $t \in \overline{Acc}$. Then we obtained $\mathcal{A}^c$ by determinization and complementation of the automaton $\mathcal{A}'$.

The automaton $\mathcal{A}'$ is defined as follows (with ϵ-transition). The set of states Q' is defined to be Q, the initial state q'_0 is q_0, the (non deterministic) transition function δ' is defined, for every $q \in Q$ and every $R \subseteq Q \times Q \times \{0, 1\}$ by

$$\delta'(q, R) = \{q' \in Q : (q, q', 1) \in R\}$$

and

$$\delta'(q,\epsilon) = \{q' \in Q : (q,q',0) \in R\}$$

and, for every $q \in Q$, $\Omega'(q) = \Omega(q) + 1$.

By construction of the automaton $\mathcal{A}'$ one has $L(\mathcal{A}') = \overline{Acc}$, i.e. the automaton $\mathcal{A}$ recognizes the set of words of A^∞ which contains traces that does not satisfies the parity conditions Ω.

Then, we define the automaton $\mathcal{A}^c$ by determinization and complementation of the automaton $\mathcal{A}'$. Safra's construction [157] says that determinization can be made in such a way to ensures that $|Q^c| = O(|Q|!)$ and, if $\Omega(Q) = \{1\}$ or $\{0,1\}$ then $\Omega^c(Q^c) = \Omega(Q)$.

□

Remark. When $\Omega(Q) = \{0\}$, $Acc = Q^\omega$ and $Acc^n = (Q^n)^\omega$ so there is no need to build the automaton $\mathcal{A}^c$.

Proof of Theorem 4.1.3.1 (end).

If we require that the automaton $\mathcal{A}^n$ is a parity automaton then we can normalize it applying Lemma 3.1.2.4 with ω-automaton $\mathcal{A}^c$.

□

4.1.4 The emptiness problem and the finite model property

In this section, we prove that, given a non deterministic parity automaton $\mathcal{A}$, the problem of deciding if there is a model $M \in L(\mathcal{A})$ - called the emptiness problem - linearly reduces to solving a parity game $Sat_{\mathcal{A}}$. Moreover, since positional strategies suffice to solve parity games, this also show that every automaton that recognizes a graph, recognizes a a finite one. This property is called the finite model property (see e.g. [59, 56, 167, 58] for the modal case).

4.1.4.1 Definition (Satisfiability game). Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, Acc^{\mathcal{A}}\rangle$ be a non deterministic automaton.

Given $(q,a) \in Q \times \Sigma$ and $R \subseteq \mathcal{P}(Q \times \Sigma)$, we say that pair $((q,a),R)$ is satisfiable when there exists a depth one Σ-tree M and a mapping $f : Succ(r^M) \to Q$ such that $\lambda^M(r^M) = a$ and $\langle r^M, f^{-1}\rangle_M \models \delta(q)$.

We define the satisfiability game graph $Sat_{\mathcal{A}} = \langle V_P, V_E, T_P, T_E, Acc\rangle$ to be the game defined by putting $V_P = Q \times \Sigma$, $V_E = \mathcal{P}(V_P)$. Set T_P is defined to be the set of pairs $((q,a),R) \in V_P \times V_E$ such that pair $((q,a),R)$ is satisfiable. Set T_E is defined to be the set of pairs $(R,(q,a)) \in V_E \times V_P$ such that $(q,a) \in R$. Acceptance condition Acc is defined to be the set $\{p \in (V_P + V_E)^\omega : \pi_1 \circ \pi_{V_P}(p) \in Acc^{\mathcal{A}}\}$, i.e. set of plays that induce an accepting sequence of states.

Remark. When the automaton $\mathcal{A}$ is in normal form as in definition 4.1.2.1, for every state $q \in Q$, we have

$$\delta(q) \equiv \bigwedge_{a \in \Sigma} a(r) \to \varphi_{q,a}$$

with $\varphi_{q,a}$ a disjunction of formulas of $CNT(Q)$. It follows that a pair $((q,a),R)$ is satisfiable if and only if one of the disjunct occurring in φ_a, of the form

$$\varphi_{(q,a,R)} \equiv \exists \overline{x}\, \mathit{diff}(\overline{x}) \wedge \bigwedge_{i\in\{1,\cdots,k\}} T(r,x_i) \wedge q_i(x_i) \wedge \left(\forall z\, \mathit{diff}(z,\overline{x}) \Rightarrow \bigvee_{q\in Q_z} q(z) \right)$$

is such that, for every $i \in \{1,\ldots,k\}$, $(q_i,a_i) \in R$ for some a_i in Σ and, for every pair $(q',a') \in R$, $q' \in \{q_1,\cdots,q_k\} \cup Q_z$.

In other words, satisfiable pairs in $Sat_{\mathcal{A}}$ can just be read from the definition of transition specification in $\mathcal{A}$. One can even restrict to minimal w.r.t. inclusion.

4.1.4.2 Lemma (Emptiness). *Let $\mathcal{A}$ be a non deterministic parity automaton. Language $L(\mathcal{A})$) of graphs recognizable by the automaton $\mathcal{A}$ is non empty if and only if there is some $a \in \Sigma$ such that position (q_0,a) is winning for the player P in the satisfiability game $Sat_r A$.*

Proof. Assume that $L(\mathcal{A}) \neq \emptyset$. Since $L(\mathcal{A})$ is closed under unraveling, we can assume that there is some tree $M \in L(\mathcal{A})$. Let then $\rho : V^M \to Q$ be an accepting functional run of $\mathcal{A}$ on M and let $a_0 = \lambda^M(r^M)$.

Strategy for the player P from position (q_0,a_0) is defined as a strategy with memory $\sigma = \langle V^M, m_0, c, \delta_u\rangle$ with:

1. memory $V^M \cup \{\bot\}$,
2. initial memory $m_0 = r^M$,
3. choice function $c : V^M \times V_P \to V_E$ defined, for every $v \in V^M$ and $(q,a) \in V_P$ by, $c(v,(q,a)) = R_v$ with $R_v = \{(\rho(v'),\lambda^M(v')) \in V_P : v' \in Succ(v)\}$ when $(\rho(v),\lambda^M(v)) = (q,a)$ and $c(v,(q,a)) = \bot$ otherwise,
4. and update function δ_u defined for every $v \in V^M$ by, for every $R \in V_E$, $\delta_u(v,R) = v$ and, for every $(q,a) \in V_P$, $\delta_u(v,(q,a)) = v'$ for some $v' \in Succ(v)$ such that $(q,a) = (\rho(v'),\lambda^M(v'))$ and $\delta_u(v,(q,a)) = \bot$ otherwise.

One can check that σ is a well defined winning strategy for the player P from position (q_0,v_0).

More precisely, one can check, by induction on the length of plays, that, for every counter strategy τ, given play $p = \sigma * \tau(q_0,v_0)$, one has $\delta_u^*(m_0,p) \neq \bot$ and, if p ends in a position (q,v), then, given $v = \delta_u^*(m_0,p)$, one has $(\rho(v),\lambda^M(v)) = (q,a)$.

Moreover, because ρ is an accepting functional run, $\langle v, \rho^{-1} \cap Succ(v)\rangle_M \models \delta(q)$. It follows that pair $((q,a),R_v)$ is satisfiable, or, in other word, $((q,a),R_v)$ is a well defined move in $Sat_{\mathcal{A}}$.

Conversely, assume that the player P has a winning strategy σ in the game $Sat_{\mathcal{A}}$ from a position (q_0,a_0) for some $a_0 \in \Sigma$. In order to prove the lemma, we define a Σ-graph M_σ proving that $M \in L(\mathcal{A})$ by defining an functional accepting run $\rho : V^M \to Q$ of $\mathcal{A}$ on M_σ.

Without lost of generality, we may assume that strategy σ is positional and, for each $(q,a) \in Q \times a$, $\sigma(q,a)$ is minimal w.r.t. inclusion. Let then $V_\sigma \subseteq Q \times \Sigma$ be the set of the player P positions that are reached from position (q_0,a_0) in plays of $Beh(Sat_{\mathcal{A}},(q_0,a_0),\sigma)$.

For each $(q,a) \in V_\sigma$, let $\varphi_{q,a,R}$ with $R = \sigma(q,a)$ be the formula of the form

$$\varphi_{(q,a,R)} \equiv \exists \overline{x}\, \mathit{diff}(\overline{x}) \wedge \bigwedge_{i \in \{1,\cdots,k\}} T(r,x_i) \wedge q_i(x_i) \wedge \left(\forall z\, \mathit{diff}(z,\overline{x}) \Rightarrow \bigvee_{q \in Q_z} q(z) \right)$$

with R (minimal) of the form $\bigcup_{i\in[1,k]}\{(q_i,a_i)\}$ (see remark page 59) and let $V_{q,a}$ be the set $V_{q,a} = \{(q_i,i,a_i) \in Q \times I\!N \times \Sigma : i \in [1,k]\}$.

We then define $M_\sigma = \langle V^{M_\sigma}, r^{M_\sigma}, T^{M_\sigma}, \lambda^{M_\sigma}\rangle$ as follows:

1. $V^{M_\sigma} = \{(q_0,0,a_0)\} \cup \bigcup_{(q,v)\in V_\sigma} V_{q,\sigma}$,
2. $r^{M_\sigma} = (q_0,0,a_0)$,
3. $T^{M_\sigma} = \{((q,k,a),(q',k',a)) \in V^{M_\sigma} \times V^{M_\sigma} : (q',k',a) \in V_{q,a}\}$,
4. $\lambda^{M_\sigma}(q,k,a) = a$.

We define then $\rho_\sigma : V^{M_\sigma} \to Q$ by, for every $(q,k,a) \in V^M$, $\rho_\sigma(q,k,a) = q$.

One can easily check that ρ_σ is a functional accepting run of $\mathcal{A}$ on M_σ. The satisfaction of initial and local conditions immediately follows from the construction. The satisfaction of global condition follows from the fact that the mapping $f : r^M.(V^M)^\infty \to \{(q,a_0)\}.(V_E.V_P)^\infty$ defined, for each path

$$p = (q_0,k_0,a_0).\cdots.(q_i,k_i,a_i).\cdots$$

by

$$f(p) = (q_0,a_0).\sigma(q_0,a_0).\cdots.(q_i,a_i).\sigma(q_i,a_i).\cdots$$

is a bijection between paths in M_σ emanating from the root r^{M_σ} in M_σ to plays in $Beh(Sat_\mathcal{A},(q_0,a_0),\sigma)$ with, obviously, for every infinite path p, $\rho(p) \in Acc^\mathcal{A}$ if and only if $f(p) \in Acc^{Sat_\mathcal{A}}$.

□

Observe that graph M_σ built above in finite. It follows:

4.1.4.3 Corollary (Finite model property). *Every satisfiable mu-calculus formula α has a finite model.*

Proof. By Theorem 3.2.2.1 the formula α is equivalent to an alternating automaton $\mathcal{A}_\alpha$ which, in turn, applying Theorem 4.1.3.1, is equivalent to a non deterministic automaton $\mathcal{A}^n_\alpha$. Then, following the above construction, $L(\mathcal{A}^n_\alpha) \neq \emptyset$ if and only if there is some finite $M \in L(\mathcal{A}^n_\alpha)$.

□

4.2 MSO and bisimulation invariance

In this section, we prove that over trees, alternating counting automata and MSO sentences are equally expressive. As a corollary, this prove that the counting bisimulation invariant fragment of MSO equal the counting mu-calculus. We also establish then that the bisimulation invariant fragment of MSO equals the modal mu-calculus.

Remember that a class $\mathcal{C}$ of graphs is *bisimulation closed* (resp. *counting bisimulation closed*) if whenever $M \in \mathcal{C}$ and M' is bisimilar (resp. counting bisimilar) to M then $M' \in \mathcal{C}$.

Then, a sentence φ is *bisimulation invariant* (resp. *counting bisimulation invariant*) if the class of transition systems it defines is bisimulation closed (resp. counting bisimulation closed).

The notion of bisimulation invariance (or counting bisimulation invariance) is extended to arbitrary formulas $\varphi(X_1, \cdots, X_n)$ with free set variables X_1, ..., X_n, by considering such formulas as sentences on graphs built with set of predicate symbols $Pred' = Pred \cup \{X_1, \cdots, X_n\}$. Since fixed point formulas, which we will consider later, may have free set variables, we implicitly consider this extension of graphs to $Pred'$ whenever there is no ambiguity.

4.2.1 Counting bisimulation invariance

4.2.1.1 Theorem (Walukiewicz [181, 182]). *A language of tree L is recognizable by a counting alternating automaton if and only if it is definable in MSO.*

Proof. By definition, any language of trees recognizable by an alternating automaton is definable in MSO. It can even be shown that a monadic Σ_3 (or monadic Π_3) suffice to express the existence of an accepting run on a tree.

It remains to show that every language of trees definable in MSO is recognizable by an alternating automaton. For this, we proceed by induction on the syntactic structure of MSO formulas. However, for convenience, we first define a "reduced" set of MSO formulas which is still equivalent to full MSO as far as definability by sentence is concerned.

More precisely, consider the set of MSO formulas defined as the smallest set containing the "atomic" formulas $r(L)$, $L \subseteq L'$ and $T(L, L')$ for every symbol L and $L' \in Pred \cup Var$ and closed under negation, disjunction and existential set quantification. The meaning of these new "atomic" formulas is given by the fixed point formulas L, $\nu X.((L \Rightarrow L') \wedge \Box X)$ and $\nu X.(\Box X \wedge (L \Rightarrow (\Box L')))$.

One can translate arbitrary MSO sentences into equivalent sentences of this reduced set. In fact, other boolean connectives and universal quantification can be encoded with disjunction, existential quantifier and negation. For FO-variable and FO-quantifiers, the key idea is to use, instead, singletons and quantification over singletons. In fact, with $empty(X) \equiv \forall Y X \subseteq Y$, we can specify that set X is a singletons with $sing(X) \equiv \neg empty(X) \wedge \forall Y, Y \subseteq X \Rightarrow (X \subseteq Y) \vee empty(Y)$. Then, FO-quantifications can be simulated by set quantifications, i.e. every formula of the form $\exists x \varphi(x)$ (resp. $\forall x \varphi$) is equivalent to the formula $\exists X sing(X) \wedge \varphi(X/x)$ (resp. $\forall X sing(X) \Rightarrow \varphi(X)$).

It remains to prove that every MSO formula of this reduced set is equivalent, over trees, to an alternating automaton.

We prove this result by induction on the syntactic complexity of these formulas (considering free set variables as constant predicates whenever needed). We know already that (1) definability by fixed formulas implies recognizability by alternating automata, and (2) fixed point formulas are closed under boolean connectives. It suffices to prove that the class of recognizable languages is closed under projection which is, in language theory, the counterpart of existential set quantification.

More precisely, let $\Sigma_X = \mathcal{P}(Pred \cup \{X\})$ and let $\Sigma = \mathcal{P}(Pred)$. We define $\pi_X : \Sigma_X \to \Sigma$ to be the projection that maps every letter $a \in \Sigma_X$ to $\pi_X(a) = a \cap Pred \in \Sigma$. The mapping π_X is extended to Σ_X-graph by defining, for every Σ_X-graph M, the projection

$\pi_X(M)$ to be the Σ_X-graph obtained from graph M just by changing the labeling function $\lambda^M : V^M \to \Sigma_X$ to the labeling function $\lambda^{\pi_X(M)} : V^M \to \Sigma$ defined by $\lambda^{\pi_X(M)} = \pi_X \circ \lambda^M$.

We have to prove that:

4.2.1.2 Lemma (Projection). *For every alternating automaton $\mathcal{A}$ on the alphabet Σ_X there is an automaton $\pi_X(\mathcal{A})$ on the alphabet $\pi_X(\Sigma_X) = \Sigma$ such that, for every Σ_X-tree M, $M \in L(\pi_X(\mathcal{A}))$ if and only if $\pi_X(M) \in L(\mathcal{A})$.*

Moreover, if the automaton $\mathcal{A}$ is a closed, open, weak or Büchi automata, then so is the automaton $\pi_X(\mathcal{A})$.

Proof. By applying Theorem 4.1.3.1, we can assume that the automaton

$$\mathcal{A} = \langle Q, \Sigma_X, q_0, \delta, \Omega \rangle$$

is non deterministic.

By definition of non deterministic automata, for each state $q \in Q$, $\delta(q)$ is (equivalent to) a formula of the form

$$\delta(q) \equiv \bigwedge_{a \in \Sigma_X} a(r) \to \varphi_{q,a}$$

with for every $a \in \Sigma_X$,

$$a(x) \equiv \bigwedge_{p \in a} p(x) \wedge \bigwedge_{p \notin a} \neg p(x)$$

and $\varphi_{q,a} \in CNT(Q)$.

Let define $\delta_X(q)$ to be the formula

$$\delta_X(q) \equiv \bigwedge_{a \in \Sigma} a(r) \to \bigvee_{b \in \pi_X^{-1}(a)} \varphi_{q,b}$$

We then define the automaton $\pi_X(\mathcal{A})$ to be $\pi_X(\mathcal{A}) = \langle Q, \Sigma, q_0, \delta_X, \Omega \rangle$. We claim that, for every Σ_X-tree M, $M \in L(\mathcal{A})$ if and only if $\pi_X(M) \in L(\pi_X(\mathcal{A}))$.

In fact, one can easily show that for every mapping $\rho : V^M \to Q$, ρ is a functional accepting run of $\mathcal{A}$ on M if and only if ρ is a functional accepting run of $\pi_X(\mathcal{A})$ on $\pi_X(M)$. □

This concludes the proof of theorem 4.2.1.1. The fact that the automaton $\pi_X(\mathcal{A})$ can be a closed, open, or, respectively, Büchi automaton when the original (non alternating !) automaton $\mathcal{A}$ is closed, open or, respectively, Büchi follows the fact that simulating alternating the automaton $\mathcal{A}$ by a non deterministic automaton $\mathcal{A}^n$ can be made, as stated in Theorem 4.1.3.1, preserving these complexity of acceptance conditions.

□

4.2.1.3 Theorem (Walukiewicz [181], J. and Lenzi [99]). *The counting bisimulation invariant fragment of MSO equals the counting mu-calculus.*

Proof. Let φ be a counting bisimulation invariant MSO formula. Applying Theorem 4.2.1.1 above, there exists a (counting) automaton $\mathcal{A}_\varphi$ and thus, by applying Theorem 3.2.3.1, a counting mu-calculus formula α_φ such that, for every tree M, $M \models \varphi$ if and only if $M \models \alpha_\varphi$.

Now, since both formulas φ and α_φ (applying Lemma 3.2.4.1) are counting bisimulation invariant, it follows, applying Theorem 1.3.3.4, that formulas φ and α_φ are equivalent on arbitrary graphs.

Conversely, all counting mu-calculus formulas are, by definition, definable in MSO so we are done.

□

4.2.2 Bisimulation invariance

We are now ready to prove a theorem analogous to Theorem 4.2.1.3 with bisimulation in place of counting bisimulation.

4.2.2.1 Theorem (J. and Walukiewicz [104]). *The bisimulation invariant fragment of MSO equals the modal mu-calculus.*

Proof. The remainder of this section is dedicated to the proof of this theorem.

4.2.2.2 Definition (Counter saturated formulas). Let α be a counting or modal mu-calculus formula α. We define the formula $\widehat{\alpha}$, called the counter saturation of formula α, to be the modal mu-calculus formula obtained from the formula α by replacing every counting modality $\Box_i$ or $\Diamond_i$ by the corresponding non counting modality $\Box$ or $\Diamond$.

The formulas α and $\widehat{\alpha}$ are semantically related as follows.

4.2.2.3 Lemma (Saturation lemma). *For every infinite cardinal κ, for every model M, $M \models \widehat{\alpha}$ if and only if $M^\kappa \models \alpha$.*

Proof. Let M be a model and let $t : V^{M^\kappa} \to V^M$ be the mapping that maps ω-indexed path of V^{M^κ} to its target in V^M. Let also $\mathcal{A}_\alpha = \langle Q, \Sigma, q_0, \delta, \Omega\rangle$ be the (flat) automaton defined in the proof of Theorem 3.2.2.1 for the formula α. By definition of $\widehat{\alpha}$, the (flat) automaton $\mathcal{A}_{\widehat{\alpha}} = \langle \widehat{Q}, \Sigma, \widehat{q}_0, \widehat{\delta}, \widehat{W}\rangle$ defined in the proof of Theorem 3.2.2.1 for the formula $\widehat{\alpha}$ can also be defined from the (flat) automaton $\mathcal{A}_\alpha$ as follows: there is a bijection $f : Q \to \widehat{Q}$ such that $f(q_0) = \widehat{q}_0$, for every $q \in Q$, $\widehat{\delta}(f(q))$ equals to the formula $\widehat{f}(\delta(q))$ obtained from the formula $\delta(q)$ by replacing every counting modality by a non counting modality and every state q by its image $f(q)$, and $\Omega(q) = \widehat{\Omega}(f(q))$.

Then it is sufficient to prove that that there is a winning strategy for the player P in the game $G(\mathcal{A}_{\widehat{\alpha}}, M)$ if and only if there is a winning strategy for the player P in the game $G(\mathcal{A}_\alpha, M^\kappa)$.

It occurs that the bijection $f : Q \to \widehat{Q}$ that maps every state $q \in Q$ to state $f(q) = \widehat{q} \in \widehat{Q}$, and the saturating morphism $g : V^{M^\kappa} \to V$ that maps every κ-indexed path $p \in V^{M^\kappa}$ to its target $g(p) \in V^M$, induce a PE-morphism R from $G(\mathcal{A}_\alpha, M^\kappa)$ to $G(\mathcal{A}_{\widehat{\alpha}}, M)$ that suffice, applying Lemma 2.3.2.2, to prove the result.

The PE-morphism R is defined to be the union of the set of pair of the player P positions the form $((q,p),(f(q),g(p))) \in V_P^{\mathcal{G}(\mathcal{A}_\alpha, M^\kappa)} \times V_P^{\mathcal{G}(\mathcal{A}_{\widehat{\alpha}}, M)}$ with $(q,p) \in Q \times V^{M^\kappa}$, and the set of pair of the player E positions the form $(m, g \circ m \circ f^{-1}) \in V_E^{\mathcal{G}(\mathcal{A}_\alpha, M^\kappa)} \times V_E^{\mathcal{G}(\mathcal{A}_{\widehat{\alpha}}, M)}$ with $m : Q \to \mathcal{P}(V^{M^\kappa})$.

The fact that R is a PE-morphism essentially follows from the following property of f and g. For every $(q,p) \in Q \times V^{M^\kappa}$, for every $m : Q \to \mathcal{P}(V^{M^\kappa})$, if

$$\langle p, m\rangle_{M^\kappa} \models \delta(q)$$

then

$$\langle g(p), g \circ m \circ f^{-1}\rangle_M \models \delta^\kappa(f(q))$$

and, for every $m : \widehat{Q} \to \mathcal{P}(V^M)$, if

$$\langle g(p), m\rangle_M \models \widehat{\delta}(f(q))$$

then

$$\langle p, g^{-1} \circ m \circ f\rangle_{M^\kappa} \models \delta(q)$$

If $\delta(q)$ is a conjunction, disjunction or an atomic predicate then this is just obvious. If $\delta(q)$ is a counting modality of the form $\Diamond_n q'$ or $\Box_n q'$ for some integer n (hence $\widehat{\delta}(f(q))$ is of the form $\Diamond f(q')$ or $\Box f(q')$), then the claim follows from the fact that, for each successor $v' \in Succ(v)$ with $v = g(p)$ in the graph M, the vertex p in the graph M^κ has infinitely many (hence more than n) successors of the form $p.k.v'$ with $k \in \kappa$.

□

4.2.2.4 Corollary (J. and Walukiewicz [104]). *For every infinite cardinal κ, MSO and the modal mu-calculus are equivalent on the class of κ-expansions of graphs.*

Proof. On trees, MSO and the counting mu-calculus are equivalent (see Theorem 4.2.1.1). Now, on κ-expansions of graphs, since κ-expansion, up to isomorphism, are involutive, by Lemma 4.2.2.3, the mu-calculus and the counting mu-calculus are equi-expressive.

□

Remark. Starting from a non deterministic counting automaton $\mathcal{A}_\alpha$ with transition specification built with formulas of $CNT(Q)$ of the form

$$\exists \overline{x}\, \mathit{diff}(\overline{x}) \wedge \bigwedge_{i\in\{1,\cdots,k\}} T(r,x_i) \wedge q_i(x_i) \wedge \left(\forall z\, \mathit{diff}(z,\overline{x}) \Rightarrow \bigvee_{q\in Q_z} q(z) \right)$$

where $\overline{x} = x_1, \cdots, x_k$, possibly empty with $k = 0$, is a we obtain, by saturation, the modal automaton $\mathcal{A}_{\widehat{\alpha}}$ with transition specification built with formulas of $MDL(Q)$ of the form

$$\exists x_1, \ldots, x_k. \bigwedge_{i\in\{1,\cdots,k\}} T(r,x_i) \wedge q_{j_i}(x_i) \wedge \forall z \bigvee_{q\in Q_z} q(z)$$

i.e. the formula obtained just by dropping the *diff* predicates. The resulting normalization of mu-calculus formulas was studied in [104, 95] where an satisfiability algorithm was proposed. Although not in the non deterministic normal form - $\mathcal{A}_{\widehat{\alpha}}$ is even not non alternating ! - , the automaton $\mathcal{A}_{\widehat{\alpha}}$ satisfies the emptiness lemma (Lemma 4.1.4). In fact, the formula $\widehat{\alpha}$ has a model if and only if the game $Sat_{\mathcal{A}_{\widehat{\alpha}}}$ is winning for the player P from the initial position.

4.3 Applications

In this section we review several applications of the counting or modal bisimulation invariance characterization theorems.

In particular, the uniform interpolation is shown to hold for the counting or modal mu-calculus. This leads to a generalization of a result obtained by D'Agostino and Hollemberg [2, 90] characterization of the modal mu-calculus by means of bisimulation quantifiers.

4.3.1 Quantified fixed point formulas

We consider here fixed point calculus extended by set quantifiers.

4.3.1.1 Definition (Quantified fixed point formulas). We defined the set of *quantified fixed point formulas* as the set built from counting (or modal) fixed point connectives - with same construction rules - and, additionally, existential (or universal) set quantifiers.

Semantics of quantified fixed point formulas is defined as for fixed point formulas by means of a translation in MSO with the extra translation rule, given any quantified fixed point formula α, any set variable X,

$$\varphi_{\exists X\alpha} \equiv \exists X \varphi_\alpha(r)$$

and

$$\varphi_{\forall X\alpha} \equiv \forall X \varphi_\alpha(r)$$

4.3.1.2 Lemma. *Quantified counting (resp. modal) formulas are strictly more expressive than counting (resp. modal) formulas.*

Proof. For the modal case, consider, the formula

$$\exists X((\Diamond X \wedge Y) \wedge (\Diamond \neg X \wedge Y))$$

It states that the root has two distinct successors in Y. This property is not invariant under bisimulation. It follows that, applying Theorem 4.2.2.1, it is not definable in the modal mu-calculus.

Observe that it is however equivalent to the counting formula $\Diamond_2 Y$.

For the counting case, consider the *diamond properties*, defined by formula

$$\forall X((\Box(\Box X)) \vee (\Box(\Box \neg X)))$$

It states that there is a single vertex at distance two from the root. This property is not counting bisimulation invariant. It follows that, applying Theorem 4.2.1.3, it is not definable in the counting mu-calculus.

□

4.3.1.3 Lemma. *Quantified counting or modal fixed point formulas are equally expressive.*

Proof. Generalizing the above example, one can check that counting modalities can be simulated by quantified non counting modalities.

□

Remark. On trees, quantified counting fixed formulas are equivalent to MSO since this is already true for the counting mu-calculus that is less expressive.

However, on graphs, the exact expressive power of quantified fixed point formulas is unknown.

4.3.2 Uniform interpolation and bisimulation quantifiers

He consider a weaker notion of set quantification that is first defined in semantical terms by means of the notion of uniform interpolant.

4.3.2.1 Definition. We say that the modal (resp. counting) mu-calculus has the uniform interpolation property, when, for every formula $\alpha(X)$ of the modal (resp. counting) mu-calculus, there is a formula $\exists_B X\alpha(X)$ of the modal mu-calculus (resp. a formula $\exists_C X\alpha(X)$ of the counting mu-calculus) called the *uniform interpolant* of α in variable X such that, for every formula β of the modal (resp. counting) mu-calculus, if

$$\models_{Pred\cup\{X\}} \beta \to \alpha(X)$$

then

$$\models_{Pred} \beta \to \exists_B X\alpha(X) \text{ (resp. } \models_{Pred} \beta \to \exists_C X\alpha(X))$$

4.3.2.2 Theorem (D'Agostino and Hollenberg [2]). *The modal or counting mu-calculus has the uniform interpolation property.*

Proof. Consider first the counting mu-calculus. Let α be a counting fixed point formula.

If we were only considering trees, $\exists X\alpha$ is the uniform interpolant of the formula α. By Theorem 4.2.1.1 and Theorem 3.2.3.1, $\exists X\alpha$ is equivalent, on trees, to a counting fixed point formula $\exists_C\alpha(X)$ that, by counting bisimulation invariance, is, in the counting mu-calculus, the expected uniform interpolant of the formula α in X.

Now, the case of the modal mu-calculus is similar. In fact, for an arbitrary infinite cardinal κ, MSO and modal mu-calculus sentences, on the class of graph κ-expansion of Σ-graphs. It follows that a similar proof can be made for the modal mu-calculus proving thus, that the modal mu-calculus formula $\exists_B X\alpha$ actually defined to be the saturation $\widehat{\exists_C X\widehat{\alpha}}$ is, in the modal mu-calculus, the uniform interpolant of the formula α in X.

□

In the sequel, quantifier $\exists_C$ (resp. $\exists_B$) are called counting bisimulation (resp. bisimulation) quantifiers. They really act like set quantifiers in the following sense.

4.3.2.3 Lemma. *For every counting (resp. modal) fixed point formula α with free variable X, every Σ-graph M, $M \models \exists_C\alpha$ (resp. $M \models \exists_B\alpha$) if and only if there exists a model M' counting bisimilar to M (resp. bisimilar to M) such that $M \models \exists X\alpha$.*

Proof. Take $M' = T(M)$ for the counting bisimulation case and $M' = M^\kappa$ for the bisimulation case.

□

One may ask what is the expressive of the language built out modalities and bisimulation (or counting bisimulation) quantifiers. It occurs that, in order to built an interesting language, one also need transitive modalities (as in PDL).

4.3.2.4 Theorem (Hollenberg [90]). *The modal (resp. counting) mu-calculus is equivalent to the language obtained from the constant predicates and closed under boolean operations, modalities (resp. counting modalities), transitive modalities, and bisimulation quantifier (resp. counting bisimulation quantifier).*

Proof. By transitive modalities, we mean modality $\Box^*$ interpreted as follows: $\Box^*\alpha \equiv \mu X.(\alpha \wedge \Box X)$.

On trees (resp. κ-expanded trees), counting bisimulation (resp. bisimulation) quantifiers have equi-expressive power. It follows that it suffices to avoid fixed points expression in the translation of basic MS formulas in the proof of Theorem 4.2.1.1.

But this is easy since the two formulas that were translated in fixed point expressions are $L \subseteq L'$ that can be defined by $\Box^*(\neg L \vee L')$, and $T(L, L')$ that can be defined by $\Box^*(\neg L \vee \Box L')$.

□

Remark. In the modal case, the above language is called BQL and is the closure of PDL by bisimulation quantifiers [90].

4.3.3 Temporal logics and mu-calculus

Another main application of the bisimulation invariance characterization is that it shows that the modal mu-calculus is (essentially) the most expressive program logic since (most) program logics are (1) bisimulation invariant and (2) translatable in MSO. Generally, points (1) and (2) are fairly easy to check while, for instance with program logic like CTL^* or worse $ECTL^*$ [49], a direct translation into mu-calculus formulas (or into alternating automata) can be difficult to provide.

4.4 References and notes

Bisimulation invariance was first considered by Van Benthem in FO [20, 19]. Invariance under unwinding was considered in the context of MSO on graph by Courcelle[45, 47]. The bisimulation invariance characterization of MSO was first established by the author in collaboration with Walukiewicz [104]. The counting bisimulation case was first stated in [99], but it follows from Walukiewicz's characterization of MSO on trees by means of non deterministic automata with counting [181, 182].

Using similar techniques, especially reusing the notion of κ-unravelings, Rabinovitch and Möller [129] extend a result of Hafer and Thomas [82] about the expressive power of CTL* on the binary tree. The study of guarded fixed point logics [79, 75, 74, 73, 76] also led the authors to a bisimualtion invariance characterization result [87, 77].

Bisimulation invariance is also considered in the context of descriptive complexity [143].

Chapter 5

More in the monadic hierarchy

In the previous chapter, we have shown that the bisimulation invariant fragment of monadic second order logic equals the mu-calculus. In this chapter, we show that this relationship is richer than expected: as announced in [99], the first levels of the bisimulation invariant fragment of the monadic quantifier alternation depth hierarchy of MSO equal, one by one, the first levels of the fixpoint alternation hierarchy of the mu-calculus [32, 11]. More precisely, Van Benthem first shows that the bisimulation invariant fragment of first order logic (FO, the 0th level of the monadic hierarchy) equals modal logic (the 0th level of the mu-calculus hierarchy). This equality holds up to the level Σ_2 of the monadic hierarchy [99] and we show that it cannot hold higher in the hierarchy.

The purpose of this chapter is to give a clear and complete proof of these correspondence theorems. They are extracted from the published presentation [99, 100, 101] of these results. Observe that, by Corollary 3.2.2.3, we already know that for every $n \in \mathbb{N}$, the mu-calculus levels NC_n and N_n are included into monadic Σ_1 (resp. the mu-calculus levels MC_n and M_n are included into monadic Π_n). The question we aim at answering in this section is thus to characterize the expressive power of the bisimulation (or counting bisimulation) invariant fragments of the various levels of the monadic quantifier alternation depth hierarchy.

5.1 Monadic Σ_1

Here, we consider the languages of trees which are closed in the topological sense. We prove that the languages of finitely branching trees accepted by modal or counting automata that are closed in the topological sense, are exactly those definable by means of (counting or modal) fixed point formulas of the nu-level.

5.1.1 Monadic Σ_1 and closed languages

More precisely, we prove the following theorem.

5.1.1.1 Theorem (J. and Lenzi [100]). *For every language L of finitely branching finite and infinite trees, the following properties are equivalent:*

1. *L is definable by an existential MSO sentence which is bisimulation (resp. counting bisimulation) invariant over graphs,*

2. *L is definable by an FO-closed existential MSO sentence which is bisimulation (resp. counting bisimulation) invariant over graphs,*
3. *L is definable in the nu-level of the modal (resp. counting) mu-calculus,*
4. *L is the projection of a locally testable tree language and L is bisimulation closed (resp. counting bisimulation closed),*
5. *L is closed in the prefix topology and recognizable by a modal (resp. counting) finite state tree automaton,*
6. *L is recognizable by a modal (resp. counting) finite state tree automaton of index zero.*

The equivalence between (1) and (6) is a non trivial logical characterization of languages of infinite trees recognizable in a naive sense: by means of finite state automata without any infinitary criterion. Observe that for finite structures such as finite words, trees or grids, recognizability by finite state automata is captured by full existential MSO [174].

FO-closed existential MSO mentioned in (2) is obtained from existential MSO by allowing arbitrary FO quantifiers to be inserted among existential set quantifiers. This fragment, considered in [4], is interesting because it is more robust and, over arbitrary graphs, strictly more expressive than existential MSO. For instance, it is closed under FO transformations. Yet, the equivalence between (1) and (2) shows that it behaves like existential MSO as far as bisimulation invariance is concerned. This result contrasts with the non equivalence observed over trees without the bisimulation invariance requirement [13].

The equivalence between (1) and (3) extends van Benthem's result on FO and modal logic [20], and refines the result obtained by Walukiewicz and the author with MSO and full mu-calculus [104].

The equivalence between (3) and (6) is a classical result (see e.g. [103, 181] for the arguments).

The equivalences between (4), (5) and (6) are easy generalization of known results in the case of the binary tree (see e.g [132, 169]). Proofs are given here for technical reasons and for completeness.

5.1.2 Recognizable closed languages

The following lemma asserts that the restriction to finitely branching trees is harmless, in the sense that recognizable languages are characterized by the finitely branching trees they contain.

5.1.2.1 Lemma (Emerson and Street [167]). *Two recognizable languages of trees are equal if and only if they contain the same set of finitely branching trees.*

The first step in the proof of Theorem 5.1.1.1 is then given by the following statement.

5.1.2.2 Theorem (J. and Lenzi [99]). *For every MSO-definable language L of finitely branching Σ-labeled trees, the following properties are equivalent:*

1. *L is closed in the prefix topology (resp. closed in the prefix topology and closed under bisimulation),*
2. *L is recognized by a counting (resp. modal) automaton of index zero.*

Proof. Let L be an MSO-definable language. Applying Theorem 3.2.2.1, there is a modal or counting (depending on whether L is bisimulation closed or not) flat automaton $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \Omega\rangle$ such that $L = L(\mathcal{A})$.

Without loss of generality, we may assume that every state $q \in Q$ is productive, i.e. for every state q there is at least one tree which is recognized from this state. We may also assume that any transition is productive, i.e. for every state q, every $a \in \Sigma$ such that $\delta(q, a) = a(r) \wedge \delta(q)$ is satisfiable, there is at least one (finitely branching) tree $T_{q,a}$ such that the root of $T_{q,a}$ is labeled by a and there exists an accepting run of $\mathcal{A}$ with initial state q instead of q_0 over the tree $T_{q,a}$. Since non-productive states or transitions cannot occur in an accepting run, all such states or transitions can be deleted from $\mathcal{A}$ without altering the accepted language $L(\mathcal{A})$.

Let then $\overline{\mathcal{A}} = \langle Q, \Sigma, q_0, 0, \delta\rangle$ be the automaton obtained from $\mathcal{A}$ just replacing the priority function Ω with the constant function 0.

To prove the equivalence, it is sufficient to prove that (over finitely branching trees) $L(\overline{\mathcal{A}})$ is the topological closure $\overline{L(\mathcal{A})}$ of $L(\mathcal{A})$.

We prove first that $L(\overline{\mathcal{A}})$ is closed. As the prefix topology can be defined by a metric, it is sufficient to show that if $\{M_n\}_{n\in\mathbb{N}}$ is a sequence of trees in $L(\overline{\mathcal{A}})$ which converges to a tree M then $M \in L(\overline{\mathcal{A}})$.

For each $n \in \mathbb{N}$, let ρ_n be an accepting run of $\overline{\mathcal{A}}$ over M_n. Considering the sequence $\{\langle M_n, \rho_n\rangle\}_{n\in\mathbb{N}}$ of $\mathcal{P}(Pred \cup Q)$-labeled trees, we know that its induced skeleton sequence converges (since $\{M_n\}_{n\in\mathbb{N}}$ converges). Applying Theorem 1.3.4.2 shows that it has a converging subsequence. The limit of that subsequence must be of the form $\langle M, \rho\rangle$ where M is the limit of $\{M_n\}_{n\in\mathbb{N}}$ and ρ is a run of $\overline{\mathcal{A}}$ over M. Now, as $\overline{\mathcal{A}}$ is of index zero, the run ρ is accepting hence $M \in L(\overline{\mathcal{A}})$.

To continue the proof, we observe that the inclusion $\overline{L(\mathcal{A})} \subseteq L(\overline{\mathcal{A}})$ is immediate as $L(\mathcal{A}) \subseteq L(\overline{\mathcal{A}})$ and $L(\overline{\mathcal{A}})$ is closed. It remains thus to show that $L(\overline{\mathcal{A}}) \subseteq \overline{L(\mathcal{A})}$.

Let now M be a finitely branching tree in $L(\overline{\mathcal{A}})$ and let ρ be an accepting run of $\overline{\mathcal{A}}$ over M. It is sufficient to show that there is a sequence $\{M_n\}_{n\in\mathbb{N}}$ of (finitely branching) trees in $L(\mathcal{A})$ which converges to M. For each $n \in \mathbb{N}$, let us define M_n as the tree obtained from the finite tree $P_n(M)$ by attaching, under each leaf v of $P_n(M)$, the tree $T_{\rho(v),\lambda(v)}$ (with a root labeled $\lambda(v)$ and accepted by the automaton $\mathcal{A}$ from the initial state $\rho(v)$).

By construction, each tree M_n belongs to $L(\mathcal{A})$ and the sequence $(M_n)_{n\in\mathbb{N}}$ converges to M which concludes the proof.

Strictly speaking, in the case where $\mathcal{A}$ is a modal automaton, we must consider in this proof runs over the κ-expansions M_n^κ of the M_ns for $\kappa = |Q|$ instead of runs on the M_ns themselves. However, this makes no difference in the argument as the κ-expansion permutes with limits.

□

This theorem gives the equivalence between (5) and (6) in Theorem 5.1.1.1. In the binary case, a very similar result is obtained by Mostowski [130].

Observe that, as a consequence of this proposition, we also have:

5.1.2.3 Corollary. *Every MSO-definable language of trees which is closed in the prefix topology is definable by means of an MS formula of the form*

$$\exists \overline{X} X_0(r) \wedge \forall x \varphi_\alpha(x, \overline{X})$$

where X_0 is one of the variables occurring in $\overline{X}$, and $\alpha(\overline{X})$ is a depth one counting formula.

Proof. Let L be an MSO-definable language of trees closed in the prefix topology. Applying Theorem 5.1.2.2, let $\mathcal{A}$ be a counting (or modal) automaton of rank 0 recognizing this language. The formula of the desired form is then obtained as follows. It expresses the existence of an accepting run of the automaton $\mathcal{A}$ with each variable X_q (one per state q) in $\overline{X}$ encoding the set of vertices labeled by state q (with X_0 encoding the initial state) and $\varphi_\alpha(x, \overline{X})$ describing the (local) transition specification.

□

Following the standard terminology [171], this corollary can be restated as follows : closed MSO definable languages of infinite trees are projection of locally testable languages of trees. Here, by locally testable, we mean languages that are defined by universally quantified local FO-formulas.

The corollary 5.1.2.3 proves that both (5) or (6) imply (4) in Theorem 5.1.1.1.

To conclude the proof of Theorem 5.1.1.1 it remains to prove that language of finitely branching trees defined by bisimulation (resp. counting bisimulation) invariant formulas of EMSO (1) or CEMSO (2) are recognizable and closed in the prefix topology (5).

In order to do so, we prove in Section 5.1.3, by applying Łos Theorem to existential second-order logic (ESO), that classes of graphs definable in ESO are closed under ultraproduct. We also prove that the ultraproduct of any converging sequence of finitely branching trees is counting bisimilar with its limit. And in Section 5.1.4 we apply this result to EMSO and CEMSO as both are fragments of ESO.

5.1.3 Ultraproducts

Let I be a set. An *ultrafilter* over I is a set $U \subseteq \mathcal{P}(I)$ of subsets of I such that $I \in U$, $\emptyset \notin U$, and U is closed under the following rules: for every A and $B \subseteq I$, if $A \in U$ and $A \subseteq B$ then $B \in U$; if A and $B \in U$ then $A \cap B$ in U; and either $A \in U$ or $I \setminus A \in U$. An ultrafilter U is *principal* if it contains a finite set, and non-principal otherwise. Observe that a non-principal ultrafilter over I contains all co-finite subsets of I.

With the help of the axiom of choice (or the Zorn Lemma) one can prove [41] that if I is an infinite set then there is a non-principal ultrafilter over I.

Assuming I is an infinite set, let U be an ultrafilter over I, and let $\{M_i\}_{i\in I}$ be an I-indexed collection of FO-structures over some relational vocabulary τ. The ultraproduct $\Pi_i^U M_i$ of $\{M_i\}_{i\in I}$ modulo U is defined as the quotient of the product structure $\Pi_i M_i$ under the congruence $\simeq_U$ defined, for every u and $v \in dom(\Pi_i M_i)$ by $u \simeq_U v$ when the set $\{i \in I : u_i = v_i\}$ belongs to U. This construction is motivated by the following theorem.

5.1.3.1 Theorem (Łos). *For every FO sentence φ over the vocabulary τ, $\Pi_i^U M_i$ is a model of φ if and only if $\{i \in I : M_i \models \varphi\}$ belongs to U.*

As this holds for an arbitrary vocabulary τ, it leads to the following corollary.

5.1.3.2 Corollary. *For every formula φ of existential second-order logic on the vocabulary τ, if $\{i \in I : M_i \models \varphi\} \in U$ then $\Pi_i^U M_i \models \varphi$.*

Proof. By standard syntactic arguments, we can always assume φ is of the form $\exists R \psi(R)$ with $\psi(R)$ a FO formula over the vocabulary $\tau \cup \{R\}$. For each $i \in I$, let R_i be any interpretation of R over $dom(M_i)$ such that $M_i \models \psi(R_i)$ if and only if $M_i \models \exists R \psi(R)$.

Assuming that $\{i \in I : M_i \models \exists R\psi\}$ belongs to U and considering ψ (resp. $\{\langle M_i, R_i\rangle\}_{i\in I}$) as a FO sentence (resp. an indexed collection of FO-structures) over the vocabulary $\tau \cup \{R\}$, Łos theorem can be applied to show that the ultraproduct $\Pi_i^U \langle M_i, R_i\rangle$ satisfies $\psi(R)$. It follows that, given R_U the congruence closure of $\Pi_i R_i$, by definition of an ultraproduct, $\Pi_i^U M_i \models \psi(R_U)$ and hence $\Pi_i^U M_i \models \exists R\psi(R)$.

□

5.1.3.3 Lemma (J. and Lenzi [100]). *Let $\{M_n\}_{n\in\mathbb{N}}$ be a sequence of finitely branching Σ-labeled trees. Assume that $\{M_n\}_{n\in\mathbb{N}}$ converges to a limit $M \in FBT(Pred)$. Let U be a non-principal ultrafilter over $\mathbb{N}$. The two structures M and $Reach(\Pi_n^U M_n)$ are isomorphic.*

Proof. Let h be a strictly positive integer. We show that $P_h(M)$ and $P_h(\Pi_n^U M_n)$ are isomorphic.

Since M is the limit of $\{M_n\}_{n\in\mathbb{N}}$, there is a number n_h such that, for every $n \geq n_h$, $P_h(M_n)$ and $P_h(M)$ are isomorphic. As $P_h(M)$ is finite, there is also a FO formula φ_h such that, given any model N, $N \models \varphi_h$ if and only if $P_h(N)$ is isomorphic with $P_h(M)$. Then, as U is non-principal, the co-finite set $\{n \in \mathbb{N} : M_n \models \varphi_h\}$ belongs to U. By applying Łos' Theorem, we get $\Pi_n^U M_n \models \varphi_h$, and hence $P_h(\Pi_n^U M_n) \models \varphi_h$, so it is isomorphic to $P_h(M)$.

Since this holds for arbitrary $h > 0$, this implies in particular that $Reach(\Pi_n^U M_n) = \bigcup_h P_h(\Pi_n^U M_n)$ is finitely branching and thus, the result now follows from Lemma 1.3.4.1.

□

5.1.4 Applications to bisimulation invariance

As EMSO and CEMSO are both fragments of ESO, we have:

5.1.4.1 Lemma (J. and Lenzi [100]). *Let L be a language of finitely branching trees definable by a bisimulation or counting bisimulation invariant EMSO or CEMSO sentence. Then L is both recognizable and topologically closed.*

Proof. As bisimulation invariance implies counting bisimulation invariance, we only need to prove this Lemma for counting bisimulation invariant sentences.

Let φ be a EMSO or CEMSO counting bisimulation invariant sentence. Let L be the class of (finitely branching) trees that satisfy φ. Since both EMSO and CEMSO are fragments of MSO, L is recognizable by Theorem 3.2.2.1.

Now, let $\{M_n\}_{n\in\mathbb{N}}$ be a sequences of finitely branching trees in L that converges towards a finitely branching tree M. In order to conclude the proof, we have to show that $M \in L$.

In order to do so, let U be a non principal ultrafilter over $\mathbb{N}$, and let N be the ultraproduct $\Pi_n^U M_n$. By Corollary 5.1.3.2 the class of models of φ is closed under ultraproduct so N satisfies φ. By Lemma 5.1.3.3, we also have that the limit M of limit of $\{M_n\}_{n\in\mathbb{N}}$ is isomorphic to $Reach(N)$ hence counting bisimilar to N. Now, since φ is counting bisimulation invariant, this shows that $M \models \varphi$, that is, $M \in L$.

□

In other words, Lemma 5.1.4.1 proves that both (1) or (2) imply (5) in Theorem 5.1.1.1. As the implications from (4) to (1) and (1) to (2) are immediate for syntactic reasons, this concludes the proof of our main Theorem.

5.2 Monadic Σ_2

In this section, we show that the bisimulation invariant fragment of the monadic Σ_2 level of the monadic hierarchy equals the $\nu\mu$-level of the mu-calculus hierarchy also known as the Büchi level.

We shall also mention that, in the binary tree, the main result presented in this report has already been announced by Lenzi [115] with quite a long and technical proof argument. Later a quite simpler argument, but for a slightly weaker result, has been given by Skurczynski [165]. This last result is weaker since it handles monadic Σ_2 formulas over the binary tree with FO kernels (called principal formulas) that are weaker than arbitrary FO-formulas as we are using here.

Still, in the following generalization of the binary case, we adopt several arguments from Skurczynski work and, quite distinctly, we handle FO-formulas by means of topological considerations obtaining thus a simple though presumably unknown yet automata theoretic bound on the expressive power of FO logic on trees.

At last, one shall observe that the monadic Σ_2 case does not follow (say by simple inductive argument) from the former characterization of the bisimulation invariant fragment of monadic Σ_1 [100] and section 5.1. In fact, the monadic Σ_1 kernel of a bisimulation invariant monadic Σ_2-formula is not necessarily invariant. And, over trees, the bisimulation invariant fragment of monadic Σ_1 is strictly weaker than full monadic Σ_1 as illustrated, for instance, by the formula $\exists x p(x)$ that is not bisimulation invariant.

5.2.1 Monadic Σ_2 and Büchi languages

5.2.1.1 Theorem (J. and Lenzi [101]). *The bisimulation (resp. counting bisimulation) invariant fragment of the level Σ_2 of the monadic hierarchy equals the $\nu\mu$-level of the modal (resp. counting) mu-calculus hierarchy.*

Proof. This results is proved through the remainder of this section.

□

Observe that the $\nu\mu$-level N_2 (or NC_2) of the modal (or counting) mu-calculus is known, in various settings, to be as expressive as (modal) tree automata with Büchi conditions [139, 103].

This result refines Rabin's own logical characterization of Büchi definable properties of the binary tree as projections of weak MSO properties [153].

As a side result, we also prove and use the fact that the languages of trees definable by first-order formulas (FO-formulas) are boolean combination of topologically closed definable languages, i.e. boolean combination of languages of the μ-level of the mu-calculus hierarchy [100].

The proof goes as follows. We successively show that every FO-formula over trees is equivalent to a weak non deterministic automaton. By projection, this shows that this holds as well for monadic Σ_1 formulas. Then, by complementation and simulation à la Muller and Schupp, we prove that every monadic Π_1 formula is equivalent to a Büchi automaton and thus, by projection again, every monadic Σ_2 formula as well. Then, the counter-saturation technique that has been developed in [104] (see also Lemma 4.2.2.3) can be applied in order to conclude the proof.

5.2.2 Weak alternating automata

Weak automata are obtained from Büchi automata by restricting the structure of automata. They play a fundamental role in the analysis of the expressive power of FO and monadic Σ_1 on trees. They have been used as a characterization of weak MSO (or the alternation free mu-calculus) on trees [135, 15, 130, 111].

5.2.2.1 Definition. A Büchi alternating automaton $\mathcal{A} = \langle Q, \Sigma_1 \times \Sigma_2, q_0, \delta, \Omega \rangle$ is called *weak automaton* if there is a partially ordered set I and a partition of Q, $Q = \bigcup_{i \in I} Q_i$, such that, given $F = \Omega^{-1}(0)$, F is a union of some of the Q_i's, and for every $q \in Q_i$, $\delta(q) \in FOL^+(\bigcup_{j \leq i} Q_j)$: this means that along plays, the index decreases, or remains the same.

The sets Q_i included in F are called the accepting components of the automaton. The other Q_i's are called rejecting components.

5.2.2.2 Lemma. *The tree languages definable by weak alternating automata are closed under complementation.*

Proof. Let $\mathcal{A} = \langle Q = \bigcup_{i \in I} Q_i, \Sigma, q_0, F, \delta \rangle$ be a weak alternating automaton. We define the automaton $\mathcal{B}$ to be the automaton

$$\mathcal{B} = \langle Q = \bigcup_{i \in I} Q_i, \Sigma, q_0, Q \setminus F, \delta^d \rangle$$

where $\delta^d(q)$ is the dual formula of $\delta(q)$: $\delta^d(q) = \neg\delta(q)[\neg q/q : q \in Q]$.

By Lemma 3.1.3.3, the automaton $\mathcal{B}$ recognizes the complement of the language recognized by the automaton $\mathcal{A}$. It remains to show that the automaton $\mathcal{B}$ is a weak automaton. In general, the winning condition of the dual of a Büchi automaton shall be a co-Büchi condition, i.e. the dual condition of the Büchi condition. However, the weakness assumption makes Büchi and co-Büchi condition equivalent.

More precisely, in the model-checking game, a play is winning for the Marker with the Büchi condition when infinitely many accepting states occur. By duality, a play is winning for the Marker with the co-Büchi condition when finitely many non accepting states occur. In general, this makes that Büchi automata are not closed under complement. However, under the assumption of weakness, there cannot be an infinite alternation of accepting and rejecting states in the model-checking games, so Büchi or co-Büchi criteria *are* equivalent.

We conclude the proof by applying Lemma 3.1.3.3.

□

5.2.2.3 Lemma. *The tree languages definable by weak non deterministic automata are closed under union and intersection.*

Proof. Here we cannot first state that alternating weak automata are closed under boolean connectives and then apply simulation theorem (Theorem 4.1.3.1) since the simulating non deterministic automata will not be, in general, weak automata.

The idea is to show closure under union (or intersection) by means of an explicit parallel product construction.

More precisely, given two weak automata $\mathcal{A} = \langle Q = \bigcup_{i \in I} Q_i, \Sigma, q_0, F, \delta \rangle$ and $\mathcal{B} = \langle Q' = \bigcup_{j \in J} Q'_j, \Sigma, q'_0, F', \delta' \rangle$. we define the automaton $\mathcal{C} = \langle Q \times Q', \Sigma, (q_0, q'_0), F'', \delta'' \rangle$

where $F'' = F \times Q' \cup Q \times F'$ for the union (resp. $F'' = F \times F'$ for the intersection), and, given π_1 and π_2 the projections of the Cartesian product $Q \times Q'$, the transition function defined in such a way that, for each $(q, q') \in Q \times Q'$, each $a \in \Sigma$, a marking m satisfies $\delta''((q, q'), \sigma)$ if the projected marking $\pi_1(m)$ satisfies $\delta(q, \sigma)$ and the projected marking $\pi_2(m)$ satisfies $\delta'(q', \sigma)$.

The automaton $\mathcal{C}$ does functionally recognize the union (resp. the intersection) of the languages functionally recognized by $\mathcal{B}$ and $\mathcal{B}$. Moreover, it is a weak automaton with $Q'' = \bigcup_{(i,j)\in I\times J} Q_i \times Q'_j$ with $I \times J$ ordered with the product order.

□

5.2.2.4 Lemma. *The tree languages definable by weak non deterministic automata are closed under projection.*

Proof. The construction given in the proof of Lemma 4.2.1.2 for non deterministic automata just applies similarly to weak non deterministic automata.

□

5.2.2.5 Theorem (Muller et al. [137, 103, 94, 111]). *Any alternating weak (resp. closed or open) automaton $\mathcal{A}$ is equivalent to a non deterministic Büchi (resp. non deterministic closed or open) automaton $\mathcal{A}_n$.*

Proof. The proof follows form Theorem 4.1.3.1 since weak automata are particular cases of Büchi automata.

□

Remark. Over binary trees, this result follows from Muller and Schupp's construction [137]. For the modal mu-calculus, it follows from the construction given in [103]. It relies on standard techniques presented in a quite general setting in [94, 16]. It has also been proved again in the modal case with slightly different techniques in [111].

5.2.3 Proof of the monadic Σ_2 case

In this section, we prove Theorem 5.2.1.1 by a series of lemmas that easily follows from the results presented in the previous sections.

5.2.3.1 Lemma (From FO to non deterministic weak). *On arbitrary trees, every first order formula is equivalent to a non deterministic weak automaton.*

Proof. In order to do so, applying Gaifman theorem, we first show that FO-formulas define languages of trees that are finite boolean closure of closed languages in the sense of the prefix topology. Then, in turn, classical results of automata theory ensure that these languages are definable by means of non deterministic weak automata.

Let d be a positive integer. A FO-formula $\varphi(x)$ with a single free variable x is called *basic d-local* when all quantifications in $\varphi(x)$ are relativized to vertices at distance at most d from x, i.e. vertices reachable from x by a undirected path of length at most d.

5.2.3.2 Theorem (Gaifman [64]). *Let φ be a FO-formula on trees. There exist $d \geq 0$ such that φ is equivalent to a finite boolean combination of formulas of the form*

$$\exists x_1 \cdots x_n \bigwedge_{i\neq j} d(x_i, x_j) > d \wedge \varphi_i(x_i)$$

where $\varphi_1(x)$, ..., $\varphi_n(x)$ are basic d-local formulas and $d(x_i, x_j) > d$ means that there is no undirected path between x_i and x_j of length smaller than or equal to d.

Then we have:

5.2.3.3 Corollary (J. and Lenzi [101]). *Every FO-definable language of tree is a finite boolean combination of closed languages.*

Proof. The negation of a formula φ of the form

$$\varphi \equiv \exists x_1 \cdots x_n \bigwedge_{i \neq j} d(x_i, x_j) > d \wedge \varphi_i(x_i)$$

defines a closed language. In fact, assume there is a sequence of trees $\{T_n\}_{n\in\omega}$ that converges towards a limit T. If $T_n \models \neg\varphi$ for every $n \in \omega$ then $T \models \neg\varphi$. Otherwise, if $T \models \varphi$ there is a finite depth h such that the satisfiability of φ is witnessed by vertices that belong to the h-prefix of T. Since the sequence $\{T_n\}_{n\in\omega}$ converges towards T, there is also an N such that for every $n \geq N$, T_n and T have isomorphic h-prefix hence $T_n \models \varphi$ which contradicts the hypothesis.

□

Now, we have:

5.2.3.4 Lemma. *FO-definable closed languages are definable by means of finite closed non deterministic automata. And, similarly, FO-definable open languages are definable by finite open non deterministic automata.*

Proof. The case of closed languages is proved in [100] and section 5.1. By complementation lemma (see Lemma 5.2.2.2), this also shows that FO-definable open languages are recognizable by means of open alternating automata. But then, the Simulation Theorem 5.2.2.5 shows that these languages are then recognizable by means of open non deterministic automata.

□

Since closed and open non deterministic automata are special case of non deterministic weak automata this, by applying also Lemma 5.2.2.3, concludes the proof of Lemma 5.2.3.1.
□

Remark. We could have given an explicit construction of such an automaton. In fact, from Gaifman normal formal form, it is quite an easy exercise. Remind however that the satisfiability problem for a FO-formula on tree is non elementary. Hence, the automaton translation of a FO-formula is also non elementary.

Then we prove:

5.2.3.5 Lemma (From Π_1 to weak). *On arbitrary trees, every monadic Π_1 formula is equivalent to a weak automaton.*

Proof. Since weak non deterministic automata are closed under projection
(Lemma 5.2.2.4) we know, by applying Lemma 5.2.3.1 that languages definable by monadic Σ_1 formulas are recognizable by means of alternating weak automata. Then, applying Lemma 5.2.2.2 we conclude the proof.

□

Now we have:

5.2.3.6 Lemma (From Σ_2 to Büchi). *On arbitrary trees, every monadic Σ_2 formula is equivalent to a Büchi automaton.*

Proof. By applying Lemma 5.2.3.5, every Π_1 formula is equivalent to a weak alternating automaton. So, by applying Simulation Theorem 5.2.2.5, it is also equivalent to a non deterministic Büchi automaton. Now, closure under projection (Lemma 4.2.1.2) concludes the proof.

□

We prove then the analogous of Theorem 5.2.1.1 for counting bisimulation.

5.2.3.7 Theorem (J. and Lenzi [101]). *The counting bisimulation invariant fragment of monadic Σ_2 equals the $\nu\mu$-level of the counting mu-calculus.*

Proof. Since any $\nu\mu$-formula of the counting mu-calculus is equivalent to a (counting bisimulation invariant) monadic Σ_2 formula, it is sufficient to prove the converse.

Let φ be a monadic Σ_2 formula counting bisimulation invariant.

First, by 5.2.3.6, we know that, over trees, the formula φ is equivalent to a Büchi automaton. By applying Theorem 3.2.3.1 it is thus equivalent, over trees, to a $\nu\mu$-formula α of the counting mu-calculus.

Since both φ and α are counting bisimulation invariant, by Lemma 3.2.4.1, we conclude that they are equivalent on arbitrary models.

□

For bisimulation invariance, we have:

5.2.3.8 Lemma (Saturation). *For arbitrary infinite cardinal κ, on κ-expansions of trees (resp. on trees), every monadic Σ_2 formula is equivalent to a modal Büchi automaton (resp. a counting Büchi automaton).*

Proof. Let φ be a monadic Σ_2 formula. By Lemma 5.2.3.6, over κ-expansions, the formula φ is equivalent to a Büchi automaton. But, following Lemma 4.2.2.3, on κ-expansions, Büchi automata are equivalent to modal Büchi automata.

□

Proof Theorem 5.2.1.1 (end). This concludes the proof of Theorem 5.2.1.1. In fact, every modal (resp. counting) $\nu\mu$-formula is equivalent to a bisimulation invariant (resp. counting bisimulation) monadic Σ_2 formula, it is sufficient to prove the converse.

Let φ be a monadic Σ_2 formula bisimulation invariant.

First, by Lemma 5.2.3.8, we know that, over κ-expansions of trees, the formula φ is equivalent to a modal Büchi automaton. By applying Theorem 3.2.3.1 it is thus equivalent, over trees, to a $\nu\mu$-formula α of the modal mu-calculus. Then, by Lemma 3.2.4.2 they are equivalent on arbitrary models. The counting bisimulation invariant case is analogous.

□

5.3 Beyond monadic Σ_2

After the two last characterization of the bisimulation invariant fragment levels of the monadic hierarchy, one may ask if their is a similar correspondence for higher levels.

5.3.1 Bisimulation invariance in monadic Σ_3

5.3.1.1 Theorem (J. and Lenzi [99]). *For each integer $k > 2$ there exists a bisimulation invariant formula of monadic Σ_3 that does not belong to the kth level of the mu-calculus hierarchy.*

Proof. From [32, 33], we know that, given an integer k, expressing the fact that a position in an arbitrary parity game with sets of parity indices $[0, k]$ is winning for the player P cannot be done with any mu-calculus formula of the level N_k. From [11], we know that this is still the case restricted to games of degree two.

Observe that in monadic second order logic, this set of winning positions may also be difficult to define. In fact, it somehow requires, at least implicitly, to check the existence of a (positional) strategy for the player P which is winning for every plays starting in the distinguished position. And a positional winning strategy is defined are a peculiar set of edges. In general, edge set quantification is not even definable in MSO.

Still we prove Theorem 5.3.1.1 by encoding any binary game G with priorities $[0, k-1]$ into a bisimulation closed class of game graphs $\mathcal{C}_G$ (on a more complex signature), called $\{l, r\}$-games, in such a way that the initial position in G is winning for player P if and only if the initial position of any graph $G' \in \mathcal{C}_G$ is winning for the player P.

More precisely:

5.3.1.2 Definition ($\{l, r\}$-games). Given $Pred_k$ defined by $Pred_k = \{p_l, p_r, p_0, \cdots, p_k\}$, the class of $\{l, r\}$-games graphs with k-priorities is defined to be the class of $\mathcal{P}(Pred_k)$-graph M such that:

1. $\{p_l^M, p_r^M\}$ is a partition of the set of vertices reachable from r^M,
2. $\{p_0^M, \cdots, p_k^M\}$ is also a partition the set of vertices reachable from r^M.

The game, from the initial position r^M, is played by the player P and E as follows: from any vertex u reachable from the root,

1. the player P chooses $x \in \{l, r\}$,
2. the player E answer by choosing some $v \in Succ^M(u)$ such that $x \in \lambda^M(v)$,

and the play go on from the new position v.

An infinite play is thus a infinite path and parity conditions, encoded by the disjoint predicates $p_0, \ldots, p_k$, applies to define the winner.

Observe that the class of $\{l, r\}$-game with $k + 1$ priorities is bisimulation closed and definable in N_1.

Moreover:

5.3.1.3 Lemma. *The class of $\{l, r\}$-game with $k + 1$-priorities, such that the initial position is winning for the player P, is definable in (the bisimulation invariant fragment of) monadic Σ_3.*

Proof. Since $\{l, r\}$-games are (encoding of) parity games, one may only check the existence of positional winning strategy. Now, for this, it suffice to select, by means of an existentially quantified set X, all vertex from which the player P plays, say, l. Checking then that the resulting strategy is winning amount to check that the minimal parity condition on any

cycle reachable from the root when the player P follows the strategy given by set X is even.

It shall be clear that this can be defined by means of a monadic Π_2 formula. More precisely, we show that the converse property, stating that the strategy allows a non accepting path, can be defined by the following monadic Σ_2 property.

In fact, it amount to check that there is a singleton $P_x = \{x\}$ with odd priority $2n+1 \in [0,k]$ such that the following property holds

$$\mu Y.\Diamond_X Y \vee (P_x \wedge \Diamond_X \mu Z.P_{k\geq 2n+1} \wedge (\Diamond Z \vee P_x))$$

with

$$\Diamond_X Y \equiv (X \to (\Diamond Y \wedge P_l)) \vee (\neg X \to (\Diamond Y \wedge P_r))$$

and $P_{k\geq 2n+1} = \bigvee_{k\geq 2n+1} P_k$. More precisely, this N_1 definable property check that there is a path computable with the player P strategy defined by set X, from the source r^M to x, and then there is a cycle from x to x, again compatible with the player P strategy, that only contains vertices of priority higher or equal than $2n+1$.

□

Observe now that any binary game G with priorities $[0,k]$ can be encoded into an equivalent $\{l,r\}$-game G', with two successors per vertex, one labeled by l, the other label by r. Moreover, solving the game G equivalently amounts to solve the game G' or even any other $\{l,r\}$-game G'' in the bisimulation class $\mathcal{C}_G$ of G'.

Now, any fixed-point formula that define the class of winning $\{l,r\}$-games defines, when restricted to these encodings, the class of (encoding of) winning binary games. Since this formula can easily be translated into a fixed point formula with same complexity that solve binary games, Arnold's result [11] applies, telling us that this formula cannot belong N_k.

□

In other words, no other equivalence similarly relates levels of the mu-calculus hierarchy with levels of the monadic hierarchy.

5.3.2 The monadic complexity of the mu-calculus

The question whether the mu-calculus is equivalent to the bisimulation invariant fragment of monadic Σ_k, for some integer $k > 2$, remains, strictly speaking, open. However, the following theorem, which is a consequence of the work of Courcelle [46] who shows that, on a quite general class of graphs, this is already true with monadic Σ_3.

5.3.2.1 Theorem (J. and Lenzi [99]). *Over the class of graphs of bounded degree (or bounded tree-width) every mu-calculus formula can be translated into a monadic Σ_3 formulas.*

Proof. The proof of Theorem 5.3.2.1 is also almost done. Indeed, from the proof of previous lemmas it is clear that *with one existential quantification over sets of edges* the winning position for the player P can be expressed as a monadic Σ_3 unary predicate. But it also follows from Lemma 3.2.2.1 that checking a fixpoint formula α on a graph M can be done via checking the existence of a winning strategy in $\mathcal{G}(\mathcal{A}_\alpha, M)$. This model-checking game can be defined, by means of monadic Σ_1 transduction[43], on graph M itself. And, if the input graph is of bounded degree (or bounded tree-width) then the resulting parity game is also of bounded degree (or bounded tree-with). Now Courcelle shows that over

graphs with bounded degree (or tree-width) quantification over edges can be "simulated" by quantifications over vertices via, again, a monadic Σ_1 transduction. Altogether, this says that over graphs of bounded degree (or bounded tree-width) mu-calculus formulas can be translated into monadic Σ_3 formulas. This concludes the proof.

□

Chapter 6

The finite model case

In 1974 R. Fagin proves that the properties of structures which are in NP are exactly the same as those expressible by existential second order sentences, known also as Σ_1^1 sentences, i.e. sentences of the form: *there exist relations R such that φ*, where R is a relation symbol (possibly of high arity) and φ is a first order formula. This result gives birth to an all research area : *finite model theory*.

In this research field, monadic Σ_1 on finite graphs (also called monadic NP in this context) is also studied. The reason is the belief that it can serve as a training ground for attacking the "real problems" like whether NP equals co-NP. In fact, some NP-complete decision problem on graphs are definable in monadic Σ_1. And it is not hard to show [61] that monadic NP is different from monadic co-NP. A much stronger result has even been proved by Matz and Thomas [125]. They show that the monadic hierarchy, the natural monadic counterpart of the polynomial hierarchy, is strict (a property is in the k-th level of the monadic hierarchy if it is expressible by a sentence of monadic second order logic where all the second order quantifiers are at the beginning and there are at most $k-1$ alternations between second order existential and second order universal quantifiers).

In this chapter, we are interested in studying the bisimulation invariant fragment of MSO *on finite graphs*. Though we do not obtain strong characterization as in the case of arbitrary graphs, we still achieved several results that show that the situation on finite graphs is radically different from the former situation.

We show that, on finite representation of words, the bisimulation invariant fragment of monadic Σ_1 equals the bisimulation invariant fragment of full monadic second order logic and capture regular (words) languages. This work is a detailed version of a joint work with Anuj Dawar [50]. This characterization extends to a counter-example of the property that, in the finite, the bisimulation invariant fragment of monadic Σ_1 would be equal to the first level (NC_1) of the counting mu-calculus, a result established for arbitrary finite and infinite graphs in section 5.1.

Studying then the notion of tiling systems or, rather graph acceptors, that generalizes the notion of automata on finite strings or trees [48, 173], we established various properties for these devices when they recognize bisimulation closed classes of graphs. This is extracted from a joint work with Anuj Dawar [50].

Last, we investigate some graph transformation that do provide separation results in the boolean or FO-closure of the monadic alternation depth hierarchy. This is extracted

from a joint work with Jerzy Marcinkowski [102].

6.1 On finite representations of words

In mathematical logic, an infinite word (an ω-word) is often encoded as a labeling of the positive integers; the ith letter of the word being defined as the color of integer $i-1$ in the encoding.

Now, everybody understands that if an infinite word is ultimately periodic (of the form $u.v^\omega$), then it can be encoded into a *finite graph*: a finite path (encoding prefix u) followed by a cycle (encoding period v). As a consequence, any property of (ultimately periodic) words can be rephrased and verified on the finite encodings of these words (called lassos in the sequel).

Observe that, as opposed to the infinite case, finite representation of eventually periodic words do have a distinguished vertex, the knot of the lasso, that can be used as sort of a pivot in order to express words properties. The purpose of this section is to study the related increase of expressiveness in logic.

6.1.1 MSO on finite encodings of words

Here, we study monadic second order logic on finite representation of ultimately periodic words.

6.1.1.1 Definition. Given a language of infinite words $L \subseteq \Sigma^\omega$, let define the *kernel* of language L to be to be the set of all ultimately periodic words of L,

$$kern(L) = \{u.v^\omega : u, v \in \Sigma^+, u.v^\omega \in L\}$$

Our main result is:

6.1.1.2 Theorem (Dawar and J. [50]). *A bisimulation closed class of finite unary graphs C is definable by a MSO-sentence φ if and only if there is an ω-regular languages $L \subseteq \Sigma^\omega$ such that $L_C = kern(L)$.*

Moreover, such a formula φ can always be choose in the monadic Σ_1 level of the monadic second order logic hierarchy.

In terms of descriptive complexity:

6.1.1.3 Corollary. *Restricted to bisimulation invariant properties on finite unary graphs, the monadic quantifier alternation depth hierarchy collapses to the level monadic Σ_1.*

We first show that for every regular languages of infinite words, there is an existential monadic formula that defines, in the finite, the set of all graph encodings of the ultimately periodic words of this language.

6.1.1.4 Theorem (Dawar and J. [50]). *For every regular ω-language $L \subseteq \Sigma^\omega$ there is a monadic Σ_1 formula φ_L such that for every finite unary graph M, $M \models \varphi_L$ if and only if $w_M \in L$.*

Proof. Let L be an ω-regular language.

Observe first, that there is a nondeterministic finite Büchi automaton

$$\mathcal{A}_L = \langle Q, q_0, \delta, F \rangle$$

that recognizes L and such that, for every infinite word of L of the form $u.v^\omega$, there is an accepting state $q \in F$ such that, there is a path in $\mathcal{A}_L$ from q_0 to q reading u, and a cycle in $\mathcal{A}_L$ from q to q reading v. In fact, following [146], such an automaton can be taken as the Büchi automaton one canonically build out of an ω-semigroup recognizing L.

Now, the formula φ_L can be defined as follows: there is a collection of disjoint sets X_q, one per state $q \in Q$, such that: (i) $r \in X_{q_0}$; (ii) for each q, for each vertex $x \in X_q$, x has a single successor y and there is a state $q' \in \delta(\lambda(x))$ such that $y \in X_{q'}$ with $\lambda(x) = \{p \in Pred : p(x)\text{holds}\}$; and (iii) any element with two predecessors in the union of the X_qs belongs to some X_q with $q \in F$.

The formula φ_L defined in such a way (1) does check that there is a unique path from the root, (2) necessarily defines on this path a labeling that is a run of the automaton $\mathcal{A}_L$, and (3), *since M is finite*, there is a vertex on this path with two predecessors labeled by an accepting state that ensures this run is accepting.

□

One may ask whether a converse of this theorem holds. More precisely, given an MSO-formula φ, let L_φ be the language of all infinite words encoded by the unary finite models of φ, i.e.

$$L_\varphi = \{w_M \in \Sigma^\omega : M \models \varphi, M \text{ unary and finite}\}$$

One may ask, for instance, if there is some ω-regular language $L \subseteq \Sigma^\omega$ such that $L_\varphi = kern(L)$; recall that, by construction, all words in L_φ are ultimately periodic !

The answer is no; there are MSO-sentence φ on unary graphs such that L_φ is not the kernel of an ω-regular language. In fact, on the alphabet $\Sigma = \{a, b\}$, take the formula φ that defines unary graphs such that, on the unique path from the root, there is a single b on the cycle. One has $L_\varphi = \{a^m.(b.a^n)^\omega \in \Sigma^\omega : m \in \mathbb{N}, n \in \mathbb{N}\}$ but, by pumping lemma, for every ω-regular language L such that $L_\varphi \subseteq L$, there are some integers m, n and $p > n$ such that $a^m.(b.a^n.b.a^p)^\omega \in L$ henceforth $L_\varphi \neq kern(L)$.

We shall prove below that, provided the formula φ is counting bisimulation invariant in the finite, the answer to this question becomes positive.

6.1.2 The monadic second order theory of lassos

The remaining of the proof of Theorem 6.1.1.2 is presented here. It goes through the study, and characterization, of monadic second order logic on unary graphs that are canonical coding of ultimately periodic words. These graphs are called lassos.

6.1.2.1 Definition (Lassos). A unary graph M is called a *lasso* when the root of M has no predecessor and every other vertex but a single one (called the knot) has a single predecessor while the knot vertex has two predecessors.

Observe that any finite unary graph if bisimilar to a lasso. Moreover, any lasso M is completely characterized by the two non empty finite words u and respectively v (in the alphabet Σ) that are described by the (acyclic) path emanating from the root to the knot of M (excluding the knot) and respectively by the cyclic path emanating from the knot

to itself (excluding it when returning to it). Writing $M_{u,v}$ for such a lasso we do have the following characterization of MSO on lassos. This characterization follows from the Decomposition Theorem proved for MSO in [122].

6.1.2.2 Theorem (Shelah [163], Makowski and Rave [122]). *For every monadic formula φ, there exists a finite set of pairs of regular languages $\{(U_i, V_i) \subseteq \Sigma^+ \times \Sigma^+\}_{i\in I}$ such that:*

$$M_{u,v} \models \varphi \text{ iff } (u,v) \in \bigcup_{i\in I} U_i \times V_i \quad ((u,v) \in \Sigma^+ \times \Sigma^+)$$

Proof. The mapping that maps every pair of non empty finite words $(u,v) \in \Sigma^+ \times \Sigma^+$ to the lasso $M_{u,v}$ is a FO-definable transduction. It follows, by Decomposition Theorem [122] that there is a finite set of pairs of MSO-formulas $\{(\varphi_i, \psi_i)\}_{i\in I}$ over finite Σ-words such that for every two words u and $v \in \Sigma^+$,

$$K_{u,v} \models \varphi$$

if and only if there is some $i \in I$ such that $u \models \varphi_i$ and $v \models \psi_i$. By applying Büchi theorem, for every $i \in I$, the MSO-formulas φ_i and ψ_i do define regular languages U_i and V_i. This conclude the proof of the theorem.

□

Remark. In particular, $L_\varphi \subseteq kern(\bigcup_{i\in I} U_i.(V_i)^\omega)$, but the inclusion may be strict.

In fact, let φ be the formula that check, on the two letter alphabet $\Sigma = \{a,b\}$, that cycles of lassos are uniformly colored. We do have $L_\varphi = (a+b)^*(a^\omega + b^\omega)$. But φ is also characterized by the languages $U = (a+b)^+$ and $V = a^+ + b^+$ with $U.V^\omega = (a+b)^\omega$.

On a more conceptual point of view, the inclusion may be strict for the following reason.

6.1.2.3 Lemma. *Let U and V be two languages of finite non empty words and let $w \in U.V^\omega$. Word w is ultimately periodic if and only if there is $u_0 \in U$ and $v_1, \ldots, v_m, v_{m+1}, \ldots, v_{m+n} \in V$ such that*

$$w = u_0.v_1.\cdots.v_m.(v_{m+1}.\cdots.v_{m+n})^\omega$$

i.e. $w = u.v^\omega$ with $u \in U.V^+$ and $v \in V^+$.

In particular, given some $i \in I$, $u \in U_i.V_i^+$ and $v \in V_i^+$, nothing ensures that $M_{u,v} \models \varphi$ (equivalently $u.v^\omega \in L_\varphi$).

So far, looking for the converse of theorem 6.1.1.4, we haven't considered invariance under counting bisimulation.

6.1.2.4 Theorem (Dawar and J. [50]). *For every MSO-formula φ, counting bisimulation-invariant on finite graphs and true only on unary graphs, there exists a finite set of pairs of regular languages $(D_t, E_t) \subseteq \Sigma^+ \times \Sigma^+$ such that, for every lassos $M_{u,v}$:*

$$M_{u,v} \models \varphi \text{ iff } \exists r \in D_t.E_t^+, \exists s \in E_t^+ \text{ such that } u.v^\omega = r.s^\omega$$

Proof. Let φ be a formula as above and let $(U_i, V_i)_{i\in I}$ be the regular languages obtained by applying Theorem 6.1.2.2.

.

6.1.2.5 Lemma. *For every $i \in I$, every $(u, v) \in U_i \times V_i$, there is a triple $t = (j, r, s) \in I \times \Sigma^+ \times \Sigma^+$ such that:*

1. *$r.s^\omega = u.v^\omega$ (hence $M_{u,v}$ and $M_{r,s}$ are counting bisimilar),*
2. *for every $n > 0$, $r.s^n \in U_j$ and $s^n \in V_j$.*

Proof. Let i, u and v be as above. One has $M_{u,v} \models \varphi$. By invariance of φ, for each integer $k > 0$, one also has $M_{u.v^k,v^k} \models \varphi$. Hence, applying Theorem 6.1.2.2 for each integer $k > 0$ there is some $i_k \in I$ such that $(u.v^k, v^k) \in U_{i_k} \times V_{i_k}$. Since this is true for infinitely many k and I is finite, there is some $j \in I$ such that $j = i_k$ for infinitely many k.

Moreover, since U_j and V_j are regular, this implies that there is some $p > 0$ such that $j = i_{p.n}$ for every $n > 0$. We conclude the proof by taking $r = u.v^p$ and $s = v^p$.

□

For every such a triple $t = (j, r, s)$, called special, let define the languages

$$D_t = [r]_{U_j}.([s]_{U_j} \cap [s]_{V_j})$$

and

$$E_t = ([s]_{U_j} \cap [s]_{V_j})$$

with the *congruence class* $[w]_L$ of a finite word $w \in \Sigma^+$ with respect to a language $L \subseteq \Sigma^+$ defined to be the set of words

$$[w]_L = \{w' \in \Sigma^+ | \forall u, v \in \Sigma^*, u.w.v \in L \Leftrightarrow u.w'.v \in L\}$$

By construction, since U_j and V_j are regular languages, then D_t and E_t are also regular languages. Also, even though there are infinitely many special triple, there are still finitely many D_ts and E_ts. In fact, when L is regular there are finitely many (regular) languages of the form $[w]_L$ for $w \in \Sigma^*$.

Moreover:

6.1.2.6 Lemma. *For every special triple $t = (j, r, s)$, $r.s \in D_t$, $s \in E_t$, $D_t.E_t^+ \subseteq U_j$ and $E_t^+ \subseteq V_j$.*

Proof. Let $t = (j, r, s)$ be a special triple as above.

Recall first that, for every u and $v \in \Sigma^+$, every $L \subseteq \Sigma^+$, we have $[u]_L.[v]_L \subseteq [u.v]_L$. It follows that:

$$D_t.E_t^+ \subseteq \bigcup_{n>0} [r.s^n]_{U_j}$$

and

$$E_t^+ \subseteq \bigcup_{n>0} [s^n]_{V_j}$$

Moreover, we know that for every $u \in \Sigma^+$ and $L \subseteq \Sigma^+$, if $u \in L$ then $[u]_L \subseteq L$. So we conclude the proof of the lemma by observing that, following Lemma 6.1.2.5, we *do* have, for every $n > 0$, $r.s^n \in U_j$ and $s^n \in V_j$.

□

Theorem 6.1.2.2 and Lemmas 6.1.2.5 and 6.1.2.6 conclude the proof of Theorem 6.1.2.4

□

6.1.2.7 Corollary. *For every MSO-formula φ, counting bisimulation-invariant on finite graphs and true only on unary graphs, there exists a regular languages L such that $L_\varphi = kern(L)$.*

Proof. Take $L = \bigcup_{t\in T} D_t.(E_t)^\omega$ as given in Theorem 6.1.2.4.

Let us show first that $L_\varphi \subseteq kern(L)$. Let $w \in L_\varphi$. By definition, there is u and v such that $u.v^\omega = w$ and $M_{u,v} \models \varphi$. By applying Lemma 6.1.2.5, this means that there is $t = (j, r, s)$ such that $u.v^\omega = r.s^\omega$ with $r.s \in D_t$ and $s \in E_t$ hence $w = r.s^\omega \in D_t.E_t^\omega$.

Conversely, let w be an ultimately periodic word in L, i.e. $w \in kern(L)$. By definition of L, this means that there is a special triple $t = (j, r, s)$ such that $w \in D_t.(E_t)^\omega$. By applying Lemma 6.1.2.3, this means that $w = u.v^\omega$ with $u \in D_t.E_t^+$ and $v \in E_t^+$. By applying Lemma 6.1.2.6, this means $w = u.v^\omega$ with $u \in U_j$ and $v \in V_j$, hence $M_{u,v} \models \varphi$ and thus $w \in L_\varphi$.

□

6.1.3 Application to bisimulation invariance in the finite

As a corollary, putting together the two theorem above:

6.1.3.1 Corollary. *Every MSO-formula counting bisimulation invariant on finite unary graphs is equivalent to a monadic Σ_1 formula.*

Moreover, since class of unary graphs equivalent to regular languages are definable, within the class of unary graphs, in the (counting or modal) mu-calculus:

6.1.3.2 Corollary. *The counting bisimulation invariant fragment of monadic Σ_1 on finite unary graphs is equivalent on finite unary graph to the mu-calculus L_μ.*

Thus, in restriction to finite unary graphs, we get that monadic Σ_1 is not only not equivalent to N_1 the first alternation level of L_μ, but it is equivalent to all of L_μ. This result is rather unexpected since, on arbitrary (finite or infinite) unary graphs, the counting bisimulation invariant fragment of monadic Σ_1 only induces topologically closed regular languages [100], i.e. languages definable by finite Büchi automata with only accepting states. This is another striking illustration that finite model theory can be dramatically different to the infinite variety.

Observe however that there is no hope to extend Corollary 6.1.3.2 to arbitrary finite graphs.

In fact, the mu-calculus formula

$$\mu X.(p \vee (\neg p \to \Box X))$$

that defines the set of vertices from which there is a (directed) path to a vertex where p holds, is bisimulation invariant but not definable by a mon. $\Sigma_1$formula. Otherwise, this would imply that directed reachability would also be definable in mon. $\Sigma_1$and that is not the case [3].

6.2 Monadic Σ_1 on finite graphs

It is an open question whether a version of the expressive completeness result stated in Theorem 4.2.2.1 is true if we restrict ourselves to finite structures. That is, is it the case

that every sentence of MSO that is bisimulation-invariant *on finite structures* is equivalent, *again on finite structures* to a sentence of modal mu-calculus ?

This statement has a weaker hypothesis and conclusion than the original theorem and is therefore not a consequence of it. It has been the subject of much recent investigation. The corresponding finite versions of the equivalence between monadic Σ_1 and N_1 for bisimulation invariant properties and of MSO and the counting mu-calculus for counting bisimulation also remain open. One related result that is known to carry over into the finite is the theorem of van Benthem[20] that every first-order definable property that is invariant under bisimulation is definable in propositional modal logic. It has been shown by Rosen [155] that this statement is still true when we restrict ourselves to finite structures.

This leads us to consider graph acceptors [173], which are known to capture monadic Σ_1 on finite structures. We show that when the properties concerned are bisimulation invariant, simple graph acceptors suffice. More precisely, we show that if a sentence φ of mon. Σ_1 is invariant under bisimulation then there is a class of structures, including representatives of all bisimulation classes, on which φ is characterized by a *tree-like* graph acceptor of *radius one* (these terms are made precise below).

One might expect that this normal form could be further refined so that the tiles are what we call *forward looking*. This would establish that bisimulation invariant properties of mon. Σ_1 can be expressed in N_1. However, such a methodology would also yield the result for the counting case, which is refuted by the counterexample obtained on unary structures (see Corollary 6.1.3.2).

6.2.1 Graph acceptors

It is known[64, 161] that monadic Σ_1 formulas can only define *local* properties. Indeed, such formulas can be characterized by *graph acceptors* [173], which are a generalization of automata operating on finite graphs rather than finite strings or trees.

6.2.1.1 Definition (k-local FO-formulas). Given a positive integer k, we say an FO-formula φ is a *k-local formula* around a first-order variable x if it is equivalent to the formula obtained from φ by restricting every quantifier in φ to the k-neighborhood of x, i.e. replacing every subformula of the form $\forall y\psi$ (resp. $\exists y\psi$) in φ by one of the form $\forall y(d(x,y) \leq k) \rightarrow \psi$ (resp. $\exists y(d(x,y) \leq k) \wedge \psi$). A *local formula* is one that is k-local for some k.

Remark. Note for every modal (or counting modal) formula α of modal depth k, the FO translation $\varphi_\alpha(x)$ is k-local around x. Indeed, it is k-local and *forward-looking*, in that we can restrict the quantifiers to the *directed* k-neighborhood by replacing $\forall y\psi$ by $\forall y(d_d(x,y) \leq k) \rightarrow \psi$, etc.

Adapting the terminology of Thomas [48, 173]:

6.2.1.2 Definition (graph acceptors). A graph acceptor $\mathcal{A}$ is a pair

$$\mathcal{A} = \langle Q, \varphi(x) \rangle$$

where Q is a finite set of states, and $\varphi(x)$ is a FO-formula local around x built on the vocabulary of Σ-graphs extended with state predicates of Q.

An accepting run of graph acceptor $\mathcal{A}$ on Σ-graph M is a mapping $\rho : V^M \to Q$ such that the $\Sigma \times Q$-labeled graph $\langle r^M, \rho^{-1} \rangle_M$ satisfies the *tiling constraint* $\varphi(x)$, i.e. $\langle r^M, \rho^{-1} \rangle_M \models \forall x \varphi$.

The set of finite graphs on which $\mathcal{A}$ has an accepting run is written $L(\mathcal{A})$.

When the tiling constraint is k-local, we say that k is the *radius* of the graph acceptor. When the tiling constraint is equivalent to a modal formula (with forward and backward modalities), we say that the graph acceptor is *tree-like*. One can check that when no backward modalities occur in the tiling constraint, a graph acceptor is just a closed (modal counting) alternating tree automaton.

When a sentence is (counting) bisimulation invariant, its truth in a model only depends on the submodels induced by the vertices reachable from the root. The following proposition is a consequence.

6.2.1.3 Lemma. *Every (counting) bisimulation invariant sentence φ of mon. Σ_1 is equivalent, on the class of finite structures, to a graph acceptor.*

Proof. Immediate consequence of Theorem 3.4 in [161].

□

Remark. Defining graph acceptors that characterizes monadic Σ_1, Thomas had to the definition above some occurrence constraint [173], i.e. extra constraints stating that some number of disjoint k-neighborhoods satisfies extra FO local properties. With the above definition, following Schwentick and Barthelman result, we do only characterize properties of monadic Σ_1 that only depends on the subgraph of M connected to its source r^M.

6.2.1.4 Lemma (Schwentick and Barthelman[161]). *Every formula α of NC_1 is equivalent to a graph acceptor.*

Proof. Let $\mathcal{A}_\alpha$ be a closed flat automaton equivalent to α (as given, say, by Theorem 3.2.2.1). For arbitrary graph M, one can express the existence of a positional winning strategy (without infinitary condition since $\alpha \in NC_1$) on game $\mathcal{G}(\mathcal{A}_\alpha, M)$ by the existence of a labeling $\rho : V^M \to \mathcal{P}(Q^{\mathcal{A}_\alpha})$ such that, for every $v \in V^M$ and every $q \in \rho(v)$, one has $\langle v \rangle_M, \rho^{-1} \models \delta^{\mathcal{A}_\alpha}(a)$. It suffices then to take for tiling constraint the formula $\varphi_\beta(x)$ where $\beta = \bigwedge_{q \in Q^{\mathcal{A}_\alpha}} \delta^{\mathcal{A}_\alpha}(q)$. Since the automaton $\mathcal{A}_\alpha$ is flat, the tiling constraint is 1-local (and forward tree-like).

□

6.2.2 Bisimulation invariant graph acceptors

Now, our aim is to push the construction that transforms a (counting) bisimulation invariant graph acceptor into a tree automaton as far as it can go on finite structures. We show that any such graph acceptor is equivalent to a tree-like graph acceptor of radius 1 on a sufficiently rich class of graphs.

We say that a graph is k-acyclic when it contains no undirected cycle of length less than $k+1$. We first show that for every structure $\mathcal{K}$ and positive integer k, we can find a k-acyclic structure that is counting bisimilar to $\mathcal{K}$ but contains no undirected cycles of length smaller than k. The construction is similar to that of acyclic covers in [144].

6.2.2.1 Definition (Powergraph). For a finite graph $\mathcal{K} = \langle V, r, E, \{p^{\mathcal{K}}\}_{p \in Pred}\rangle$ define its *powergraph* $2^{\mathcal{K}}$ to be the graph $2^{\mathcal{K}} = \langle V', r', E', \{p^{\mathcal{K}'}\}_{p \in Pred}\rangle$ defined by $V' = V \times 2^V$ (where 2^V denotes the set of maps $V \to \{0,1\}$), $r' = (r, \overline{0})$, there is an edge E' from a vertex (v, f) to a vertex (w, g) whenever $(v, w) \in E$ and g equals the function defined from f by taking, for each $u \in V$, $g(u) = f(u)$ when $u \neq w$ and $g(w) = 1 - f(w)$, and with, for each $p \in Pred$, $p^{\mathcal{K}'} = \{(v, \overline{b}) \in V' : v \in p^{\mathcal{K}'}\}$.

6.2.2.2 Lemma. *Graphs $\mathcal{K}$ and $2^{\mathcal{K}}$ are counting bisimilar and, if $\mathcal{K}$ is k-acyclic for some k then $2^{\mathcal{K}}$ is $2k$-acyclic.*

Proof. (sketch) The mapping $h : V' \to V$ that maps each vertex (v, f) in $2^{\mathcal{K}}$ to the vertex $h(v, f) = v$ in $\mathcal{K}$ induces a counting bisimulation. Now, consider an undirected cycle in the graph $2^{\mathcal{K}}$. Along any edge from (v, f) to (w, g), f and g must differ in exactly one bit. Thus, for the cycle to return to its starting point, all bits that are changed must flip at least twice. This then maps via h to a cyclic path in $\mathcal{K}$ where all vertices occur at least twice.

□

6.2.2.3 Corollary. *For each positive integer k and every graph $\mathcal{K}$, there is a k-acyclic graph $\mathcal{K}'$ counting bisimilar to $\mathcal{K}$.*

Proof. By iterating the powergraph construction.

□

Now, we obtain the following

6.2.2.4 Theorem. *Every bisimulation invariant formula φ is equivalent, on k-acyclic graphs, to a formula φ'' of the form $\varphi'' \equiv \exists Y_1 \ldots \exists Y_m \forall x \psi'$ with ψ' a 1-local tree-like constraint.*

Proof. Let φ be a counting bisimulation invariant monadic Σ_1 formula. By applying Lemma 6.2.1.3, we may assume that φ is a graph acceptor of the form $\varphi \equiv \exists X_1 \ldots \exists X_l \forall x \psi$ with ψ k-local.

We claim the following : let ψ_a be the k-local FO formula asserting that the k-neighbourhood of x is acyclic. The formula φ is equivalent, over k-acyclic graphs, to the formula, $\varphi' \equiv \exists X_1 \ldots \exists X_l \forall x (\psi \wedge \psi_a)$. It is an immediate consequence of the definition.

The last argument is based on the observation that the Hintikka type (see [55]) of a tree centered on a node c is completely determined by the atomic propositions that are true at c and the Hintikka types of the subtrees rooted at the neighbors of c. Thus, by introducing a fresh set of second-order quantifiers (logarithmic in the number of Hintikka types), it is not difficult to build the formula φ''.

□

As the constraint ψ' is tree-like of radius 1, it can be described by a counting modal formula with forward and backward modalities.

Remark. If this formula were equivalent to one without backward modalities, then one could show that we can obtain a formula θ of NC_1 that is equivalent to φ on k-acyclic graphs. As φ is invariant under counting bisimulation on finite structures by hypothesis and θ by definition and since the class of k-acyclic graphs contains representatives of all bisimulation classes on finite structures, it follows that θ and φ are equivalent on the class of all finite structures. Thus, we would have proved that every formula of monadic

Σ_1 invariant under counting bisimulation is equivalent to a formula of NC_1. This would contradict Corollary 6.1.3.2 since the class of graphs counting bisimilar to unary graphs is definable by a monadic Σ_1 formula.

6.3 Variation on FO-closures

An important part of research in the area of monadic NP is devoted to the possibility of expressing different variations of graph connectivity. Already Fagin's proof that monadic NP is different from monadic co-NP is based on the fact that connectivity of undirected graphs is not expressible by a sentence in monadic Σ^1_1, while non-connectivity obviously is. Then de Rougemont [51] and Schwentick [160] proved that connectivity is not in monadic NP even in the presence of various built-in relations.

However, as observed by Kanellakis, the property of reachability (for undirected graphs) is in monadic NP (reachability is the problem if, for a given graph and two distinguished nodes s and t, there is a path from s to t in this graph). It follows that connectivity, although not in monadic NP, is expressible by a formula of the form $\forall x \forall y \exists_2 \overline{P} \varphi$. This observation leads to the study of *closed monadic NP* [4], the class of properties expressible by a sentence with quantifier prefix of the form $(\exists_2{}^*(\exists\forall)^*)^*$, and of the *closed monadic hierarchy*, the class of properties expressible by a sentence with quantifier prefix of the form $((\exists_2{}^*(\exists\forall)^*)^*(\forall_2{}^*(\exists\forall)^*)^*)^*$.

We present here an inductive and compositional technology for proving some non expressibility results for monadic second order logic. In particular, our technology gives an alternative simple solution to all the technical problems described in the citation from [4] above. But unlike the construction in [4], which is specific for first order/Boolean closure of monadic NP, our technology is universal: it deals with first order/Boolean closure of most monadic classes.

It also appears that - with minor modifications - the above inductive constructions can also be applied inside Kozen's mu-calculus [110]. This constitutes a first small step towards trying to understand, over finite models, the (descriptive) complexity (in terms of patterns of FO and/or monadic quantifiers' prefix) of properties definable in the mu-calculus.

6.3.1 Monadic EF-games

All the structures we consider in this report are finite graphs (directed or not). The signature of the structures may also contain some additional unary relations ("colors") and constants (s and t).

6.3.1.1 Definition. 1. *A pattern of a monadic game* (or just *pattern*) is any word over the alphabet $\{\forall, \exists, \forall_2, \exists_2, \oplus\}$.

2. If w is a pattern then the pattern $\overline{w}$ (dual to w) is inductively defined as $\forall\overline{v}$, $\exists\overline{v}$, $\forall_2\overline{v}$, $\exists_2\overline{v}$ or $\oplus\overline{v}$ if w equals $\exists v$, $\forall v$, $\exists_2 v$, $\forall_2 v$ or $\oplus v$ respectively. The dual of the empty word is the empty word.

$\forall$ and $\exists$ still keep the meaning of universal and existential first order quantifiers, while $\forall_2$ and $\exists_2$ are universal and existential monadic second order (set) quantifiers. As you will soon see $\oplus$ should be understood as a sort of boolean closure of a game. We will use the abbreviation FO for the regular expression $(\forall + \exists)$.

6.3.1.2 Definition. Let P and R be two relational structures over the same signature. Let w be some pattern. An Ehrenfeucht-Fraïssé the game with pattern w over (P, R) is then the following game between 2 players, called Spoiler and Duplicator:

1. If w is the empty word then the game is over and Duplicator wins if the substructures induced in P and in R by all the constants in the signature are isomorphic. Spoiler wins if they are not isomorphic.
2. If w is nonempty then:
   (a) If $w = \exists v$ ($w = \forall v$) for some v then a new constant symbol c is added to the signature, Spoiler chooses the interpretation of c in P (R resp.) and then Duplicator chooses the interpretation of c in R (P resp.). Then they play the game with pattern v on the enriched structures.
   (b) If $w = \exists_2 v$ ($w = \forall_2 v$) for some v then a new unary relation symbol C is added to the signature, Spoiler chooses the interpretation of C in P (R resp.) and then Duplicator chooses the interpretation of C in R (P resp.) Then they play the game with pattern v on the enriched structures.
   (c) If $w = \oplus v$ for some v then Spoiler can decide if he prefers to continue with the game with pattern v or rather with $\overline{v}$. Then they play the game with the pattern chosen by Spoiler.

The part of the game described by item (a) is called a first order round, or pebbling round. The part described by item (b) is a second order round, or coloring round.

6.3.1.3 Definition. We say that a property (i.e a class of structures) $\mathcal{S}$ is expressible by a pattern w if for each two structures $P \in \mathcal{S}$ and $R \notin \mathcal{S}$ Spoiler has a winning strategy in the game with pattern w on (P, R). If W is a set of patterns then we say that $\mathcal{S}$ is expressible in W if there exists a $w \in W$ such that $\mathcal{S}$ is expressible by w.

The following theorem illustrates the links between games and logics. We skip its proof as well known (see for example [55] and [4]):

6.3.1.4 Theorem. *The following statements hold:*

1. *Monadic NP is exactly the class of properties expressible by $\exists_2{}^* FO^*$;*
2. *The boolean closure of monadic NP is exactly the class of properties expressible by $\oplus \exists_2{}^* FO^*$;*
3. *The first order closure of monadic NP is exactly the class of properties expressible by $FO^* \oplus \exists_2{}^* FO^*$;*
4. *2k-th level of the monadic hierarchy is exactly the class of properties expressible by $(\exists_2{}^* \forall_2{}^*)^k FO^*$;*
5. *2k-th level of the closed monadic hierarchy is exactly the class of properties expressible by $(FO^* \exists_2{}^* \forall_2{}^*)^k FO^*$;*
6. *Closed monadic NP is exactly the class of properties expressible by $(FO^* \exists_2{}^*)^*$;*

The last theorem motivates:

6.3.1.5 Definition. *A non trivial class of game patterns* (or just *class*) is a set of game patterns denoted by a regular expression without union over the alphabet $\{\oplus, \exists_2, \forall_2, FO\}$, which ends with FO^* and contains at least one $\forall_2{}^*$ or $\exists_2{}^*$

In the sequel, every class of game patterns we consider is non trivial.

The techniques we are going to present are inductive and compositional. *Inductive* means here that we will assume as a hypothesis that there is a property expressible by some class of patterns W_1 but not by W and then, under this hypothesis, we will prove that there is a property expressible in the class V_1W_1 but not in the class VW where V_1 and V will be some (short) prefixes. The word *compositional* means here that the pair of structures (P_{VW}, R_{VW}) (on which Duplicator has a winning strategy in a VW-game) will be directly constructed from the pair of structures (P_W, R_W) (on which Duplicator has a winning strategy in a W-game). For this construction we do not need to know anything about the original structures.

In the sequel, we will assume that all our structures are connected and that the signature contains a constant s (for *source*). This is possible thanks to the following natural definition and obvious lemma:

6.3.1.6 Definition. Let $\mathcal{S}$ be a property of structures (with the signature without constant s). Then $cone(\mathcal{S})$ is the property of structures (with the same signature, enriched with constant s): *For every x distinct from s there is an edge from s to x and the substructure induced by all the vertices distinct from s has the property $\mathcal{S}$.*

6.3.1.7 Lemma. *If $\mathcal{S}$ is expressible by w then $cone(\mathcal{S})$ also is. If $\mathcal{S}$ is not expressible by w then there is a pair of connected structures (P, R) (see Definition 6.3.1.8 below) such that P has the property $cone(\mathcal{S})$, R does not, and Duplicator has a winning strategy in the w-game on (P, R).*

□

Now we introduce some notations for graph operations. As we just mentioned we assume that all the graphs we are dealing with are connected and have some distinguished node s. Some of them will also have another distinguished node t (for *target*).

6.3.1.8 Definition.

1. Let U denote the graph containing just two vertices, s and t, and one edge $E(s, t)$.
2. If $\mathcal{A}$ is a set of graphs, then $\Sigma^s_{P\in\mathcal{A}}P$ ($\Sigma^{st}_{P\in\mathcal{A}}P$) is the union of all graphs in $\mathcal{A}$ with all the s vertices identified (resp. and all the t vertices identified). We will use also the notation $\Sigma^s_c P$ ($\Sigma^{st}_c P$) if $\mathcal{A}$ contains just c copies of the same structure P. If there are only two elements, say P and R in $\mathcal{A}$, then we write $P{+}R$ (or $P{+\!\!+}R$) instead of of $\Sigma^s_{P\in\mathcal{A}}P$ (or $\Sigma^{st}_{P\in\mathcal{A}}P$).
3. If P is a graph with constants s and t then $P.R$ (or PR for short) is the graph being a union of P and R with t of P identified with s of R (so that s of the new graph is the s of P and the t of the new graph is the t of R if it exists.
4. If $\mathcal{A}$ is a set of graphs then the graph $\Sigma^s_{P\in\mathcal{A}}(UP)$ will be called a *connected set of graphs*. If there are just two elements in $\mathcal{A}$ then we will call it a *connected pair of graphs*.

Let us start with an obvious lemma, which would remain true even without the assumption that the relations introduced during the second order rounds are unary:

6.3.1.9 Lemma. *If the graphs P and R are isomorphic then Duplicator has a winning strategy in the w-game on (P, R) whatever w is.*

□

The following Lemmas 6.3.1.10-6.3.1.12 are not much harder to prove that Lemma 6.3.1.9 but the assumption that games are monadic is crucial here:

6.3.1.10 Lemma. *If Duplicator has winning strategies in w-games on (P_1, R_1) and on (P_2, R_2) then he also has winning strategies in w-games on (P_1+P_2, R_1+R_2), on $(P_1{+\!\!+}P_2, R_1{+\!\!+}R_2)$ and on (P_1P_2, R_1R_2).*

□

6.3.1.11 Lemma. *For every structure P and pattern w there exists a number n such that provided $m \geq n$ then Duplicator has winning strategies in the w-games on $(\Sigma^s_m P, \Sigma^s_{m+1} P)$ and $(\Sigma^{st}_m P, \Sigma^{st}_{m+1} P)$*

Proof. Induction on the structure of w. Use the fact that for a structure P of some fixed size there are only finitely many colorings of it, so if we have enough copies some colorings must repeat many times.

□

6.3.1.12 Lemma. *Let P be a connected pair of structures P_1 and P_2 and let R be a connected pair of structures R_1 and R_2. Suppose for some (non trivial[1]) class V there exists $v \in V$ such that Spoiler has a winning strategy on the v-games on (P_1, R_1) and on (P_1, R_2). Then there exists $w \in \exists V$ such that Spoiler has a winning strategy in the w-games on (P, R).*

Proof. The strategy of Spoiler is to take as his first constant the source of P_1 in P. Duplicator must answer either with the source of R_1 or of R_2, and so he must make a commitment on which of the two structures is going to play the role of P_1 in R now. The cases are symmetric, so let us assume he decides on R_1. Then Spoiler uses his strategy for the v-game on (P_1, R_1) to win the game. Actually, Spoiler must force Duplicator to moves only inside the structures P_1 and R_1. This can achieved with one more coloring round (at any time in the v-game) subsequently playing a w-game for some $w \in V$ since V is non trivial. The next remark makes this observation more precise.

□

Remark. After the first round, when Spoiler picks the source of P_1 and Duplicator answers by the source of R_1, Spoiler must force Duplicator to restrict the moves of the remaining game only to the structures P_1 and R_1. In other words, Spoiler needs to be sure that each time he picks a constant inside P_1 (R_1) Duplicator actually answers with a constant inside R_1 (P_1). This can be secured with the use of an additional coloring round: Spoiler paints P_1 (or R_1, he is as happy with a $\exists_2$ round as with a $\forall_2$ one) with some color leaving the rest of P unpainted. Duplicator must answer by painting R_1 (P_1) with this color, leaving the rest of R unpainted. Otherwise, this will be detected by Spoiler with the use of the final first order rounds. Notice that the additional coloring round can take place at any moment of the game, and so that the strategy is available for Spoiler for some $\exists V$-game since V is a nontrivial class of patterns.

---

1. see Definition 6.3.1.5

6.3.2 A tool for the boolean closure

Let $\mathcal{S}$ be any property. Then, a connected pair of structures $UP{+}UR$ will be called $\mathcal{SS}$ if both the structures P and R belong to $\mathcal{S}$, $\overline{\mathcal{S}}\mathcal{S}$ if exactly one of them belongs to $\mathcal{S}$ and $\overline{\mathcal{SS}}$ otherwise.

6.3.2.1 Definition. For a property $\mathcal{S}$ define $bool(\mathcal{S})$ as the property: *the structure is a connected set of connected pairs of structures, and at least one of those pairs is* $\overline{\mathcal{S}}\mathcal{S}$.

6.3.2.2 Lemma. *Suppose a property $\mathcal{S}$ is not expressible in class W, but both $\mathcal{S}$ and its complement $\overline{\mathcal{S}}$ are expressible in some other class V. Then $bool(\mathcal{S})$ is not expressible in $\oplus W$ but is expressible in $\exists\exists V$.*

Proof. Let us first show that there exists $w \in V$ such that, provided $P \in bool(\mathcal{S})$ and $R \notin bool(\mathcal{S})$, Spoiler has a winning strategy in the $\exists\exists w$-game on (P, R). This will prove that property $bool(\mathcal{S})$ is expressible by $\exists\exists V$.

First observe that if R is not a connected set of pairs then either the vertices of R at distance less than 2 from s do not form a tree, or there is a vertex at distance 2 from s whose degree is not 3, or R is not connected, or there is a vertex x at distance 2 from s such that the structure resulting from removing x (and all the three adjacent edges) from R has less than 3 connected components. In each of those cases Spoiler can win some game in $\exists V$ for every nontrivial V.

If R is a connected set of pairs then in his first move Spoiler takes as his constant the source of some $\overline{\mathcal{S}}\mathcal{S}$ pair in P. Duplicator must answer by showing a source of some pair in R. There are two cases: either Duplicator shows a source of some $\mathcal{SS}$ pair in R or a source of some $\overline{\mathcal{SS}}$ pair in R. In each of the two cases we may think that one pair of structures has been selected in P and one in R. Spoiler can restrict the game to the two selected pairs (see Remark 6.3.1). Then we use Lemma 6.3.1.12 to finish the proof.

Now we will show that whatever a pattern $\oplus w$ is, where $w \in W$, there exist two structures $P \in bool(\mathcal{S})$ and $R \notin bool(\mathcal{S})$ such that Duplicator has a winning strategy in the $\oplus w$-game on (P, R). Let (P_1, R_1) be such a pair of structures that $P_1 \in \mathcal{S}$, $R_1 \notin \mathcal{S}$ and Duplicator has a winning strategy in the w-game on (P_1, R_1). Let c be some huge constant. Let $R = \Sigma^s_c\,(U(UP_1{+}UP_1){+}U(UR_1{+}UR_1))$. So R is a connected set of $2c$ connected pairs, c of them are $\overline{\mathcal{SS}}$ and c are $\mathcal{SS}$. Obviously, $R \notin bool(\mathcal{S})$. Let $P = R{+}U(UP_1{+}UR_1)$ be R with one more pair, a $\mathcal{S}\overline{\mathcal{S}}$ one, so that $P \in bool(\mathcal{S})$.

Now, if Spoiler in his first move decides to play the game w on P and R then remark that P is $Q_1{+}Q_2{+}Q_3$ where $Q_1 = \Sigma^s_c\,(U(UP_1{+}UP_1))$, $Q_2 = \Sigma^s_c\,(U(UR_1{+}UR_1))$ and $Q_3 = U(UR_1{+}UP_1)$ while R is $Q_4{+}Q_5{+}Q_6$ where $Q_4 = \Sigma^s_c\,(U(UP_1{+}UP_1))$, $Q_5 = \Sigma^s_{c-1}\,(U(UR_1{+}UR_1))$ and $Q_6 = U(UR_1{+}UR_1)$. We know that Duplicator has a winning strategies in w-games on (Q_1, Q_4) (by Lemma 6.3.1.9), on (Q_2, Q_5) (by Lemma 6.3.1.11) and on (Q_3, Q_6) (by Lemma 6.3.1.10, since he has a winning strategy in a w-game on (P_1, R_1)). So, again by Lemma 6.3.1.10 he has a winning strategy in w-game on (P, R).

If Spoiler decides in his first round to continue with $\overline{w}$ rather than w then take Q_1, Q_2, Q_3 as before but $Q_4 = \Sigma^s_{c-1}\,(U(UP_1{+}UP_1))$, $Q_5 = \Sigma^s_c\,(U(UR_1{+}UR_1))$ $Q_6 = U(UP_1{+}UP_1)$ and use the same reasoning, using the fact that Duplicator has a winning strategy in the $\overline{w}$-game on (R_1, P_1).

□

6.3.3 A tool for first order quantifiers

Now the signature of our structures will contain additional unary relation symbol G (for *gate*). For a given structure P, and for two its vertices x, y, such that $G(y)$ holds let $P_{x,y}$ be the structure consisting of the connected component of $P - \{x\}$, containing y as its source. $P - \{x\}$ is here understood to be the structure resulting from P after removing x and all its adjacent edges. So $P_{x,y}$ could be read as "the structure you enter from x crossing the gate y"

6.3.3.1 Definition. Let $\mathcal{S}$ be some property of structures. Then reach($\mathcal{S}$) will be the following property (of a structure P): *there is a path from s to t such that for every x on this path it holds that (i) $x \notin G$ and (ii) for every y such that $E(x, y)$ and $G(y)$ the structure $P_{x,y}$ has the property $\mathcal{S}$.*

By a *path from s to t* we mean a subset H of the set of vertices of the structure such that $s, t \in H$, each of s and t has exactly one adjacent vertex in H and each element of H which is neither s nor t has exactly 2 adjacent vertices in H. The fact that H is a path is expressible by FO^*.

6.3.3.2 Lemma. *1. Suppose a property $\mathcal{S}$ is not expressible in some class W. Then reach($\mathcal{S}$) is not expressible in FO^*W;*

2. Suppose a property $\mathcal{S}$ is expressible in some class W. Then reach($\mathcal{S}$) is expressible in the class $\exists_2\forall\forall W$.

Proof.

1. First of all we will show that if $\mathcal{S}$ is not expressible in W, then also reach($\mathcal{S}$) is not expressible in W. For a given $w \in W$ there are structures P and R such that $P \in \mathcal{S}$, $R \notin \mathcal{S}$ and Duplicator has a winning strategy in the w-game on (P, R). Consider a structure T whose only elements are s, t, x, y, whose edges are $E(s, x), E(x, t), E(x, y)$ and for which $G(y)$ holds. Let P_0 be the union of T and P, with y of T identified with s of P. The s and t of P_0 are s and t of T. Let R_0 be the structure constructed in the same way from T and R. Then obviously $P_0 \in reach(\mathcal{S})$, $R_0 \notin reach(\mathcal{S})$ and Duplicator has a winning strategy in the w-game on (P_0, R_0). Notice that both P_0 and R_0 have the following property :

(*) (property of structure Q) if x is reachable from s or from t by a path disjoint from G and if y is such that $G(y)$ and $E(x, y)$ then Q_{xy} contains neither s of Q nor t of Q.

Now let P and R be structures, both satisfying (*) and such that $P \in reach(\mathcal{S})$, $R \notin reach(\mathcal{S})$ and Duplicator has a winning strategy in a w-game on (P, R). In order to prove our claim it is enough (by induction) to construct structures (P_1, R_1) both satisfying (*) and such that $P_1 \in reach(\mathcal{S})$, $R_1 \notin reach(\mathcal{S})$ and Duplicator has a winning strategy in a $\forall\exists w$-game on (P_1, R_1). Let n be a huge enough constant. Define: $R_1 = (\Sigma^{st}_n(PR))+\!\!+(\Sigma^{st}_n(RP))$ and $P_1 = R_1+\!\!+PP$. Obviously $P_1 \in reach(\mathcal{S})$ and $R_1 \notin reach(\mathcal{S})$ hold. Now will show a winning strategy for Duplicator in a $\forall\exists w$-game on (P_1, R_1). In his first round Spoiler selects some constant in R_1. Duplicator answers with the same constant in P_1 (this is possible since R_1 can be viewed as a subset of P_1). Now notice that after this first round R_1 can be seen as

$$RP+\!\!+PR+\!\!+(\Sigma^{st}_{n-1}(PR))+\!\!+(\Sigma^{st}_{n-1}(RP))$$

and P_1 as

$$RP+\!\!+PR+\!\!+(\Sigma^{st}_{n-1}(PR))+\!\!+(\Sigma^{st}_{n-1}(RP))+\!\!+PP$$

where the constant selected in the first round is in the first $RP{+}{+}PR$, both in R_1 and in P_1. By Lemma 6.3.1.9 and Lemma 6.3.1.10 it is now enough to show that Duplicator has a winning strategy in the remaining $\exists w$-game on (P_2, R_2) where

$$P_2 = \Sigma^{st}_{n-1}(PR)){+}{+}(\Sigma^{st}_{n-1}(RP)){+}{+}PP$$

and

$$R_2 = \Sigma^{st}_{n-1}(PR)){+}{+}(\Sigma^{st}_{n-1}(RP))$$

Let Spoiler select some constant in P_2.

If Spoiler selects a constant in $\Sigma^{st}_{n-1}(PR)){+}{+}(\Sigma^{st}_{n-1}(RP))$ then Duplicator answers with the same constant in R_2 and then wins easily. The only interesting case is when Spoiler selects his constant in PP. Suppose it is selected in the first P (the other case is symmetric). Then Duplicator answers by selecting the same constant in the P of some PR in R_2. Notice that $P_2 = Q_1{+}{+}Q_2{+}{+}(\Sigma^{st}_{n-1}(RP))$ and $R_2 = Q_3{+}{+}Q_4{+}{+}(\Sigma^{st}_{n-1}(RP))$, where $Q_1 = PP$, $Q_2 = \Sigma^{st}_{n-1}(PR))$, $Q_3 = PR$ and $Q_4 = \Sigma^{st}_{n-2}(PR))$, and where some constant is already fixed in the first P of Q_1 and in the P of Q_3. Now the w-game remains to be played. But since Duplicator has a winning strategy in the w-game on (P, R) he also has (by Lemmas 6.3.1.9 and 6.3.1.10) a winning strategy in a w-game on (Q_1, Q_3). By Lemma 6.3.1.11 he has a winning strategy in a w-game on (Q_2, Q_4) and so, again by Lemma 6.3.1.10 we get a winning strategy for Duplicator in the $\exists w$-game on (P_2, R_2).

2. Suppose $P \in reach(\mathcal{S})$ and $R \notin reach(\mathcal{S})$. Spoiler, in his first move fixes a path in P, as in the definition of $reach(\mathcal{S})$. Duplicator answers selecting a set in R. If the set selected by Duplicator is not a path from s to t then Spoiler only needs some fixed number of first order rounds to win. If it is such a path then there must be some x on the path, and some y such that $E(x, y)$, $G(y)$ hold in R and $R_{x,y} \notin \mathcal{S}$. Now Spoiler uses his two first order universal rounds to fix those x and y. Duplicator answers with some two points z, t in P such that $E(z, t)$ and $G(t)$ hold in P. But, since $P \in reach(\mathcal{S})$ it turns out that $P_{z,t} \in \mathcal{S}$, so Spoiler can use rounds of the remaining w-game to secure a win (a trick from Remark 6.3.1 will be needed here to restrict the w-game to $P_{x,y}, R_{z,t}$).

□

Remark. The role of predicate G is not crucial for the construction above. It could be replaced by a graph gadget if the reader wishes to see $\mathcal{P}_2$ being a property of undirected uncolored graphs.

Another way to avoid the unary relation G (as suggested by Larry Stockmeyer) is to define $reach(S)$ as: there is a path from s to t such that for every x on this path and every y such that $E(x, y)$ and y is not on this path, the structure $P_{x,y}$ has the property $\mathcal{S}$.

6.3.4 Applications

As the first application of our toolkit we reprove the following result.

6.3.4.1 Theorem (Ajtai, Fagin and Stockmeyer [4]). *There exists property $\mathcal{P}_1$ expressible in $FO^*\exists_2{}^*FO^*$ but not in $\oplus\exists_2{}^*FO^*$. There exists property $\mathcal{P}_2$ expressible in $\exists_2 FO^*\exists_2{}^*FO^*$ but not in $FO^* \oplus \exists_2{}^*FO^*$.*

Proof. Let $Cted$ be the property of connectivity. It is well known that $Cted$ is not expressible in $\exists_2{}^*FO^*$ but both $Cted$ and its complement are expressible in $\forall\exists_2{}^*FO^*$. now take $\mathcal{P}_1 = bool(cone(Cted))$ and $\mathcal{P}_2 = reach(bool(cone(Cted)))$. Use Lemmas 6.3.2.2 and 6.3.3.2 to finish the proof.

□

A new result we can prove is that even if the hierarchy inside closed monadic NP collapses, it does not collapse on a first order level:

6.3.4.2 Theorem (J. and Marcinkowski [102]). *If there is a property expressible in FO^*W but not in W, where $W = (\exists_2{}^*FO^*)^k$ then there is a property expressible in $\exists_2 FO^*W$ but not in FO^*W.*

Proof. This follows immediately from Lemma 6.3.3.2

□

Several similar results can be proved for the closed monadic hierarchy or reproved for the monadic hierarchy (see [125] and [124] sections 4.4 and 4.5).

It is interesting to remark that the inductive constructions presented here are also definable (with minor and insignificant variations) inside Kozen's propositional μ-calculus [110].

More precisely, given some unary predicates S, one may define in the μ-calculus the new predicates that depend on S: $Bool(S) = \Diamond(\Diamond S \wedge \Diamond\neg S)$ and $Reach(S) = \mu X.(\Box(G \Rightarrow S) \wedge (\Diamond X \vee T))$ which almost denote the same constructions (here the "target" constant t is replaced by the set of "possible targets" T and the "source" constant s is the implicit free FO variable in any mu-calculus formula).

From Lemmas 6.3.2.2 and 6.3.3.2 (which extend to these definitions inside the mu-calculus) and the fact that (the mu-calculus version of) directed reachability : $dreach = \mu X.(\Diamond X \vee T)$ is not expressible in $\exists_2{}^*FO^*$ while both $dreach$ and its complement are expressible in $\exists_2\forall\exists_2{}^*FO^*$, one has :

6.3.4.3 Corollary (J. and Marcinkowski [102]). *There are properties $\mathcal{R}_1$ and $\mathcal{R}_2$ definable in monadic μ-calculus such that $\mathcal{R}_1$ is expressible in $FO^*\exists_2 FO^*\exists_2{}^*FO^*$ but not in $\oplus\exists_2{}^*FO^*$ and $\mathcal{R}_2$ is expressible in $\exists_2 FO^*\exists_2 FO^*\exists_2{}^*FO^*$ but not in $FO^* \oplus \exists_2{}^*FO^*$.*

Proof. Take $\mathcal{R}_1 = Bool(dreach)$ and $\mathcal{R}_2 = Reach(dreach)$ and apply Lemmas 6.3.2.2 and 6.3.3.2 to finish the proof.

□

Chapter 7

Distributed games

Distributed infinite discrete games, as defined in [128], is a recent multiplayer extension of discrete two player infinite games. It generalizes to infinite plays a related notion of multiplayer games with partial information defined by Peterson and Reif [148, 147].

The main motivation for their introduction is to provide an abstract framework for distributed synthesis problems, in which most known decidable cases [17, 112, 116, 120, 178, 150] can be encoded and solved uniformly [128].

In the present chapter, we review the definition of distributed synthesis problems, distributed n-process games, and we show how these notions are related to each other. We show that classical results from automata theory can be used efficiently to simplify games. More precisely, we use alternating tree automata composition, and simulation of an alternating automaton by a non-deterministic one, as two central tools for giving a simple proof of a known decidable case : the pipeline case [128] also known, in the context of Peterson and Reif finite games as hierarchical games [148, 147]. Last, we then provide new complexity and decidability results on distributed games.

Many results presented here have been obtained with Julien Bernet [25] and shall be part of his PhD thesis (in preparation) [24].

For convenience, we use in this chapter strategy trees (as defined in Section 2.1.2) instead of strategies. In fact, it occurs that the notion of strategy tree much better suits tree automata techniques on games and distributed synthesis.

7.1 Distributed synthesis

The purpose of distributed games is to provide a uniform setting into which distributed synthesis problems can be encoded. We review in this section a definition of distributed synthesis problem adapted from Pnueli and Rosner [150] [1] Our goal here is to give a zero delay semantics to not necessarily acyclic distributed architecture. This generalizes most of the cases previously presented in the literature[25, 62, 96, 128, 112, 120, 121, 150]

1. This approach extend the DEA master thesis of Xavier Briand made in 2001 under the supervision of the author.

7.1.1 Sequential functions

Synthesis problems makes extensive use of sequential functions. We review here this notion establishing also the equivalence with an alternative representation of sequential functions that plays a fundamental role in synthesis with zero-delay and loop-back.

7.1.1.1 Definition (Sequential functions). A (deterministic) *sequential function* with input alphabet A and output alphabet B is any mapping

$$g : A^* \to B^*$$

such that $g(\epsilon) = \epsilon$ and there exists a mapping

$$f : A^+ \to B$$

called the *kernel* of g, such that, for every $w \in A^*$ and every $a \in A$, $g(w.a) = g(w).f(w.a)$.

Remark. Since any sequential function $g : A^* \to B^*$ is completely defined by its kernel $f : A^+ \to B$, we may write f^* in place of g.

7.1.1.2 Definition (Finitely generated seq. functions). For all function $f : A^+ \to B$, for all $u \in A^*$, let

$$f_u : A^+ \to B$$

be the function defined, for all $v \in A^+$, by $f_u(v) = f(u.v)$. A sequential function $f^* : A^* \to B^*$ if *finitely generated* when the set

$$\{f_u \in A^+ \to B : u \in A^*\}$$

is finite. In this case, we also say that f has *finite memory.*

This definition is generally considered to be sufficient for distributed synthesis purposes [150]. In fact, synthesizing an architecture component program amount to synthesize the (finite memory) kernel of a finitely generated sequential function. Moreover, these kernels can just be seen as complete B-labeled A-trees (*with arbitrary value at the root*) and automata theory applies for giving solution to (some) distributed synthesis problems (on acyclic architecture) even with MSO specification of the expected global behaviors [112, 128, 25, 62].

However, in order to handle zero-delay and cyclic architecture, we need a more manageable equivalent definition.

7.1.1.3 Definition. The *functional kernel* of a sequential function

$$f^* : A^* \to B^*$$

is the mapping

$$F_f : A^* \to (A \to B)$$

defined, for every $u \in A^*$ and $a \in A$, by

$$(F_f(u))(a) = f(u.a)$$

7.1.1.4 Lemma. *The mapping that maps every (total) function $f : A^+ \to B$ to the (total) function $F_f : A^* \to (A \to B)$ is one to one and onto.*

Proof. The converse construction that maps any (total) function $F : A^* \to (A \to B)$ to the (total) function $f_F : A^+ \to B$ defined, for every $u \in A^*$, $a \in A$ by $f_F(u.a) = F(u)(a)$ is obviously the inverse of the mapping $f \mapsto F_f$.

□

In other words, every sequential function is (also) completely determined by its functional kernel.

Remark. Specifying sequential functions, one may equivalently specifies their kernel or their functional kernel. Actually, the translation from one formalism to the other is even a FO-transduction. In particular, for every MSO formula φ specifying properties of (kernels) of sequential functions, there is an MSO formula $\widehat{\varphi}$ such that, for every $f : A^+ \to B$, $f \models \varphi$ if and only if $F_f \models \widehat{\varphi}$. One can check that this also applies quite similarly to tree automata (equivalent to MSO specifications [152]) with no increase in automata size.

The following example of a two-player game first us that sequential functions can be seen as strategies in two player games.

7.1.1.5 Definition. Given two alphabets A and B, let define the A/B-game $\mathcal{G}_{A,B} = \langle V_P, V_E, T_P, T_E, p_0, Acc\rangle$ to be the two player game defined by $V_E = (A \to B)$, $V_P = A \cup \{*, \bot\}$, $T_P = (V_P - \{\bot\}) \times V_E$, $T_E = V_E \times (V_P - \{*\})$, $p_0 = * \in V_P$ and $Acc = (V_P + V_E)^\omega$.

7.1.1.6 Lemma. *The mapping that maps every (finite memory) strategy tree $\sigma : (V_P)^+ \to V_E$ for player P in game $\mathcal{G}_{A,B}$ to the (finitely generated) sequential function $f^*_\sigma : A^* \to B^*$ defined, for every $u \in A^*$, every $a \in A$, by $f^*_\sigma(u.a) = (\sigma(*.u))(a)$ is an onto mapping.*

In other words, defining a finitely generated sequential function amounts to, equivalently, defining a finite memory non blocking strategy for player P in game $\mathcal{G}_{A,B}$.

7.1.2 Distributed architectures

Adapted from Pnueli and Rosner's definition [150], we define here a notion of architecture where every process write on a single output channel but may read several input or (other processes) output channels. As a result, notations are slightly different. This shall however cause no particular difficulty.

Our purpose here is to give zero delay semantics to cyclic architecture as well. This is a main (and non trivial) difference with Pnueli and Rosner's presentation.

7.1.2.1 Definition. A *distributed architecture* $\mathcal{D}$ is defined as a tuple

$$\mathcal{D} = \langle I, P, r, \{A_c\}_{c \in I \cup P}\rangle$$

with finite set I of input channels, disjoint finite set P of processes and output channels (every process write on one output channel and every output channel is written by one process so we identify processes and output channels), reading mapping $r : P \to \mathcal{P}(I \cup P)$ that maps every process $p \in P$ to the set channel $r(p)$ where process p read input values, and, for every channel $i \in I \cup P$, finite alphabet A_i of possible channel values. We always

assume that all these sets are disjoint one from the other. We also always assume that $I \subseteq \bigcup\{r(p) : p \in P\}$, i.e. any input is read by at least one process.

Given architecture $\mathcal{D}$, the *dependency relation* on processes is defined to be the least transitive relation $\prec_{\mathcal{D}} \subseteq P \times P$ such that, for every $(p_1, p_2) \in P \times P$, if $p_2 \in r(p_1)$ then $p_1 \prec_{\mathcal{D}} p_2$.

We say that architecture $\mathcal{D}$ is an *acyclic distributed architecture* when relation $\prec_{\mathcal{D}}$ is a (strict) partial order relation. In this case, we write $\preceq_{\mathcal{D}}$ the induced order relation called the dependency order.

In the sequel, for every set of channels $C \subseteq I \cup P$, we shall write $A_C = \Pi_{c \in C} A_c$, i.e. A_C is the (product) alphabet of the (parallel) channels of C seen as a single bigger channel.

A typical example of distributed architecture is the *pipeline architecture* [150, 112].

7.1.2.2 Definition. A *pipeline architecture* is an acyclic distributed architecture such that (1) dependency order $\preceq_{\mathcal{D}}$ is a linear order and, (2) for every process p, if $I \cap r(p) \neq \emptyset$ then $r(p) = I$ (hence p is maximal in $\langle P, \preceq_{\mathcal{D}} \rangle$ and henceforth greatest since $\preceq_{\mathcal{D}}$ is linear).

Adapted from Peterson and Reif, one can also define the hierarchical architecture.

7.1.2.3 Definition. A *hierarchical architecture* is a distributed architecture when (1) dependency relation is total, (2) every pair of mutually dependent processes have the same inputs, and (3) only maximal processes read I, i.e. for every process p_1 and $p_2 \in P$, (1) either $(p_1, p_2) \in \prec_{\mathcal{D}}$ or $(p_2, p_1) \in \prec_{\mathcal{D}}$, (2) if $(p_1, p_2) \in \prec_{\mathcal{D}} \cap \prec_{\mathcal{D}}^{-1}$ then $r(p_1) = r(p_2)$ and (3) if $r(p_1) \cap I \neq \emptyset$ then $r(p_1) = I$.

Observe that a pipeline architecture is a hierarchical architecture. The converse is false since, in particular, a hierarchical architecture is not necessarily acyclic.

7.1.3 Zero-delay semantics

Giving a zero-delay semantics to distributed architecture could be, in presence of loops, a difficult task. However, with our hypothesis, there is no internal channel. This means that the sequence of values that occurs on output channels are completely determined by architectures global behaviors. And the question of defining global behaviors that are realizable among architecture components follow quite easily. Formally:

7.1.3.1 Definition. A *global behavior* of distributed architecture $\mathcal{D} = \langle I, P, r, \{A_c\}_{c \in I \cup P} \rangle$ is defined to be any sequential function $f^* : A_I^* \to A_P^*$. We say that the behavior f^* is *distributed* when, for each process $p \in P$, there is a sequential function $f_p^* : A_{r(p)}^* \to A_p^*$ that computes the output defined by f^* on channel p by reading all and only the inputs on channels $r(p)$.

More precisely, behavior f^* is distributed when, for every input sequence $u_I \in A_I^*$ given the output sequence $u_P \in A_P^*$ computed by f^*, i.e.

$$u_P = f^*(u_I)$$

given, for every channel $c \in I \cup P$, the sequence $u_c \in A_c^+$ of corresponding sequence of values on channel c,i.e.

$$u_c = \begin{cases} \pi_{A_c}(u_I) & \text{if } c \in I \\ \pi_{A_c}(u_P) & \text{if } c \in P \end{cases}$$

given, for every $p \in P$, the sequence $u_{r(p)} \in A^+_{r(p)}$ of inputs red by process p, i.e. for every $k \in [0, |u_I| - 1]$,

$$u_{r(p)}(k) = (u_c(k))_{c \in r(p)}$$

the following equations must be satisfied: for every $p \in P$,

$$u_p = f^*_p(u_{r(p)})$$

In this case, set $\{f_p\}_{p \in P}$ is called a *distributed realization* of f. Observe that a distributed global behavior f is completely determined by its distributed realization.

A distributed realization $\{f_p\}_{p \in P}$ of global behavior f is called a *finite distributed realization* when, moreover, for every $p \in P$, f_p has finite memory.

7.1.4 Local criteria for distributed realization

The previous definition is not much manageable: we need a local criteria that will tell us, given a distributed architecture $\mathcal{D} = \langle I, P, r, \{A_c\}_{c \in I \cup P} \rangle$, when a global behavior $f^* : A^*_I \to A^*_P$ will have a distributed realization.

7.1.4.1 Definition. Given a mapping $F : A_I \to A_P$, called an global *one-step behavior*, we say that F is *locally realizable* in $\mathcal{D}$ when there exists a set of mapping $\{f_p : A_{r(p)} \to A_p\}_{p \in P}$, called a *local realization* of F, such that, for all $a = (a_c)_{c \in I} \in A_I$, given $(a_c)_{c \in P} = F(a)$, one has, for each $p \in P$, $a_p = f_p((a_c)_{c \in r(p)})$.

This notion of one-step distributed realization leads us to a necessary and sufficient condition for an architecture behaviors to be distributed.

7.1.4.2 Theorem (J. [97]). *A global behavior $f^* : A^*_I \to A^*_P$ of architecture $\mathcal{D}$ has a distributed realization if and only if, for every $u \in A^*_I$, the mapping $F_f(u)$ is locally realizable.*

Proof. By induction on the length of u.

□

In other words, computing a distributed realization of sequential function $f^* : A^*_I \to A^*_P$ of architecture $\mathcal{D}$ amounts to, for every $u \in A^*_I$, finding a local realization of the functional kernel F_f of function f at position u. Observe that doing so, we defined, partially, functional kernels of process behaviors.

7.1.5 Distributed synthesis problems

Now comes the general definition of distributed synthesis problems.

7.1.5.1 Definition. Given a specification φ - say defined in MSO - of global behaviors, the (finite) *distributed synthesis problem* $P = \langle \mathcal{D}, \varphi \rangle$ is to find a global behavior f that satisfies φ and has a (finite memory) distributed realization.

Remark. In their presentation, Pnueli and Rosner distinguish internal channels, whose behaviors are not mentioned in the global specification φ from external channels whose behaviors can be explicitly described in φ. It shall be clear that Pnueli and Rosner approach is essentially a subcase of the present one since nothing enforces the global specification φ to speak about such or such channel.

But this generalization comes with a price. It may require a more subtle analysis of the specification φ. The idea in the present approach is to delay this analysis even further. Distributed synthesis problems are encoded, in the next section, into distributed games which behaviors, in turn, can/must be analyzed not only regarding the global specification φ but also regarding the local constraints that can be specified about the behavior of such or such process.

More generally, the distributed games approach aims at studying communication flows (or even knowledge flows) at the semantic level - looking at the environment possible moves - while, in Pnueli and Rosner's settings, environment behaviors is already partially specified at the syntactic level - in the shape of the given architecture -.

7.1.5.2 Theorem (Kupferman and Vardi [112]). *The (finite) distributed synthesis problem for pipeline architecture and MSO specification is decidable and non elementary complete.*

Proof. Non elementary hardness follows from [147] and an non elementary decision algorithm in given in [112].

□

Remark. In this case, it is also shown that existence of distributed realization coincides with existence of finite realization.

7.1.5.3 Theorem (Pnueli and Rosner [150]). *The finite distributed synthesis problem for architecture with two processes and LTL specification is, in general, undecidable and Σ_1^0-hard.*

Remark. It is an easy observation but that is worth pointed out that every synthesis problem $\langle \mathcal{D}, \varphi \rangle$ can be simulated by a equivalent synthesis problem $\langle \mathcal{D}', \varphi' \rangle$ where $\mathcal{A}'$ is a totally disconnected architecture, i.e. every process has a single external input and a single output. In fact, the synthesis problem $\langle \mathcal{D}'', \varphi' \rangle$ can be defined as follows. Every channel c in the architecture $\mathcal{D}$ is duplicated in the architecture $\mathcal{D}'$ into a channel c_I that simulate the inputs of channel c and a channel c_O to that simulates the outputs of channel c. Then, the specification φ' only need to say that specification φ must hold when values occurring on channels c_I and c_O are the same. This way, solving the synthesis problem $\langle \mathcal{D}', \varphi' \rangle$ one only need to consider to the case when architecture $\mathcal{D}'$ does behave implicitly like architecture $\mathcal{D}$.

This suggests that the definition of an architecture is, to some extent, arbitrary in the sense that synthesis problems should be attacked directly at the semantic level by a direct analysis of the implicit communication that may be forced into the specification φ.

One can also observe moreover that in the construction described above, the specification φ' can essentially be expressed from specification with extra FO connectives. This tells us that standard restriction of the expressive power of the specification language, say only handling FO formulas, will be of no help.

7.2 Distributed games

Distributed games are a special kind of multiplayer games with partial information [148, 147]. Roughly speaking, an n-process distributed game is an $n+1$ player game played

between n local cooperating process players that have their own view of the play against an environment player that has a global view of the game but, eventually, restricted moves.

For instance, modeling a point to point communication between two processes, the environment may be forced to either transmit the message or loose it. Then we knows that communication through a fair loose/transmit point to point communication channel can be solved by various process protocols such as the alternating bit protocol. In other words, restricting the environment moves may help the team of processes to have a winning strategy.

7.2.1 Distributed arenas

The worst case for the process team is when every moves of the environment player is allowed, provided it is compatible with each local game that is played between himself and the local process. This idea is modeled by the notion of free asynchronous product of local two players arenas. Any other distributed games will be a sub-case of this worst case essentially restricting the environment player capacity to move.

7.2.1.1 Definition (Free asynchronous product). Given two arenas

$$\mathcal{G}_1 = \langle V_P^{\mathcal{G}_1}, V_E^{\mathcal{G}_1}, T_P^{\mathcal{G}_1}, T_E^{\mathcal{G}_1} \rangle$$

and

$$\mathcal{G}_2 = \langle V_P^{\mathcal{G}_1}, V_E^{\mathcal{G}_2}, T_P^{\mathcal{G}_2}, T_E^{\mathcal{G}_2} \rangle$$

the *free (asynchronous) product* of $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined to be the arena

$$\mathcal{G}_1 \otimes \mathcal{G}_2 = \langle V_P^{\mathcal{G}_1 \otimes \mathcal{G}_2}, V_E^{\mathcal{G}_1 \otimes \mathcal{G}_2}, T_P^{\mathcal{G}_1 \otimes \mathcal{G}_2}, T_E^{\mathcal{G}_1 \otimes \mathcal{G}_2} \rangle$$

defined as follows:

- *environment positions* : $V_E^{\mathcal{G}_1 \otimes \mathcal{G}_2} = V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2}$,
- *processes positions* : $V_P^{\mathcal{G}_1 \otimes \mathcal{G}_2} = (V_E^{\mathcal{G}_1} \cup V_P^{\mathcal{G}_1}) \times (V_E^{\mathcal{G}_2} \cup V_P^{\mathcal{G}_1}) - (V_E^{\mathcal{G}_1} \times V_E^{\mathcal{G}_2})$,
- *processes moves* : $T_P^{\mathcal{G}_1 \otimes \mathcal{G}_2}$ is the set of all pairs $(p, e) \in (V_P^{\mathcal{G}_1 \otimes \mathcal{G}_2} \times V_E^{\mathcal{G}_1 \otimes \mathcal{G}_2})$ such that, for $i = 1$ and $i = 2$:
  - **either** $p[i] \in V_P^{\mathcal{G}_i}$ and $(p[i], e[i]) \in T_P^{\mathcal{G}_1}$ (the process i is *active in* p),
  - **or** $p[i] \in V_E^{\mathcal{G}_i}$ and $p[i] = e[i]$ (the process i is *inactive in* p),
- and *environment moves* : $T_E^{\mathcal{G}_1 \otimes \mathcal{G}_2}$ is the set of all pairs $(e, p) \in (V_E^{\mathcal{G}_1 \otimes \mathcal{G}_2} \times V_P^{\mathcal{G}_1 \otimes \mathcal{G}_2})$ such that, for $i = 1$ and $i = 2$:
  - **either** $p[i] \in V_P^{\mathcal{G}_i}$ and $(e[i], p[i]) \in T_P^{\mathcal{G}_i}$ (the environment player activates the process i),
  - **or** $p[i] \in V_E^{\mathcal{G}_i}$ and $p[i] = e[i]$ (the environment player keeps the process i inactive).

This definition generalizes to n arenas $\mathcal{G}_1, \ldots, \mathcal{G}_n$. The resulting product is written $\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$.

Remark. The free asynchronous product of simple arenas is already, as we shall see soon, a (simple case of) distributed arenas. Before defining arbitrary distributed arenas, let us comment this definition.

We want to make n processes to play against a single environment player. In the product $\mathcal{G} = \mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$ of n arenas $\mathcal{G}_1, \ldots, \mathcal{G}_n$, such a play can be defined as follows.

From a global environment position $e \in V_E^{\mathcal{G}}$, the environment player selects a set of local games by taking some $I \subseteq \{1, \cdots, n\}$ and moves to a process position $p \in V_P^{\mathcal{G}}$ obtained by choosing a local move in all selected local games. By definition of $V_P^{\mathcal{G}}$, observe that I must not be empty (otherwise the resulting position will not belong to P) hence the environment player activates at least one (local) process.

From such a processes position $p \in V_P^{\mathcal{G}}$, for each $i \in I$, the process i answers the environment player move by choosing a local move from $p[i]$ in $V_P^{\mathcal{G}_i}$. Other processes remain idle.

We want all processes to play according to their own local view of the (global) arena this is why we will always allows all possible compositions of processes local moves. However, one shall observe that if we also allow the environment player to play any composition of local moves, the resulting global game will sound (forgetting asynchronism) like n local games that are played independently from each other.

This remark leads us to the following notion of distributed arenas.

7.2.1.2 Definition (Distributed Arena). A *distributed arena* is a free asynchronous product where the possible environment moves may have been restricted. More precisely, given two arenas $\mathcal{G}_1$ an $\mathcal{G}_2$, a (two-process) distributed arena built upon the arenas $\mathcal{G}_1$ and $\mathcal{G}_2$ is any simple arena $\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}} \rangle$ with $V_P^{\mathcal{G}} = V_P^{\mathcal{G}_1 \otimes \mathcal{G}_2}$, $V_E^{\mathcal{G}} = V_E^{\mathcal{G}_1 \otimes \mathcal{G}_2}$, $T_P^{\mathcal{G}} = T_P^{\mathcal{G}_1 \otimes \mathcal{G}_2}$, $T_E^{\mathcal{G}} \subseteq T_E^{\mathcal{G}_1 \otimes \mathcal{G}_2}$.

These definitions extend to n-process distributed arenas.

Since a distributed arena is built upon n simple arenas, we need a definition to speak about its local components:

7.2.1.3 Definition (Projection). Given a n-process distributed arena

$$\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}} \rangle$$

with $V_E^{\mathcal{G}} = V_E^{\mathcal{G}_1} \times \ldots \times V_E^{\mathcal{G}_n}$ and $V_P^{\mathcal{G}} = V^{\mathcal{G}_1} \times \ldots \times V^{\mathcal{G}_n} - V_E^{\mathcal{G}}$ given a non empty set $I \subseteq \{1, \ldots, n\}$, define the *canonical projection* $\mathcal{G}[I]$ of $\mathcal{G}$ on I as the arena

$$\mathcal{G}[I] = \langle V_P^{\mathcal{G}[I]}, V_E^{\mathcal{G}[I]}, T_P^{\mathcal{G}[I]}, T_E^{\mathcal{G}[I]} \rangle$$

given by:

- $V_P^{\mathcal{G}[I]} = V_P^{\mathcal{G}}[I] - V_E^{\mathcal{G}}[I]$ (possibly smaller than $V_P^{\mathcal{G}}[I]$),
- $V_E^{\mathcal{G}[I]} = V_E^{\mathcal{G}}[I]$,
- $T_P^{\mathcal{G}[I]} = T_P^{\mathcal{G}}[I] \cap (V_P^{\mathcal{G}}[I] \times V^G g_E[I])$,
- and $T_E^{\mathcal{G}[I]} = T_E^{\mathcal{G}}[I] \cap (V_E^{\mathcal{G}}[I] \times V_P^{\mathcal{G}}[I])$,

In particular, for each $i \in [1, n]$, we write $\mathcal{G}[i]$ for the ith projection of the game $\mathcal{G}$.

Remark. Observe that an n-process distributed arena $\mathcal{G}$ as above can always be seen as a distributed arena built upon the games $\mathcal{G}[1]$, ..., $\mathcal{G}[n]$. Moreover, in the same way Cartesian product of sets is (up to isomorphism) associative, given an arbitrary non empty set $I \subset \{1, \ldots, n\}$, given $\overline{I} = \{1, \ldots, n\} - I$ (non empty), the n-process distributed arena $\mathcal{G}$ can, as well, be seen as a distributed arena built upon the two (distributed) arenas $\mathcal{G}[I]$ and $\mathcal{G}[\overline{I}]$.

7.2.2 Distributed games and strategies

We define below distributed games and distributed strategies.

7.2.2.1 Definition (Distributed Games). A n-process *distributed game* $\mathcal{G}$ is a tuple

$$V\mathcal{G} = \langle V_P, V_E, T_P, T_E, v_0, Acc \rangle$$

where $\langle V_P, V_E, T_P, T_E \rangle$ is a n-process distributed arena, $v_0 \in V_P$ is the initial position, and $Acc \subseteq (V_E.V_P)^\omega$ is the (regular) winning infinitary condition.

A distributed game is a particular case of simple game. It follows that previous notions of plays and strategies are still defined. However, in order to avoid confusion with what may happen in the local arena a distributed game is build upon, we shall speak now of a *global play* and a *global strategy*.

A play $w \in (V_E.V_P)^+$ is said to be active for the process i when w ends in a position $p \in V_P$ such that $p[i] \in V_P[i]$.

7.2.2.2 Definition (Local and distributed Strategies). Given a n-tuple of local strategies $\{\sigma_i : (V_P[i])^+ \to V_E[i]\}_{i \in \{1,\ldots,n\}}$, the induced global strategy

$$\sigma_1 \otimes \ldots \otimes \sigma_n : (V_P)^+ \to V_E$$

is defined as follows: for every play of the form $w.p \in (V_P)^+$, given the set $I \subseteq \{1, \ldots, n\}$ of active processes in the global processes position p (i.e. $I = \{i \in \{1, \ldots, n\} : p[i] \in V_P[i]\}$), define $\sigma(w.p) = e$ by:

– $e[i] = \sigma_i((w.p)[i])$ when $i \in I$, i.e. when process i is active in position p,

– $e[i] = p[i]$ when $i \in \{1, \ldots, n\} - I$, i.e. when process i is inactive in position p.

(provided everything is well-defined, otherwise $\sigma(w.p)$ is left undefined).

A global strategy $\sigma : (V_E.V_P)^+ \to V_E$ is a *distributed strategy* if σ equals the composition $\sigma_1 \otimes \ldots \otimes \sigma_n$ of some n local strategies.

Note that global strategies are not always distributed. Moreover, there are distributed games in which the processes have a winning strategy, but no winning distributed strategy.

From this, we can derive the following fact: distributed games are not determined, in the sense that even when the environment player does not have a winning strategy, the processes may not have a winning *distributed* strategy.

7.2.3 Synchronous vs asynchronous distributed games

Asynchronicity in distributed games is a very convenient characteristic to encode more easily phenomena that do occur in distributed synthesis problems: a process may perform a local action without the other processes even knowing that any action is performed. The question we address here is what is the price to pay for such increase in expressiveness.

7.2.3.1 Definition (Synchronous distributed games). Given an n-process distributed game $\mathcal{G} = \langle V_P^{\mathcal{G}}, V_E^{\mathcal{G}}, T_P^{\mathcal{G}}, T_E^{\mathcal{G}}, Acc \rangle$ we say that $\mathcal{G}$ is *synchronous* when

$$T_E^{\mathcal{G}} \subseteq V_E^{\mathcal{G}} \times \left(\prod_{i \in [1,n]} V_P^{\mathcal{G}}[i] \right)$$

i.e. the environment player moves always activate every process.

Remark. Observe that in an n-process synchronous distributed games, one can restrict the process positions to $\prod_{i\in[1,n]} V_P^{\mathcal{G}}[i] \subset V_E^{\mathcal{G}}$ without any change for plays starting in such positions or environment In the sequel, a position in $\prod_{i\in[1,n]} V_P^{\mathcal{G}}[i]$ or $V_E^{\mathcal{G}}$ is called a *synchronous position* as opposed to positions in $V_P^{\mathcal{G}} - \prod_{i\in[1,n]} V_P^{\mathcal{G}}[i]$.

7.2.3.2 Theorem (Bernet and J. [24]). *Every asynchronous distributed game $\mathcal{G}$ can be simulated by a synchronous game $\widehat{\mathcal{G}}$ with (almost) same position in such a way that there is a distributed finite memory strategy in the game $\mathcal{G}$ from synchronous position $p \in V^{\mathcal{G}}$ if and only if there is a distributed finite memory strategy in the game $\widehat{\mathcal{G}}$ from the (copy of the) same position $p \in V^{\widehat{\mathcal{G}}}$.*

Proof. Let $\mathcal{G}$ be an asynchronous n-process distributed game. We built an equivalent synchronous game $\widehat{G}$ from the game $\mathcal{G}$ as follows.

The first step is to make the asynchronicity explicit, i.e. every asynchronous move in $\mathcal{G}$ from a local position $e_i \in V_E^{\mathcal{G}}[i]$ is mimicked in the game $\widehat{\mathcal{G}}$ by a move of process P_i in $\mathcal{G}$ is put now in a distinguished copy $\widehat{e}_i \in V_P^{\widehat{\mathcal{G}}}[i]$ of the idle position e_i. From position $\widehat{e}_i$, the process P_i can only move back to local position e_i.

However, making the asynchronicity explicit gives new information to processes, i.e. a process may now, for instance, count the number of successive environment player moves that leave it idle.

The second step is thus to disable this counting (at least with finite memory strategy) by adding, from every synchronous position $e \in V_E^{\mathcal{G}}$ an environment move "asynchronous everywhere" to $\widehat{e} \in V_P^{\widehat{\mathcal{G}}}$ defined, for every $i \in [1, n]$, by $\widehat{e}[i] = \widehat{e}[i]$.

Disabling, in winning condition, infinitely many successive move asynchronous everywhere ensures that the game $\widehat{\mathcal{G}}$ that is equivalent to the game $\mathcal{G}$ w.r.t. finite memory distributed winning strategies as shown by pumping lemma arguments.

The detail of the proof is given in Julien Bernet's PhD[24].

□

7.3 Tree Automata and Distributed Games

A special form of alternating automata, that fits especially our purpose of applying automata techniques to distributed is also reviewed.

7.3.1 Tree automata for distributed games

Given two finite alphabets D and Σ, a Σ-*labeled* D-*tree* (also called D,Σ-*tree*) is a partial function $D^* \to \Sigma$ whose domain is closed under prefix operation. In the sequel, elements of Σ are called *labels* and elements of D are called *directions*.

The following definition is a variation on Muller and Schupp's original definition of alternating automata [137]. Our goal is to have a tree-transducer like automata definition, even for alternating automata.

7.3.1.1 Definition (Alternating tree automata). A finite (D, Σ)-*alternating tree automaton* is a tuple:

$$\mathcal{A} = \langle Q = Q^{\forall} \uplus Q^{\exists}, D, \Sigma, q_0, \delta = \delta^{\forall} \cup \delta^{\exists}, \Omega \rangle$$

where Q is a finite set of states, $q_0 \in Q^{\exists}$ is the initial state, $\delta^{\forall} : Q^{\forall} \times D \to \mathcal{P}(Q^{\exists})$ and $\delta^{\exists} : Q^{\exists} \times \Sigma \to \mathcal{P}(Q^{\forall})$ are the transition functions, and $\Omega : Q^{\forall} \to \omega$ is the parity condition.

An automaton $\mathcal{A}$ is a *non deterministic automaton* when $|\delta^{\forall}(q, d)| \leq 1$ (for every $q \in Q^{\forall}$, $d \in D$).

7.3.1.2 Definition (Runs). A *run* of an automaton $\mathcal{A} = \langle Q, D, \Sigma, i, \delta, \Omega \rangle$ over a Σ-labeled D-tree $t : D^* \to \Sigma$ is a $Q^{\forall}$-labeled $(D \times Q^{\exists})$ tree $\rho : (D \times Q^{\exists})^* \to Q^{\forall}$ such that:

- $\rho(\epsilon) \in \delta^{\exists}(q_0, t(\epsilon))$,
- for every $w \in Dom(\rho)$, if $\rho(w) = q$, then for every direction $d \in D$ such that $a = t(w[1].d)$ is defined, and for every existential state $q_1 \in \delta^{\forall}(q, d)$, there exists a universal state $q_2 \in \delta^{\exists}(q_1, a)$ such that $\rho(w.(d, q_1)) = q_2$.

A tree t *is accepted* by $\mathcal{A}$ if and only if there exists a run ρ of $\mathcal{A}$ over t such that for every infinite branch w in ρ: $states_{\rho}(w) \in Acc$. Denote by $L(\mathcal{A})$ the language of all trees that are accepted by $\mathcal{A}$. The *size* of an automaton $\mathcal{A}$ is denoted by $|\mathcal{A}|$.

Remark. Observe that these tree automata (both alternating and non alternating), if slightly unusual, have the same expressive power as their standard counterpart, as in [137] or in Chapter 3 in this text.

More precisely, given the automaton

$$\mathcal{A} = \langle Q = Q^{\forall} \uplus Q^{\exists}, D, \Sigma, q_0, \delta = \delta^{\forall} \cup \delta^{\exists}, Acc \subseteq Q^{\omega} \rangle$$

one may assume, without lost of generality, that Acc only depends on states of $Q^{\exists}$. Then, we define the alternating automaton

$$\mathcal{A}' = \langle Q', D, \Sigma, q_0', \delta', Acc' \rangle$$

with $Q' = Q^{\exists}$, $q_0' = q_0$, for each $q \in Q'$, $Acc' = \pi_{Q^{\exists}}(Acc)$ and, for each $q \in Q'$,

$$\delta'(q) = \bigwedge_{a \in \Sigma} a(r) \to \bigvee_{q' \in \delta^{\exists}(q,a)} \forall z \left(\bigwedge_{d \in D} T_d(r, z) \to \bigwedge_{q'' \in \delta^{\forall}(q', d)} q''(z) \right)$$

The equivalence between the automaton $\mathcal{A}$ and the automaton $\mathcal{A}'$ shall be a direct consequence of definitions.

With this alternating automata definition, a definition actually closer to Muller and Schupp original definition [136], simulation still holds:

7.3.1.3 Theorem (Muller and Schupp [137]). *Every alternating tree automaton $\mathcal{A}$ is equivalent to a non deterministic tree automaton $\mathcal{A}'$, with $|\mathcal{A}'| \leq 2^{2^{|\mathcal{A}|}}$ (with Muller acceptance condition).*

Since the runs of an automaton on trees are themselves trees, automata act as tree transducers and can be sequentially combined.

7.3.1.4 Definition (Automata Composition). Given two tree automata

$$\mathcal{A}_1 = \langle Q_1, D_1, \Sigma_1, q_{0,1}, \delta_1, Acc_1 \rangle$$

and

$$\mathcal{A}_2 = \langle Q_2, D_2, \Sigma_2, q_{0,2}, \delta_2, Acc_2 \rangle$$

such that automaton $\mathcal{A}_2$ is non deterministic, $D_2 = D_1 \times Q_1^{\exists}$ and $\Sigma_2 = Q_1^{\forall}$, we define the composition of $\mathcal{A}_1$ followed by $\mathcal{A}_2$ to be the automaton

$$\mathcal{A}_2 \circ \mathcal{A}_1 = \langle T(Q), D_1, \Sigma_1, T(q_0), T(\delta), T(Acc) \rangle$$

defined as follows:

- $T(Q^{\exists}) = Q_1^{\exists} \times Q_2^{\exists}$; $T(Q^{\forall}) = Q_1^{\forall} \times Q_2^{\forall}$;
- $q_0 = (q_{0,1}, q_{0,2})$,
- $(q_1', q_2') \in T(\delta^{\forall})((q_1, q_2), d) \Leftrightarrow \begin{cases} q_1' \in \delta^{\forall}(q_1, d) \\ \{q_2'\} = \delta_2^{\forall}(q_2, (d, q_1')) \end{cases}$
- $(q_1', q_2') \in T(\delta^{\exists})((q_1, q_2), a) \Leftrightarrow \begin{cases} q_1' \in \delta^{\exists}(q_1, a) \\ q_2' \in \delta_2^{\exists}(q_2, q_1') \end{cases}$
- $T(Acc) = \{w \in T(Q)^{\omega} \mid w[1] \in Acc_1 \wedge w[2] \in Acc_2\}$

7.3.1.5 Theorem (Bernet and J. [25]). *For every tree $t : D_1^* \to \Sigma_1$, $t \in L(\mathcal{A}_2 \circ \mathcal{A}_1)$ if and only if there exists an accepting run $\rho : (D_1 \times Q_1^{\exists})^* \to Q_1^{\forall}$ of $\mathcal{A}_1$ over t such that $\rho \in L(\mathcal{A}_2)$.*

Proof. The proof, although tedious, is not complicated, and is therefore omitted here. Observe that it is crucial that $\mathcal{A}_2$ is non-alternating ; nevertheless, by applying Theorem 7.3.1.3, one can always assume that is is the case.

□

7.3.2 Distributed games with external conditions

Our purpose here is to mix games and automata by allowing games infinitary conditions to be defined by an arbitrary tree-automaton that specifies winning strategies. Strictly speaking, this is no longer a game since the notion of winning play is lost. But, as we shall see, this definition make sense when dealing with distributed games.

7.3.2.1 Definition (External Winning Condition). A *game with external winning condition* is a tuple

$$\mathcal{G} = \langle V_P, V_E, T_P, T_E, v_0, \mathcal{A} \rangle$$

where $\langle V_P, V_E, T_P, T_E \rangle$ is a game arena, $v_0 \in V_E \cup V_P$ is the initial position, and $\mathcal{A}$ is a (V_P, V_E)-tree automaton. In such a game, a strategy tree σ is winning (from initial position v_0) if, position v_0, it is belongs to $L(\mathcal{A})$. This definition extends to distributed games.

In the sequel, in order to avoid confusion, a game with a winning condition defined as in section 7.2.1 is called *game with internal winning condition.*

For two player games, this definition does not bring much added value as illustrated by the following lemma:

7.3.2.2 Lemma. *Every two player game $\mathcal{G}$ with external (non deterministic) condition $\mathcal{A}$ is equivalent to a two player game $\mathcal{A} \times \mathcal{G}$ with internal condition with $|\mathcal{A} \times \mathcal{G}| = |\mathcal{G}|.|\mathcal{A}|$.*

Proof. Given a game $\mathcal{G}$ and an automaton $\mathcal{A} = \langle Q^\forall \uplus Q^\exists, P, E, q_0, \delta = \delta^\forall \cup \delta^\exists, Acc\rangle$. We assume that the initial position v_0 belongs to $V_P^\mathcal{G}$. A similar construction can be made with $v_0 \in V_E^\mathcal{G}$.

The Game $\mathcal{A} \times \mathcal{G}$ is defined as follows:
- $V_P^{\mathcal{A}\times\mathcal{G}} = Q^\exists \times V_P^\mathcal{G}$,
- $V_E^{\mathcal{A}\times\mathcal{G}} = Q^\forall \times V_E^\mathcal{G}$,
- $T_P^{\mathcal{A}\times\mathcal{G}}$ is the set of pair of the form $((q_1, p), (q_2, e)) \in V_P^{\mathcal{A}\times\mathcal{G}} \times V_E^{\mathcal{A}\times\mathcal{G}}$ such that $(p, e) \in T_P^\mathcal{G} g$ and $q_2 \in \delta^\exists(q_1, e)$,
- $T_E^{\mathcal{A}\times\mathcal{G}}$ is the set of pair of the form $((q_2, e), (q_3, p)) \in V_E^{\mathcal{A}\times\mathcal{G}} \times V_P^{\mathcal{A}\times\mathcal{G}}$ such that $(e, p) \in T_E^\mathcal{G} g$ and $q_3 \in \delta^\forall(q_2, p)$,
- $Acc^\mathcal{G} = \{w \in (V^{\mathcal{A}\times\mathcal{G}})^\omega : \pi_{Q^\forall \cup Q^\exists}(w) \in Acc.$

Then, one can easily check that there is a non blocking strategy σ from position $v_0 \in V_P^\mathcal{G}$ such that $\sigma \in L(\mathcal{A})$ if and only if their is a winning strategy from position $(q_0, v_0) \in V_P^{\mathcal{A}\times\mathcal{G}}$ in the game $\mathcal{A} \times \mathcal{G}$.

□

Remark. In this construction, the player P in the game $\mathcal{A}\times\mathcal{G}$ build, in parallel, a strategy σ in the game $\mathcal{G}$ and an accepting run of the automaton $\mathcal{A}$ on σ. Observe that it is essential that the automaton $\mathcal{A}$ is non deterministic, otherwise nothing would ensure, in the many copies of the automaton $\mathcal{A}$ running along the path of the strategy tree, that it is really the same strategy tree that is built by the player P.

For distributed games, the above construction may not preserve distributed strategies since viewing automata states may give them knowledge that were not known. Still, an external winning condition on a distributed game $\mathcal{G}$ can be internalized at the price of adding one more process.

7.3.2.3 Theorem (Bernet and J. [25]). *For every n-process game $\mathcal{G}$ with external winning condition, there exists a $(n+1)$-process game $\mathcal{G}'$ with internal winning condition such that $\mathcal{G}'[1, \ldots, n] = \mathcal{G}$, and such that the processes have a (distributed) winning strategy σ in the game $\mathcal{G}$ if and only if the processes have a (distributed) winning strategy of the form $\sigma \otimes \sigma'$ in the game $\mathcal{G}'$.*

Proof. (sketch) Let $\mathcal{G} = \langle V_P, V_E, T_P, T_E, v_0, \mathcal{A}\rangle$ (where $\mathcal{A} = \langle Q^\forall \uplus Q^\exists, P, E, q_0, \delta = \delta^\forall \cup \delta^\exists, Acc\rangle$) be a distributed game with external winning condition with initial condition $v_0 \in V_E^\mathcal{G}$. The case $v_0 \in V_P^\mathcal{G}$ is essentially the same up to minor initialization details.

The game $\mathcal{G}' = \langle V_P', V_E', T_P', T_E', v_0', Acc\rangle$ is defined as follows. The positions and the winning condition are given by:
- $V_P' = (V_E \times (Q^\exists \times V_E)) \cup (V_P \times (Q^\exists \times \{\#\}))$,
- $V_E' = (V_E \times Q^\exists) \cup (V_E \times Q^\forall)$,
- $v_0' = (v_0, q_0)$,
- $Acc = \{w \in (V_E'.V_P')^\omega \mid \pi_{Q^\forall \cup Q^\exists}(w) \in Acc\}$

and moves are (repeatedly) defined by: from an environment position $(e, q) \in V_E \times Q^\exists$ (or the initial position):

1. first, the environment player (deterministically) moves to the process position $(e,(q,e)) \in V_E \times (Q^{\exists} \times V_E)$,
2. then, the new (automaton) process locally chooses $q' \in \delta^{\exists}(q,e)$, the other processes stay idle, thus the play proceeds in $\mathcal{G}'$, to the environment position $(e,q') \in V_E \times Q^{\forall}$,
3. then, the environment player chooses $p \in T_E(e)$ and $q_1 \in \delta^{\forall}(q',p)$, and the play proceeds to the process position $(p,(q_1,\#)) \in V_P \times (Q^{\exists} \times \{\#\})$,
4. finally, the processes 1 to n (on the game $\mathcal{G}$) choose some $e_1 \in T_P(p)$, the new (automaton) process stays almost idle (he simply deletes the # sign), and the play proceeds to the environment position $(e_1,q_1) \in V_E \times Q^{\exists}$.

If ρ is an accepting run of $\mathcal{A}$ over t_σ (for some strategy σ in $\mathcal{G}$), one deduce from ρ a strategy σ' such that $\sigma \otimes \sigma'$ is winning in $\mathcal{G}'$. Conversely, if $\sigma \otimes \sigma'$ is a winning strategy in $\mathcal{G}'$, one can infer an accepting run of $\mathcal{A}$ over t_σ from σ'.

□

7.3.3 Distributed synthesis in distributed games

We prove here that distributed behaviors can be encoded as distributed strategies in distributed games.

7.3.3.1 Definition. Let $\mathcal{D} = \langle I, P, r, \{A_c\}_{c \in I \cup P}\rangle$ with $P = \{1,\dots,n\}$ an n-process distributed architecture. We define distributed game

$$\mathcal{G}_{\mathcal{D}} = \langle V_P, V_E, T_P, T_E, p_0, Acc\rangle$$

to be an n-process distributed game built from the free synchronous game $\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n$ where, for every $p \in P = \{1,\cdots,n\}$, the game $\mathcal{G}_p$ is the game $\mathcal{G}_{r(p),p}$ defined in Section 7.1.1.5 such that, following Lemma 7.1.1.6, encodes all possible behaviors of process p as non blocking strategies.

The initial position is the process position $p_0 = *^n = (*,*,\cdots,*)$, and environment player moves are defined as follows.

We say an environment position $e = (f_p)_{p\in P} \in V_E = \prod_{p\in P} V_E^{\mathcal{G}_p}$ is *coherent* when it is the local realization of a one step architecture behavior $F_e : A_I \to A_P$ (see Section 7.1.4.1). From a coherent position $e \in V_E$ witnessed by function $F_e : A_I \to A_P$, the environment player chooses an arbitrary $a = (a_i)_{i\in I}$ and, given $(a_i)_{i\in P} = F_e(b)$, moves to the position for processes $p_{e,a} = ((a_j)_{j\in r(p)})_{p\in P} \in V_P$. From an incoherent environment position e, the environment player always moves to $\bot^n = (\bot,\bot,\cdots,\bot) \in V_P$. The winning infinite plays Acc are defined to be the set of all plays.

Intuitively, distributed behaviors of architecture $\mathcal{D}$ are encoded as distributed winning strategies in the distributed game $\mathcal{G}_{\mathcal{D}}$ by enforcing every process $p \in P$ to define, step by step, in game $\mathcal{G}_p$, the local behavior process p will have in architecture $\mathcal{D}$, while the environment player checks that these choices are compatible one with the other in such a way that the resulting global behavior is well defined (and thus has a distributed realization). Lemma 7.1.1.6 and Theorem 7.1.4.2 apply to ensure this construction is correct.

Formally:

7.3.3.2 Theorem (Bernet and J. [24]). *A distributed strategy $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$ is winning in game $\mathcal{G}_\mathcal{D}$ if and only if the set $\{f_{\sigma_p}\}_{p\in P}$ of the behaviors defined on local games $\mathcal{G}_1, \ldots, \mathcal{G}_n$ by strategies $\sigma_1, \ldots, \sigma_n$ is a distributed realization of a sequential function $f^*_\sigma : A^*_I \to A^*_P$.*

In particular, strategy σ is finite memory if and only if the sequential function f^*_σ is finitely generated.

Proof. Let $\sigma : V^*_P \to V_E$ be a winning distributed strategy for the process team with $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$.

By definition, from every coherent position $e \in V_E$ there is one and only one mapping $F_e : A_I \to A_P$ locally realized by e. Moreover, for every input value $a \in A_I$, there is one and only one position $p_{e,a} \in V_P$ where environment player can move to and, moreover, value a can be read in values stored in $p_{e,a}$ hence all positions $\{p_{e,a}\}_{a\in A_I}$ are distinct one from the other.

It follows that there is a unique mapping $h_\sigma : A^*_I \to V^+_P$ such that $h_\sigma(\epsilon) = (*, \cdots, *)$ and, for every $u \in A^*_I$, for every $a \in A_I$, given $e = \sigma(h_\sigma(u))$, one has $h(u.a) = h_\sigma(u).p_{e,a}$.

We define then the mapping $F_\sigma : A^*_I \to (A_I \to A_P)$: the *functional kernel* of the sequential function induced by strategy σ, by, for every $u \in A_I$, $F_\sigma(u) = F_e$ with $e = \sigma(h_\sigma(u))$.

By construction, F_σ is the functional kernel of a realizable behavior f^*_σ of architecture $\mathcal{D}$. In fact, for every $u \in A^*_I$, the environment position $\sigma(h_\sigma(u))$ is a local realization of $F_\sigma(u)$ since it is a coherent position hence Theorem 7.1.4.2 applies.

Moreover, for every $u \in A^*_I$, by construction of game $\mathcal{G}_\mathcal{D}$, by Lemma 7.1.1.6 and by definition of distributed strategies, one also has

$$F_\sigma(u) = (F_{\sigma_p}(\pi_{A_{r(p)}}(h_\sigma(u))))_{p\in P}$$

where, for every $p \in P$, F_{σ_p} is the functional kernel of the (local) sequential function f_{σ_p} induced by the (local) strategy σ_p.

In other words, the set $\{f^*_{\sigma_p}\}_{p\in P}$ of the local behaviors induced by the local strategies $\{\sigma_p\}_{p\in P}$ is a distributed realization of f^*_σ.

Conversely, let $f^* : A^*_I \to A^*_P$ be a distributed architecture behavior realized by some set of local process behaviors $\{f_p\}_{p\in P}$.

Given, for every $p \in Pred$ a non blocking strategy σ_p in game $\mathcal{G}_p$ that, following Lemma 7.1.1.6, corresponds to behavior f_p, it is not hard to see that the distributed strategy $\sigma_f = \sigma_1 \otimes \cdots \otimes \sigma_n$ is winning in the distributed game $\mathcal{G}_\mathcal{D}$. In fact, this amounts to showing that, following strategy σ_f, no incoherent positions are ever reached from the initial position p_0. This immediately follows form the fact that the set of local behaviors $\{f_p\}_{p\in P}$ is a distributed realization of the global behavior f.

□

We then show that any n-process distributed synthesis problem with zero-delay semantics can be encoded into solving an $n+1$-process distributed game.

More formally:

7.3.3.3 Theorem (Bernet and J. [24]). *For every n-process architecture*

$$\mathcal{D} = \langle I, P, r, \{A_c\}_{c\in I\cup P}\rangle$$

and specification φ of (kernel of) sequential function from A_I^* to A_P^* there is an $n+1$-process distributed game $\mathcal{G}_{\langle \mathcal{D},\varphi\rangle}$ such that there is a (finitely generated) realizable behavior for $\mathcal{D}$ that satisfies φ if and only if there is a (finite memory) distributed strategy for the process team in game $\mathcal{G}_{\langle \mathcal{D},\varphi\rangle}$.

Proof. The idea is, for Internalization Theorem 7.3.2.3 to extend game $\mathcal{G}_{\mathcal{D}}$ with an extra player that will run a tree automaton on the process team distributed strategy σ in game $\mathcal{G}_{\mathcal{D}}$ in order to check that it induces a behavior that also satisfies the specification φ. By Theorem 7.3.3.2, the induced behavior will be (finitely generated and) distributed if and only if the strategy for the process team is (finite memory and) distributed.

□

Example (Pipeline Example Continued). Assume we start from a pipeline architecture $\mathcal{D}$ with set of process $P = \{1, \cdots, n\}$ linearly ordered by $\preceq_{\mathcal{D}}$ with $1 \prec_{\mathcal{D}} 2 \prec_{\mathcal{D}} \cdots \prec_{\mathcal{D}} n$. Let $\mathcal{G}$ be the n-process distributed game encoding $\mathcal{D}$ as above.

One can check that, in distributed $\mathcal{G}$, if the process i knows the strategy followed by every process j for each $j \preceq i$, then, from the initial position $(*, *, \cdots, *)$, the process i can deduce, at every step, the local positions of these processes.

For the process n, at the input of the pipeline, this tells us that the process n can behave like an automaton that read every other process' strategy before defining is own strategy in such a way that, composed with the others, it will satisfied the external condition $\mathcal{A}$. This is the way Kupferman and Vardi [112] do use for solving the pipeline synthesis problem. Since this property is preserved in the above encoding of pipeline architecture, we can generalize it to distributed games themselves. This leads to the notion of game leader and linear distributed game defined and studied in the next section.

7.3.4 Externalization with leader and application

The notion of leader defined below follows the intuition above. In fact, it provides a local condition that is sufficient for such a global knowledge to be available to a process player.

7.3.4.1 Definition (Leader). Given a 2-process game $\mathcal{G} = \langle V_P, V_E, T_P, T_E, p_0, \mathcal{A}\rangle$, we say that the process 2 is a leader when, for every environment position $e \in V_E$, every processes positions x and $y \in V_P$ such that both $(e, x) \in T_E$ and $(e, y) \in T_E$,

- if $x[2] = y[2]$ then $x[1] = y[1]$,
- if $x[2] \in V_E[2]$ or $y[2] \in V_E[2]$ then $x = y$.

Intuitively, the process 2 is a leader when, as soon as he knows a global environment position then, after an environment move (or several consecutive moves if the process 2 stays idle for some time), the process 2 can predict, from his own position, the global processes position of the game.

This local property has the following formulation when it comes to considering plays:

7.3.4.2 Lemma. *Let $\mathcal{G} = \langle V_P, V_E, T_P, T_E, p_0\rangle$ be a 2-process arena with initial position p_0. For every strategy σ for the processes, the restriction of $view_2$ to the plays that are consistent with σ and active for the process 2 is one-to-one.*

Proof. Immediate from the definition.

□

Rephrased in a more useful way, this observation leads to the following result:

7.3.4.3 Lemma. *For every 2-process game $\mathcal{G} = \langle V_P, V_E, T_P, T_E, p_0, \Omega\rangle$ such that the process 2 is a leader, there exists a $(V_P[1], V_E[1])$-automaton $\mathcal{A}_2$ such that for every strategies σ on $\mathcal{G}$, σ_1 on $\mathcal{G}_1$, the following propositions are equivalent:*

(1) *there exists a strategy σ_2 on $\mathcal{G}[2]$ such that $\sigma = \sigma_1 \otimes \sigma_2$*

(2) *σ_2 is an accepting run of $\mathcal{A}_2$ on σ_1.*

Proof. (sketch) We first give here a construction for $\mathcal{A}_2$ in the case both the process 1 and the process 2 are always active in the positions for processes.

The Automaton $\mathcal{A}_2 = \langle Q_2, V_P[1], V_E[1], q_{0,2}, \delta_2, Acc_2\rangle$ can be defined as follows:

– $Q_2^\forall = V_E$; $Q_2^\exists = V_P[2] \cup \{q_{0,2}\}$,

– $\delta_2^\forall(q, p_1) = \{p_2 \in Q_2^\exists : \ (q, (p_1, p_2)) \in T_E\} \quad (q \in Q_2^\forall, p_1 \in P[1])$,

– $\delta_2^\exists(p_2, e_1) = \{q \in Q_2^\forall : \ q[1] = e_1 \wedge (p_2, q[2]) \in T_P[2]\} \quad (p_2 \in Q_2^\exists, e_1 \in V_E[1])$ with $\delta_2^\exists(q_{0,2}, e_1) = \{e_0[2]\}$,

– $Acc_2 = Q_2^\omega$.

The correspondence between runs of $\mathcal{A}_2$ on strategy trees in $\mathcal{G}[1]$ and strategy trees in $\mathcal{G}$ easily follows from this construction, and from the fact that the process 2 is a leader in $\mathcal{G}$.

In the case the process 2 may be inactivated by the environment player one can check that, since the process 2 is a leader, the game $\mathcal{G}$ can be first normalized so that this no longer happens.

In the case the process 1 may be inactivated by environment player, then the construction below can be extended, defining (quite easily though tediously) an automaton $\mathcal{A}_2$ with ϵ-transition. However, the main arguments remain the same.

□

Since the previous result holds for arbitrary external condition and arbitrary strategies in $\mathcal{G}[1]$ (even if $\mathcal{G}[1]$ is itself a distributed game), it follows:

7.3.4.4 Theorem (Bernet and J. [25]).
For every $(n+1)$-process distributed game $\mathcal{G} = \langle V_P, V_E, T_P, T_E, p_0, \mathcal{A}\rangle$ with non deterministic external winning condition $\mathcal{A}$ such that the process $(n+1)$ is a leader, there is a $(V_P[1, n], V_E[1, n])$-automaton $\mathcal{A}_{n+1}$ such that the following propositions are equivalent:
(1) *the processes have a distributed winning strategy on $\mathcal{G}$.*

(2) *the processes have a distributed winning strategy in*

$$\langle \mathcal{G}[1 \ldots n], p_0[1 \ldots n], \mathcal{A} \circ \mathcal{A}_n\rangle.$$

Example (The Pipeline: End). We have already mentioned that, in the $(n+1)$-process pipeline arena $\mathcal{G}$, from any initial position, the process p_{n+1} is a leader. It follows that Theorem 7.3.4.4 applies.

Moreover, observe that the resulting n-process game arena $\mathcal{G}[1 \ldots n]$ is nothing but a n-process pipeline arena. This says that Theorem 7.3.4.4 can be applied repeatedly till the number of processes is reduced to one. Now, one can internalize the automaton, and compute a winning strategy in the resulting simple game using Theorem 2.1.1.9.

Transposed on our more abstract setting, this can be expressed as the following corollary of the theorem.

7.3.4.5 Theorem (Mohalik et al. [128], Bernet et al. [25]). *For every n-process ($n \geq 2$) distributed game $\mathcal{G}$ such that for each $i \in \{2, \ldots, n\}$ the process i is a leader in $\mathcal{G}[1 \ldots i]$, the problem of determining whether the processes have a finite winning strategy is decidable.*

Remark. At every step, the external condition we get from the composition is an alternating automaton that needs to be simulated by a non alternating one so that the composition can be iterated. This means that the complexity of solving the pipeline architecture synthesis problem by means of its encoding into a distributed game is a tower of exponents of depth at least the number of components in the pipeline. This (bad) complexity was expected, since this problem is non-elementary [150].

7.4 Complexity of distributed games

In general, solving distributed games - equivalently distributed synthesis problem [150] - is undecidable [147]; However, with two distributed processes only playing against the environment, the decidability problem was left open [128].

In this section, we show that two-process distributed game are undecidable. Moreover, we show that, depending on the infinitary winning condition, solving two-process distributed is Σ_1^0-complete for reachability conditions, Π_1^0-complete for safety conditions, and, more generally, the hierarchy induced by Mostowski's weak infinitary conditions [132] defines, level by level, complete problems for the arithmetical hierarchy. Last, we show that distributed games with Büchi infinitary conditions or higher are Σ_1^1-complete.

7.4.1 Tilings, quasi-tilings and two players games

In order to prove lower bounds results in next section, we will use finite and infinite tilings [21, 83].

7.4.1.1 Definition. Let $\{n, s, w, e\}$ be the four cardinal directions of the plan. Given a finite set of color C with a distinguished color $\#$ called the border color, a tile is a mapping $t : \{n, s, w, e\} \to C$ that assign to each cardinal direction a color C with the additional requirement that $t(s) \neq \#$ and $t(w) \neq \#$, i.e. color $\#$ will only be used to defined $East$ or $North$ borders.

Given a finite set T of tiles, a *tiling* is a partial function $m : \omega \times \omega \to T$ such that $dom(m) = [0, M-1] \times [0, N-1]$ for some $(M, N) \in \omega \times \omega$ when m is a finite tiling or $dom(f) = \omega \times \omega$ when m is an infinite tiling, such that: for all $(i, j) \in dom(m_\sigma)$, *N/S-compatibility:* if $(i, j+1) \in dom(m)$ then $m(i, j)(n) = m(i, j+1)(s)$, *$E/W$-compatibility:* if $(i+1, j) \in dom(m)$ then $m(i, j)(w) = m(i+1, j)(e)$, *$E$-border condition:* $(i+1, j) \notin dom(m)$ if and only if $m(i, j)(e) = \#$, and *N-border condition:* $(i, j+1) \notin dom(m)$ if and only if $m(i, j)(n) = \#$.

7.4.1.2 Theorem (Berger [21], Harel [83]). *Given a set of colors C and a set T of tiles with a distinguished tile $t_0 \in T$: (1) the problem of finding M and N and a finite $M \times N$-tiling m such that $m(0, 0) = t_0$ is Σ_1^0-complete, (2) the problem of finding an infinite tiling m such that $m(0, 0) = t_0$ is Π_1^0-complete, and (3) the problem of finding an infinite tiling m such that $m(0, 0) = t_0$ and one given color , say* red*, occurs infinitely often is Σ_1^1-complete.*

7.4.1.3 Definition. A function $m : \omega \times \omega \to T$ is a *quasi-tiling* when it satisfies N/S-compatibility on every column, *East* border condition on every line, and Est/W-compatibility on the first line but not necessarily the other conditions.

It occurs that, for every finite set of tile T and initial tile $t_0 \in T$, there exists a two-player game $\mathcal{G}_{T,t_0}$ that encodes all quasi-tiling $m : \omega \times \omega \to T$ as non blocking strategies for player E. Formally:

7.4.1.4 Definition. Given a finite set of color C, a finite set of C-colored tiles T and an initial tile t_0, let $\mathcal{G}_{T,t_0} = \langle V_P, V_E, p_0, T_P, T_E, Acc$ be the two player game defined by:

- $V_P = (\{e, n\} \times T \times \{P\}) \cup \{\bot\}$ and $V_E = (\{e, n\} \times T \times \{P\}) \cup \{*\}$,
- T_P is the set of all pairs of the form $((d, t, P), (d, t', E)) \in V_P \times V_E$ such that, if $d = e$ then $t'(w) = t(e)$ and if $d = n$ then $t'(s) = t(n)$ and $t'(e) = \#$ if and only if $t(e) = \#$,
- T_E is the set of all pairs of the form $(*, (x, t_0, P))$ or $((x, t, E), \bot)$ plus all pairs of the form $((d, t, P), (d', t, E)) \in V_P \times V_E$ such that, if $d = e$ then $d' \in \{d, n\}$ and if $d = n$ or $t(e) = \#$ and then $d' = n$,
- $p_0 = *$ and $Acc = (V_P + V_E)^\omega$.

7.4.1.5 Lemma (J. [97]). *For every non blocking strategy $\sigma : V_P^+ \to V_E$, in game $\mathcal{G}_{T,t_0}$, there is a unique quasi-tiling $m_\sigma : \omega \times \omega \to T$ such that, for all $(i, j) \in \omega \times \omega$, $(i, j) \in dom(m_\sigma)$ if and only if there is counter strategy $\tau : V_E^+ \to V_P$ such that $\pi_1 \circ \pi_{V_P}(\sigma * \tau) = e^i.n^j$ and $(\sigma * \tau)(i + j) = m_\sigma(i, j)$.*

Conversely, for every quasi-tiling m such that $m(0, 0) = t_0$ there is a non blocking strategy σ_m is game $\mathcal{G}_{m,t_0}$ such that $m_{\sigma_m} = m$.

Proof. By construction, in every play, player E task is to chose, at every step, a direction e or n and, when direction n has been chosen, or when the left border is reached, to choose repeatedly direction n. It follows that every (blocking) strategy for player E that avoids position $\bot$ can be described by (1) choosing some $(i, j) \in \omega \times \times$ and (2) playing the successive directions described by the word $e^i.n^j$ - provided player P does not create the *East* border. In front of player E, player P strategy just amounts to choose, for every $(i', j') \leq_x (i, j)$ a tile $t_{i',j'}$. It shall be clear that this choice is independent from (i, j) so we can define $m_\sigma(i', j') = t_{i',j'}$.

The fact that m_σ is a quasi-tiling immediately follows from game G_{T,t_0} definition. The converse property is also immediate.

□

Remark. Observe that, in game $\mathcal{G}_{T,t_0}$, player P chooses to define a tiling bounded in the *East* direction by choosing the first tile t such that $t(e) = \#$.

7.4.2 completeness results

7.4.2.1 Theorem (J. [97]). *The problem of finding a winning distributed strategy in a 2-process distributed game with safety condition is Π_1^0-complete.*

Proof. It shall be clear that solving a safety distributed game is Π_1^0. It remains to prove that it is also Π_1^0-hard. In order to do so, we encode the infinite tiling problem into a safety distributed game.

Let T be a finite set of tile and let $t_0 \in T$ be a given initial tile. The idea is to build a distributed game $\mathcal{G}$ from the free product $\mathcal{G}_{T,t_0} \otimes \mathcal{G}_{T,t_0}$ with safety condition in such a way that player E checks that (1) if a distributed strategy $\sigma_1 \otimes \sigma_2$ is non blocking then $\sigma_1 = \sigma_2$, and (2) a distributed strategy of the form $\sigma \otimes \sigma$ is winning if and only if the quasi-tiling $m_\sigma : \omega \times \omega \to T$ is a tiling of $\omega \times \omega$ (i.e. it satisfies moreover the E/W-compatibility condition.

More precisely, we assume, without lost of generality, that in every position in game $\mathcal{G}_{T,t_0}$ there is a (fourth) additional component that count, modulo 2, the number of moves performed by player E from the initial position $*$.

Environment moves are then restricted from the moves defined in game $\mathcal{G}_{T,t_0} \otimes \mathcal{G}_{T,t_0}$ as follows. From the initial position $(*,*)$ player E has only two kind of strategies: (1) player E, at every rounds, plays synchronously in the same direction in both local games; in both components, the number of player E moves are always equal modulo 2; relying on this information, player E checks that Process players' strategies are, on both side, equal; if not player E moves to position $(\bot, \bot)$, (2) player E first moves to the $East$, asynchronously, in the second component only, and at every other rounds, plays synchronously in the same direction in both local games; in both component, the number of player E moves are distinct, modulo 2; relying on this information, player E checks that Process players local strategies are compatible along the E/W axis, i.e. whenever tiles t_1 and t_2 are chosen by players P_1 and P_2, player E checks that $t_1(e) = t_2(w)$; if not, or, again, if any player move to a tile that contains the border color $\#$, and only in these cases, player E moves to position $(\bot, \bot)$. Additionally, in both cases, if ever a Process player choose a tile that contains the border color $\#$, then Environment also moves to position $(\bot, \bot)$.

The winning condition for the Process team is to avoid position $(\bot, \bot)$. This is a safety condition.

Since players P_1 and P_2 stay unaware of the initial choice made by player E, a winning distributed strategy is (1) to play the same strategy σ in both game, (2) to be sure that the induced quasi-tiling m_σ with $m_\sigma(t_0)$ (see Lemma 7.4.1.5) also satisfies the E/W-compatibility condition henceforth m_σ is a solution of the encoded infinite tiling problem.

Conversely, for all infinite tiling m such that $m(0,0) = t_0$ one can check that $\sigma_m \otimes \sigma_m$ is a winning distributed strategy.

□

In particular, solving distributed safety game is not in Σ_1^0. Since checking that a given finite memory strategy is distributed and winning is decidable, one also has:

7.4.2.2 Corollary. *The problem of solving finite distributed safety games with* finite-memory *distributed winning strategy is also undecidable.*

7.4.2.3 Theorem (J. [97]). *The problem of finding a winning distributed strategy in a two-process distributed game with reachability condition is Σ_1^0-complete.*

Proof. Again, it shall be clear that this problem is Σ_1^0. It remains to prove that it is Σ_1^0-hard. In order to do so, we encode into reachability distributed games the finite tiling problem.

The encoding is similar to the encoding in the proof of Theorem 7.4.2.1) except that (1) player E now allows players P_1 and P_2 to play tiles that contains the border color $\#$ and (2) the winning condition for Process team is to reach, at the end of every local play, a tile t with $t(n) = \#$.

It shall be clear that there is a winning distributed winning strategy in the new (reachability) distributed game $\mathcal{G}$ if and only if there is a finite tiling m such that $m(0,0) = t_0$

□

7.4.2.4 Theorem (J. [97]). *For every integer $n > 0$, the problem of solving two-process distributed weak game with Mostowski range $[0, n-1]$ (resp. $[1, n]$) is Π^0_n-complete (resp. Σ^0_n-complete).*

Proof. Observe first that a similar statement holds for Alternating Turing Machine with infinite computation. Intuitively, alternation allows a machine to (1) guess the answer of an oracle and, at the same time, to (2) start a computation of the oracle (or its complement) that checks the guessing is correct. By construction, since no acknowledgment is expected, the resulting infinitary conditions are weak in the sense of Mostowski[132].

For distributed games, we can proceed similarly by induction on $n > 0$ by means of (encoding of) tiling systems. The ground case $n = 1$ is solved by Theorem 7.4.2.1 and Theorem 7.4.2.3. The main technicality in building a Π^0_{n+1}-complete (resp. Σ^0_{n+1}-complete) game is to transmit, from the safety (resp. reachability) calling distributed game, arguments to the pair of called oracles (Σ^0_n and Π^0_n-complete). This can be achieved with tiling-like encoding, forcing the current line of the calling distributed game to be the first line of the called oracle (by means of adequate constraints on the possible player E moves). The resulting infinitary condition is weak as in the case of Alternating Turing Machine.

□

7.4.2.5 Theorem (J. [97]). *The problem of solving two-process (or more) distributed game with Büchi condition (or higher) is Σ^1_1-complete.*

Proof. It shall be clear that solving arbitrary n-process distributed game is Σ^1_1.

Conversely, from the encoding of the infinite tiling problem (see proof of Theorem 7.4.2.1), the idea is to add in local game $\mathcal{G}_{T,t_0}$ a non deterministic tree automaton [152, 78] that checks that, given local strategy σ followed by player P, the induced quasi-tiling m_σ (seen as a a sub tree of the binary tree $t : (e+t)^* \to T$) uses infinitely many tiles with color *red*.

Such an automaton can be defined with Büchi acceptance criterion that, in turn, defines the winning condition for every Process team.

□

Remark. As a conclusion, one may ask if similar results can be proved restricting to synchronous distributed two player games. It turns out that the answer is yes. In fact this can be proved by an encoding, in local games, of the two possible *test equality* or *test e/w-compatibility* modes for player E. In this encoding, one has to ensure that neither player P_1 nor player P_2 can read (or even deduce) what is the operating mode of player E. This makes this encoding not trivial.

It is also probably the case, following for instance [72], that with variants of these undecidable infinite distributed games, one can define similarly decidable classes of distributed games that are complete for every level of the polynomial hierarchy and, possibly with more processes, even higher. In fact, the finite $M \times N$ tiling problem with bounded M and N (given as inputs) in known to be NP-complete[83], and the pipeline distributed game is known to be decidable but non-elementary complete [147].

7.5 References and notes

Peterson and Reif [147, 148] initiate the research on multiplayer games of incomplete information, considering finite games, and introducing the notion of *hierarchical games*. Subsequent results on solving distributed synthesis (such as [150], [112]) essentially used the same ideas and techniques, except in the fact that they consider infinite plays and/or branching time specifications.

The relationships between distributed games (presented here), other (and more general) versions of distributed games[66, 67], multi-player games with partial information [147, 148], distributed synthesis [150, 112, 128], control theory [17, 119, 120, 121, 22, 156, 117, 116, 154] and other true concurrent model based approaches such as MSCs [68, 69, 69] need to be investigated further more. In particular, it is not clear at the moment how the complexity lower-bounds obtained in Section 7.4 can be applied to these various settings.

Chapter 8

Perspectives

In this chapter, we are reviewing some open problems and potential research directions that could be followed. On purpose, they are closely related with what is presented in this report. Of course, plenty of other very interesting research problems could be stated in the general field of logic, automata and games (even restricted to monadic second order logic). For instance, among many other very interesting results and works in progress presented in the GAMES network, recent advances in algebraic characterization of tree languages sound very promising and potentially quite close to the underlying research objectives of the work presented here.

8.1 Two player games

Whether parity games can be solved in $PTIME$ or not, is, in this field, the very hot problem. A better understanding of the mathematical properties of parity games and winning strategies in parity games is also a related relevant issue.

Relational fixed point algorithm to define/compute winning positions. Except Vöge and Jurdzinski's algorithm whose complexity is unknown [179], it seems that all other algorithms that compute the set of winning positions in parity games amount, more and less implicitly, to define winning positions by means of monadic second order formulas. By the Bisimulation Invariance Theorem 4.2.2.1, these formulas are, in turn, equivalent to mu-calculus formulas, which, by strictness of the hierarchy [32], have arbitrary complexity. In other words, on the conceptual point of view, solving parity games amounts, in these approaches, ..., to solve parity games...

One way to escape this reflexive point of view while staying at the symbolic level of defining the set of winning positions by means of formulas, could be to consider fixed point formulas on k-ary relations - say binary - that, provided their fixed point alternation depth is bounded, are computable in PTIME. For instance, this could consist in defining, for every integer n a canonical parity game $\mathcal{G}_n$ of size n such that, whenever a position is winning in a parity game of size n, it can be related by a simulation relation (in $\mathcal{G}_n \; E \rightleftharpoons^P \mathcal{G}$ or even in $\mathcal{G}_n \; E \vdash^P \mathcal{G}$) that would be definable by means of (relational) fixed point formula !

An attempt with a very special kind of synchronous simulation relation already leads to the algorithm presented in section 2.2. Building more clever simulation could be tried !

Complete proof systems for game. We have defined above several notions of simulation relations. The reader who is familiar with proof theory may have noticed the similarity between generalized simulation relations and sequent calculus. In fact, the player P positions can be seen as disjunctions, the player E positions can be seen as conjunctions, and our definition of generalized simulation implements somehow Gentzen sequent calculus rules. As we have proved the soundness of our definition (see Lemma 2.3.3.5) one may ask if there is some underlying completeness result to expect.

A first problem is to define what would be a valid sequent. A possible definition would be to say that, given two tuples of positions $\overline{x}$ and $\overline{y}$ in (finite) parity games, a sequent $\overline{x} \vdash \overline{y}$ is valid if, whenever there a winning strategy from all positions in $\overline{x}$, then there exists a winning strategy from at least one position in $\overline{y}$. In fact, this definition would only make sense by extending games with free variables so that positions in games would be evaluated in propositional formulas with free variables.

With this definition, there is a chance that our generalized simulation relation is complete in the sense that whenever a sequent $\overline{x} \vdash \overline{y}$ is valid then there is a generalized simulation relation from $\overline{x}$ to $\overline{y}$, i.e. the position $(\overline{x}, \overline{y})$ is winning in the underlying generalized simulation game.

However, no local constraint in players moves in the game $\mathcal{G}_1 \ {}_E\vdash^P \mathcal{G}_2$ handle the winning conditions.

Normalizing parity games. Looking for complete proof systems for games, one may use Walukiewicz result on the completeness of Kozen's axiomatization of the mu-calculus [180]. In fact, Kozen's axiomatization handles locally, in inference rules, infinitary conditions trough rules for handling least and greatest fixed point constructs. Moreover, positions in finite parity games with free variables can be seen as systems of least and greatest boolean equations. They are thus encodable in boolean mu-calculus formulas and, thus, Walukiewicz's completeness result applies.

However, in such an encoding there is both: (1) an unraveling of the finite parity games - mu-calculus formulas are tree shaped - and (2) a normalization of priorities - parity conditions are implicitly defined by the alternation of least and greatest fixed point constructions.

In our definition of generalized simulation games, we only proposes a global transfer condition. This weakness can be explained by the fact that priorities in parity games are somehow quite arbitrarily spread on positions. There may even no hope to define a complete set of inference rule that will, locally, by means of priorities comparison, guarantee the global transfer condition to be fulfilled.

The underlying open problem could be the following : what are the possible priority labeling of a given game that preserve winning plays ? Is there, among them, a canonical labeling ? A positive answer to the latter may help to understand better the structure of parity games and, possibly, will lead to a complete axiomatization of games sequents with local handling of infinitary conditions.

In such an attempt, ω-semi-group theory [146] may help since it provides canonical representation of the (regular) set of winning plays.

8.2 Automata and finite graphs

There have been many proposals to extend the word or tree automata theory to finite graphs. The fact that graph acceptors in their most general definition - with occurrence constraints - do capture monadic Σ_1 definable graph properties [173] is probably a good point in favor of this particular extension. However, comparing mu-calculus formulas to graph acceptors suggests that the question is still not settled on finite relational structure. Surprisingly enough, it leads us towards an alternative proposed by Courcelle[42, 44].

Another related and more precise question is the following: are there MSO definable properties, bisimulation invariant *on finite graphs*, that are *not* definable in the modal mu-calculus ? This question remains open despite several attempts [144, 50]. Observe that modal mu-calculus properties can be checked in PTIME while there are monadic Σ_1 definable NP-complete problems. Solving this question could very well amount to showing that there are NP hard problems definable in bisimulation invariant fragment of MSO in the finite.

Extending graph acceptors with edge labeling. Lemma 6.2.1.4 and corollary 6.1.3.2 show that graph acceptors are expressive enough to capture the level NC_1 of the counting mu-calculus hierarchy but also expressive enough to capture formulas that are at least $NC_2 \cap MC_2$ "hard" - since infinite word languages are Büchi $\cap$ co-Büchi "hard" - . On the other hand, graph acceptors are not expressive enough to capture reachability that is definable by means of a least fixed point in MC_1 (even M_1).

Observe that the definition of graph acceptor aims at extending to finite graphs the notion of automata on finite strings or trees. One may note however that on strings or trees, edge (resp. edge set) quantification is definable by means of vertex (resp. vertex set) quantification - say in the directed case, coding every edge by its target vertex. This encoding is no longer available on arbitrary graph. This suggests that the notion of graph acceptors may be extended to graph acceptors with edge labeling, i.e. moving from existential MSO_1 formulas to existential MSO_2 formulas has defined by Courcelle[44].

Are these graphs acceptors with edge labeling expressive enough to capture not only directed reachability but even, say, the alternation free mu-calculus ? The question of characterizing the bisimulation invariant fragment of these extended graph acceptors may also be simpler.

Graph acceptors with infinitary conditions. In the same way infinite tree automata are defined by extending finite tree automata adding infinitary conditions - say regular conditions or even parity conditions -, one may extend similarly graph acceptors by adding a similar infinitary condition on infinite labeled paths.

This track was already investigated in collaboration with Dietmar Berwanger. It occurs that: (1) only adding *regular* infinitary conditions to graph acceptors fails to capture the counting mu-calculus, (2) only adding parity conditions to *extended* graph acceptors with edge labeling fails to capture the counting mu-calculus, (3) adding both *regular* infinitary condition to *extended* graph acceptors with edge labeling do capture - at least - the counting mu-calculus.

The exact expressive power of these extended graph acceptors with (or without) regular infinitary conditions remains to be understood.

8.3 Distributed games

This research track is an attempt to pull down the many formalisms that deals with discrete program synthesis into a common, minimalist, formalism. Today, it is just too new an attempt to claim whether it should fail or succeed. Many things remain to be done. Some research directions are listed below.

Universality of distributed games. What known and solved problems are encodable and solvable within distributed games ?

Relationship with other theories. What tools, provided by other theories, e.g. automata theory, proof theory, concurrency theory, logic of Knowledge, are applicable - can be imported - to distributed games ?

Applicability. What concrete software - or hardware - problems are encodable and/or solvable in distributed games ?

Trying to answer these questions may lead to add structures or new concepts to the definition of distributed games in order to handle, for instance, arbitrary data, communication - or knowledge - flows, etc...

Bibliography

[1] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, 1991.

[2] G. D' Agostino and M. Hollenberg. Logical questions concerning the mu-calculus: Interpolation, Lyndon and łos-Tarski. *Journal of Symbolic Logic*, 65(1):310–332, 2000.

[3] M. Ajtai and R. Fagin. Reachability is harder for directed rather than undirected finite graphs. *Journal of Symbolic Logic*, 55:113–150, 1990.

[4] M. Ajtai, R. Fagin, and L. Stockmeyer. The closure of monadic NP. *Journal of Computer and System Sciences*, 60:660–716, 2000.

[5] R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *Proc. 9th Conferance on Concurrency Theory*, LNCS, Nice, September 1998. Springer-Verlag.

[6] H.R. Andersen. Model checking and boolean graphs. *Theoretical Comp. Science*, 126:3–30, 1994.

[7] A. Arnold. Sémantique des processus communicants. *RAIRO-Informatique Théorique*, 15:103–139, 1981.

[8] A. Arnold. Systèmes de transitions finis et sémantique des processus communicants. *TSI*, 9:193–216, 1989.

[9] A. Arnold. *Finite Transition Systems*. Masson, Prentice-Hall, 1994.

[10] A. Arnold. An initial semantics for the μ-calculus on trees and rabin's complementation lemma. *Theoretical Comp. Science*, 148:121–132, 1995.

[11] A. Arnold. The mu-calculus alternation-depth hierarchy over the binary tree is strict. *Theoretical Informatics and Applications*, 33:329–339, 1999.

[12] A. Arnold and A. Dicky. An algebraic characterization of transition system equivalences. *Information and Computation*, 82:198–229, 1989.

[13] A. Arnold, G. Lenzi, and J. Marcinkowski. The hierarchy inside closed monadic Σ_1 collapses on the infinite binary tree. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 157–166. IEEE Computer Society, 2001.

[14] A. Arnold and D. Niwiński. Fixed point characterization of Büchi automata on infinite trees. *J. Inf. Process. Cybern. EIK*, 26:451–459, 1990.

[15] A. Arnold and D. Niwiński. Fixed points characterization of weak monadic logic definable sets of trees. In *Tree, Aut. and Languages*, pages 159–188. Elsevier, 1992.

[16] A. Arnold and D. Niwiński. *Rudiments of mu-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

[17] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controlers with partial observation. *Theoretical Comp. Science*, 303(1):7–34, 2003.

[18] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, volume 398 of *LNCS*, pages 62–74. Springer-Verlag, 1989.

[19] J. van Benthem. *Languages in Action. Catégories, Lambdas and Dynamic Logic*, volume 130 of *Studies in Logic*. North-Holland, 1991.

[20] J.F.A.K. van Benthem. *Modal Correspondance Theory*. PhD thesis, University of Amsterdam, 1976.

[21] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66:1–72, 1966.

[22] A. Bergeron. A unified approach to control problems in discrete event processes. *RAIRO-ITA*, 27:555–573, 1993.

[23] J. A. Bergstra and J. W. Klop. Process theory based on bisimulation semantics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, 1988.

[24] J. Bernet. *Jeux discrets pour la spécification et la synthèse de processus communiquants*. PhD thesis, LaBRI, Université de Bordeaux I, to be defended.

[25] J. Bernet and D. Janin. Automata theory for discrete distributed games. In *Foundation of Computing Theory*, volume 3623 of *LNCS*, pages 540–551. Springer-Verlag, 2005.

[26] J. Bernet, D. Janin, and I. Walukiewicz. Permissive strategies: from parity games to safety games. *Theoretical Informatics and Applications (RAIRO)*, 36:261–275, 2002.

[27] O. Bernholtz and O. Grumberg. Branching time temporal logic and amorphous tree automata. In *Conf. on Concurrency Theory (CONCUR)*, volume 715 of *LNCS*, pages 262–277. Springer-Verlag, 1993.

[28] D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2001*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 70–84. Springer-Verlag, 2001.

[29] D. Berwanger and E. Grädel. Fixed-point logics and solitaire games. *Theory of Computing Systems*, 37:675–694, 2004.

[30] Dietmar Berwanger. *Games and Logical Expressiveness*. PhD thesis, RWTH Aachen, 2005.

[31] A. Blass. A game semantics for linear logic. *Ann. Pure Appl. Logic*, 56(1-3):183–220, 1992.

[32] J. Bradfield. The alternation hierarchy for Kozen's mu-calculus is strict. In *Conf. on Concurrency Theory (CONCUR)*, volume 1119 of *LNCS*. Springer-Verlag, 1996.

[33] J. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Comp. Science*, 195:133–153, 1998.

[34] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[35] J. R. Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1962.

[36] J. R. Büchi. State strategies for games in $F_{\sigma\delta} \cap G_{\delta\sigma}$. *Journal of Symbolic Logic*, 48:1171–1198, 1983.

[37] J. R. Büchi and L. H. Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34(2):166–170, 1969.

[38] J. R. Büchi and L. H. Landweber. Solving sequencial conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[39] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[40] I. Castellani. Bisimulations and abstraction homomorphisms. *Journal of Computer and System Sciences*, 34:210–235, 1987.

[41] C. C. Chang and H. J. Keisler. *Model Theory*. Studies in Logic and the Foundations of Mathematics vol. 73. North–Holland, 1990.

[42] B. Courcelle. The monadic second order logic of graph I: Recognizable sets of finite graphs. *Inf. and Comp.*, 85:12–75, 1990.

[43] B. Courcelle. Monadic second order definable graph transductions: A survey. *Theoretical Comp. Science*, 126:53–75, 1994.

[44] B. Courcelle. The monadic second-order logic of graphs VI: On several representations of graphs by logical structures. *Discrete Applied Mathematics*, 54:117–149, 1994.

[45] B. Courcelle. The monadic second-order logic of graphs IX: Machines and their behaviours. *Theoretical Comp. Science*, 149, October 1995.

[46] B. Courcelle. The monadic second-order logic of graphs XIV: uniformly sparse graphs and edge set quantifications. *Theoretical Comp. Science*, 149, October 1995.

[47] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[48] S. Seibert D. Giammarresi, A. Restivo and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125(1):32–45, 1996.

[49] M. Dam. CTL* and ECTL* as a fragments of the modal μ-calculus. In *CAAP'92*, volume 581 of *LNCS*, pages 145–165. Springer-Verlag, 1992.

[50] A. Dawar and D. Janin. On the bisimulation invariant fragment of monadic Σ_1 in the finite. In *Found. of Soft. tech and Theor. Comp. Science*, volume 3328 of *LNCS*, pages 224–237. LNCS, 2004.

[51] M. de Rougemont. Second-order and inductive definability on finite structures. *Zeitschrift fuer Mathematische Logik und Grundlagen der Mathematik*, 33:47–63, 1987.

[52] E. W. Dijkstra. *A Discipline Of programming.* Automatic Computation. Prentice-Hall, 1976.

[53] S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games ? In *IEEE Symp. on Logic in Computer Science (LICS).* IEEE-press, 1997.

[54] C. Jutla E. A. Emerson and A. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 385–396. Springer-Verlag, 1993.

[55] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 2 edition, 1995.

[56] E. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theor. Comp. Science (vol. B)*, pages 995–1072. Elsevier, 1990.

[57] E. A. Emerson and E. M. Clark. Characterizing correctness properties of parallel programs using fixpoints. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 95 of *LNCS*, pages 169–181. Springer-Verlag, 1980.

[58] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 328–337, 1988.

[59] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 368–377, 1991.

[60] E.A. Emerson. Automated temporal reasonning about reactive systems. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency : Structure versus automata*, volume 1043 of *LNCS*, pages 111–120. Springer-Verlag, 1996.

[61] R. Fagin. Monadic generalized spectra. *Zeitschrift fuer Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[62] B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 321–330, 2005.

[63] A. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *7th Ann. ACM Symp. on Principles of Programming Languages*, pages 163–173, 1980.

[64] H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North-Holland, 1982.

[65] D. Gale and F. M. Stewart. Infinite games with perfect information. *Ann. Math. Studies*, 28:245–266, 1953.

[66] P. Gastin, B. Lerman, and M. Zeitoun. Causal memory distributed games are decidable for series-parallel systems. In *Proceedings of FSTTCS'04*, LNCS. Springer-Verlag, 2004. To appear.

[67] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games and distributed control for asynchronous systems. In *Proceedings of LATIN'04*, volume 2976 of *LNCS*, pages 455–465. Springer-Verlag, 2004.

[68] B. Genest, M. Minea, A. Muscholl, and D. Peled. Specifying and verifying partial order properties using template MSCs. In *Proceedings of FoSSaCS'04*, volume 2987 of *LNCS*, pages 195–210. Springer-Verlag, 2004.

[69] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High level MSCs: realizability and model-checking. In P. Widmayer et al., editors, *Proc. of the 29th International Colloquium on Automata, Languages and Programming, ICALP'02*, volume 2380 of *Lect. Notes Comp. Sci.*, pages 657–668, Malaga, Spain, 2002. Springer.

[70] R.J. van Glabbeek. The linear time – branching time spectrum (extended abstract). In J.C.M. Baeten and J.W. Klop, editors, *Conf. on Concurrency Theory (CONCUR)*, volume 458 of *LNCS*, pages 278–297. Springer-Verlag, 1990.

[71] R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, *Conf. on Concurrency Theory (CONCUR)*, volume 715 of *LNCS*, pages 66–81. Springer-Verlag, 1993.

[72] E. Grädel. Domino games and complexity. *SIAM J. on Computing*, 19:787–804, 1990.

[73] E. Grädel. Decision procedures for guarded logics. In *Proceedings of 16th International Conference on Automated Deduction, CADE '99*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 31–51. Springer-Verlag, 1999.

[74] E. Grädel. On the restrainning power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[75] E. Grädel. Why are modal logics so robustly decidable? In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Current Trends in Theoretical Computer Science, Entering the 21st Century*, pages 393–498. World Scientific, 2001.

[76] E. Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Comp. Science*, 288:129 – 152, 2002.

[77] E. Grädel, C. Hirsch, and M. Otto. Back and Forth Between Guarded and Modal Logics. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000*, pages 217–228, 2000.

[78] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics and Infinite Games*, volume 2500 of *LNCS Tutorial*. Springer, 2002.

[79] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science, LICS '99*, pages 45–54. IEEE Computer Society Press, 1999.

[80] I. Guessarian and W. Niar-Dinedane. An automaton characterization of fairness in SCCS. In R. Cori and M. Wirsing, editors, *Symp. on Theor. Aspects of Computer Science (STACS)*, volume 294 of *LNCS*, pages 356–372. Springer-Verlag, 1988.

[81] Y. Gurevich and L. Harrington. Trees, automata and games. In *14th ACM Symp. on the Theory of Computing*, pages 60–65, 1982.

[82] T. Hafer and W. Thomas. Computationnal tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 267 of *LNCS*, pages 269–279. Springer-Verlag, 1987.

[83] D. Harel. Effective transformations on infinite trees, with applications to high undecidability. *J. ACM*, 33(1):224–248, 1986.

[84] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[85] M. Hennessy and R. Milner. Observing nondeterminism and concurrency. In J. de Backer and M. van Leeuwen, editors, *7th Int. Coll. on Aut., Lang. and Prog.*, volume 85 of *LNCS*, pages 299–309. Springer-Verlag, 1980.

[86] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. Assoc. Comput. Mach.*, 32:137–161, 1985.

[87] C. Hirsch. *Guarded Logics: Algorithms and Bisimulation.* PhD thesis, RWTH Aachen, 2002.

[88] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.

[89] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[90] M. Hollenberg. *Logic and Bisimulation.* PhD thesis, Utrecht University, 1998.

[91] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.

[92] D. Janin. Some results about logical descriptions of non deterministic behaviours. In *XIIIth Found. of Soft. Tech. and Theo. Comp. Sci.*, volume 761 of *LNCS*. Springer-Verlag, 1993.

[93] D. Janin. *Propriétés logiques du non déterminisme et mu-calcul modal.* PhD thesis, Université de Bordeaux I, 1996.

[94] D. Janin. Automata, tableaus and a reduction theorem for fixpoint calculi in arbitrary complete lattices. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 172–182, 1997.

[95] D. Janin. When the satisfiability problem for fixpoint expressions in complete ordered sets is simple. In *CSL'97, (communication only)*, 1997.

[96] D. Janin. On the complexity of distributed synthesis. In *submited*, 2005.

[97] D. Janin. *Contribution aux fondements des méthodes formelles: jeux, logique et automates.* Thèse d'habilitation soumise à Université de Bordeaux I, Novembre 2005.

[98] D. Janin and G. Lenzi. On the structure of the monadic logic of the binary tree. In *Mathematical Found. of Comp. Science (MFCS)*, volume 1672 of *LNCS*, pages 310–320. Springer-Verlag, 1999.

[99] D. Janin and G. Lenzi. Relating levels of the mu-calculus hierarchy and levels of the monadic hierarchy. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 347–356. IEEE Computer Society, 2001.

[100] D. Janin and G. Lenzi. On the logical definability of topologically closed recognizable languages of infinite trees. *Computing and Informatics*, 21:185–203, 2002.

[101] D. Janin and G. Lenzi. On the relationship between weak monadic second order logic on arbitrary trees, with application to the mu-calculus. *Fundam. Inform.*, 61(3-4):247–265, 2004.

[102] D. Janin and J. Marcinkowski. A toolkit for first order extension of monadic games. In *Symp. on Theor. Aspects of Computer Science (STACS)*, volume 2010 of *LNCS*. Springer-Verlag, 2001.

[103] D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In *Mathematical Found. of Comp. Science (MFCS)*, volume 969 of *LNCS*. Springer-Verlag, 1995.

[104] D. Janin and I. Walukiewicz. On the expressive completeness of the modal mu-calculus with respect to monadic second order logic. In *Conf. on Concurrency Theory (CONCUR)*, volume 1119 of *LNCS*, pages 263–277. Springer-Verlag, 1996.

[105] A. Joyal. Free lattices, communication, and money games. In *10th international congress of logic, methodology and philosophy of science*, Logic and scientific methods, pages 29–68, 1997.

[106] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. Technical Report RS-94-7, Basic Research in Computer Science, 1994.

[107] M. Jurdziński. Deciding the winner in parity games is in UP∩co-UP. *Information Processing Letters*, 68(3), 1998.

[108] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Symp. on Theor. Aspects of Computer Science (STACS)*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, feb. 2000.

[109] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, 1968.

[110] D. Kozen. Results on the propositional μ-calculus. *Theoretical Comp. Science*, 27:333–354, 1983.

[111] O. Kupferman and M. Vardi. $\Pi_2 \cap \Sigma_2 = AFMC$. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 2719 of *LNCS*, pages 697–713. Springer-Verlag, 2003.

[112] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 389–398, 2001.

[113] G. Lenzi. A hierarchy theorem for the mu-calculus. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 1099 of *LNCS*, pages 87–97. Springer-Verlag, 1996.

[114] G. Lenzi. *The Mu-calculus and the Hierarchy Problem*. PhD thesis, Scuola Normale Superiore, Pisa, 1997.

[115] G. Lenzi. A new logical characterization of Büchi automata. In *Symp. on Theor. Aspects of Computer Science (STACS)*, volume 2010 of *LNCS*. Springer-Verlag, 2001.

[116] F. Lin and M. Wonham. Decentralized control and coordination of discrete event systems with partial observation. *IEEE Transactions on automatic control*, 33(12):1330–1337, 1990.

[117] F. Lin and M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on automatic control*, 33(12):1330–1337, 1990.

[118] D. E. Long, A. Browne, E. C. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evalutation of fixpoint expressions. In *Computer Aided Verification (CAV)*, volume 818 of *LNCS*, pages 338–350. Springer-Verlag, 1994.

[119] P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, 2001.

[120] P. Madhusudan and P. Thiagarajan. Distributed control and synthesis for local specifications. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 2076 of *LNCS*. Springer-Verlag, 2001.

[121] P. Madhusudan and P.S. Thiagarajan. A decidable class of asynchrnous distributed controlers. In *CONCUR'02*, volume 2421 of *LNCS*, pages 145–160. Springer-Verlag, 2002.

[122] J.A. Makowski and E. Ravve. Incremental model checking for decomposable strutures. In J. Wiedermann and P. Hajek, editors, *Mathematical Found. of Comp. Science (MFCS)*, volume 969 of *LNCS*, pages 540–551. Springer-Verlag, 1995.

[123] D. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

[124] O. Matz. *Dot-Depth and Monadic Quantifier Alternation over Pictures*. PhD thesis, RWTH Aachen, 1999.

[125] O. Matz and W. Thomas. The monadic quantifier alternation hierarchy over finite graphs is infinite. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 236–244, 1997.

[126] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[127] R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[128] S. Mohalik and I. Walukiewicz. Distributed games. In *Found. of Soft. tech and Theor. Comp. Science*, volume 2914 of *LNCS*, pages 338–351. Springer-Verlag, 2003.

[129] F. Moller and A. Rabinovich. On the expressive power of CTL*. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 360–369, 1999.

[130] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Fifth Symposium on Computation Theory*, number 208 in LNCS, pages 157–168. Springer-Verlag, 1984.

[131] A. W. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki, 1991.

[132] A. W. Mostowski. Hierarchies of weak automata on weak monadic formulas. *Theoretical Comp. Science*, 83:323–335, 1991.

[133] A. A. Muchnik. Games on infinite trees and automata with dead-ends: a new proof for the decidability of the monadic second order theory of two successors. Bull. *EATCS*, 42:220–267, 1992. (traduction d'un article en Russe de 1984).

[134] D. Muller. Infinite sequences and finite machines. In *Fourth Annual Symp. IEEE, Switching Theory and Logical Design*, pages 3–16, 1963.

[135] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theoretical Comp. Science*, 97:233–244, 1992.

[136] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Comp. Science*, 54:267–276, 1987.

[137] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Comp. Science*, 141:67–107, 1995.

[138] D. Niwiński. On fixed point clones. In *13th ICALP*, pages 464–473, 1986.

[139] D. Niwiński. Fixed points vs. infinite generation. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 402–409, 1988.

[140] D. Niwiński. Fixed point characterization of infinite behavior of finite state systems. *Theoretical Comp. Science*, 189:1–69, 1997.

[141] D. Niwiński and I. Walukiewicz. Games for the mu-calculus. *Theoretical Computer Sciences*, 163:99–116, 1997.

[142] T. Heyman O. Grumberg and A. Schuster. Distributed symbolic model checking for mu-calculus. In *Computer Aided Verification (CAV)*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[143] M. Otto. Bisimulation-invariant Ptime and higher-dimensional mu-calculus. *Theoretical Comp. Science*, 224:237–265, 1999.

[144] M. Otto. Modal and guarded characterisation theorems over finite transition systems. In *Proc. of the 17th IEEE Symp. on Logic in Computer Science (LICS)*, pages 371–380, 2002.

[145] D. Park. Concurrency and automata on infinite sequences. In *5th GI Conf. on Theoret. Comput. Sci.*, pages 167–183, Karlsruhe, 1981. LNCS 104.

[146] D. Perrin and J.E. Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *Semigroups, Formal Languages and Groups*, NATO Advanced Study Institute, pages 49–72. Kluwer academic, 1995.

[147] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *20th Annual IEEE Symposium on Foundations of Computer Sciences*, pages 348–363, october 1979.

[148] G.L. Peterson, J.H. Reif, and S. Azhar. Decision algorithms for multiplayer noncooperative games of incomplete information. *Computers and Mathematics with Applications*, 43:179–206, january 2002.

[149] A. Pnueli. The temporal logic of programs. In *18th Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[150] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *IEEE Symposium on Foundations of Computer Science*, pages 746–757, 1990.

[151] V. Pratt. A decidable μ-calculus. In 22^{nd} *Symp. on Foundations of Comput. Sci.*, pages 421–427, 1981.

[152] M. O. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[153] M. O. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundation of Set Theory*, pages 1–23. North Holland, 1970.

[154] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[155] E. Rosen. Modal logic over finite structures. *Journal of Logic, Language and Information*, 6:427–439, 1997.

[156] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[157] S. Safra. On the complexity of ω-automata. In *FOCS*, 1988.

[158] S. Safra. Exponential determinization of ω-automtata with strong fairness acceptance conditions. In *Proc. 24th ACM Symp. on Theory of Computer Science*, 1992.

[159] L. Santocanale. The alternation hierarchy for the theory of mu-lattices. *Theory and Applications of Categories*, 9:166–197, 2002.

[160] T. Schwentick. Graph connectivity, monadic *np* and built-in relations of moderate degree. In *Int. Call. on Aut. and Lang. and Programming (ICALP)*, volume 944 of *LNCS*, pages 405–416. Springer-Verlag, 1995.

[161] T. Schwentick and K. Barthelmann. Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics and Theoretical Computer Science*, 3:109–124, 1999.

[162] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, sept. 1996.

[163] S. Shelah. The monadic second order theory of order. *Annals of Mathematics*, 102:379–419, 1975.

[164] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, Mass., 1967.

[165] J. Skurczyński. A characterization of Büchi tree automata. *Information Processing Letters*, 81:29–33, 2002.

[166] C. S. Stirling. Modal and temporal logics. In S.Abramsky, D.Gabbay, and T.Maibaum, editors, *Handbook of Logic in Comuter Science*, pages 477–563. Oxford University Press, 1991.

[167] R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.

[168] J. Stupp. The lattice-model is recursive in the original model. Technical report, Institute of Mathematics, The Hebrew University, Jerusalem, Israel, 1975.

[169] M. Takahashi. The greatest fixed points and rational omega-tree languages. *Theoretical Comp. Science*, 44:259–274, 1986.

[170] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[171] W. Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theor. Comp. Science (vol. B)*, pages 133–191. Elsevier, 1990.

[172] W. Thomas. On the synthesis of strategies in infinite games. In *Symp. on Theor. Aspects of Computer Science (STACS)*, number 944 in LNCS. Springer-Verlag, 1995.

[173] W. Thomas. Automata theory on trees and partial orders. In M. Dauchet M. Bidoit, editor, *TAPSOFT'97*, number 1214 in LNCS, pages 20–38. Springer-Verlag, 1997.

[174] W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.

[175] B. A. Trakhtenbrot. Finite automata and the logic of monadic predicates. *Dokl. Akad. Nauk SSSR*, 140:326–329, 1961.

[176] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Siberian Mathematical Journal*, 3:103–131, 1962. English translation in AMS Transl. 59 (1966), 23–55.

[177] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. System Sci.*, 32:183–221, 1986.

[178] A. Vincent. Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité. In Hermès, editor, *Modélisation des Systèmes Réactifs*, pages 87–98, 2001.

[179] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV'00*, volume 1855 of *LNCS*. Springer-Verlag, 2000.

[180] I. Walukiewicz. Completeness of Kozen's axiomatization of the propositional μ-calculus. In *IEEE Symp. on Logic in Computer Science (LICS)*, 1995.

[181] I. Walukiewicz. Monadic second order logic on tree-like structures. In *Symp. on Theor. Aspects of Computer Science (STACS)*, volume 1046 of *LNCS*, pages 401–414. Springer-Verlag, 1996.

[182] I. Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Comp. Science*, 275(1-2):311–346, 2002.

[183] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. In *Journées Montoises d'Informatique Théorique, Marne-la-Vallée*, March 2000.

[184] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.

[185] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Comp. Science*, 158:343–359, 1996.

Index