On Games in Formal Verification

Loredana Sorrentino

Università degli studi di Napoli "Federico II"
Dipartimento di Matematica e Applicazioni "R. Caccioppoli"

Dottorato in **Scienze Computazionali e Informatiche**
Ciclo XXVII

A thesis submitted in fulfillment of the degree of
Doctor in Compute Science

Submission: March 31, 2015
Defense: Napoli, to be added
Revised version: March 31, 2015

Supervisor: Prof. Dr. Aniello Murano

Acknowledgments

"*Tutto scorre...* Niente rimane così se non per alcuni momenti, per degli istanti. La vita è mutamento, trasformazione, un continuo divenire, evolvere... Perfino chi sta leggendo adesso questa tesi, non appena avrà finito di leggerla, non sarà più lo stesso in quanto la trasformazione è un processo inarrestabile quanto irreversibile".

Insomma, il tempo passa! Come tutte le cose, anche questo periodo di dottorato volge al termine, chiudendosi un altro ciclo della mia vita personale ed accademica. Proprio per questo motivo, cominciare a scrivere questi ringraziamenti è stata la cosa più difficile.

Non è facile citare e ringraziare, in poche righe, tutte le persone che, a vario titolo, mi hanno affiancata, incoraggiata supportata e "sopportata". Ad ogni modo, come ho sempre fatto in qualsiasi altra situazione, ci proverò, sperando di riuscirci nel migliore dei modi.

Innanzitutto voglio ringraziare il Prof. **Aniello Murano** essendo stato per me grande esempio di intelligenza, professionalità ed amore per la ricerca, per aver creduto in me, incoraggiandomi ogni qualvolta ho avuto qualche perplessità. Egli è sempre stato disponibile a offrirmi il suo valido contributo durante tutte le fasi del mio lavoro di ricerca, preoccupandosi che io facessi esperienze costruttive e utili alla mia crescita professionale.

Ringrazio, inoltre, il Dott. **Fabio Mogavero** per avermi seguito in maniera molto professionale durante questi anni.

Ringrazio di cuore il mio collega ed amico **Giuseppe Perelli** con il quale ho condiviso un'infinità di piccoli momenti più o meno positivi, di serenità o di tristezza, di sconforto o di entusiasmo, non facendomi mai mancare qualche bella parola di incoraggiamento quando mi sentivo sfiduciata.

Ringrazio il Dott. **Sasha Rubin** per avermi fornito interessanti ed utili consigli riguardo la stesura di questo elaborato.

Ringrazio, inoltre, tutti gli amici, le amiche ed i miei **co-autori**, per aver condiviso con me grandi e piccoli momenti di vita quotidiana ed accademica.

Infine, ringrazio di cuore mio marito e tutta la mia famiglia per essermi stati sempre vicini, ognuno a proprio modo, in questi anni e dalle cui sorprendenti manifestazioni di affetto ho tratto la forza per superare i momenti più difficili, ritrovando ogni volta gli stimoli e la giusta carica per dedicarmi a questo che io amo definire "Progetto di vita".

Contents

Introduction

Game theory [Mye91, DL$^+$08, RB94] is a powerful framework for decision-making where two or more players (or agents) take some decisions (as an opportune combination of actions) in order to achieve a certain goal. Agents can be individuals, groups, firms, or any combination of them. They can play in an adversarial or collaborative manner.

In the last years, game theory has come to the fore as a powerful tool in formal-system verification [CE81, CGP02, KVW00, QS81]. In particular, it has been usefully applied in checking the reliability of reactive and embedded systems. In formal verification, to check whether a system satisfies a desired behavior we check instead, by means of a suitable algorithm, whether a mathematical model of the system *meets* a formal specification describing the systems [CGP02]. As far the system modeling concerns, we mainly distinguish between *closed* and *open* systems [HP85]. The former are characterized from the fact that their behavior is completely determined in by their internal states. We also say that closed systems admit only one source of non-determinism, *i.e.*, the internal one. An open system, instead, is characterized by an ongoing interaction with an external environment on which the whole behavior of the system model relies. Hence, open systems admit two sources of non-determinism, one from the environment and one from the system itself. Closed systems are usually modeled via *Kripke* structures or *labeled-state transition systems*. Open systems, instead, require more involved structures in which one has to explicitly take into consideration the interaction between the system and the external environment. This reasoning also applies in the more general setting in which the agents consist of the interaction of several entities. As far as the specification concerns, we distinguish between the cases in which it is given explicitly, for example, via a formula of a logic, or implicitly along with a condition over the model. The classical *reachability* question over some specific states is an example of the latter. The algorithm to decide whether the system model "meets" the specification strongly relies on the specific setting one considers.

In this thesis, we use games to model and reason about open-system design and verification. We both consider the case of internal and external specifications. In the first case, we restrict to the case in which it is defined directly over the game structure. In the second case, we refer to some specific logics under the strategic reasoning framework. More specifically, we significantly extend some well-investigated qualitative conditions in order to address quantitative agent objectives. This kind of conditions turn out to be very useful to additionally specify how much effort an agent has to use to reach his own target or how many possible different ways he has to achieve it.

We first consider the case of internal quantitative specifications. To this aim, we start introducing the basic game model of *two-player games*. In this setting one of the players,

usually called *Player* 0 is used to represent the system, and the other one, called *Player* 1, is used to represent the external environment [PR89, Mar93, Raj97, Orn00, KVW01]. In this setting, the specification can be simply given as a *winning condition* over the game structure. Depending on the interaction between the system and its environment, the resulting game may be either turn-based (*i.e.*, system and the environment transitions are interleaved) or concurrent (*i.e.*, player transitions are taken simultaneously). In a turn-based game, the states of the game are partitioned into ones belonging to Player 0 and those ones belonging to Player 1. Then, the owner of a state determines the move to take and thus the next state of the game. In a concurrent game, conversely, the two players choose a move (i.e, *actions*) simultaneously and independently, and both choices together determine the next state of the game. Turn-based games correspond to an interleaving semantics between the system and the environment. Concurrent games correspond instead to a synchronous interaction [Luc00].

Among two-player turn-based games of infinite-duration, one model framework widely investigated in open-system verification is *Parity Games* [EJ88, EJ91, Mos87, Zie98]. In the last two decades, they have been proved to be one of the most powerful evaluation machinery for the automatic synthesis and verification of distributed and reactive systems in several real scenarios [AMM11, AMM13, AKM12, KVW01]. Their importance is also related to their strict connection with other games of infinite duration, such as, *mean* payoff, *discounted* payoff, *energy*, and *stochastic* games [Ber07, CD12, CDHR10, CHJ05, CJH04]. Noteworthy, parity games are polynomially equivalent to the model checking of specification expressed via formulas of the μ-calculus modal logic [Koz83, EJ91]. In the years, this dichotomy has been fruitfully used to solve several important theoretical questions in formal verification. It has been also used in very complicated scenarios such as the case of open-systems interacting with an external environment having only partial information about the former [KVW01, KV97]. Parity games have been also the subject of tools usefully applied in several real formal-verification scenarios [CLM15]. Solving a parity game is one of the rare problems that belongs to the complexity class UPTIME $\cap$ COUPTIME and it is a long open question whether it belongs to PTIME. The variety of algorithms that have been invented for solving parity games is surely due to the fact that many people believe it is the case.

Parity games, classically, are played on directed graphs whose nodes are labeled with priorities (namely, *colors*) and players have perfect information about the adversary moves. The players move in turn a token along the edges of the graph starting from a designated initial node. Thus, a play induces an infinite path and Player 0 wins the play if the greatest priority that is visited infinitely often is even. In the contrary case, it is Player 1 that wins the play. Parity games can express several important system requirements such as *safety* and *liveness* properties. Along an infinite play, safety requirements are used to ensure that nothing "bad" will ever happen, while liveness properties ensure that something "good" eventually happens [AH98]. Often, safety and liveness properties alone are simple

to satisfy, while it becomes a very challenging task when properties of this kind need to be satisfied simultaneously. As an example, assume we want to check the correctness of a printer scheduler that serves two users in which it is required that, whenever a user sends a job to the printer, the job is eventually printed out (liveness property) and that two jobs are never printed simultaneously (safety property). The above liveness property can be written as the LTL [Pnu77] formula $\mathtt{G}(req \rightarrow \mathtt{F}\ grant)$, where G and F stand for the classic temporal operators "always" and "eventually", respectively. This kind of question is also known in literature as a *request-response condition* [HTW08]. As explained above, in a parity game, this requirement is interpreted over an infinite path generated by the interplay of the two players. From a theoretical viewpoint, on checking whether a request is eventually granted, there is no bound on the "waiting time", namely the time elapsed until the job is printed out. In other words, it is enough to check that the system "can" grant the request, while we do not care when it happens. In a real industry scenario, instead, the request is more concrete, *i.e.*, the job must be printed out in a reasonable time bound.

In the last few years, several works have focused on the above timing aspect in system specification. In particular, we have a clear fusion between qualitative and quantitative specification. The qualitative specification concern the ω-regular specification and the quantitative specification regarding the elapsing time to reach a specific goal. In [KPV09], it has been addressed by forcing LTL to express "prompt" requirements, by means of a *prompt* operator $\mathtt{F}_p$ added to the logic. In [AHK10] the automata-theoretic counterpart of the $\mathtt{F}_p$ operator has been studied. *Prompt-Büchi* automata are introduced and it has been showed that their intersection with ω-regular languages is equivalent to co-Büchi. Successively, the prompt semantics has been lifted to ω-regular games, under the parity winning condition [CHH09], by introducing *finitary parity* games. There, the concept of "*distance*" between positions in a play has been introduced and referred as the number of edges traversed to reach a position from a given one. Then, winning positions of the game are restricted to those occurring bounded. To give few more details, first consider that, as in classic parity games, arenas have vertexes equipped with natural number priorities and in a play every odd number met is seen as a pending "*request*" that, to be satisfied, requires to meet a bigger even number afterwards along the play, which is therefore seen as a "*response*". Then, Player 0 wins the game if almost all requests are responded within a bounded distance. It has been shown in [CHH09] that the problem of determining the winner in a finitary parity game is in PTIME.

Recently, the work [CHH09] has been generalized in [FZ12] to deal with more involved prompt parity conditions. For this reason, arenas are further equipped with two kinds of edges, i*-edges* and ϵ*-edges*, which indicate whether there is or not a time-unit consumption while traversing an edge, respectively. Then, the cost of a path is determined by the number of its i-edges. In some way, the cost of traversing a path can be seen as the consumption of resources. Therefore, in such a game, Player 0 aims to achieve its goal with a bounded

resource, while Player 1 tries to avoid it. In particular, Player 0 wins a play if there is a bound b such that all requests, except at most a finite number, have a cost bounded by b and all requests, except at most a finite number, are responded. Since we now have an explicit cost associated to every path, the corresponding condition has been named *cost parity* (CP). Note that in cost parity games a finite number of unanswered requests with unbounded cost is also allowed. By disallowing this, in [FZ12], a strengthening of the cost parity condition has been introduced and named *bounded-cost parity* (BCP) condition. There, it has been shown that the winner of both cost parity and bounded-cost parity can be decided in NPTIME $\cap$ CONPTIME.

In Chapter I of this thesis, we introduce a general study of the concept of promptness in parity games that allows to put under a unique theoretical framework several of the cited variants along with new ones. Also, we describe simple polynomial reductions from all these conditions to either Büchi or parity games, which simplify all previous known procedures. In particular, they allow to lower the complexity class of *cost* and *bounded-cost parity games* recently introduced. Indeed, we provide solution algorithms showing that determining the winner of these games is in UPTIME $\cap$ COUPTIME. Our algorithm reduces the original problem to a unique parity game, which is the key point of how we gain a better result (w.r.t. the complexity class point of view).

Through the years a variety of algorithms for solving parity games has been presented. Among the others, we recall the Zielonka[Zie98] algorithm that yields a recursive approach, the Jurdziński's small progress measures algorithm[Mar00], the strategy improvement algorithm by Jens Vöge and Marcin Jurdziński[Jen00], and the big-step by Schewe[Sve07]. In addition, these algorithms have been also implemented and one of the most important platform containing all of them is PGSolver, written in OCaml [Fri09]. This platform has the merit of having declared the Zielonka algorithm the best performing in practice. In Chapter II, we deeply revisit the implementation of the recursive algorithm introducing several improvements and making use of *Scala Programming Language*. These choices have been proved to be very successful, gaining up to two orders of magnitude in running time.

In the second part of this thesis, we concentrate on games with external quantitative specifications and use them to reason about multi-agent systems. As we did for internal specifications along with parity games, we start recalling some basic concepts. In (qualitative) multi-agent systems verification, different approaches have been taken into consideration. One worth of mention is *Alternating-Time Temporal Logic* (ATL*, for short) [AHK02]. This logic allows to reason about strategies of agents having the satisfaction of temporal goals as a a payoff criterion. Formally, it is obtained as a generalization of CTL*, in which the existential `E` and the universal `A` *path quantifiers* are replaced with *strategic modalities* of the form $\langle\langle \mathrm{A} \rangle\rangle$ and $[\![\mathrm{A}]\!]$, where A is a set of *agents*. Despite its expressiveness, ATL* suffers from the strong limitation that strategies are treated only implicitly. This restriction makes the

logic less suited to formalize several important solution concepts, such as *Nash Equilibrium*. These considerations led to introduce *Strategy Logic* (SL, for short) [CHP07, MMV10], a more powerful formalism for strategic reasoning. As a key aspect, this logic treats strategies as *first-order objects* that can be determined by means of the existential $\langle\langle x \rangle\rangle$ and universal $[\![x]\!]$ quantifiers, which can be respectively read as *"there exists a strategy x"* and *"for all strategies x"*. Remarkably, a strategy in SL is a generic conditional plan that at each step prescribes an action on the base of the history of the play. Such a plan is not intrinsically glued to a specific agent but an explicit binding operator (a, x) allows to link an agent a to the strategy associated with a variable x. A common aspect about all logics mentioned above is that quantifications are either existential or universal. *Per contra*, there are several real scenarios in which "more precise" quantifications are crucially needed (see [BMM12], for an argumentation). This has attracted the interest of the formal verification community to *graded modalities*. They have been first studied in classic modal logic [Fin72] and then exported to the field of *knowledge representation* to allow quantitative bounds on the set of individuals satisfying a certain property. In particular, they are considered as *counting quantifiers* in first-order logics [GOR97] and *number restrictions* in *description logics* [HB91]. The first applications of graded modalities in formal verification concern closed systems. In [KSV02], *graded* μCALCULUS has been introduced in order to express statements about a given number of immediately accessible worlds. Successively in [FNP09b, BMM09, BMM10, BMM12], the notion of graded modalities have been extended to deal with number of paths. Among the others graded CTL (GCTL, for short) has been introduced along with a suitable axiomatization of a counting [BMM12].

In open systems verification, we are aware of just two cases in which graded modalities have been investigated: module checking for graded μCALCULUS [FMP08] and an extension of ATL along with graded path modalities (GATL, for short) [FNP09a]. These two orthogonal approaches have the merit of having introduced a counting over strategies. In particular, while the former involves a counting on one-step moves among two-agents, the latter allows for a more sophisticated counting on the histories of the game in a multi-player setting. Nevertheless, GATL suffers of several limitations. First, not surprisingly, it cannot express powerful game reasonings due to the limitation of its underlining logic ATL. Second, it is based on a very rigid counting of existential strategies only.

In Chapter III of this thesis, we introduce and study *Graded Strategy Logic* (GSL), an extension of *Strategy Logic* (SL) along with *graded quantifiers*. Our aim is introduce a formalism that is able to count the different strategies that an agent has available to verify a given formula. In GSL, by means of the existential construct $\langle\langle x \geq g \rangle\rangle \varphi$ one can state that *there are at least g strategies x satisfying φ*. As different strategies may induce the same outcome, although looking different, they need to be count as one. For this reason, we introduce a suitable equivalence relation over profiles based on the strategic behavior they

induce. We investigate some basic questions over a *vanilla* fragment of GSL. In particular, we report on positive results about the determinacy of turn-based games and the related model-checking problem, which we show to be PTIME-COMPLETE.

For the sake of clarity of exposition, every chapter is a build in a way that is self content. This means that introduction and preliminary concepts and notation are locally defined.

CHAPTER 1

On Promptness in Parity Games

Contents

In this chapter, we study several formalization of two-player parity games, under the prompt semantics, over colored (vertexes) arenas with or without weights over edges. In the sequel, we refer to the latter as *colored arenas* and to the former as *weighted arenas*. Our aim is twofold. On one side, we give a clear picture of all different extended parity conditions introduced in the literature working under the prompt assumption. In particular, we analyze their main intrinsic peculiarities and possibly improve the complexity class results related to the game solutions. On the other side, we introduce new parity conditions to work on both colored and weighted arenas and study their relation with the known ones. For a complete list of all the conditions we address the Table 1.1.

In order to make our reasoning more clear, we first introduce the concept of *non-full*, *semi-full* and *full* acceptance parity conditions. To understand their meaning, first consider again the cost parity condition. By definition, it is a conjunction of two properties and in both of them a finite number of requests (possibly different) can be ignored. For this reason, we call this condition "non-full". Consider now the bounded-cost parity condition. By definition, it is still a conjunction of two properties, but now only in one of them a finite number of requests can be ignored. For this reason, we call this condition "semi-full". Finally, a parity condition is named "full" if none of the requests can be ignored. Note that the full concept has been already addressed in [CHH09] on classic (colored) arenas. We also refer to [CHH09] for further motivations and examples.

As a main contribution in this chapter, we introduce and study three new parity conditions named *full parity* (FP), *prompt parity* (PP) and *full-prompt parity* (FPP) condition, respectively. The full parity condition is defined over colored arenas and, in accordance to the full semantics, it simply requires that all requests must be responded. Clearly, it has no meaning to talk about a semi-full parity condition, as there is just one property to satisfy. Also, the non-full parity condition corresponds to the classic parity one. See Table 1.2 for a schematic view of this argument. We prove that the problem of checking whether player 0 wins under the full parity condition is in PTIME. This result is obtained by a quadratic translation to classic Büchi games. The prompt parity condition, which we consider on both colored and weighted arenas, requires that almost all requests are responded within a bounded cost, which we name here *delay*. The full-prompt parity condition is defined accordingly. Observe that the main difference between the cost parity and the prompt parity conditions is that the former is a conjunction of two properties, in each of which a possibly different set of finite requests can be ignored, while in the latter we indicate only one set of finite requests to be used in two different properties. Nevertheless, since the quantifications of the winning conditions range on co-finite sets, we are able to prove that prompt and cost parity conditions are semantically equivalent. We also prove that the complexity of checking whether player $\exists$ wins the game under the prompt parity condition is UPTIME $\cap$ COUPTIME, in the case of weighted arenas. So, the same result holds for cost parity games and this improves the

previously known NPTIME ∩ CONPTIME result [FZ12]. The statement is obtained by a quartic translation to classic parity games. Our algorithm reduces the original problem to a unique parity game, which is the key point of how we gain a better result (w.r.t. the complexity class point of view). Obviously, this is different from what is done in [FZ12], as the algorithm there performs several calls to a parity game solver and from this approach we are not able to derive a parsimonious reduction which is necessary for the UPTIME ∩ COUPTIME result. Observe that, on colored arenas, prompt and full-prompt parity conditions correspond to the finitary and bounded-finitary parity conditions [CHH09], respectively. Hence, both the corresponding games can be decided in PTIME. We prove that for full-prompt parity games the PTIME complexity holds even in the case the arenas are weighted. Finally, by means of a cubic translation to classic parity games, we prove that bounded-cost parity over weighted arenas is in UPTIME ∩ COUPTIME, which also improves the previously known NPTIME ∩ CONPTIME result [FZ12] about this condition.

1.1 Preliminaries

In this section, we describe the concepts of two-player turn-based arena, payoff-arena, and game.

1.1.1 Arenas

An *arena* is a tuple $\mathcal{A} \triangleq \langle \mathrm{Ps}_\exists, \mathrm{Ps}_\forall, Mv \rangle$, where $\mathrm{Ps}_\exists$ and $\mathrm{Ps}_\forall$ are the disjoint sets of *existential* and *universal positions* and $Mv \subseteq \mathrm{Ps} \times \mathrm{Ps}$ is the left-total *move relation* on $\mathrm{Ps} \triangleq \mathrm{Ps}_\exists \cup \mathrm{Ps}_\forall$. The *order* of $\mathcal{A}$ is the number $|\mathcal{A}| \triangleq |\mathrm{Ps}|$ of its positions. An arena is *finite* iff it has finite order. A *path* (resp., *history*) in $\mathcal{A}$ is an infinite (resp., finite non-empty) sequence of vertexes $\pi \in \mathrm{Pth} \subseteq \mathrm{Ps}^\omega$ (resp., $\rho \in \mathrm{Hst} \subseteq \mathrm{Ps}^+$) compatible with the move relation, *i.e.*, $(\pi_i, \pi_{i+1}) \in Mv$ (resp., $(\rho_i, \rho_{i+1}) \in Mv$), for all $i \in \mathbb{N}$ (resp., $i \in [0, |\rho| - 1[$), where Pth (resp., Hst) denotes the set of all paths (resp., histories). Intuitively, histories and paths are legal sequences of reachable positions that can be seen, respectively, as partial and complete descriptions of possible outcomes obtainable by following the rules of the game modeled by the arena. An *existential* (resp., *universal*) *history* in $\mathcal{A}$ is just a history $\rho \in \mathrm{Hst}_\exists \subseteq \mathrm{Hst}$ (resp., $\rho \in \mathrm{Hst}_\forall \subseteq \mathrm{Hst}$) ending in an existential (resp., universal) position, *i.e.*, $\mathsf{lst}(\rho) \in \mathrm{Ps}_\exists$ (resp., $\mathsf{lst}(\rho) \in \mathrm{Ps}_\forall$). An *existential* (resp., *universal*) *strategy* on $\mathcal{A}$ is a function $\sigma_\exists \in \mathrm{Str}_\exists \subseteq \mathrm{Hst}_\exists \to \mathrm{Ps}$ (resp., $\sigma_\forall \in \mathrm{Str}_\forall \subseteq \mathrm{Hst}_\forall \to \mathrm{Ps}$) mapping each existential (resp., universal) history $\rho \in \mathrm{Hst}_\exists$ (resp., $\rho \in \mathrm{Hst}_\forall$) to a position compatible with the move relation, *i.e.*, $(\mathsf{lst}(\rho), \sigma_\exists(\rho)) \in Mv$ (resp., $(\mathsf{lst}(\rho), \sigma_\forall(\rho)) \in Mv$), where $\mathrm{Str}_\exists$ (resp., $\mathrm{Str}_\forall$) denotes the set of all existential (resp., universal) strategies. Intuitively, a strategy is a high-level plan for a player to achieve his own goal, which contains the choice of moves as a function of the histories of the current outcome. A path $\pi \in \mathrm{Pth}(v)$ starting at a position

$v \in \mathrm{Ps}$ is the *play* in $\mathcal{A}$ *w.r.t.* a pair of strategies $(\sigma_\exists, \sigma_\forall) \in \mathrm{Str}_\exists \times \mathrm{Str}_\forall$ ($((\sigma_\exists, \sigma_\forall), v)$-*play*, for short) iff, for all $i \in \mathbb{N}$, it holds that if $\pi_i \in \mathrm{Ps}_\exists$ then $\pi_{i+1} = \sigma_\exists(\pi_{\leq i})$ else $\pi_{i+1} = \sigma_\forall(\pi_{\leq i})$. Intuitively, a play is the unique outcome of the game given by the player strategies. The *play function* $\mathsf{play} : \mathrm{Ps} \times (\mathrm{Str}_\exists \times \mathrm{Str}_\forall) \to \mathrm{Pth}$ returns, for each position $v \in \mathrm{Ps}$ and pair of strategies $(\sigma_\exists, \sigma_\forall) \in \mathrm{Str}_\exists \times \mathrm{Str}_\forall$, the $((\sigma_\exists, \sigma_\forall), v)$-play $\mathsf{play}(v, (\sigma_\exists, \sigma_\forall))$.

1.1.2 Payoff Arenas

A *payoff arena* is a tuple $\widehat{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Pf}, \mathsf{pf} \rangle$, where $\mathcal{A}$ is the underlying arena, Pf is the non-empty set of *payoff values*, and $\mathsf{pf} : \mathrm{Pth} \to \mathrm{Pf}$ is the *payoff function* mapping each path to a value. The *order* of $\widehat{\mathcal{A}}$ is the order of its underlying arena $\mathcal{A}$. A payoff arena is *finite* iff it has finite order. The overloading of the payoff function pf from the set of paths to the sets of positions and pairs of existential and universal strategies induces the function $\mathsf{pf} : \mathrm{Ps} \times (\mathrm{Str}_\exists \times \mathrm{Str}_\forall) \to \mathrm{Pf}$ mapping each position $v \in \mathrm{Ps}$ and pair of strategies $(\sigma_\exists, \sigma_\forall) \in \mathrm{Str}_\exists \times \mathrm{Str}_\forall$ to the payoff value $\mathsf{pf}(v, (\sigma_\exists, \sigma_\forall)) \triangleq \mathsf{pf}(\mathsf{play}(v, (\sigma_\exists, \sigma_\forall)))$ of the corresponding $((\sigma_\exists, \sigma_\forall), v)$-play.

1.1.3 Games

A *(extensive-form) game* is a tuple $\mathfrak{I} \triangleq \langle \widehat{\mathcal{A}}, \mathrm{Wn}, v_\mathrm{o} \rangle$, where $\widehat{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Pf}, \mathsf{pf} \rangle$ is the underlying payoff arena, $\mathrm{Wn} \subseteq \mathrm{Pf}$ is the *winning payoff set*, and $v_\mathrm{o} \in \mathrm{Ps}$ is the designated *initial position*. The *order* of $\mathcal{G}$ is the order of its underlying payoff arena $\widehat{\mathcal{A}}$. A game is *finite* iff it has finite order. The *existential* (resp., *universal*) *player* $\exists$ (resp., $\forall$) wins the game $\mathfrak{I}$ iff there exists an existential (resp., universal) strategy $\sigma_\exists \in \mathrm{Str}_\exists$ (resp., $\sigma_\forall \in \mathrm{Str}_\forall$) such that, for all universal (resp., existential) strategies $\sigma_\forall \in \mathrm{Str}_\forall$ (resp., $\sigma_\exists \in \mathrm{Str}_\forall$), it holds that $\mathsf{pf}(\sigma_\exists, \sigma_\forall) \in \mathrm{Wn}$ (resp., $\mathsf{pf}(\sigma_\exists, \sigma_\forall) \notin \mathrm{Wn}$). For sake of clarity, given a game $\mathfrak{I}$ we denote with $\mathrm{Pth}(\mathfrak{I})$ the set of all paths in $\mathfrak{I}$ and with $\mathrm{Str}_\exists(\mathfrak{I})$ and $\mathrm{Str}_\forall(\mathfrak{I})$ the sets of strategies over $\mathfrak{I}$ for the player $\exists$ and $\forall$, respectively. Also, we indicate by $\mathrm{Hst}(\mathfrak{I})$ the set of the histories over $\mathfrak{I}$.

1.2 Parity Conditions

In this section, we give an overview about all different parity conditions we consider in this article, which are variants of classical parity games that will be investigated over both classic colored arenas (*i.e.*, with unweighted edges) and weighted arenas. Specifically, along with the known Parity (P), Cost Parity (CP), and Bounded-Cost Parity (BCP) conditions, we introduce three new winning conditions, namely Full Parity (FP), Prompt Parity (PP), and Full-Prompt Parity (FPP).

Before continuing, we introduce some notation to formally define all addressed winning conditions. A *colored arena* is a tuple $\widetilde{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$, where $\mathcal{A}$ is the underlying arena,

	Non-Prompt	Prompt
Non-Full	Parity (P)	Prompt Parity (PP) $\equiv$ Cost Parity (CP)
Semi-Full	$-$	Bounded Cost Parity (BCP)
Full	Full Parity (FP)	Full Prompt Parity (FPP)

Table 1.1: Prompt/non-prompt conditions under the full/semi-full/non-full constraints.

$\mathrm{Cl} \subseteq \mathbb{N}$ is the non-empty sets of *colors*, and $\mathsf{cl} : \mathrm{Ps} \to \mathrm{Cl}$ is the *coloring function* mapping each position to a color. Similarly, a *(colored) weighted arena* is a tuple $\overline{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$, where $\langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ is the underlying colored arena, $\mathrm{Wg} \subseteq \mathbb{N}$ is the non-empty sets of *weights*, and $\mathsf{wg} : Mv \to \mathrm{Wg}$ is the *weighting functions* mapping each move to a weight. The overloading of the coloring (resp., weighting) function from the set of positions (resp., moves) to the set of paths induces the function $\mathsf{cl} : \mathrm{Pth} \to \mathrm{Cl}^\omega$ (resp., $\mathsf{wg} : \mathrm{Pth} \to \mathrm{Wg}^\omega$) mapping each path $\pi \in \mathrm{Pth}$ to the infinite sequence of colors $\mathsf{cl}(\pi) \in \mathrm{Cl}^\omega$ (resp. weights $\mathsf{wg}(\pi) \in \mathrm{Wg}^\omega$) such that $(\mathsf{cl}(\pi))_i = \mathsf{cl}(\pi_i)$ (resp., $(\mathsf{wg}(\pi))_i = \mathsf{wg}((\pi_i, \pi_{i+1}))$), for all $i \in \mathbb{N}$. Every colored (resp., weighted) arena $\widetilde{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ (resp., $\overline{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$) induces a canonical payoff arena $\widehat{\mathcal{A}} \triangleq \langle \mathcal{A}, \mathrm{Pf}, \mathsf{pf} \rangle$, where $\mathrm{Pf} \triangleq \mathrm{Cl}^\omega$ (resp., $\mathrm{Pf} \triangleq \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$) and $\mathsf{pf}(\pi) \triangleq \mathsf{cl}(\pi)$ (resp., $\mathsf{pf}(\pi) \triangleq (\mathsf{cl}(\pi), \mathsf{wg}(\pi))$).

In the following, along a play, we interpret the occurrence of an odd priority as a "*request*" and the occurrence of the first bigger even priority at a later position as a "*response*". Then, we distinguish between *prompt* and *not-prompt* requests. In the not-prompt case, a request is responded independently from the elapsed time between its occurrence and response. Conversely, in the prompt case, the time within a request is responded has an important role. It is for this reason that we consider weighted arenas. So, a *delay* over a play is the sum of the weights over of all the edges crossed from a request to its response. We now formalize these concepts. Let $c \in \mathrm{Cl}^\omega$ be an infinite sequence of colors. Then, $\mathrm{Rq}(c) \triangleq \{i \in \mathbb{N} : c_i \equiv 1 \pmod 2\}$ denotes the set of all *requests* in c and $\mathrm{rs}(c, i) \triangleq \mathsf{min}\{j \in \mathbb{N} : i \leq j \wedge c_i \leq c_j \wedge c_j \equiv 0 \pmod 2\}$ represents the *response* to the requests $i \in \mathrm{Rs}$, where by convention we set $\mathsf{min}\emptyset \triangleq \omega$. Moreover, $\mathrm{Rs}(c) \triangleq \{i \in \mathrm{Rq}(c) : \mathrm{rs}(c, i) < \omega\}$ denotes the subset of all requests for which a response is provided. Now, let $w \in \mathrm{Wg}^\omega$ be an infinite sequence of weights. Then, $\mathrm{dl}((c, w), i) \triangleq \sum_{k=i}^{\mathrm{rs}(c,i)-1} w_k$ denotes the *delay w.r.t.* w within which a request $i \in \mathrm{Rq}(c)$ is responded. Also, $\mathrm{dl}((c, w), \mathrm{R}) \triangleq \mathsf{sup}_{i \in \mathrm{R}} \mathrm{dl}((c, w), i)$ is the supremum of all delays of the requests contained in $\mathrm{R} \subseteq \mathrm{Rq}(c)$.

As usual, all conditions we consider are given on infinite plays. Then, the winning of the game can be defined *w.r.t.* how often the characterizing properties of the winning condition are satisfied along each play. For example, we may require that *all* requests have to be responded along a play, which we denote as a *full* behavior of the acceptance condition. Also, we may require that the condition (given as a unique or a *conjunction* of properties) holds almost

everywhere along the play (*i.e.*, a finite number of places along the play can be ignored), which we denote as a *not-full* behavior of the acceptance condition. More in general, we may have conditions, given as a *conjunction* of several properties, to be satisfied in a mixed way, *i.e.*, some of them have to be satisfied almost everywhere and the remaining ones, over all the play. We denote the latter as a *semi-full* behavior of the acceptance condition. Table 1.1 reports the combination of the full, not-full, and semi-full behaviors with the known conditions of parity, cost-parity and bounded cost-parity and the new condition of prompt-parity we introduce. As it will be clear in the following, bounded cost-parity has intrinsically a semi-full behavior on weighted arenas, but it has no meaning on (unweighted) colored arenas. Also, over colored arenas, the parity condition has an intrinsic not-full behavior. As far as we known, some of these combinations have never been studied previously on colored arenas (full parity) and weighted arenas (prompt parity and full-prompt parity).

Observe that, in the following, in each graphic representation of a game, the circular nodes belong to player $\exists$ while the square nodes to player $\forall$.

1.2.1 Non-Prompt Conditions

The non-prompt conditions relate only to the satisfaction of a request (*i.e.*, its response), without taking into account the elapsing of time before the response is provided (*i.e.*, its delay). As reported in Table 1.1, here we consider as non-prompt conditions, those ones of parity and full parity. To do this, let $\mathfrak{D} \triangleq \langle \widehat{\mathcal{A}}, \mathrm{Wn}, v_0 \rangle$ be a game, where the payoff arena $\widehat{\mathcal{A}}$ is induced by a colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$.

Parity condition *(*P*)* $\mathfrak{D}$ is a *parity game* iff it is played under a parity condition, which requires that all requests, except at most a finite number, are responded. Formally, for all $c = \mathrm{Cl}^\omega$, we have that $c \in \mathrm{Wn}$ iff there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c)$ such that $\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$, *i.e.*, c is a winning payoff iff almost all requests in $\mathrm{Rq}(c)$ are responded. Consider for example the colored arena $\widetilde{\mathcal{A}_1}$ depicted in Figure 1.1, where all positions are universal, and let $\alpha + \beta$ be the regular expression describing all possible plays starting at v_0, where $\alpha = (v_0 \cdot v_1^* \cdot v_2) \cdot v_0 \cdot v_1^\omega$ and $\beta = (v_0 \cdot v_1^* \cdot v_2)^\omega$. Now, keep a path $\pi \in \alpha$ and let $c_\alpha \triangleq \mathsf{pf}(\pi) \in (1 \cdot 0^* \cdot 2) \cdot 1 \cdot 0^\omega$ be its payoff. Then, $c_\pi \in \mathrm{Wn}$, since the parity condition is satisfied by putting in R the last index in which the color 1 occurs in c_π. Again, keep a path $\pi \in \beta$ and let $c_\pi \triangleq \mathsf{pf}(\pi) \in (1 \cdot 0^* \cdot 2)^\omega$ be its payoff. Then, $c_\pi \in \mathrm{Wn}$, since the parity condition is satisfied by simply choosing $\mathrm{R} \triangleq \emptyset$. In the following, as a special case, we also consider parity games played over arenas colored only with the two priorities 1 and 2, to which we refer as *Büchi games* (B).

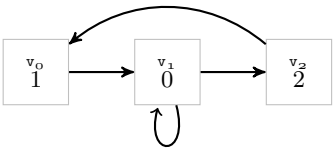

Figure 1.1: Colored Arena $\widetilde{\mathcal{A}_1}$.

Full Parity condition *(*FP*)* $\Game$ is a *full parity game* iff it is played under a full parity condition, which requires that all requests are responded. Formally, for all $c \in \mathrm{Cl}^\omega$, we have that $c \in \mathrm{Wn}$ iff $\mathrm{Rq}(c) \subseteq \mathrm{Rs}(c)$ *i.e.*, c is a winning payoff iff all requests in $\mathrm{Rq}(c)$ are responded. Consider for example the colored arena $\widetilde{\mathcal{A}_2}$ in Figure 1.2, where all positions are existential. There is a unique path $\pi = (v_0 \cdot v_1)^\omega$ starting at v_0 having payoff $c_\pi \triangleq \mathsf{pf}(\pi) = (1 \cdot 2)^\omega$ and set of requests $\mathrm{Rq}(c_\pi) = \{2n : n \in \mathbb{N}\}$. Then, $c_\pi \in \mathrm{Wn}$, since the full parity condition is satisfied as all requests are responded by the color 2 at the odd indexes. Observe that the arena of the game $\widetilde{\mathcal{A}_1}$ depicted in Figure 1.1 is not won under the *full parity* condition. Indeed, if we consider the path π with payoff $\mathsf{pf}(\pi) \in (1 \cdot 0^\omega)$, it holds that not all requests are responded.

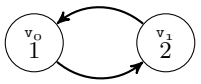

Figure 1.2: Colored Arena $\widetilde{\mathcal{A}_2}$.

1.2.2 Prompt Conditions

The prompt conditions take into account, in addition to the satisfaction of a request, also the delay before it occurs. As reported in Table 1.1, here we consider as prompt conditions, those ones of prompt parity, full-prompt parity, cost parity, and bounded-cost parity. To do this, let $\Game \triangleq \langle \widehat{\mathcal{A}}, \mathrm{Wn}, v_0 \rangle$ be a game, where the payoff arena $\widehat{\mathcal{A}}$ is induced by a (colored) weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$.

Prompt Parity condition *(*PP*)* $\Game$ is a *prompt parity game* iff it is played under a prompt parity condition, which requires that all requests, except at most a finite number of them, are responded with a bounded delay. Formally, for all $(c, w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$, we have that $(c, w) \in \mathrm{Wn}$ iff there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c)$ such that $\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$ and there exists a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c, w), \mathrm{Rq}(c) \setminus \mathrm{R}) \leq b$ holds, *i.e.*, (c, w) is a winning payoff iff almost all requests in $\mathrm{Rq}(c)$ are responded with a delay bounded by an a priori number b. Consider for example the weighted arena $\overline{\mathcal{A}_3}$ depicted in Figure 1.3. There is a unique path $\pi = v_0 \cdot (v_1 \cdot v_2)^\omega$ starting at v_0 having payoff $p_\pi \triangleq \mathsf{pf}(\pi) = (c_\pi, w_\pi)$, where $c_\pi = 3 \cdot (1 \cdot 2)^\omega$ and $w_\pi = 2 \cdot (1 \cdot 0)^\omega$, and set of requests $\mathrm{Rq}(c_\pi) = \{0\} \cup \{2n + 1 : n \in \mathrm{N}\}$. Then, $p_\pi \in \mathrm{Wn}$, since the prompt parity condition is satisfied by choosing $\mathrm{R} = \{0\}$ and $b = 1$.

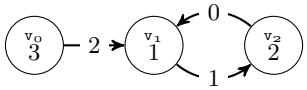

Figure 1.3: Weighted Arena $\overline{\mathcal{A}_3}$.

Full-Prompt Parity condition *(*FPP*)* $\Game$ is a *full-prompt parity game* iff it is played under a full-prompt parity condition, which requires that all requests are responded with a bounded delay. Formally, for all $(c, w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$, we have that $(c, w) \in \mathrm{Wn}$ iff $\mathrm{Rq}(c) = \mathrm{Rs}(c)$

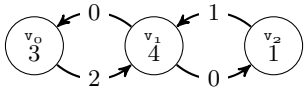

Figure 1.4: Weighted Arena $\overline{\mathcal{A}_4}$.

and there exists a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c,w),\mathrm{Rq}(c)) \leq b$ holds, *i.e.*, (c,w) is a winning payoff iff all requests in $\mathrm{Rq}(c)$ are responded with a delay bounded by an a priori number b. Consider for example the weighted arena $\overline{\mathcal{A}_4}$ depicted in Figure 1.4. Now, take a path $\pi \in v_0 \cdot v_1 \cdot ((v_0 \cdot v_1)^* \cdot (v_2 \cdot v_1)^*)^\omega$ starting at v_0 and let $p_\pi \triangleq \mathsf{pf}(\pi) = (c_\pi, w_\pi)$ be its payoff, with $c_\pi \in 3 \cdot 4 \cdot ((3 \cdot 4)^* \cdot (1 \cdot 4)^*)^\omega$ and $w_\pi \in 2 \cdot ((0 \cdot 2)^* \cdot (0 \cdot 1)^*)^\omega$. Then, $p_\pi \in \mathrm{Wn}$, since the full-prompt parity condition is satisfied as all requests are responded by color 4 with a delay bound $b = 2$. Observe that, the arena of the game $\widetilde{\mathcal{A}_3}$ depicted in Figure 1.3 is not won under the *full prompt parity* condition. Indeed, if we consider the unique path $\pi = v_0 \cdot (v_1 \cdot v_2)^\omega$ starting at v_0 having payoff $p_\pi \triangleq \mathsf{pf}(\pi) = (c_\pi, w_\pi)$, where $c_\pi = 3 \cdot (1 \cdot 2)^\omega$ and $w_\pi = 2 \cdot (1 \cdot 0)^\omega$, it holds that there exists an unanswered request at the vertex v_0.

Remark 1.2.1 *As a special case, the prompt and the full-prompt parity conditions can be analyzed on simply colored arenas, by considering each edge as having weight* 1*. Then, the two cases just analyzed correspond to the finitary parity and bounded parity conditions studied in [CHH09].*

Cost Parity condition *(*CP*) [FZ12]* $\supset$ is a *cost parity game* iff it is played under a cost parity condition, which requires that all requests, except at most a finite number of them, are responded and all requests, except at most a finite number of them (possibly different from the previous ones) have a bounded delay. Formally, for all $(c,w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$, we have that $(c,w) \in \mathrm{Wn}$ iff there is a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c)$ such that $\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$ and there exist a finite set $\mathrm{R}' \subseteq \mathrm{Rq}(c)$ and a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c,w),\mathrm{Rq}(c) \setminus \mathrm{R}') \leq b$ holds, *i.e.*, (c,w) is a winning payoff iff almost all requests in $\mathrm{Rq}(c)$ are responded and almost all have a delay bounded by an a priori number b. Consider for example the weighted arena $\overline{\mathcal{A}_5}$ in Figure 1.5. There is a unique path $\pi = v_0 \cdot {v_1}^\omega$ starting at v_0 having payoff $p_\pi \triangleq \mathsf{pf}(\pi) = (c_\pi, w_\pi)$, where $c_\pi = 1 \cdot 0^\omega$ and $w_\pi = 1^\omega$, and set of requests $\mathrm{Rq}(c_\pi) = \{0\}$. Then, $p_\pi \in \mathrm{Wn}$, since the prompt parity condition is satisfied with $\mathrm{R} = \mathrm{R}' = \{0\}$ and $b = 0$.

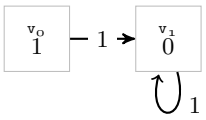

Figure 1.5: Weighted Arena $\overline{\mathcal{A}_5}$.

Bounded-Cost Parity condition *(*BCP*) [FZ12]* $\supset$ is a *bounded-cost parity game* iff it is played under a bounded-cost parity condition, which requires that all requests, except at most a finite number, are responded and all have a bounded delay. Formally, for all $(c,w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$, we have that $(c,w) \in \mathrm{Wn}$ iff there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c)$ such that $\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$ and there exists a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c,w),\mathrm{Rq}(c)) \leq b$ holds, *i.e.*, (c,w) is a winning payoff iff almost all requests in $\mathrm{Rq}(c)$ are responded and all have a delay bounded by an a priori number b. Consider for example the

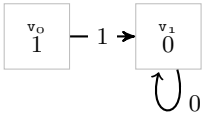

Figure 1.6: Weighted Arena $\overline{\mathcal{A}_6}$.

weighted arena $\overline{\mathcal{A}_6}$ depicted in Figure 1.6. There is a unique path $\pi = v_0 \cdot {v_1}^\omega$ starting at v_0 having payoff $p_\pi \triangleq \mathsf{pf}(\pi) = (c_\pi, w_\pi)$, where $c_\pi = 1 \cdot 0^\omega$, and set of requests $\mathrm{Rq}(c_\pi) = \{0\}$. Then, $p_\pi \in \mathrm{Wn}$, since the prompt parity condition is satisfied with $\mathrm{R} = \{0\}$ and $b = 1$.

Wn	Formal definitions		
P	$\forall c \in \mathrm{Cl}^\omega.\ c \in \mathrm{Wn}$ iff	$\exists \mathrm{R} \subseteq \mathrm{Rq}(c), \lvert\mathrm{R}\rvert < \omega.$	$\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$
FP			$\mathrm{Rq}(c) = \mathrm{Rs}(c)$
PP	$\forall (c,w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega.$ $(c,w) \in \mathrm{Wn}$ iff	$\exists \mathrm{R} \subseteq \mathrm{Rq}(c), \lvert\mathrm{R}\rvert < \omega.$	$\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c) \wedge$ $\exists b \in \mathbb{N}\,.\,\mathrm{dl}((c,w), \mathrm{Rq}(c) \setminus \mathrm{R}) \leq b$
FPP			$\mathrm{Rq}(c) = \mathrm{Rs}(c) \wedge$ $\exists b \in \mathbb{N}\,.\,\mathrm{dl}((c,w), \mathrm{Rq}(c)) \leq b$
CP		$\exists \mathrm{R} \subseteq \mathrm{Rq}(c), \lvert\mathrm{R}\rvert < \omega.$ $\exists \mathrm{R}' \subseteq \mathrm{Rq}(c), \lvert\mathrm{R}'\rvert < \omega.$	$\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c) \wedge$ $\exists b \in \mathbb{N}\,.\,\mathrm{dl}((c,w), \mathrm{Rq}(c) \setminus \mathrm{R}') \leq b$
BCP		$\exists \mathrm{R} \subseteq \mathrm{Rq}(c), \lvert\mathrm{R}\rvert < \omega.$	$\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c) \wedge$ $\exists b \in \mathbb{N}\,.\,\mathrm{dl}((c,w), \mathrm{Rq}(c)) \leq b$

Table 1.2: Summary of all winning condition (Wn) definitions.

In Table 1.2, we list all winning conditions (Wn) introduced above, along with their respective formal definitions. For the sake of readability, given a game $\mathcal{D} = \langle \widehat{\mathcal{A}}, \mathrm{Wn}, v_0 \rangle$, we sometimes use the winning condition acronym name in place of Wn, as well as we refer to $\mathcal{D}$ as a Wn game. For example, if $\mathcal{D}$ is a parity game, we also say that it is a P game, as well as, write $\mathcal{D} = \langle \widehat{\mathcal{A}}, \mathrm{P}, v_0 \rangle$.

1.3 Equivalences and Implications

In this section, we investigate the relationships among all parity conditions discussed above. For the sake of coherence, we use the names $\mathcal{A}$, $\widehat{\mathcal{A}}$, $\widetilde{\mathcal{A}}$ and $\overline{\mathcal{A}}$ to refer to arenas, payoff arenas, colored arenas and weighted arenas, respectively.

1.3.1 Positive Relationships

In this subsection, we prove all positive existing relationships among the studied conditions and report them in Figure 1.7, where an arrow from a condition Wn_1 to another condition Wn_2 means that the former implies the latter. In other words, if player $\exists$ wins a game under the condition Wn_1, then it also wins the game under the condition

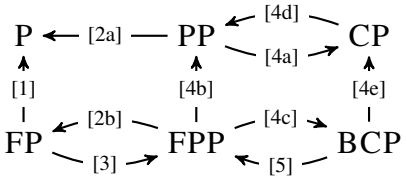

Figure 1.7: Implication Schema.

Wn_2, over the same arena. The label on the edges indicates the item of the next theorem in which the result is proved. In particular, we show that prompt parity and cost parity are semantically equivalent. The same holds for full parity and full prompt parity over finite arenas and for full prompt parity and bounded cost parity on positive weighted arenas. Also, as one may expect, fullness implies not-fullness under every condition and all conditions imply the parity one.

Theorem 1.3.1 *Let $\Game_1 = \langle \widehat{\mathcal{A}_1}, \mathrm{Wn}_1, v_0 \rangle$ and $\Game_2 = \langle \widehat{\mathcal{A}_2}, \mathrm{Wn}_2, v_0 \rangle$ be two games defined on the payoff arenas $\widehat{\mathcal{A}_1}$ and $\widehat{\mathcal{A}_2}$ having the same underlying arena $\mathcal{A}$. Then, player $\exists$ wins $\Game_2$ if it wins $\Game_1$ under the following constraints:*

1. *$\widehat{\mathcal{A}_1} = \widehat{\mathcal{A}_2}$ are induced by a colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ and $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FP}, \mathrm{P})$;*
2. *$\widehat{\mathcal{A}_1}$ and $\widehat{\mathcal{A}_2}$ are induced, respectively, by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ and its underlying colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ and*
   - *(a) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{PP}, \mathrm{P})$, or*
   - *(b) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FPP}, \mathrm{FP})$;*
3. *$\widehat{\mathcal{A}_2}$ and $\widehat{\mathcal{A}_1}$ are finite and induced, respectively, by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ and its underlying colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ and $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FP}, \mathrm{FPP})$;*
4. *$\widehat{\mathcal{A}_1} = \widehat{\mathcal{A}_2}$ are induced by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ and*
   - *(a) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{PP}, \mathrm{CP})$, or*
   - *(b) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FPP}, \mathrm{PP})$, or*
   - *(c) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FPP}, \mathrm{BCP})$, or*
   - *(d) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{CP}, \mathrm{PP})$, or*
   - *(e) $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{BCP}, \mathrm{CP})$;*
5. *$\widehat{\mathcal{A}_1} = \widehat{\mathcal{A}_2}$ are induced by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$, with $\mathsf{wg}(v) > 0$ for all $v \in \mathrm{Ps}$, and $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{BCP}, \mathrm{FPP})$.*

Proof. All items, but 3, 4d, and 5, are immediate by definition. So, we only focus on the remaining ones.

[Item 3] Suppose by contradiction that player $\exists$ wins the FP $\Game_1$ game but it does not win the FPP game $\Game_2$. Then, there is a play π in $\Game_2$ having payoff $(c, w) = \mathsf{pf}(\pi) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$ for which $\mathrm{dl}((c, w), \mathrm{Rq}(c)) = \omega$. So, there exists at least a request $r \in \mathrm{Rq}(c)$ with a delay greater than $s = \sum_{e \in Mv} \mathsf{wg}(e)$. Since the arena is finite, this implies that, on the infix of π that goes from the request r to its response, there is a move that occurs twice. So, player $\forall$

has the possibility to force another play π' having r as request and passing infinitely often through this move without reaching the response. But this is impossible, since player $\exists$ wins the FP game $\Game_1$.

[Item 4d] To prove this item, we show that if a payoff $(c, w) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$ satisfies the CP condition then it also satisfies the PP one. Indeed, by definition, there are a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c)$ such that $\mathrm{Rq}(c) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c)$ and a possibly different finite set $\mathrm{R}' \subseteq \mathrm{Rq}(c)$ for which there is a bound $b \in \mathbb{N}$ such that $\mathrm{dl}((c, w), \mathrm{Rq}(c) \setminus \mathrm{R}') \leq b$. Now, consider the union $\mathrm{R}'' \triangleq \mathrm{R} \cup \mathrm{R}'$. Obviously, this is a finite set. Moreover, it is immediate to see that $\mathrm{Rq}(c) \setminus \mathrm{R}'' \subseteq \mathrm{Rs}(c)$ and $\mathrm{dl}((c, w), \mathrm{Rq}(c) \setminus \mathrm{R}'') \leq b$, for the same bound b. So, the payoff (c, w) satisfies the PP condition, by using R'' in place of R in the definition.

[Item 5] Suppose by contradiction that player $\exists$ wins the BCP game $\Game_1$ but it does not win the FPP game $\Game_2$. Then, there is a play π in $\Game_2$ having payoff $(c, w) = \mathsf{pf}(\pi) \in \mathrm{Cl}^\omega \times \mathrm{Wg}^\omega$ for which $\mathrm{Rq}(c) \neq \mathrm{Rs}(c)$. So, since all weights are positive, there exists at least a request $r \in \mathrm{Rq}(c) \setminus \mathrm{Rs}(c) \neq \emptyset$ with $\mathrm{dl}((c, w), r) = \omega$. But this is impossible. □

The following three corollaries follow as immediate consequences of, respectively, Items 2b and 3, 4a and 4d, and 4c and 5 of the previous theorem.

Corollary 1.3.1 *Let $\Game_{\mathrm{FPP}} = \langle \widehat{\mathcal{A}_{\mathrm{FPP}}}, \mathrm{FPP}, v_0 \rangle$ be an* FPP *game and $\Game_{\mathrm{FP}} = \langle \widehat{\mathcal{A}_{\mathrm{FP}}}, \mathrm{FP}, v_0 \rangle$ an* FP *one defined on the two finite payoff arenas $\widehat{\mathcal{A}_{\mathrm{FPP}}}$ and $\widehat{\mathcal{A}_{\mathrm{FP}}}$ induced, respectively, by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ and its underlying colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$. Then, player $\exists$ wins $\Game_{\mathrm{FPP}}$ if it wins $\Game_{\mathrm{FP}}$.*

Corollary 1.3.2 *Let $\Game_{\mathrm{CP}} = \langle \widehat{\mathcal{A}}, \mathrm{CP}, v_0 \rangle$ be a* CP *game and $\Game_{\mathrm{PP}} = \langle \widehat{\mathcal{A}}, \mathrm{PP}, v_0 \rangle$ a* PP *one defined on the payoff arena $\widehat{\mathcal{A}}$ induced by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$. Then, player $\exists$ wins $\Game_{\mathrm{CP}}$ if it wins $\Game_{\mathrm{PP}}$.*

Corollary 1.3.3 *Let $\Game_{\mathrm{BCP}} = \langle \widehat{\mathcal{A}}, \mathrm{BCP}, v_0 \rangle$ be a* BCP *game and $\Game_{\mathrm{FPP}} = \langle \widehat{\mathcal{A}}, \mathrm{FPP}, v_0 \rangle$ an* FPP *one defined on the payoff arena $\widehat{\mathcal{A}}$ induced by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$, where $\mathsf{wg}(v) > 0$, for all $v \in \mathrm{Ps}$. Then, player $\exists$ wins $\Game_{\mathrm{BCP}}$ if it wins $\Game_{\mathrm{FPP}}$.*

1.3.2 Negative Relationships

In this subsection, we show a list of counterexamples to point out that some winning conditions are not equivalent to other ones. We report the corresponding result in Figure 1.8, where an arrow from a condition Wn_1 to another condition Wn_2 means that there exists an arena on which player $\exists$ wins a Wn_1 game while it loses a Wn_2 one. The label on the edges indicates the item of the next theorem in which the result is

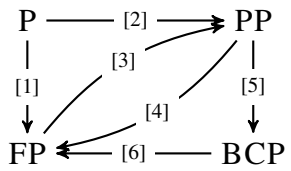

Figure 1.8: Counterexample Schema.

proved. Moreover, the following list of counter-implications, non reported in the figure, can be simply obtained by the reported ones together with the implication results of Theorem 1.3.1: (P, FPP), (P, CP), (P, BCP), (FP, FPP), (FP, CP), (FP, BCP), (PP, FPP), (CP, FP), (CP, FPP), (CP, BCP), and (BCP, FPP).

Theorem 1.3.2 *There exist two games* $\Game_1 = \langle \widehat{\mathcal{A}_1}, \mathrm{Wn}_1, v_0 \rangle$ *and* $\Game_2 = \langle \widehat{\mathcal{A}_2}, \mathrm{Wn}_2, v_0 \rangle$*, defined on the two payoff arenas* $\widehat{\mathcal{A}_1}$ *and* $\widehat{\mathcal{A}_2}$ *having the same underlying arena* $\mathcal{A}$*, such that player* $\exists$ *wins* $\Game_1$ *while it loses* $\Game_2$ *under the following constraints:*

1. $\widehat{\mathcal{A}_1} = \widehat{\mathcal{A}_2}$ *are induced by a colored arena* $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{P}, \mathrm{FP})$*;*
2. $\widehat{\mathcal{A}_2}$ *and* $\widehat{\mathcal{A}_1}$ *are induced, respectively, by a weighted arena* $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ *and its underlying colored arena* $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{P}, \mathrm{PP})$*;*
3. $\widehat{\mathcal{A}_2}$ *and* $\widehat{\mathcal{A}_1}$ *are infinite and induced, respectively, by a weighted arena* $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ *and its underlying colored arena* $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{FP}, \mathrm{PP})$*;*
4. $\widehat{\mathcal{A}_1}$ *and* $\widehat{\mathcal{A}_2}$ *are induced, respectively, by a weighted arena* $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ *and its underlying colored arena* $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{PP}, \mathrm{FP})$*;*
5. $\widehat{\mathcal{A}_1} = \widehat{\mathcal{A}_2}$ *are induced by a weighted arena* $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{PP}, \mathrm{BCP})$*;*
6. $\widehat{\mathcal{A}_1}$ *and* $\widehat{\mathcal{A}_2}$ *are induced, respectively, by a weighted arena* $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$*, with* $\mathsf{wg}(v) = 0$ *for some* $v \in \mathrm{Ps}$*, and its underlying colored arena* $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ *and* $(\mathrm{Wn}_1, \mathrm{Wn}_2) = (\mathrm{BCP}, \mathrm{FP})$.

Proof. [Item 1] Consider as colored arena $\widetilde{\mathcal{A}}$ the one underlying the weighted arena depicted in Figure 1.5, which has just the path $\pi = v_0 \cdot {v_1}^\omega$ with payoff $c = \mathsf{pf}(\pi) = 1 \cdot 0^\omega$. Then, it is immediate to see that player $\exists$ wins the P game but not the FP game, since $\mathrm{Rq}(c) \setminus \mathrm{Rs}(c) = \{0\}$.

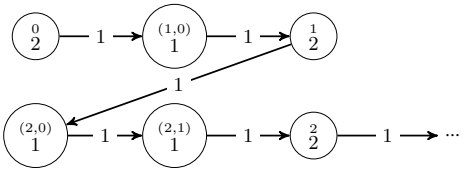

Figure 1.9: Infinite Weighted Arena $\overline{\mathcal{A}_7}$.

[Item 2] Consider as colored arena $\widetilde{\mathcal{A}}$ the one depicted in Figure 1.1 and as weighted arena $\overline{\mathcal{A}}$ the one having a weight 1 on the self loop on v_1 and 0 on the remaining moves. It is immediate to see that player $\exists$ wins the P game $\Game_1$. However, player $\forall$ has a strategy that forces in the PP game $\Game_2$ the play $\pi = \prod_{i=1}^{\omega} v_0 \cdot {v_1}^i \cdot v_2$ having payoff $(c, w) = \mathsf{pf}(\pi) = (\prod_{i=1}^{\omega} 1 \cdot 0^i \cdot 2, \prod_{i=1}^{\omega} 0 \cdot 1^i \cdot 0)$. Therefore, player $\exists$ does not win $\Game_2$, since there is no finite set $\mathrm{R} \subset \mathrm{Rq}(c)$ for which $\mathrm{dl}((c, w), \mathrm{Rq}(c) \setminus \mathrm{R}) < \omega$.

[Item 3] Consider as weighted arena $\overline{\mathcal{A}}$ the infinite one depicted in Figure 1.9 having set of positions $\mathrm{Ps} \triangleq \mathbb{N} \cup \{(i, j) \in \mathbb{N} \times \mathbb{N} : j < i\}$ and moves defined as follows: if

$j < i - 1$ then $((i, j), (i, j + 1)) \in Mv$ else $((i, j), i) \in Mv$. In addition, the coloring of the positions are $\mathsf{cl}(i) = 2$ and $\mathsf{cl}((i, j)) = 1$. Now, it is immediate to see that, on the underlying colored arena $\widetilde{\mathcal{A}}$, Player $\exists$ wins the FP game $\Im_1$, since all requests on the unique possible play $\pi = \prod_{i=0}^{\omega}(\prod_{j=0}^{i-1}(i, j)) \cdot i$ are responded. However, it does not win the PP game $\Im_2$, since $\mathrm{dl}((c, w), \mathrm{Rq}(c)) = \omega$, where $(c, w) = \mathsf{pf}(\pi) = (\prod_{i=0}^{\omega} 1^i \cdot 2, 1^\omega)$. Indeed, there is no finite set $\mathrm{R} \subset \mathrm{Rq}(c)$ for which $\mathrm{dl}((c, w), \mathrm{Rq}(c) \setminus \mathrm{R}) < \omega$.

[Item 4] Consider as weighted arena $\overline{\mathcal{A}}$ the one depicted in Figure 1.5 having just the path $\pi = v_0 \cdot {v_1}^\omega$ with payoff $(c, w) = \mathsf{pf}(\pi) = (1 \cdot 0^\omega, 0 \cdot 1^\omega)$. Player $\exists$ wins the PP game $\Im_1$, since there is just one requests, which we can simply avoid to consider. However, as already observed in Item 1, the FP game $\Im_2$ on the underlying colored arena $\widetilde{\mathcal{A}}$ is not won by the same player.

[Item 5] Consider again as weighted arena $\overline{\mathcal{A}}$ the one depicted in Figure 1.5. As already observed in Item 4, the PP game $\Im_1$ is won by player $\exists$. However, it does not win the BCP game $\Im_2$, since $\mathrm{dl}((c, w), 0) = \omega$.

[Item 6] Consider as weighted arena $\overline{\mathcal{A}}$ the one depicted in Figure 1.6 having just the path $\pi = v_0 \cdot {v_1}^\omega$ with payoff $(c, w) = \mathsf{pf}(\pi) = (1 \cdot 0^\omega, 1 \cdot 0^\omega)$. Player $\exists$ wins the BCP game $\Im_1$, since there is just one requests, which we can simply avoid to consider, and its delay is equal to 1. However, as already observed in Item 1, the FP game $\Im_2$ on the underlying colored arena $\widetilde{\mathcal{A}}$ is not won by the same player. □

1.4 Polynomial Reduction

In this section, we face the computational complexity of solving full parity, prompt parity and bounded cost parity games. Then, due to the relationships among the winning conditions described in the previous section, we propagate the achieved results to the other conditions as well.

The technique we adopt is to solve a given game through the construction of a new game over an enriched arena, on which we play with a simpler winning condition. Intuitively, the constructed game encapsulates, inside the states of its arena, some information regarding the satisfaction of the original condition. To this aim, we introduce the concepts of *transition table* and its *product* with an arena. A transition table is an automaton without acceptance condition, which is used to represent the information of the winning condition mentioned above. Then, the product operation allows to inject this information into the new arena. In particular, the transition table uses non deterministic states to let the player $\exists$ to forget some requests. This will be useful to handle the reduction from prompt parity condition.

The constructions we propose are pseudo-polynomial. However, if we restrict to the case of having only 0 and 1 as weights over the edges, then they become polynomial, due to the fact that the threshold is bounded by the number of edges in the arena. Moreover, since a

game with arbitrary weights can be easily transformed into one with weights 0 and 1, we overall get a polynomial reduction for all the cases. Note that to check whether a value is positive or zero can be done in linear time in the number of its bits and, therefore, it is linear in the description of its weights.

In the following, for a given set of colors $\mathrm{Cl} \subseteq \mathbb{N}$, we assume $\bot < i$, for all $i \in \mathrm{Cl}$. Intuitively, $\bot$ is a special symbol that can be seen as lower bound over color priorities. Moreover, we define $\mathrm{R} \triangleq \{c \in \mathrm{Cl} : c \equiv 1 \pmod 2\}$ to be the set of all possible request values in Cl with $\mathrm{R}_\bot \triangleq \{\bot\} \cup \mathrm{R}$.

1.4.1 Transition Tables

A *transition table* is a tuple $\mathcal{T} \triangleq \langle \mathrm{Sm}, \mathrm{St}, \mathsf{tr} \rangle$, where Sm is the set of *symbols*, St_Δ and St_N with $\mathrm{St} \triangleq \mathrm{St}_\Delta \cup \mathrm{St}_N$ are disjoint sets of *deterministic* and *non deterministic states*, and $\mathsf{tr} : (\mathrm{St}_\Delta \times \mathrm{Sm} \to \mathrm{St}) \cup (\mathrm{St}_N \to 2^{\mathrm{St}})$ is the *transition function* mapping either pairs of deterministic states and symbols to states or non deterministic states to sets of states. $\mathcal{T}$ is deterministic if $\mathsf{tr} : \mathrm{St}_\Delta \times \mathrm{Sm} \to \mathrm{St}$ and $\mathrm{St}_N = \emptyset$. The *order* (resp., *size*) of $\mathcal{T}$ is $|\mathcal{T}| \triangleq |\mathrm{St}|$ (resp., $\|\mathcal{T}\| \triangleq |\mathsf{tr}|$). A transition table is *finite* iff it has finite order. Let $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$ be a colored arena, where $\mathcal{A} = \langle \mathrm{Ps}_\exists, \mathrm{Ps}_\forall, Mv \rangle$ is the underlying arena and $\mathcal{T} \triangleq \langle \mathrm{Cl}, \mathrm{St}, \mathsf{tr} \rangle$ a transition table. The product $\widetilde{\mathcal{A}} \otimes \mathcal{T}$ is an arena in which vertexes are pairs of vertexes from $\widetilde{\mathcal{A}}$ and states from $\mathcal{T}$. Then, such pair belongs to the player $\exists$ iff the first component belongs to the player $\exists$ in the original arena $\widetilde{\mathcal{A}}$ or the second is a non deterministic state. Moreover, the moves are determined by the moves in $\widetilde{\mathcal{A}}$ and the transition table $\mathcal{T}$. Formally, $\widetilde{\mathcal{A}} \otimes \mathcal{T} \triangleq \langle \mathrm{Ps}^\star_\exists, \mathrm{Ps}^\star_\forall, Mv^\star \rangle$ is the *product arena* defined as follows:

- $\mathrm{Ps}^\star_\exists \triangleq \mathrm{Ps}_\exists \times \mathrm{St}_\Delta \cup \mathrm{Ps} \times \mathrm{St}_N$;
- $\mathrm{Ps}^\star_\forall \triangleq \mathrm{Ps}_\forall \times \mathrm{St}_\Delta$;
- $Mv^\star : \mathrm{Ps} \times \mathrm{St} \to \mathrm{Ps} \times \mathrm{St}$ such that $((v_1, s_1), (v_2, s_2)) \in Mv^\star$ iff $(v_1, v_2) \in Mv$ and one of the following condition holds.
  1. $s_1 \in \mathrm{St}_\Delta$ and $s_2 = \mathsf{tr}(s_1, \mathsf{cl}(v_1))$;
  2. $s_1 \in \mathrm{St}_N$, $v_1 = v_2$ and $s_2 = \mathsf{tr}(s_1)$.

Similarly, let $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$ be a weighted arena with $\mathcal{A} = \langle \mathrm{Ps}_\exists, \mathrm{Ps}_\forall, Mv \rangle$ and $\mathcal{T} \triangleq \langle \mathrm{Cl} \times \mathrm{Wg}, \mathrm{St}, \mathsf{tr} \rangle$ a transition table. Then, $\overline{\mathcal{A}} \otimes \mathcal{T} \triangleq \langle \mathrm{Ps}^\star_\exists, \mathrm{Ps}^\star_\forall, Mv^\star \rangle$ is the *product arena* as before, except for the case 1 in which we use $s_2 = \mathsf{tr}(s_1, (\mathsf{cl}(v_1), \mathsf{wg}((v_1, v_2))))$.

1.4.2 From Full Parity to Büchi

In this section, we show a reduction from full parity games to Büchi games. The reduction is done by constructing an ad-hoc transition table $\mathcal{T}$ that maintains basic informations of the

parity condition. Then, the Büchi game uses as an arena an enriched version of the original one, which is obtained as its product with the built transition table. Intuitively, the latter keeps track, along every play, of the value of the biggest unanswered request. When such a request is satisfied, this value is set to the special symbol $\bot$. To this aim, we use as states of the transition table, together with the symbol $\bot$, all possible request values. Also, the transition function is defined in the following way: if a request is satisfied then we move to state $\bot$, otherwise, we move to the state representing the maximum between the new request it reads and the previous memorized one (kept into the current state). Hence, both states and symbols in the transition table associated to the Büchi game are colors of the colored arena of the full parity game. Consider now the arena built as the product of the original one with the above described transition table and use as colors the values 1 and 2, assigned as follows: if a position contains $\bot$, color it with 2, otherwise, with 1. By definition of full parity and Büchi games, we have that a Büchi game is won over the new built arena if and only if the full parity game is won over the original arena. Indeed, over a play of the new arena, meeting a bottom symbol infinitely often means that all requests found over the corresponding play of the old arena are satisfied. The formal construction of the transition table and the enriched arena follow. For a given full parity (FP) game $\mathcal{Э} \triangleq \langle \widehat{\mathcal{A}}, \mathrm{FP}, v_{\mathrm{o}} \rangle$ induced by a colored arena $\widetilde{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl} \rangle$, we construct a deterministic transition table $\mathcal{T} \triangleq \langle \mathrm{Cl}, \mathrm{St}, \mathsf{tr} \rangle$, with set of states $\mathrm{St} \triangleq \mathrm{R}_{\bot}$ and transition function defined as follows:

- $\mathsf{tr}(r, c) \triangleq \begin{cases} \bot, & \text{if } r < c \text{ and } c \equiv 0 \,(\mathrm{mod}\ 2); \\ \mathsf{max}\{r, c\}, & \text{otherwise.} \end{cases}$

Now, let $\mathcal{A}^{\star} =< \mathrm{Ps}^{\star}_{\exists}, \mathrm{Ps}^{\star}_{\forall}, Mv^{\star} >$ be the product arena of $\widetilde{\mathcal{A}}$ and $\mathcal{T}$ and consider the colored arena $\widetilde{\mathcal{A}^{\star}} \triangleq \langle \mathcal{A}^{\star}, \{1, 2\}, \mathsf{cl}^{\star} \rangle$ such that, for all positions $(v, r) \in \mathrm{Ps}^{\star}$, if $r = \bot$ then $\mathsf{cl}^{\star}((v, r)) = 2$ else $\mathsf{cl}^{\star}((v, r)) = 1$. Then, the B game $\mathcal{Э}^{\star} = \langle \widehat{\mathcal{A}^{\star}}, \mathrm{B}, (v_{\mathrm{o}}, \bot) \rangle$, with $\widehat{\mathcal{A}^{\star}}$ induced by the colored arena $\widetilde{\mathcal{A}^{\star}}$, is such that player $\exists$ wins $\mathcal{Э}$ iff it wins $\mathcal{Э}^{\star}$.

Theorem 1.4.1 *For every* FP *game* $\mathcal{Э}$ *with* $k \in \mathbb{N}$ *priorities, there is a* B *game* $\mathcal{Э}^{\star}$*, with order* $|\mathcal{Э}^{\star}| = \mathrm{O}(|\mathcal{Э}| \cdot k)$*, such that player* $\exists$ *wins* $\mathcal{Э}$ *iff it wins* $\mathcal{Э}^{\star}$*.*

Proof. [**If**] By hypothesis, we have that player $\exists$ wins the B game $\mathcal{Э}^{\star}$ on the colored arena $\widetilde{\mathcal{A}}$, which induces a payoff arena $\widehat{\mathcal{A}}$. This means that, there exists a strategy $\sigma^{\star}_{\exists} \in \mathrm{Str}_{\exists}(\mathcal{Э}^{*})$ for player $\exists$ such that for each strategy $\sigma^{\star}_{\forall} \in \mathrm{Str}_{\forall}(\mathcal{Э}^{*})$ for player $\forall$, it holds that $\mathsf{pf}(v_{\mathrm{o}}, (\sigma^{\star}_{\exists}, \sigma^{\star}_{\forall})) \in \mathrm{B}$. Therefore, for all $\pi^{\star} \in \mathrm{Pth}(\mathcal{Э}^{\star}_{\restriction \sigma^{\star}_{\exists}})$, we have that $\mathsf{pf}(\pi^{\star}) \models \mathrm{B}$. Hence, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^{\star}})$ such that $\mathrm{Rq}(c_{\pi^{\star}}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^{\star}})$ with $c_{\pi^{\star}} = \mathsf{pf}(\pi^{\star})$. Now, construct a strategy $\sigma_{\exists} \in \mathrm{Str}_{\exists}(\mathcal{Э})$ such that, for all $\pi \in \mathrm{Pth}(\mathcal{Э}_{\restriction \sigma_{\exists}})$, there exists $\pi^{\star} \in \mathrm{Pth}(\mathcal{Э}^{\star}_{\restriction \sigma_{\exists}{}^{\star}})$, with $\pi = \pi^{\star}_{\restriction 1}$. To do this, let $ext : \mathrm{Hst}_{\exists} \longrightarrow \mathrm{R}_{\bot}$ be a function mapping each history $\rho \in \mathrm{Hst}_{\exists}(\mathcal{Э})$ to the biggest color request not yet answered along a play or to $\bot$, in case there are not unanswered requests. So, we set $\sigma_{\exists}(\rho) \triangleq \sigma^{\star}_{\exists}((\mathsf{lst}(\rho), ext(\rho)))_{\restriction 1}$, for all $\rho \in \mathrm{Hst}_{\exists}(\mathcal{Э})$.

At this point, for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\mathfrak{I})$, there is a strategy $\sigma^\star_\forall \in \mathrm{Str}_\forall(\mathfrak{I}^\star)$ such that $c_\pi \triangleq \mathsf{pf}(v_0, (\sigma_\exists, \sigma_\forall)) \in \mathrm{FP}$, $c_{\pi^\star} \triangleq \mathsf{pf}(v_0, (\sigma^\star_\exists, \sigma^\star_\forall)) \in \mathrm{B}$ and $c_\pi = (c_{\pi^\star})_{\restriction 1}$. Set $\sigma^\star_\forall$ using $\sigma_\forall$ as follows: $\sigma^\star_\forall((v, r)) = \sigma_\forall((v, r'))$ where $r' = \mathsf{tr}(r, \mathsf{cl}(v))$. Since $\mathsf{pf}(\pi^\star) \models \mathrm{B}$, we have that $c_{\pi^\star} \in (\mathrm{Cl}^* \cdot 2)^\omega$. Due to the structure of the transition table and the fact that we give a priority 2 to the vertexes in which there are not unanswered requests, we have that $\mathrm{Rq}(c_{\pi^\star}) = \mathrm{Rs}(c_{\pi^\star})$ and so $\mathrm{Rq}(c_\pi) = \mathrm{Rs}(c_\pi)$.

[**Only If**] By hypothesis, we have that player $\exists$ wins the game $\mathfrak{I}$ on the colored arena $\widetilde{\mathcal{A}}$ which induces a payoff arena $\widehat{\mathcal{A}}$. This means that, there exists a strategy $\sigma_\exists \in \mathrm{Str}_\exists(\mathfrak{I})$ for player $\exists$ such that for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\mathfrak{I})$ for player $\forall$, it holds that $\mathsf{pf}(v_0, (\sigma_\exists, \sigma_\forall)) \in \mathrm{FP}$. Therefore, for all $\pi \in \mathrm{Pth}(\mathfrak{I}_{\restriction \sigma_\exists})$, we have that $\mathsf{pf}(\pi) \models \mathrm{FP}$. Hence, $\mathrm{Rq}(c_\pi) = \mathrm{Rs}(c_\pi)$ with $c_\pi = \mathsf{pf}(\pi)$. Now, we construct a strategy $\sigma^\star_\exists \in \mathrm{Str}_\exists(\mathfrak{I}^\star)$ for player $\exists$ on $\widetilde{\mathcal{A}^\star}$ as follows: for all vertexes (v, r), where $r \in \mathrm{R}_\bot$, it holds that $\sigma^\star_\exists(v, r) = \sigma_\exists(v)$. We prove that $\mathsf{pf}(\pi^\star) \models \mathrm{B}$ for all play $\pi^\star \in \mathrm{Pth}(\mathfrak{I}^\star_{\restriction \sigma^\star_\exists})$, *i.e.*, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^\star})$ such that $\mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^\star})$ with $c_{\pi^\star} = \mathsf{pf}(\pi^\star)$. To do this, we project out π from $\pi^\star$, *i.e.*, $\pi = \pi^\star_{\restriction 1}$, whose meaning is $\pi^\star_i = (\pi_i, r_i)$, for all $i \in \mathbb{N}$. It easy to see that $\pi \in \mathrm{Pth}(\mathfrak{I}_{\restriction \sigma_\exists})$ and then $\mathsf{pf}(\pi) \models \mathrm{FP}$. By contradiction, assume that $\mathsf{pf}(\pi^\star) \not\models \mathrm{B}$. Consequently, there are no vertexes $(v, \bot)$ that appear infinitely often. This means that there exists a position $i \in \mathbb{N}$ in which there is a request $r \in \mathrm{Rq}(c_\pi)$ not satisfied. But this means $\mathsf{pf}(\pi) \not\models \mathrm{FP}$, which is impossible. □

In the following, we report some examples of arenas obtained applying the reduction mentioned above. Observe that each vertex of the constructed arena is labeled with its name (in the upper part) and, in according to the transition function, by the biggest request not responded (in the middle part) and its color (in the lower part).

Example 1.4.1 *Consider the colored arena depicted in Figure 1.10. It represents the reduction from the colored arena $\widetilde{\mathcal{A}}$ drawn in Figure 1.2 where player $\exists$ wins the* FP *game $\mathfrak{I}$ as all requests are responded. It easy to see that player $\exists$ wins also the* B *game $\mathfrak{I}^\star$ in Figure 1.10, as the vertex $(\mathtt{v}_0, \bot)$ with priority 2 is visited infinitely often.*

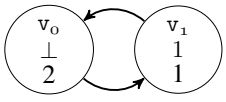

Figure 1.10: From Full Parity to Buchi.

Example 1.4.2 *Consider, now, the arena depicted in Figure 1.11. It represents a reduction from the colored arena $\widetilde{\mathcal{A}}$ drawn in Figure 1.5 where player $\exists$ loses the* FP *game $\mathfrak{I}$ as we have that the request at the vertex $\mathtt{v}_0$ is never responded. It easy to see that player $\exists$ also loses the* B *game $\mathfrak{I}^\star$ in Figure 1.11 as he visits only finitely often the vertex $(\mathtt{v}_0, \bot)$.*

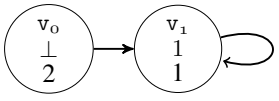

Figure 1.11: From Full Parity to Buchi.

1.4.3 From Bounded-Cost Parity to Parity

In this section, we show a construction that allows to reduce a bounded-cost parity game to a parity game. The approach we propose extends the one given in the previous section by further equipping the transition table $\mathcal{T}$ with a counter that keeps track of the delay accumulated since an unanswered request has been issued. Such a counter is bounded in the sense that if the delay exceeds the sum of weights of all moves in the original arena, then it is set to the special symbol $\divideontimes$. The idea is that if in a finite game such a bound has been exceeded then the adversarial player has taken at least twice a move with a positive weight. So, he can do this an arbitrary number of times and delay longer and longer the satisfaction of a request that therefore becomes not prompt. Thus, we use as states in $\mathcal{T}$, together with $\divideontimes$, a finite set of pairs of numbers, where the first component, as above, represents a finite request, while the second one is its delay. As first state component we also allow $\bot$, since with $(\bot, 0)$ we indicate the fact that there are not unanswered requests up to the current position. Then, the transition function of $\mathcal{T}$ is defined as follows. If a request is not satisfied within a bounded delay, then it goes and remains forever in state $\divideontimes$. Otherwise, if the request is satisfied, then it goes to $(\bot, 0)$, else it moves to a state that contains, as first component, the maximum between the last request not responded and the read color and, as second component, the one present in the current state plus the weight of the traversed edge.

Now, consider the product arena $\mathcal{A}^\star$ of $\mathcal{T}$ with the original arena and color its positions as follows: unanswered request positions, with delay exceeding the bound, are colored with 1, while the remaining ones are colored as in the original arena. Clearly, in $\mathcal{A}^\star$, a parity game is won if and only if the bounded-cost parity game is won on the original arena. The formal construction of $\mathcal{T}$ and $\mathcal{A}^\star$ follow.

For a given BCP game $\mathcal{D} \triangleq \langle \widehat{\mathcal{A}}, \mathrm{BCP}, v_{\mathrm{o}} \rangle$ induced by a weighted arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$, we construct a deterministic transition table $\mathcal{T} \triangleq \langle \mathrm{Cl} \times \mathrm{Wg}, \mathrm{St}, \mathsf{tr} \rangle$, with set of states $\mathrm{St} \triangleq \{\divideontimes\} \cup \mathrm{R}_\bot \times [0, s]$, where we assume $s \triangleq \sum_{m \in Mv} \mathsf{wg}(m)$ to be the sum of all weights of moves in $\overline{\mathcal{A}}$, and transition function defined as follows:

- $\mathsf{tr}(\divideontimes, (c, w)) \triangleq \divideontimes$;
- $\mathsf{tr}((r, k), (c, w)) \triangleq \begin{cases} (\bot, 0), & \text{if } r < c \text{ and } c \equiv 0 \,(\mathrm{mod}\ 2); \\ \divideontimes, & \text{if } k + w > s; \\ (\mathsf{max}\{r, c\}, k + w), & \text{otherwise.} \end{cases}$

Let $\mathcal{A}^\star =< \mathrm{Ps}^\star_\exists, \mathrm{Ps}^\star_\forall, Mv^\star >$ be the product arena of $\overline{\mathcal{A}}$ and $\mathcal{T}$, and $\widetilde{\mathcal{A}^\star} \triangleq \langle \mathcal{A}^\star, \mathrm{Cl}, \mathsf{cl}^\star \rangle$ the colored arena such that the state $(\mathtt{v}, \divideontimes)$ is colored with 1, while all other states are colored as in the original arena (w.r.t. the first component). Then, the P game $\mathcal{D}^\star = \langle \widehat{\mathcal{A}^\star}, \mathrm{P}, (v_{\mathrm{o}}, (\bot, 0)) \rangle$ induced by $\widetilde{\mathcal{A}^\star}$ is such that player $\exists$ wins $\mathcal{D}$ iff it wins $\mathcal{D}^\star$.

Theorem 1.4.2 *For every finite* BCP *game* $\supset$ *with* $k \in \mathbb{N}$ *priorities and sum of weights* $s \in \mathbb{N}$*, there is a* P *game* $\supset^\star$*, with order* $|\supset^\star| = \mathrm{O}(|\supset| \cdot k \cdot s)$*, such that player* $\exists$ *wins* $\supset$ *iff it wins* $\supset^\star$*.*

Proof. [**If**] By hypothesis, player $\exists$ wins the game $\supset^\star$ on the colored arena $\widetilde{\mathcal{A}}$, which induces a payoff arena $\widehat{\mathcal{A}}$. This means that there exists a strategy $\sigma^\star_\exists \in \mathrm{Str}_\exists(\supset^\star)$ for player $\exists$ such that for each strategy $\sigma^\star_\forall \in \mathrm{Str}_\forall(\supset^\star)$ for player $\forall$, it holds that $\mathsf{pf}(v_0, (\sigma^\star_\exists, \sigma^\star_\forall)) \in \mathrm{P}$. Therefore, for all $\pi^\star \in \mathrm{Pth}(\supset^\star_{\restriction \sigma^\star_\exists})$, we have that $\mathsf{pf}(\pi^\star) \models \mathrm{P}$, hence, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^\star})$ such that $\mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^\star})$ with $c_{\pi^\star} = \mathsf{pf}(\pi^\star)$. Now, we construct a strategy $\sigma_\exists \in \mathrm{Str}_\exists(\supset)$ such that, for all $\pi \in \mathrm{Pth}(\supset_{\restriction \sigma_\exists})$, there exists $\pi^\star \in \mathrm{Pth}(\supset^\star_{\restriction \sigma_{\exists}{}^\star})$, *i.e.*, $\pi = \pi^\star_{\restriction_1}$. Let $ext : \mathrm{Hst}_\exists \longrightarrow (\mathrm{R}_\bot \times \mathbb{N})$ be a function mapping each history $\rho \in \mathrm{Hst}_\exists(\supset)$ to a pair of values representing, respectively, the biggest (color) request not yet answered along the history and the sum of the weights over the crossed edges, from the last response of the request. So, we set $\sigma_\exists(\rho) \triangleq \sigma^\star_\exists((\mathsf{lst}(\rho), ext(\rho)))_{\restriction 1}$, for all $\rho \in \mathrm{Hst}_\exists(\supset)$. At this point, for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\supset)$, there is a strategy $\sigma^\star_\forall \in \mathrm{Str}_\forall$ such that $(c_\pi, w_\pi) \triangleq \mathsf{pf}(v_0, (\sigma_\exists, \sigma_\forall)) \in \mathrm{BCP}$, $c_{\pi^\star} \triangleq \mathsf{pf}(v_0, (\sigma^\star_\exists, \sigma^\star_\forall)) \in \mathrm{P}$ and $c_\pi = (c_{\pi^\star})_{\restriction 1}$. Set $\sigma^\star_\forall$ using, trivially, $\sigma_\forall$ as follows: $\sigma^\star_\forall((\mathsf{v}, (r, k))) = (\sigma_\forall(\mathsf{v}), (r^{'}, k^{'}))$ where $(r', k') = \mathsf{tr}((r, k), (\mathsf{cl}(\mathsf{v}), \mathsf{wg}((\mathsf{v}, \sigma_\forall(\mathsf{v})))))$. Let $b = \mathsf{max}\{k \in \mathbb{N} \mid \exists\, i \in \mathbb{N}, \exists\, v \in \mathrm{St}(\supset), r \in \mathrm{R}_\bot . (\pi^\star)_i = (v, (r, k))\}$ be the maximum value the counter can have and $s = \sum_{e \in Mv} \mathsf{wg}(e)$ the sum of weights of edges over the weighted arena $\overline{\mathcal{A}}$. Since $\mathsf{pf}(\pi^\star) \models \mathrm{P}$, by construction, we have that there is no state $(\mathsf{v}, \divideontimes)$ in $\pi^\star$. Moreover, all states $(\mathsf{v}, (r, k))$ in $\pi^\star$ have $k \leq b \leq s$. In other words, b corresponds to the delay within which the request is satisfied. Thus, there exists both a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_\pi)$ such that $\mathrm{Rq}(c_\pi) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_\pi)$ and a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c_\pi, w_\pi), \mathrm{Rq}(c_\pi)) \leq b$.

[**Only If**] By hypothesis, player $\exists$ wins the game $\supset$ on the weighted arena $\overline{\mathcal{A}}$, which induces a payoff arena $\widehat{\mathcal{A}}$. This means that there exists a strategy $\sigma_\exists \in \mathrm{Str}_\exists(\supset)$ for player $\exists$ such that for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\supset)$ for player $\forall$, it holds that $\mathsf{pf}(v_0, (\sigma_\exists, \sigma_\forall)) \in \mathrm{BCP}$. Therefore, for all $\pi \in \mathrm{Pth}(\supset_{\restriction \sigma_\exists})$, we have that $\mathsf{pf}(\pi) \models \mathrm{BCP}$. Hence, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_\pi)$ such that $\mathrm{Rq}(c_\pi) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_\pi)$ and a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c_\pi, w_\pi), \mathrm{Rq}(c_\pi)) \leq b$, where $(c_\pi, w_\pi) = \mathsf{pf}(\pi)$. Let s be the sum of weights of edges in the original arena $\overline{\mathcal{A}}$, previously defined. Now, we construct a strategy $\sigma^\star_\exists \in \mathrm{Str}_\exists(\supset^\star)$ for player $\exists$ on $\overline{\mathcal{A}^\star}$ as follows: for all vertexes $(v, (r, k))$, where $r \in \mathrm{R}_\bot$ and $k \in [0, s]$, it holds that $\sigma^\star_\exists(((v, (r, k)))) = (\sigma_\exists(v), (r^{'}, k^{'}))$ where $(r^{'}, k^{'}) = \mathsf{tr}((r, k), (\mathsf{cl}(\mathsf{v}), \mathsf{wg}((\mathsf{v}, \sigma_\exists(\mathsf{v})))))$. We want to prove that $\mathsf{pf}(\pi^\star) \models \mathrm{P}$, for all plays $\pi^\star \in \mathrm{Pth}(\supset^\star_{\restriction \sigma^\star_\exists})$, *i.e.*, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^\star})$ such that $\mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^\star})$ with $c_{\pi^\star} = \mathsf{pf}(\pi^\star)$. To do this, first suppose that, for all plays $\pi^\star \in \mathrm{Pth}(\supset^\star_{\restriction \sigma^\star_\exists})$, $\pi^\star$ does not cross a state of the kind $(\mathsf{v}, \divideontimes) \in \mathrm{St}(\supset^\star)$ and projects out π from $\pi^\star$, *i.e.*, $\pi = \pi^\star_{\restriction_1}$. It easy to see that $\pi \in \mathrm{Pth}(\supset_{\restriction \sigma_\exists})$ and, so, $\mathsf{pf}(\pi) \models \mathrm{BCP}$. Consequently, $\mathsf{pf}(\pi) \models \mathrm{P}$. Now, due to our assumption, the colors in $\mathsf{pf}(\pi)$ and $\mathsf{pf}(\pi^\star)$ are the same, *i.e.*, $c_\pi = c_{\pi^\star}$. Thus, it holds that $\mathsf{pf}(\pi^\star) \models \mathrm{P}$. It remains to see that our assumption is the only possible one, *i.e.*, it is impossible to find a path $\pi^\star \in \mathrm{Pth}(\supset^\star_{\restriction \sigma^\star_\exists})$,

containing a state of the the kind $(\mathtt{v}, \circledast) \in \mathrm{St}(\mathfrak{Z}^\star)$. By contradiction, assume that there exists a position $i \in \mathbb{N}$ in which there is a request $r \in \mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R}$ not satisfied within delay at most s. Moreover, let j be the first position in which a state of kind $(\mathtt{v}, \circledast)$ is traversed. Between the states $(\mathtt{v}_i, (r_i, k_i)) = (\pi^\star)_i$ and $(\mathtt{v}_j, (r_j, k_j)) = (\pi^\star)_j$, there are no states whose color is an even number bigger than $\mathsf{cl}(\mathtt{v}_i)$. Then, it holds that $\sum_{h=i}^{j} \mathsf{wg}(h) > s$, *i.e.*, at least one of the edges is repeated. Let l and $l^{'}$ with $l < l^{'}$ be two positions in π in which the same edge is repeated, *i.e.*, $(\pi_l, \pi_{l+1}) = (\pi_{l'}, \pi_{l'+1})$. Observe that $\mathsf{wg}((\pi_{l'}, \pi_{l'+1})) > 0$ since otherwise we would not have exceeded the bound s. Furthermore, $\pi_{l+1} = \pi_{l+1'}$ is necessarily a state of player $\forall$. So, he has surely a strategy forcing the play π to pass infinitely often through the edge $(\pi_{l'}, \pi_{l'+1})$. This means that $\mathsf{pf}(\pi) \not\models \mathrm{BCP}$, which is impossible.

□

In the following, we report some examples of arenas obtained applying the reduction mentioned above. Observe that, each vertex of the constructed arena is labeled with its name (in the upper part) and, in according to the transition function, by the pair containing the biggest request not responded and the counter from the last request not responded (in the middle part) and its color (in the lower part).

Example 1.4.3 *Consider the weighted arena depicted in Figure 1.12. It represents the reduction from the weighted arena $\overline{\mathcal{A}}$ drawn in Figure 1.6, where player $\exists$ wins the* BCP *game $\mathfrak{Z}$ as the request at the vertex $\mathtt{v}_0$ is not responded but it has a bounded delay equals to 1. It easy to see that player $\exists$ wins also the* P *game $\mathfrak{Z}^\star$ obtained from the same weighted arena $\overline{\mathcal{A}}$ as he can visit infinitely often the vertex $(\mathtt{v}_1, (1, 1))$ having priority 0 but only finitely often the vertex $(\mathtt{v}_0, (\bot, 0))$ with priority 1.*

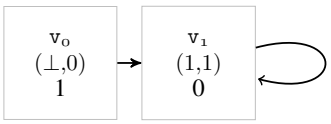

Figure 1.12: From Bounded-Cost Parity to Parity.

Example 1.4.4 *Consider the weighted arena in Figure 1.13. It represents the reduction from the weighted arena $\overline{\mathcal{A}}$ drawn in Figure 1.3 where player $\exists$ loses the* BCP *$\mathfrak{Z}$ since the request at the vertex $\mathtt{v}_0$ is never responded and there is a unique play in which the delay is incremented by 1 in an unbounded way. It easy to see that player $\exists$ loses also the* P *game $\mathfrak{Z}^\star$ obtained from the same weighted arena $\overline{\mathcal{A}}$ as there exists a unique play where the special states $(\mathtt{v}_2, \circledast)$ and $(\mathtt{v}_1, \circledast)$ with priority 1, are the only ones visited infinitely often.*

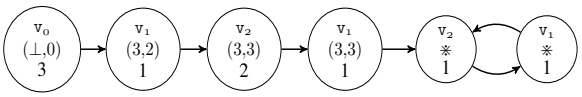

Figure 1.13: From Bounded-Cost Parity to Parity.

1.4.4 From Prompt Parity to Parity and Büchi

Finally, we show a construction that reduces a prompt parity game to a parity game. In particular, when the underlying weighted arena of the original game has only positive weights, then the construction returns a Büchi game. Our approach extends the one proposed for the above BCP case, by further allowing the transition table $\mathcal{T}$ to guess a request value that is not meet anymore along a play. This is done to accomplish the second part of the prompt parity condition, in which a finite number of requests can be excluded from the delay computation. To do this, first we allow $\mathcal{T}$ to be nondeterministic and label its states with a flag $\alpha \in \{D, \exists\}$ to identify, respectively, deterministic and existential states. Then, we enrich the states by means of a new component $d \in [0, h]$, where $h \triangleq |\{v \in \mathrm{Ps} : \mathsf{cl}(v) \equiv 1 \pmod 2\}|$ is the maximum number of positions having odd priorities. So, d represents the counter of the forgotten priority, which it is used to later check the guess states. The existential states belong to player $\exists$. Conversely, the deterministic ones belong to player $\forall$. As initial state we have the tuple $(\mathtt{v}, ((\bot, 0, D), 0)))$ indicating that there are not unanswered and forgotten requests up to the current deterministic position. The transition function over a deterministic state is defined as follows. If a request is not satisfied in a bounded delay, (*i.e.*, the delay exceeds the sum of the weights of all moves in the original arena) then it goes and remains forever in state $(\mathtt{v}, \divideontimes)$ with priority 1; if the request is satisfied then it goes to $(\mathtt{v}, ((\bot, d, D), 0))$ indicating that in this deterministic state there is not an unanswered request and the sum of the weight of the edges is 0); otherwise it moves to an existential state that contains, as first component, the triple having the maximum between the last request not responded and the read color, the counter of forgotten priority, and a flag indicating that the state is existential. Moreover, as a second component, there is a number that represents the sum of the weights of the traversed edges until the current state. The transition function over an existential state is defined as follows. If d is equal to h (*i.e.*, the maximum allowable number of positions having an odd priority), then the computation remains in the same (deterministic) state; otherwise, the computation moves to a state in which the second component is incremented by the weight of the crossed edge. Note that the guess part is similar to that one performed to translate a nonderministic co-Büchi automaton into a Büchi one [KMM06]. Finally, we color the positions of the obtained arena as follows: unanswered request positions, with delay exceeding the bound, are colored by 1, while the remaining ones are colored as in the original arena. In case the weighted arena of the original game has only positive weights, then one can exclude a priory the fact that there are unanswered requests with bounded delays. So, all these kind of requests can be forgotten in order to win the game. Thus, in this case, it is enough to satisfy only the remaining ones, which corresponds to visit infinitely often a position containing as second component the symbol $\bot$. So it is enough to color these positions with 2, all the remaining ones with 1, and play on this arena a Büchi condition. The formal construction of the transition table and the enriched arena follow.

For a PP game $\mathfrak{D} \triangleq \langle \widehat{\mathcal{A}}, \mathrm{PP}, v_{\mathrm{o}} \rangle$ induced by an arena $\overline{\mathcal{A}} = \langle \mathcal{A}, \mathrm{Cl}, \mathsf{cl}, \mathrm{Wg}, \mathsf{wg} \rangle$, we build a transition table $\mathcal{T} \triangleq \langle \mathrm{Cl} \times \mathrm{Wg}, \mathrm{St}, \mathsf{tr} \rangle$, with sets of states $\mathrm{St}_\Delta \triangleq \{\divideontimes\} \cup \mathrm{Z}_D \times [0, s]$ and $\mathrm{St}_N \triangleq \mathrm{Z}_\exists \times [0, s]$, where we assume $s \triangleq \sum_{m \in Mv} \mathsf{wg}(m)$ to be the sum of all weights of moves in the original arena and $\mathrm{Z}_\alpha \triangleq \mathrm{R}_\perp \times [0, h] \times \alpha$, and its transition function defined as follows:

- $\mathsf{tr}(\divideontimes, (c, w)) \triangleq \divideontimes$;
- $\mathsf{tr}(((r, d, D), k), (c, w)) \triangleq \begin{cases} ((\perp, d, D), 0), & \text{if } r < c \wedge c \equiv 0 \,(\mathrm{mod}\ 2); \\ \divideontimes, & \text{if } k + w > s; \\ ((\mathsf{max}\{r, c\}, d, \exists), k + w), & \text{otherwise.} \end{cases}$
- $\mathsf{tr}(((r, d, \exists), k)) \triangleq \begin{cases} \{((r, d, D), k)\}, & \text{if } d = h; \\ \{((r, d, D), k), ((\perp, d + 1, D), 0)\}, & \text{otherwise.} \end{cases}$

Observe that, the set Z_α is the Cartesian product of the biggest unanswered request, the counter of the forgotten priority and, a flag indicating whether the state is deterministic or existential.

Let $\mathcal{A}^\star = \overline{\mathcal{A}} \otimes \mathcal{T}$ be the product arena of $\overline{\mathcal{A}}$ and $\mathcal{T}$ and consider the colored arena $\widetilde{\mathcal{A}^\star} \triangleq \langle \mathcal{A}^\star, \mathrm{Cl}, \mathsf{cl}^\star \rangle$ such that, for all positions $(v, t) \in \mathrm{Ps}^\star$, if $t = \divideontimes$ then $\mathsf{cl}^\star((v, t)) = 1$ else $\mathsf{cl}^\star((v, t)) = \mathsf{cl}(v)$. Then, the P game $\mathfrak{D}^\star = \langle \widehat{\mathcal{A}^\star}, \mathrm{P}, (v_{\mathrm{o}}, ((\perp, 0, D), 0)) \rangle$ induced by $\widetilde{\mathcal{A}^\star}$ is such that player $\exists$ wins $\mathfrak{D}$ iff it wins $\mathfrak{D}^\star$.

Theorem 1.4.3 *For every* PP *game* $\mathfrak{D}$ *with* $k \in \mathbb{N}$ *priorities and sum of weights* $s \in \mathbb{N}$*, there is a* P *game* $\mathfrak{D}^\star$*, with order* $|\mathfrak{D}^\star| = \mathrm{O}(|\mathfrak{D}|^2 \cdot k \cdot s)$*, such that player* $\exists$ *wins* $\mathfrak{D}$ *iff it wins* $\mathfrak{D}^\star$*.*

Proof. [**If**] By hypothesis, player $\exists$ wins the game $\mathfrak{D}^\star$ on the colored arena $\widetilde{\mathcal{A}}$, which induces a payoff arena $\widehat{\mathcal{A}}$. This means that there exists a strategy $\sigma^\star_\exists \in \mathrm{Str}_\exists(\mathfrak{D}^\star)$ for player $\exists$ such that for each strategy $\sigma^\star_\forall \in \mathrm{Str}_\forall(\mathfrak{D}^\star)$ for player $\forall$, it holds that $\mathsf{pf}(v_{\mathrm{o}}, (\sigma^\star_\exists, \sigma^\star_\forall)) \in \mathrm{P}$. Therefore, for all $\pi^\star \in \mathrm{Pth}(\mathfrak{D}^\star_{\restriction \sigma^\star_\exists})$, we have that $\mathsf{pf}(\pi^\star) \models \mathrm{P}$. Hence, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^\star})$ such that $\mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^\star})$ with $c_{\pi^\star} = \mathsf{pf}(\pi^\star)$. Now, we construct a strategy $\sigma_\exists \in \mathrm{Str}_\exists(\mathfrak{D})$ such that, for all $\pi \in \mathrm{Pth}(\mathfrak{D}_{\restriction \sigma_\exists})$, there exists $\pi^\star \in \mathrm{Pth}(\mathfrak{D}^\star_{\restriction \sigma_\exists{}^\star})$, *i.e.*, $\pi = \pi^\star_{\restriction 1}$. Let $ext : \mathrm{Hst}_\exists \longrightarrow (\mathrm{R}_\perp \times [0, h] \times D) \times \mathbb{N}$ be a function mapping each history $\rho \in \mathrm{Hst}_\exists(\mathfrak{D})$ to a tuple of values that represent, respectively, the biggest color request along the history ρ that is both not answered and not forget by $\sigma_\exists{}^\star$, the number of odd priorities that are forgotten, and the sum of the weights over the crossed edges since the more recent occurrence of one of the following two cases: the last response of a request or the last request that is forgotten. So, we set $\sigma_\exists(\rho) \triangleq \sigma^\star_\exists((\mathsf{lst}(\rho), ext(\rho)))_{\restriction 1}$, for all $\rho \in \mathrm{Hst}_\exists(\mathfrak{D})$. At this point, for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\mathfrak{D})$, there is a strategy $\sigma^\star_\forall \in \mathrm{Str}_\forall$ such that $(c_\pi, w_\pi) \triangleq \mathsf{pf}(v_{\mathrm{o}}, (\sigma_\exists, \sigma_\forall)) \in \mathrm{PP}$, $c_{\pi^\star} \triangleq \mathsf{pf}(v_{\mathrm{o}}, (\sigma^\star_\exists, \sigma^\star_\forall)) \in \mathrm{P}$ and $c_\pi \triangleq (c_{\pi^{\star\prime}})_{\restriction 1}$ where $\pi^{\star\prime}$ is obtained

from $\pi^\star$ by removing the vertexes of the form $(v,((r,d,\exists),k))$ that are the vertexes in which it is allowed to forget a request. Now, set $\sigma^\star_\forall$ using $\sigma_\forall$ as follows: $\sigma^\star_\forall(\mathtt{v},((r,d,\alpha),k)) = (\sigma_\forall(\mathtt{v}),((r^{'},d^{'},\alpha^{'}),k^{'}))$ where $((r^{'},d^{'},\alpha^{'}),k^{'}) = \mathsf{tr}(((r,d,\alpha),k),(\mathsf{cl}(\mathtt{v}),\mathsf{wg}((\mathtt{v},\sigma_\forall(\mathtt{v})))))$. Let $b = \mathsf{max}\{k \in \mathbb{N} \mid \exists\, i \in \mathbb{N}, \exists\, v \in \mathrm{St}(\mathfrak{I}), r \in \mathrm{R}_\perp, d \in [0,h], \alpha \in \{D,\exists\}.(\pi^\star)_i = (v,((r,d,\alpha),k))\}$ be the maximum value that the counter can have and $s = \sum_{e \in Mv} \mathsf{wg}(e)$ the sum of weights of edges over the weighted arena $\overline{\mathcal{A}}$. Since $\mathsf{pf}(\pi^\star) \models \mathrm{P}$, by construction we have that there is no state $(\mathtt{v},\divideontimes)$ in $\pi^\star$. Moreover, all states $(\mathtt{v},((r,d,\alpha),k))$ in $\pi^\star$ have $k \leq b \leq s$. Thus, there exists both a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_\pi)$ such that $\mathrm{Rq}(c_\pi) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_\pi)$ and a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c_\pi,w_\pi),\mathrm{Rq}(c_\pi) \setminus \mathrm{R}) \leq b$.

[**Only If**] By hypothesis, player $\exists$ wins the game $\mathfrak{I}$ on the weighted arena $\overline{\mathcal{A}}$, which induces a payoff arena $\widehat{\mathcal{A}}$. This means that there exists a strategy $\sigma_\exists \in \mathrm{Str}_\exists(\mathfrak{I})$ for player $\exists$ such that, for each strategy $\sigma_\forall \in \mathrm{Str}_\forall(\mathfrak{I})$ for player $\forall$, it holds that $\mathsf{pf}(v_{\mathrm{o}},(\sigma_\exists,\sigma_\forall)) \in \mathrm{PP}$. Therefore, for all $\pi \in \mathrm{Pth}(\mathfrak{I}_{\restriction\sigma_\exists})$ we have that $\mathsf{pf}(\pi) \models \mathrm{PP}$. Hence, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_\pi)$ such that $\mathrm{Rq}(c_\pi) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_\pi)$ and there exists a bound $b \in \mathbb{N}$ for which $\mathrm{dl}((c_\pi,w_\pi),\mathrm{Rq}(c_\pi) \setminus \mathrm{R}) \leq b$, with $(c_\pi,w_\pi) = \mathsf{pf}(\pi)$. Let $h \triangleq |\{v \in \mathrm{Ps} : \mathsf{cl}(v) \equiv 1 \,(\mathrm{mod}\ 2)\}|$ be the maximum number of positions having odd priorities. Moreover, let s be the sum of all weights of moves in the original game $\mathfrak{I}$, previously defined. Now, we construct a strategy $\sigma^\star_\exists \in \mathrm{Str}_\exists(\mathfrak{I}^\star)$ for player $\exists$ on $\overline{\mathcal{A}^\star}$ as follows. For all vertexes $(v,((r,d,D),k)) \in \mathrm{St}_N(\mathfrak{I}^\star)$, we set $\sigma^\star_\exists(v,((r,d,D),k)) = (\sigma_\exists(v),((r^{'},d^{'},\alpha^{'}),k^{'}))$ where $((r^{'},d^{'},\alpha^{'}),k^{'}) = \mathsf{tr}(((r,d,D),k),(\mathsf{cl}(\mathtt{v}),\mathsf{wg}((\mathtt{v},\sigma_\exists(\mathtt{v})))))$. Moreover, for all vertexes $(v,((r,h,\exists),k)) \in \mathrm{St}_N(\mathfrak{I}^\star)$, we set $\sigma^\star_\exists(v,((r,h,\exists),k)) = (\sigma_\exists(v),((r,h,D),k))$. Now, let $frg : \mathrm{St}_N \to \mathbb{N}$ be a function such that $frg(v)$ is the maximum odd priority that player $\exists$ can forget, *i.e.*, the highest odd priority that can be crossed only finitely often in $\mathfrak{I}_{\restriction\sigma_\exists}$ starting at v. At this point, if $d < h$, *i.e.*, it is still possible to forget other $h - d$ priorities, then we set $\sigma^\star_\exists(v,((r,d,\exists),k)) = (\sigma_\exists(v),((\perp,d+1,D),0))$ if $r \leq frg(v)$, otherwise, $\sigma^\star_\exists(v,((r,d,\exists),k)) = (\sigma_\exists(v),((r,d,D),k))$. We want to prove that $\mathsf{pf}(\pi^\star) \models \mathrm{P}$, for all play $\pi^\star \in \mathrm{Pth}(\mathfrak{I}^\star_{\restriction\sigma^\star_\exists})$, *i.e.*, there exists a finite set $\mathrm{R} \subseteq \mathrm{Rq}(c_{\pi^\star})$ such that $\mathrm{Rq}(c_{\pi^\star}) \setminus \mathrm{R} \subseteq \mathrm{Rs}(c_{\pi^\star})$ with $c_{\pi^\star} = \mathsf{pf}(\pi^\star)$. Starting from $\pi^\star$, we construct $\pi^{\star^{'}}$ by removing the vertexes of the form $(v,((r,d,\exists),k))$ that are the vertexes in which is allow to forget a request . Then, we project out π from $\pi^{\star^{'}}$, *i.e.*, $\pi = \pi^{\star^{'}}_{\restriction 1}$. It easy to see that $\pi \in \mathrm{Pth}(\mathfrak{I}_{\restriction\sigma_\exists})$ and, so, $\mathsf{pf}(\pi) \models \mathrm{PP}$. Consequently, $\mathsf{pf}(\pi) \models \mathrm{P}$. The colors in $\mathsf{pf}(\pi)$ and $\mathsf{pf}(\pi^{\star^{'}})$ are the same, *i.e.*, $c_\pi = c_{\pi^{\star'}}$. Thus, it holds that $\mathsf{pf}(\pi^{\star^{'}}) \models \mathrm{P}$ and so $\mathsf{pf}(\pi^\star) \models \mathrm{P}$. At this point, it just remains to see that our assumption is the only possible one, *i.e.*, it is impossible to find a path $\pi^\star \in \mathrm{Pth}(\mathfrak{I}^\star_{\restriction\sigma^\star_\exists})$ containing a state of the the kind $(\mathtt{v},\divideontimes) \in \mathrm{St}(\mathfrak{I}^\star)$. To do this, we use the same reasoning applied in the proof of Theorem 1.4.2. □

It is worth observing that the estimation on the size of $\mathfrak{I}^\star$ in Theorem 1.4.3 is quite coarse since several type of states can not be reached by the initial position.

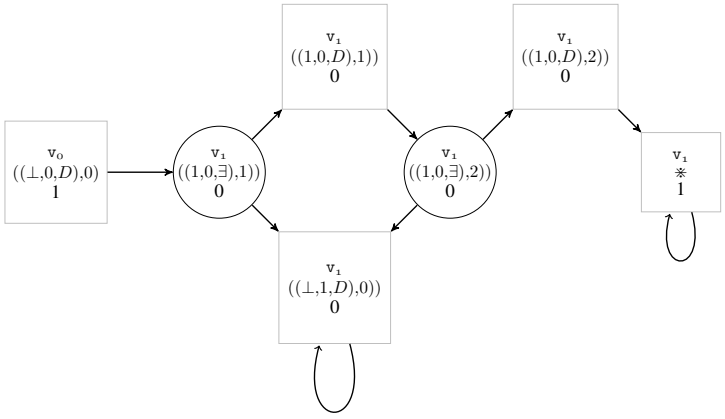

Figure 1.14: From Prompt Parity to Parity (Figura 5).

In the following, we report some examples of arenas obtained by applying the reduction mentioned above. Observe that each vertex of the constructed arena is labeled by its name (in the upper part) and, according to the transition function, by the tuple containing the biggest request not responded, the maximum number of forgotten positions having odd priorities in the original arena, a flag indicating a deterministic or an existential state, a counter from the last request not responded (in the middle part), and its color (in the lower part).

Example 1.4.5 *Consider the weighted arena depicted in Figure 1.14. It represents the reduction from the arena drawn in Figure 1.5. In this example, player* $\exists$ *wins the* PP *game* $\supset$ *because only the request at the vertex* $\mathtt{v}_0$ *is not responded and this request is not traversed infinitely often. Moreover, as previously showed, player* $\exists$ *also wins the* P *game* $\supset^\star$ *obtained from the same weighted arena in Figure 1.14. In more details, starting from the initial vertex* $(\mathtt{v}_0, ((\bot, 0, D), 0))$ *with priority* 1*, player* $\forall$ *moves the token to the existential vertex* $(\mathtt{v}_1, ((1, 0, \exists), 1))$ *having priority* 0*. At this point, player* $\exists$ *has two options: he can forget or not the biggest odd priority crossed up to now. In the first case, he moves to the vertex* $(\mathtt{v}_1, ((\bot, 1, D), 0))$*, having priority* 0*, where player* $\forall$ *can only cross infinitely often this vertex, letting player* $\exists$ *to win the game. In the other case, he moves to the vertex* $(\mathtt{v}_1, ((1, 0, D), 1))$ *with priority* 0 *from which player* $\forall$ *moves to the vertex* $(\mathtt{v}_1, ((1, 0, \exists), 2))$ *having priority* 0*. From this vertex, player* $\exists$ *can still decide either to forget or not the biggest odd priority crossed up to now. In the first case player* $\exists$ *wins the game by crossing infinitely often the vertex* $(\mathtt{v}_1, ((\bot, 1, D), 0))$ *with priority* 0*. In the other case, he loses the game and so he will never take such a move. In conclusion, player* $\exists$ *has a winning strategy against every possible strategy of the player* $\forall$*.*

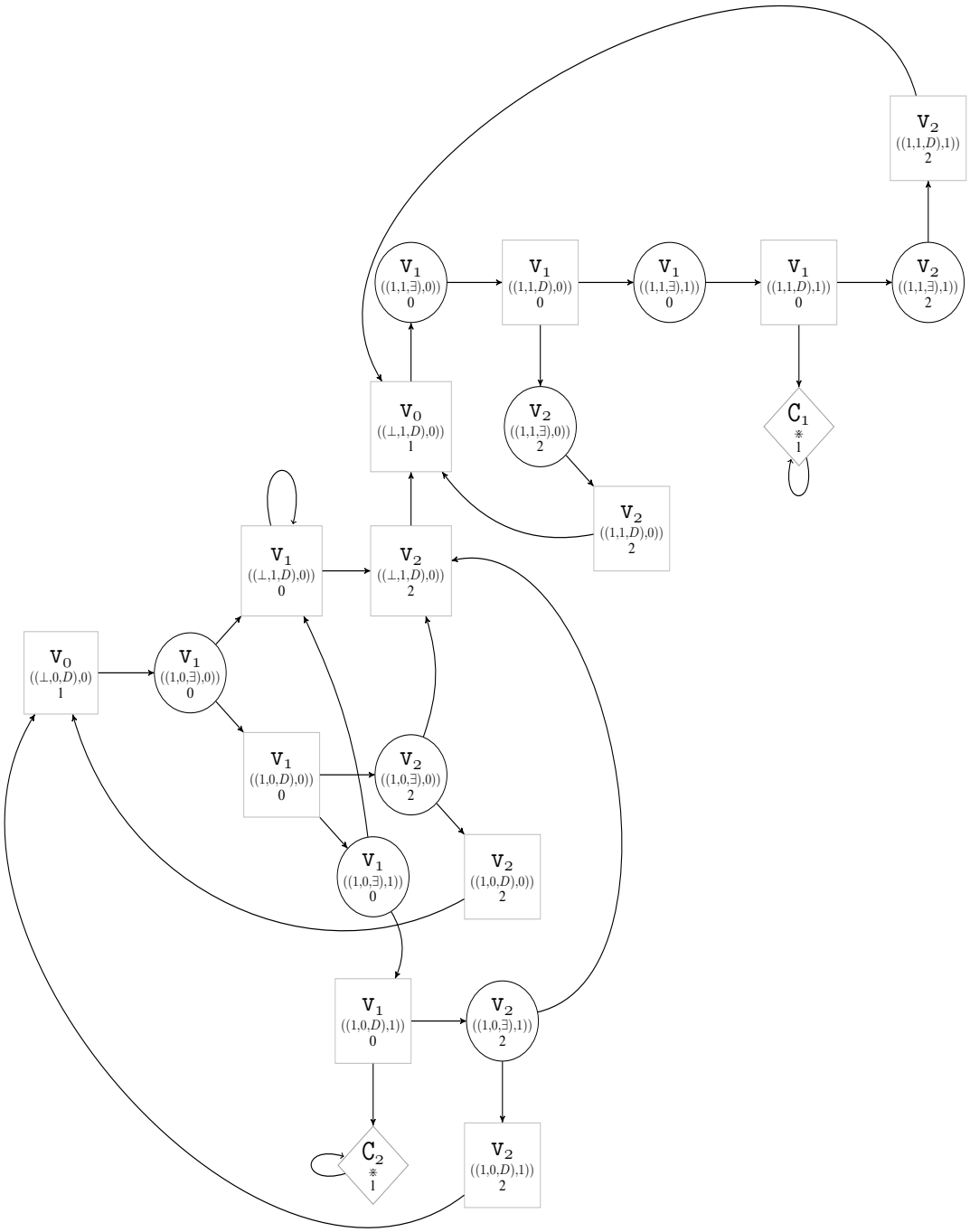

Figure 1.15: From Prompt Parity to Parity.

Example 1.4.6 *Consider the weighted arena depicted in Figure 1.15. This arena represents the reduction from the arena in Figure 1.1. In this example, player* $\exists$ *loses the* PP $\Im$ *against any possible move of the opponent because the delay between the request and its response is unbounded. Moreover, as proved, player* $\exists$ *loses also the* P *game* $\Im^\star$ *obtained from the same weighted arena* $\overline{\mathcal{A}}$*, against any possible move of the opponent. In detail, we have that the game starts in the vertex* $(\mathtt{v}_0, ((\bot, 0, D), 0))$ *having priority* 1*. At this point, player* $\forall$ *is obliged to go to the vertex* $(\mathtt{v}_1, ((1, 0, \exists), 0))$ *with priority* 0*. Then, player* $\exists$ *has two options that are either to forget or not forget the biggest odd priority crossed.*

1. *In the first case he goes to the vertex* $(\mathtt{v}_1, ((\bot, 1, D), 0))$ *having priority* 0*. From this vertex, player* $\forall$*, in order to avoid losing, does not cross this vertex infinitely often, but he moves the token in the vertex* $(\mathtt{v}_2, ((\bot, 1, D), 0))$ *having priority* 2*. From this vertex, player* $\forall$ *is obliged to move the token in the vertex* $(\mathtt{v}_0, ((\bot, 1, D), 0))$ *with priority* 1 *and yet to the vertex* $(\mathtt{v}_1, ((1, 1, \exists), 0))$ *having priority* 1*. At this point, player* $\exists$ *can move the token only to the vertex* $(\mathtt{v}_1, ((1, 1, D), 0))$ *with priority* 0*, which belong to player* $\forall$*. Then, this player, moves the token in the vertex* $(\mathtt{v}_1, ((1, 1, \exists), 1))$ *with priority* 0*. From this vertex, player* $\exists$ *can only move the token to the vertex* $(\mathtt{v}_1, ((1, 1, D), 1))$ *from which player* $\forall$ *wins the game by forcing the token to remain in the diamond vertex* $(\mathtt{C}_1, \divideontimes)$ *which we use to succinctly represent a strong connected component, fully labeled by* 1*, from which player* $\exists$ *cannot exit.*

2. *In the other case, player* $\exists$ *goes to the vertex* $(\mathtt{v}_1, ((1, 0, D), 0))$ *having priority* 0*. At this point, player* $\forall$ *may decide to go either in the vertex* $(\mathtt{v}_2, ((1, 0, \exists), 0))$ *having priority* 2 *or in the vertex* $(\mathtt{v}_1, ((1, 0, \exists), 1))$ *with priority* 0*. From the vertex* $(\mathtt{v}_2, ((1, 0, \exists), 0))$*, player* $\exists$ *can decide either to forget or not the biggest odd priority crossed.*

   (a) *In the first case, player* $\exists$ *moves the token to the vertex* $(\mathtt{v}_2, ((\bot, 1, D), 0))$ *having priority* 2 *and the play continues as in step* 1*, starting from this vertex.*

   (b) *In the other case, player* $\exists$ *moves the token to the vertex* $(\mathtt{v}_2, ((1, 0, D), 0))$ *belonging to the player* $\forall$ *which moves the token at the initial vertex. At this point, player* $\exists$ *moves the token to the initial vertex* $(\mathtt{v}_0, ((\bot, 0, D), 0))$ *having priority* 1*. From this vertex, player* $\forall$ *goes to the vertex* $(\mathtt{v}_1, ((1, 0, \exists), 0))$ *with priority* 0*. Then, player* $\exists$ *can either forget or not the biggest odd priority crossed. From this state, we have already seen that he can win the game.*

   From the vertex $(\mathtt{v}_1, ((1, 0, \exists), 1))$ *with priority* 0*, player* $\exists$ *can:*

   (a) *decide to forget the biggest odd priority and then to move the token to the vertex* $(\mathtt{v}_1, ((\bot, 1, D)\, 0))$ *having priority* 0*. At this point, the play continues as in step* 1 *starting from this vertex.*

(b) decide to not forget the biggest odd priority and then to move the token to the vertex $(\mathtt{v}_1, ((1, 0, D), 1))$ *belonging to the player* $\forall$*, which force the token to remain in the diamond vertex* $(\mathtt{C}_2, \divideontimes)$ *having priority* 1*, winning the game.*

In case the weighted arena $\overline{\mathcal{A}}$ is positive, *i.e.*, $\mathsf{wg}(v) > 0$ for all $v \in \mathrm{Ps}$, we can improve the above construction as follows. Consider the colored arena $\widetilde{\mathcal{A}^\star} \triangleq \langle \mathcal{A}^\star, \{1, 2\}, \mathsf{cl}^\star \rangle$ such that, for all positions $(v, t) \in \mathrm{Ps}^\star$, if $t = ((\bot, d, D), 0)$ for some $d \in [0, h]$ then $\mathsf{cl}^\star((v, t)) = 2$ else $\mathsf{cl}^\star((v, t)) = 1$. Then, the B game $\supset^\star = \langle \widehat{\mathcal{A}^\star}, \mathrm{B}, (v_0, ((\bot, 0, D), 0)) \rangle$ induced by $\widetilde{\mathcal{A}^\star}$ is such that player $\exists$ wins $\supset$ iff it wins $\supset^\star$.

By means of a proof similar to the one used to prove Theorem 1.4.3 , we obtain the following.

Theorem 1.4.4 *For every* PP *game* $\supset$ *with* $k \in \mathbb{N}$ *priorities and sum of weights* $s \in \mathbb{N}$ *defined on a positive weighted arena, there is a* B *game* $\supset^\star$*, with order* $|\supset^\star| = \mathrm{O}(|\supset|^2 \cdot k \cdot s)$*, such that player* $\exists$ *wins* $\supset$ *iff it wins* $\supset^\star$*.*

1.5 Discussion

Recently, promptness reasonings have received large attention in system design and verification. This is due to the fact that, while from a theoretical point of view questions like "a specific state is eventually reached in a computation" have a clear meaning and application in formal verification, in a practical scenario, such a question results useless if there is no bound over the time the required state occurs. This is the case, for example, when we deal with liveness and safety properties. The question becomes even more involved in the case of reactive systems, well modeled as two-player games, in which the response can be procrastinated later and later due to an adversarial behavior.

In this work, we studied several variants of two-player parity games working under a prompt semantics. In particular, we gave a general and clean setting to formally describe and unify most of such games introduced in the literature, as well as to address new ones. Our framework helped us to investigate peculiarities and relationships among the addressed games. In particular, it helped us to come up with solution algorithms that have as core engine and main complexity the solution of a parity or a Büchi game. This makes the proposed algorithms very efficient. With more details, we have considered games played over colored and weighted arenas. In colored arenas, vertexes are colored with priorities and the parity condition asks whether, along paths, every odd priority (a request) is eventually followed by a bigger even priority (a response). In addition, weighted arenas have weights over the edges and consider as a delay of a request the sum of the edges traversed until its response occurs. Also, we have differentiated conditions depending on whether it occurs not-full (all requests, but a finite number, have to be satisfied), full (all requests have to be satisfied) or

semi-full (the condition is a conjunction of two request properties, one behaving full and the other not-full).

As games already addressed in the literature, we studied the cost parity and bounded-cost parity ones and, for both of them, we provided algorithms that improve their known complexity. As new parity games, we investigated the full parity, full-prompt parity, and prompt parity ones. We showed that full parity games are in PTIME, prompt parity and cost parity are equivalent and both in UPTIME ∩ COUPTIME. The latter improves the known complexity class result of [FZ12] to solve cost parity games because our algorithm reduces the original problem to a unique parity game, while the one in [FZ12] performs "several calls" to a parity game solver. Tables 1.1 and 1.2 report the formal definition of all conditions addressed in the chapter along with the full/not-full/semi-full behavior. Tables 1.3 summarizes the achieved results. In particular, we use the special arrow $\hookleftarrow$ to indicate that the result is trivial or an easy consequence of another one in the same row.

Conditions	Colored Arena	(Colored) Weighted arena
Parity (P)	UPTIME ∩ COUPTIME [Jur98]	$\hookleftarrow$
Full Parity (FP)	PTIME [Thm 1.4.1]	$\hookleftarrow$
Prompt Parity (PP)	PTIME [Thm 1.4.4]	UPTIME ∩ COUPTIME [Thm 1.4.3]
Full Prompt Parity (FPP)	$\hookleftarrow$	PTIME [FP + Cor 1.3.1]
Cost Parity (CP)	PTIME [PP + Cor 1.3.2]	UPTIME ∩ COUPTIME [PP + Cor 1.3.2]
Bounded Cost Parity (BCP)	PTIME [FPP + Cor 1.3.3]	UPTIME ∩ COUPTIME [Thm 1.4.2]

Table 1.3: Summary of all winning condition complexities.

As future work, there are several directions one can investigate. For example, one can extend the same framework in the context of multi-agent systems. Recently, a (multi-agent) logic for strategic reasoning, named *Strategy Logic* [MMPV12] has been introduced and deeply studied. This logic has as a core engine the logic LTL. By simply considering as a core logic a prompt version of LTL [KPV09], we get a prompt strategy logic for free. More involved, one can inject a prompt μ-calculus modal logic (instead of LTL) to have a proper prompt parity extension of Strategy Logic. Then, one can investigate opportune restrictions to the conceived logic to gain interesting complexities for the related decision problems. Overall, we recall that Strategy Logic is highly undecidable, while several of its interesting fragments are just to 2EXPTIME-COMPLETE. As another direction for future work, one may think to extend the prompt reasoning to infinite state systems by considering, for example, pushdown parity games [Wal01, ALM$^+$13, BSW03]. However, this extension is rather than an easy task as one needs to rewrite completely the algorithms we have proposed.

CHAPTER 2

Solving Parity Games in Scala

Contents

Parity games [EJ91, Zie98] are abstract infinite-duration games that represents a powerful mathematical framework to address fundamental questions in computer science and mathematics. They are strict connected with other games of infinite duration, such as *mean* and *discounted* payoff, *stochastic*, and *multi-agent* games [Ber07, CDHR10, CHJ05, CJH04].

In formal system design and verification [CGP02, KVW00], parity games arise as a natural evaluation machinery to automatically and exhaustively check for reliability of distributed and reactive systems [AMM13, AKM12, KVW01]. More specifically, in formal verification, *model-checking* techniques [CE81, QS81] allow to verify whether a system is correct with respect to a desired behavior by checking whether a mathematical model of the system meets a formal specification of the expected execution. In case the latter is given by means of a μ-calculus formula [Koz83], the model checking problem can be translated, in linear-time, into a parity game [EJ91]. Hence, every parity game solver can be used in practice as a model checker for a μ-calculus specification (and vice-versa). Using this approach, *liveness* and *safety* properties can be addressed in a very elegant and easy way [MMS13]. Also, this offers a very powerful machinery to check for component software reliability [AMM13, AKM12].

In the basic settings, parity games are two-player turn-based games, played on directed graphs, whose nodes are labeled with *priorities* (i.e., natural numbers). The players, named *player* 0 and *player* 1, move in turn a token along graph's edges. Thus, a play induces an infinite path and player 0 wins the play if the greatest priority visited infinitely often is even; otherwise, player 1 wins the play.

The problem of finding a winning strategy in parity games is known to be in UPTime $\cap$ CoUPTime [Jur98] and deciding whether a polynomial time solution exists or not is a long-standing open question. Aimed to find the right complexity of parity games, as well as come out with solutions working efficiently in practice, several algorithms have been proposed in the last two decades. In Table 2.1, we report the most common ones along with their known computational complexities, where parameters n, e, and d denote the number of nodes, edges, and priorities in the game, respectively (for more details, see [Fri09, Oli09]).

Condition	Complexity
Recursive [Zie98]	$O(e \cdot n^d)$
Small Progress Measures [Mar00]	$O(d \cdot e \cdot (\frac{n}{d})^{\frac{d}{2}})$
Strategy Improvement [Jen00]	$O(2^e \cdot n \cdot e)$
Dominion Decomposition [Mar08]	$O(n^{\sqrt{n}})$
Big Step [Sve07]	$O(e \cdot n^{\frac{1}{3}d})$

Table 2.1:
Parity algorithms along with their computational complexities.

All above mentioned algorithms have been implemented in *PGSolver*, written in *OCaml* by Oliver Friedman and Martin Lange [Fri09, Oli09], a collection of tools to solve, benchmark

and generate parity games. Noteworthy, PGSolver has allowed to declare the Zielonka Recursive Algorithm as the best performing to solve parity games in practice, as well as explore some optimizations such as decomposition into strong connect components, removal of self-cycles on nodes, and priority compression [Ada09, Mar00].

Despite the enormous interest in finding efficient algorithms for solving parity games, less emphasis has been put on the choice of the programming language. Mainly, the scientific community relies on OCaml as the best performing programming language to be used in this setting and PGSolver as an optimal and *de facto* platform to solve parity games. However, starting from graphs with a few thousand of nodes, even using the Zielonka's algorithm, PGSolver would require minutes to decide the given game, especially on dense graphs. Therefore a natural question that arises is whether there exists a way to improve the running time of PGSolver. We identify three research directions to work on, which specifically involve: the algorithm itself, the way it is implemented, and the chosen programming language. As a result we introduce, in this chapter, a slightly improved version of the Classic Zielonka Algorithm along with a heavily optimized implementation in *Scala Programming Language* [Ode04, Ode08]. Scala is a high-level language, proven to be well performing [Hun11], with object and functional oriented features, that recently has come to the fore with useful applications in several fields of computer science including *formal verification* [Bar11]. Our experiments show that, by using all Scala features extensively, we are able of gaining two order of magnitude in running time with respect to the implementation of the Zielonka's algorithm in PGSolver.

In details, the main goal of this work is the design and development of a new tool for solving parity games, based on an improved version of the Zielonka Recursive Algorithm, with performance in mind. Classical Zielonka algorithm requires to decompose the graph game into multiple smaller arenas, which is done by computing, in every recursive call, the *difference* between the current graph and a given set of nodes. This operation (Algorithm 2.1, lines 10 and 15) turns out to be quite expensive as it requires to generate a new graph at each iteration. Somehow such a difference operation has the flavor of the complicancy of complementing automata in formal verification [Tho90]. Remarkably, our improved version guarantees that the original arena remains immutable by tracking the removed nodes in every subsequent call and checking, in constant time, whether a node needs to be excluded or not. Casting this idea in the above automata reasoning, it is like enriching the state space with two flags $(removed, \neg removed)$, instead of performing a complementation.

In this chapter we consider and compare four implementations. The Classic (*C*) and Improved (*I*) Recursive (*R*) algorithms implemented in Scala (*S*) and OCaml (*O*). Using random generated games, we show that *IRO* gains an order of magnitude against *CRO, as well as CRS* against *CRO*. Remarkably, we show that these improvements are cumulative by proving that *IRS* gains two order of magnitude against *CRO*.

We have been able to achieve this kind of performance optimization by deeply studying the way the classic Recursive algorithm has been implemented in PGSolver and concentrating on the following tasks of the algorithm, which we have deeply improved: finding the maximal priority, finding all nodes with a given priority, and removing a node (including related edges) from the graph. Parsing the graph in Scala, we allocate an *Array,* whose size is fixed to the number of nodes of the graph. In addition we populate at the same time the adjacency list and incidence list for each node, which avoids to build a transposed graph. We make also use of an open source Java library called *Trove* that provides a fast and lightweight implementation of the *java.util* Collection API.

Finally, we want to remark that, among all programming languages, we have chosen to investigate Scala as it shares several modern and useful programming language aspects. Among the others, Scala carries functional and object-oriented features, compiles its programs for the JVM, is interoperable with Java and an high-level language with a concise and clear syntax. The results we obtain strongly support our choice and allow to declare Scala as a clear winner over OCaml, in terms of performance.

2.1 Preliminaries

For reasons of self content, in this section we briefly introduce the notion of of Parity Games, the classic version of Zielonka Recursive Algorithm and its implementation in PGSolver.

A *Parity Game* can be defined as a tuple $G = (V, V_0, V_1, E, \Omega)$ where

- $(V,\ E)$ forms a directed graph whose set of nodes is partitioned into $V = V_0 \cup V_1$;
- V_0 and V_1 are two non empty sets of nodes , where $V_0 \cap V_1 = \emptyset$;
- $\Omega : V \to N$ is the *priority function* that assigns to each node a natural number called the *priority* of the node.

We assume E to be *total*, i.e. for every node $v \in V$, there is a node $w \in V$ such that $(v, w) \in E$. In the following we also write vEw in place of $(v, w) \in E$ and use $vE := \{w \mid vEw\}$. Parity games are played between two players called *Player 0* and *Player 1*. Starting in a node $v \in V$, both players construct an infinite path (the *play*) through the graph as follows. If the construction reaches, at a certain point, a finite sequence $v_0...v_n$ and $v_n \in V$ then player i selects a node $w \in v_nE$ and the play continues with the sequence $v_0...v_nw$. Every play has a unique winner, defined by the priority that occurs infinitely often. Precisely, the *winner* of the play $v_0v_1v_2...$ is player i iff $max\{p \mid \forall j \,.\, \exists k \geq j \,:\, \Omega(v_k) = p\}\, mod\, 2 = i$.

Strategy A *strategy* for player i is a partial function $\sigma : V^*V \to V$, such that, for all sequences $v_0...v_n$ with $v_{j+1} \in v_jE$, with $j = 0, ..., n{-}1$, and $v_n \in V_i$ we have that

$\sigma(v_0...v_n) \in v_nE$. A play $v_0v_1...$ *conforms* to a strategy σ for player i if, for all j we have that, if $v_j \in V_i$ then $v_{j+1} = \sigma(v_0...v_j)$.

A strategy σ for player i (σ_i) is a winning strategy in node v if player i wins every play starting in v that conforms to the strategy σ. In that case, we say that player i *wins* the game G starting in v. A strategy σ for player i is called *memoryless* if, for all $v_0...v_n \in V^*V_i$ and for $w_0...w_m \in V^*V_i$, we have that if $v_n = w_m$ then $\sigma(v_0...v_n) = \sigma(w_0...w_m)$. That is, the value of the strategy on a path only depends on the last node on that path. Starting from G we construct two sets $W_0, W_1 \subseteq V$ such that W_i is the set of all nodes v such that Player i wins the game G starting in v. Parity games enjoy *determinacy* meaning that for every node v either $v \in W_0$ or $v \in W_1$ [EJ91]. The problem of solving a given parity game is to compute the sets W_0 and W_1, as well as the corresponding *memoryless* winning strategies, σ_0 for Player 0 and σ_1 for Player 1 on their respective winning regions. The construction procedure of winning regions makes use of the notion of *attractor*.

Attractor Let $U \subseteq V$ and $i \in \{0, 1\}$. The i-attractor of U is the least set W s.t. $U \subseteq W$ and whenever $v \in V_i$ and $vE \cap W \neq \emptyset$, or $v \in V_{1-i}$ and $vE \subseteq W$ then $v \in W$. Hence, the i-attractor of U contains all nodes from which Player *i* can move "towards" U and Player $1 - i$ *must* move "towards" U. The i-attractor of U is denoted by $Attr_i(G, U)$. Let A be an arbitrary attractor set. The game $G \setminus A$ is the game restricted to the nodes $V \setminus A$, *i.e.* $G \setminus A = (V \setminus A, V_0 \setminus A, V_1 \setminus A, E \setminus (A \times V \cup V \times A), \Omega_{|V \setminus A})$. It is worth observing that the totality of $G \setminus A$ is ensured from A being an attractor. Formally, for all $k \in \mathbb{N}$, the i-attractor is defined as follows:

$$
\begin{aligned}
&Attr_i^0(U) = U \ ; \\
&Attr_i^{k+1}(U) = Attr_i^k(U) \cup \{v \in V_i \mid \exists\, w \in Attr_i^k(U)\ s.t.\ vEw\} \\
&\qquad\qquad \cup \{v \in V_{1-i} \mid \forall\, w \,:\, vEw \Rightarrow w \in Attr_i^k(U)\} \ ; \\
&Attr_i(U) = \textstyle\bigcup_{k \in \mathbb{N}} Attr_i^k(U) \ .
\end{aligned}
$$

2.1.1 The Zielonka Recursive Algorithm

Here, we describe the Zielonka Recursive Algorithm, using the basic concepts introduced in the above, and we make some observations regarding its implementation in PGSolver. The algorithm to solve parity games introduced by Zielonka comes from Mc-Naughton's work [McN93]. The Zielonka Recursive Algorithm [McN93], as reported in Figure 2.1, uses a divide and conquer technique to sove parity games. It constructs the winning sets for both players using the solution of subgames. It removes the nodes with the highest priority from the game, together with all nodes (and edges) attracted to this set. The algorithm $win(G)$ takes as input a graph G and, after a number of recursive calls over ad hoc built subgames,

```
1
2  function win(G):
3          if V == ∅:
4                  (W_0, W_1) = (∅, ∅)
5          else:
6                  d = max priority in G
7                  U = {v ∈ V | priority(v) = d}
8                  p = d % 2
9                  j = 1 − p1
10                 A = Attr_p(U)
11                 (W_0', W_1') = win(G \ A)
12                 if W_j' == ∅:
13                         W_p = W_p' ∪ A
14                         W_j = ∅
15                 else:
16                         B = Attr_j(W_1^j)
17                         (W_0', W_1') = win(G \ B)
18                         W_p = W_p'
19                         W_j = W_j' ∪ B
20 return (W_0, W_1)
```

Figure 2.1: Zielonka Recursive Algorithm

returns the winning sets (W_0, W_1) for player 0 and player 1, respectively. The running time of this algorithm is exponential in the number of priorities.

2.2 PGsolver Analisys and Improved Algorithm

PGSolver is a tool developed by Oliver Friedman and Martin Lange [Fri09, Oli09]. This tool is from some years now, the de-facto tool for solving parity games. It contains implementations in OCaml of about 10 algorithms, including the one implemented in this study: Zielonka's Recursive Algorithm. For benchmarking purposes, PGSolver comes with tools to generate different kind of games, from random to special cases, and lets customize the kind of the game wanted passing as command line flags the number of nodes, available priorities, minimum and number of edges. In [Ant14], we used PGSolver as a comparison to generate and then benchmark multiple games. PGSolver offers many optimizations techniques found in literature for example Jurdziński suggests to perform SCCs decomposition of a graph. Other optimizations are related to detection of special cases such as self-cycle games, one-parity game, one-player game, and priority compression and propagation. The step of priority compression attempts to reduce the number of priorities in a parity game, while the priority propagation aimed to increase priority but to reduce the range of priorities in a game and therefore compress the overall priorities. The tool itself allowed to explore the previously hidden area of practical process. For example, contrary to common believe, an increasing large number of priorities does not necessarily impose a great difficulty in practice, this observation was also confirmed in [Ant14]. The recursive algorithm by Zielonka, was declared the best

performing one when compared to the other algorithms if no optimizations or preprocessing steps were applied. Also, SCC decomposition was proven to be highly profitable alongside the elimination of self-cycles. It is important to note that not every optimization can speed up every algorithm. For example, the recursive algorithm can achieve best result from the elimination of self-cycles and priority compression; this is highly reasonable due to the fact that without self-cycle elimination it would require more recursive calls. This work is a result of a deep analysis of PGSolver's capabilities in solving parity game in an efficient and performant manner. In more details, even using the Zielonka's Recursive Algorithm, with SCC decompositions enabled PGSolver would require minutes to decide games with few thousands of nodes, especially on dense graphs. Our investigation starts with the way Zielonka's Recursive has been implemented: the graph data structure is represented as a fixed length Array of tuples, where every tuple contains information about a node, such as the player, priority and adjacency list. Before every recursive call is performed, the implementation performs the difference between the actual graph and the attractor set, outputting a new graph as well as building the transposed graph. In addition the attractor function implemented in PGSolver uses a TreeSet as data structure guaranteeing only logarithmic search, inserts and removals. One may say that the added complexity for making a new graph or building the transposed is still linear time on the actual graph, but it is worth noting that general-purpose memory allocators are very expensive as the per-operation cost floats around one hundred processor cycles [Gay98]. Through these years many efforts have been made to improve memory allocation writing custom allocators from scratch, a process known to be difficult and error prone [Ber01, Ber13].

In more detail, our implementation our implementation focuses on three key point of the Recursive Zielonka Algorithm. The first point is that in which the algorithm computes the difference between the graph and the attractor, returning a new graph (see lines 11 and 17, Figure 2.1). The second point is that in which, in every call the attractor function builds the transposed graph (see line 10, Figure 2.1). The last point is that in which the attractor calculates the number of successors for the opponent player, in every iteration, possibly visiting several times the same node (see line 16, Figure 2.1). Thanks to these arguments and with the aim of performance optimizations, a slightly improved version of the recursive algorithm has been obtained. The improved algorithm and its attractor function are listed, respectively, in Figure 2.2 and 2.3.

Let G be a graph. Removing a node from G and building the transposed graph takes time $\Theta(|V| + |E|)$. Thus dealing with dense graph such operation takes $\Theta(|V|^2)$. In order to reduce the running time complexity caused by these graph operations, we a requirement for immutability of the graph G ensuring that every recursive call uses the graph without applying any modification to the state of the graph. Therefore, to construct the sub-games, in the recursive calls, we keep track of each node that is going to be removed from the graph,

```
function win (G):
        T = G.transpose()
        Removed = {}
        return winI(G, T, Removed)

function winI(G, T, Removed):
        if |V| == |Removed|:
                return (∅, ∅)
        W = (∅, ∅)
        d = maximal priority in G
        U = { v ∈ V | priority(v) = d }
        p = d % 2
        j = 1 − p
        W' = (∅, ∅)
        A = Attr(G, T, Removed, U, p)
        (W'_0, W'_1) = winI(G, T, Removed ∪ A)
        if W'_j == ∅:
                W_p = W'_p ∪ A
                W_j = ∅
        else:
                B = Attr(G, T, Removed, W'_j, j)
                (W'_0, W'_1) = winI(G, T, Removed ∪ B)
                W_p = W'_p
                W_j = W'_j ∪ B
        return (W_0, W_1)
```

Figure 2.2: Improved Recursive Algorithm

```
function Attr(G, T, Removed, A, i):
    tmpMap = []
    for x = 0 to |V|:
        if x ∈ A tmpMap = 0
        else tmpMap = −1
    index = 0
    while index < |A|:
        for v_0 ∈ adj(T, A[index]):
            if v_0 ∉ Removed:
                if tmpMap[v_0] == −1:
                    if player(v_0) == i:
                        A = A ∪ v_0
                        tmpMap[v_0] = 0
                    else:
                        adj_counter = −1
                        for x ∈ adj(G, v_0):
                            if (x ∉ Removed):
                                adj_counter += 1
                        tmpMap[v_0] = adj_counter
                        if adj_counter == 0:
                            A = A ∪ v_0
                else if (player(v_0) == j
                        and tmpMap[v_0] > 0):
                    tmpMap[v_0] −= 1
                    if tmpMap[v_0] == 0:
                        A = A ∪ v_0
    return A
```

Figure 2.3: Improved Recursive Attractor

```
def win(G: GraphWithSets)
: (ArrayBuffer[Int], ArrayBuffer[Int]) = {
          val W = Array(ArrayBuffer.empty[Int], ArrayBuffer.empty[Int])
          val d = G.max_priority()
          if (d > -1) {
            val U = G.priorityMap.get(d) .filter(p => !G.exclude(p))
            val p = d % 2
            val j = 1 - p
            val Wl = Array(ArrayBuffer.empty[Int], ArrayBuffer.empty[Int])
            val A = Attr(G, U, p)
            val res = win(G -- A)
            Wl(0) = res._1
            Wl(1) = res._2
            if (W(j).size == 0) {
              W(p) = Wl(p) ++= A
              W(j) = ArrayBuffer.empty[Int]
                    } else {
               val B = Attr(G, Wl(j), j)
               val res2 = win(G -- B)
              Wl(0) = res2._1
              Wl(1) = res2._2
              W(p) = Wl(p)
              W(j) = Wl(j) ++= B
    }
  }
  (W(0), W(1))
}
```

Figure 2.4: Improved Algorithm in Scala

adding all of them to a set called Removed. The improved algorithm is capable of checking if a given node is excluded or not in constant time as well as it completely removes the need for a new graph in every recursive call.

2.3 Scala Implementations

Scala is the programming language designed by Martin Odersky, the codesigner of Java Generics and main author of javac compiler. Scala defines itself as a scalable language, statically typed, a fusion of an object-oriented language and a functional one. It runs on the Java Virtual Machine (JVM) and supports every existing Java library. Scala is a purely object-oriented language in which, like Java and Smalltalk, every value is an object and every operation is a method call. In addition Scala is a functional language where every function is a first class object, also is equipped with efficient immutable data structures, with a strong selling point given by Java interoperability. However, it is not a purely functional language as objects may change their states and functions may have side effects. The functional aspects are perfectly integrated with the object-oriented features. The combination of both styles makes possible to express new kinds of patterns and abstractions. All these features make Scala programming language as a clever choice to solve these tasks, in a strict comparison with other programming languages available such as C, C++ or Java. Historically, the first generation of the JVM was entirely an interpreter; nowadays the JVM uses a Just-In-Time (JIT) compiler, a complex process aimed to improve performance at runtime. This process

can be described in three steps: (1) source files are compiled by the Scala Compiler into Java Bytecode, that will be feed to a JVM; (2) the JVM will load the compiled classes at runtime and execute proper computation using an interpreter; (3) the JVM will analyze the application method calls and compile the bytecode into native machine code. This step is done in a lazy manner: the JIT compiles a code path when it knows that is about to be executed. JIT removed the overhead of interpretation and allows programs to start up quickly, in addition this kind of compilation has to be fast to prevent influencing the actual performance of the program. Another interesting aspect of the JVM is that it verifies every class file after loading them. This makes sure that the execution step does not violate some defined safety properties. The checks are performed by the verifier that includes a complete type checking of the entire program. The JVM is also available on all major platforms and compiled Java executables can run on all of them with no need for recompilation. The Scala compiler scalac compiles a Scala program into Java class files. The compiler is organized in a sequence of successive steps. The first one is called the front-end step and performs an analysis of the input file, makes sure that is a valid Scala program and produces an attributed abstract syntax tree (AST); the back-end step simplifies the AST and proceeds to the generation phase where it produces the actual class files, which constitute the final output. Targeting the JVM, the Scala Compiler checks that the produced code is type-correct in order to be accepted by the JVM bytecode verifier. In [20], published by Google, Scala even being an high level language, performs just 2.5x slower than C++ machine optimized code. In particular it has been proved to be even faster than Java. As the paper notes: *"While the benchmark itself is simple and compact, it employs many language features, in particular high level data structures, a few algorithms, iterations over collection types, some object oriented features and interesting memory allocation patterns"*.

2.3.1 Improved Algorithm in Scala

In this section we introduce our implementation of the Improved Recursive Algorithm in Scala, listed as Figure 2.4 and Figure 2.5. Aiming at performance optimizations we use a *priority HashMap* where every *key* is a certain priority and a *value* is a set of each node v where $priority(v) = key$. As fast and JVM-Optimized *HashMaps* and *ArrayLists* we use the ones included in the open source library *Trove*. In addition, using the well known *strategy pattern* [GHJV94] we open the framework for further extentions and improvements. The intended purpose of our algorithm is to assert that the performance of existing tools for solving parity games can be improved using the improved algorithm and choosing Scala as the programming language. We rely on Scala's internal features and standard library making heavy use of the dynamic *ArrayBuffer* data structure. In order to store the arena we use an array of *Node* objects. The Node class contains: a list of adjacent nodes, a list of incident nodes, its priority and the player; the data structure also implements a factory method called "$--(set : ArrayBuffer[Int])$" that takes an ArrayBuffer of integers as input, flags all the

```
def Attr(G: GraphWithSets,
   A: ArrayBuffer[Int], i: Int)
   : ArrayBuffer[Int] = {
  val tmpMap = Array
    .fill[Int](G.nodes.size)(-1)
  var index = 0
  A.foreach(tmpMap(_) = 0)
  while (index < A.size) {
    G.nodes(A(index))
      .<~.foreach(v0 => {
        if (!G.exclude(v0)) {
          val flag = G.nodes(v0).player == i
          if (tmpMap(v0) == -1) {
            if (flag) {
              A += v0
              tmpMap(v0) = 0
            } else {
              val tmp = G.nodes(v0)
               .~>
               .count(x => !G.exclude(x)) - 1
              tmpMap(v0) = tmp
              if (tmp == 0) A += v0
            }
          } else if (!flag && tmpMap(v0) > 0){
            tmpMap(v0) -= 1
            if (tmpMap(v0) == 0) A += v0
          }
        }
    })
    index += 1
  }
  A
}
```

Figure 2.5: Improved Attractor in Scala

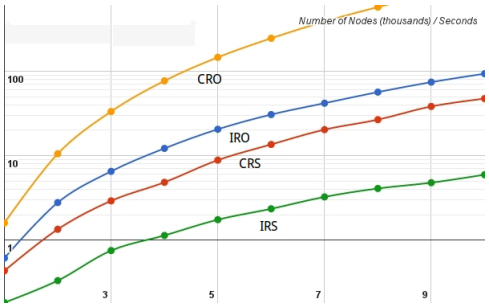

Figure 2.6: Random Games Chart in Logarithmic Scale

nodes in the array as excluded, and returns the reference to the new graph. In addition, there is also a method called $max\_priority()$ that will return the maximal priority in the graph and the set of nodes with that priority. The Attractor function makes deeply use of an array of integers named $tmpMap$ that is preallocated using the number of nodes in the graph with a negative integer as default value; we use $tmpMap$ when looping through every node in the set A given as parameter, to keep track of the number of successors for the opponent player. We add a node $v \in V$ to the attractor set when its counter (stored in *tmpMap[v]*) reaches 0 ($adj(v) \subseteq A$ and $v \in V_{opponent}$) or if $v \in V_{player}$; using an array of integers, or an HashMap, to serve this purpose, guarantees a constant time check if a node was already visited and ensures that the count for the opponent's node adjacency list takes place one time only. These functions are inside a singleton object called *ImprovedRecursiveSolver* that extends the *Solver* interface.

2.4 Benchmarks

In this section we study, analyze and evaluate the running time of our four implementations: *Classic Recursive in OCaml* (*CRO*), *Classic Recursive in Scala* (*CRS*), *Improved Recursive in OCaml* (*IRO*) and *Improved Recursive in Scala* (*IRS*). We have run our experiments on multiple instances of random parity games. We want to note that *IRS* does not apply any preprocessing steps to the arena before solving. All tests have been run on an Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, with 16GB of Ram (with no Swap available) running Ubuntu 14.04. Precisely, we have used 100 random arenas generated using PGSolver of each of the following types, given $N = i \times 1000$ with i integer and $1 \leq i \leq 10$ and a timeout set at 600 seconds. In the following, we report six tables in which we show the running time of all experiments under fixed parameters. Throughout this section we define abo_T when the program has been aborted due to excessive time and abo_M when the program has been killed by the Operating System due to memory consumption. In Figure 2.6 we also report the trends of the four implementations using a logarithmic scale with respect to *seconds*. This figure is based on the

averages of all results reported in the tables below.

N nodes, N colors, $adj(\frac{N}{2},\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	0.204	1.99	0.505	0.752
2×10^3	**0.456**	13.208	1.918	3.664
3×10^3	**1.031**	41.493	2.656	6.147
4×10^3	**1.879**	96.847	6.728	15.966
5×10^3	**2.977**	183.589	12.616	27.272
6×10^3	**3.993**	306.104	19.032	41.051
7×10^3	**4.989**	486.368	27.05	50.367
8×10^3	**6.103**	abo_T	36.597	70.972
9×10^3	**7.287**	abo_T	55.171	97.216
10×10^3	**8.468**	abo_T	68.303	113.36

N nodes, N colors, $adj(1,\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	**0.179**	1.21	0.454	0.583
2×10^3	**0.389**	8.075	1.173	2.366
3×10^3	**0.868**	25.097	2.656	6.147
4×10^3	**1.279**	57.186	4.23	10.452
5×10^3	**2.273**	108.983	9.206	20.377
6×10^3	**2.772**	183.884	12.562	27.489
7×10^3	**3.748**	291.077	17.942	37.521
8×10^3	**3.942**	418.377	22.105	47.502
9×10^3	**4.989**	593.721	23.93	61.593
10×10^3	**6.413**	abo_T	42.408	80.508

N nodes, 2 colors, $adj(\frac{N}{2},\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	**0.189**	1.98	0.481	0.702
2×10^3	**0.469**	12.941	1.55	3.17
3×10^3	**1.046**	41.584	3.995	7.428
4×10^3	**1.712**	96.545	5.378	13.823
5×10^3	**2.414**	181.225	11.273	22.575
6×10^3	**3.458**	307.233	16.472	35.269
7×10^3	**4.612**	484.159	26.448	49.311
8×10^3	**6.003**	abo_T	28.968	65.674
9×10^3	**7.03**	abo_T	43.666	85.909
10×10^3	**8.938**	abo_T	57.18	110.814

N nodes, 2 colors, $adj(1,\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	**0.159**	1.226	0.385	0.468
2×10^3	**0.341**	7.965	1.004	2.162
3×10^3	**0.797**	25.114	2.305	6.014
4×10^3	**1.123**	56.422	3.699	9.421
5×10^3	**1.704**	108.584	6.12	14.971
6×10^3	**2.243**	182.935	10.099	22.621
7×10^3	**3.324**	286.503	13.898	32.335
8×10^3	**3.95**	430.265	19.743	44.281
9×10^3	**4.597**	abo_T	28.742	56.81
10×10^3	**5.651**	abo_T	33.639	71.434

N nodes, $\sqrt{N}$ colors, $adj(\frac{N}{2},\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	**0.204**	1.978	0.468	0.71
2×10^3	**0.456**	13.114	1.575	3.203
3×10^3	**1.031**	41.493	3.868	7.492
4×10^3	**1.621**	96.55	5.744	13.97
5×10^3	**2.439**	183.589	10.72	22.98
6×10^3	**3.372**	307.426	15.978	34.78
7×10^3	**4.662**	485.826	26.432	48.875
8×10^3	**6.499**	abo_T	34.741	66.423
9×10^3	**7.147**	abo_T	48.915	86.645
10×10^3	**8.988**	abo_T	56.656	111.492

N nodes, $\sqrt{N}$ colors, $adj(1,\ N)$

N	*IRS*	*CRO*	*CRS*	*IRO*
1×10^3	**0.162**	1.218	0.384	0.475
2×10^3	**0.344**	7.947	1.034	2.195
3×10^3	**0.788**	25.029	2.406	5.944
4×10^3	**1.105**	57.307	3.835	9.608
5×10^3	**1.678**	108.623	6.34	15.165
6×10^3	**2.281**	182.154	9.871	22.859
7×10^3	**3.193**	285.28	14.338	32.536
8×10^3	**4.185**	422.74	20.362	44.515
9×10^3	**5.009**	599.071	24.347	57.022
10×10^3	**5.76**	abo_T	35.024	72.291

2.4.1 Trends Analysis for Random Arenas

The speedup obtained by our implementation of the Improved Recursive Algorithm is in most cases quite noticeable. Figure 2.7 shows the running time trend for Improved and Classic Algorithm on each platform. The seconds are limited to $[0, 100]$. As a result we show that even with all preprocessing steps enabled in PGSolver, *IRS* is capable of gaining two orders of magnitude in running time.

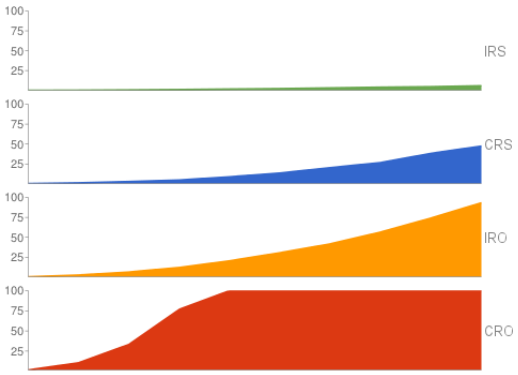

Figure 2.7: Trends Chart

2.4.2 Trends Analysis for Special Games

Focusing on Classic Recursive in PGSolver and our Improved Recursive in Scala, here we show the running times for non-random games generated by PGSolver. In particular we use four types of non-random games, these experiments have been run against PGSolver using the Classic Recursive Algorithm with all optimizations disabled and all solutions were matched to ensure correctness.

Clique[n] games are fully connected games without self-loops, where *n* is the number of nodes. The set of nodes is partitioned into V_0 and V_1 having the same size. For all $v \in V_p$, $priority(v)\,\%\,2 = p$. For our experiments we set $n = 2^k$where $8 \leq k \leq 14$. Table below reports the running time for our experiments and these results are drawn in Figure 2.8.

n	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
IRS	0.05	0.07	0.12	0.46	1.18	4.87	17.39
CRO	0.09	0.61	4.37	29.58	229.78	abo_T	abo_M

In **Ladder[n]** game, every node in V_0 has priority 2 and every node in V_1 has priority 1. In addition, each node $v \in V$ has two successors: one in V_0 and one in V_1, which form a node pair. Every pair is connected to the next pair forming a *ladder* of pairs. Finally, the last pair is connected to the top. The parameter *n* specifies the number of node

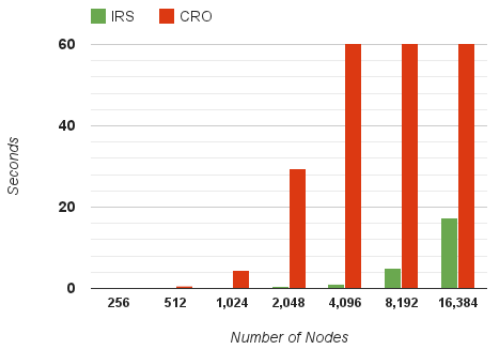

Figure 2.8: Clique Trends

pairs. For our tests, we set $n = 2^k$where $7 \leq k \leq 19$ and report our experiments in the table below whose trend is drawn in Figure 2.9.

Figure 2.9 shows better performance for *CRO* than *IRS* using low-scaled values as input parameter. This behaviour is not surprising as there is a *warming-up* time required by the Java Virtual Machine.

n	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
IRS	0.01	0.02	0.03	0.05	0.08	0.11	0.13	0.15	0.19	0.25	0.38	0.48	0.93
CRO	0.00	0.00	0.01	0.01	0.03	0.06	0.13	0.3	0.65	1.39	2.93	6.21	11.71

Model Checker Ladder[n] consists of overlapping blocks of four nodes, where the parameter *n* specifies the number of desidered blocks. Every node is owned by player 1, $V_1 = V$ and $V_0 = \emptyset$, and the nodes are connected such that every cycle passes through a single point of colour 0. For our experiments we set $n = 2^k$ where $7 \leq k \leq 19$, report our experiments in the table below and draw the trends in Figure 2.10.

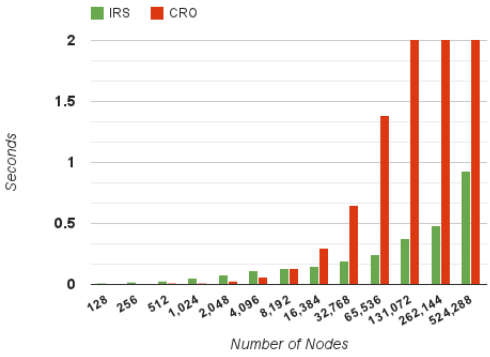

Figure 2.9: Ladder Trends

n	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
IRS	0.01	0.02	0.03	0.04	0.07	0.12	0.14	0.16	0.19	0.21	0.26	0.39	0.65
CRO	0.00	0.00	0.01	0.01	0.02	0.05	0.10	0.22	0.47	0.99	2.12	4.16	8.31

Jurdzinski[n, m] games are designed to generate the worst-case behaviour for the Small Progress Measure Solver [Mar00]. The parameter *n* is the number of layers, where each layer has *m* repeating blocks that are interconnected as described in [Mar00].

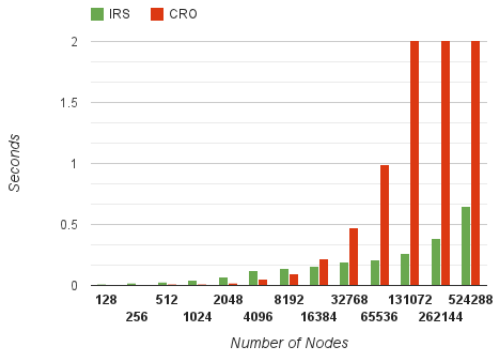

Figure 2.10: Model Checker Ladder Trends

As this game takes two parameters, in our test we ran two experiments: one where *n* is fixed to 10 and $m = 10 \times 2^k$, for $k = 1, 2, 3, 4, 5$ and one where m is fixed to 10 and $n = 10 \times 2^k$, for $k = 1, 2, 3, 4, 5$. The results of our expreriments are reported in the

m	10×2^1	10×2^2	10×2^3	10×2^4	10×2^5
IRS	0.21	0.48	1.54	4.55	15.31
CRO	0.23	0.79	3.14	15.77	65.85
n	10×2^1	10×2^2	10×2^3	10×2^4	10×2^5
IRS	0.28	0.77	3.02	30.02	232.24
CRO	0.42	2.94	22.69	184.12	abo_T

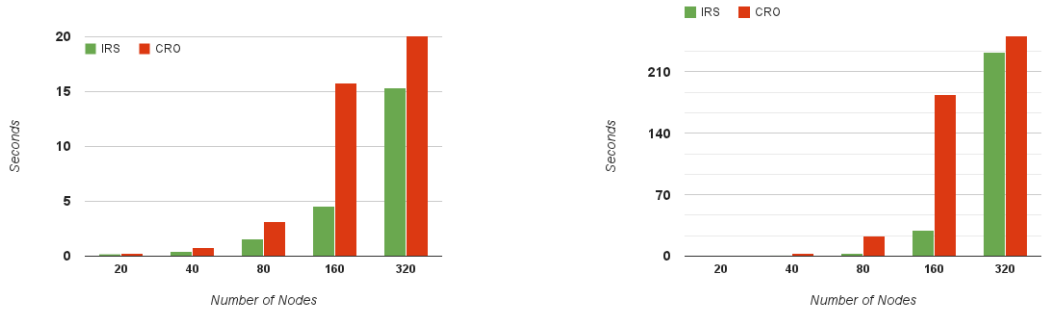

Table 2.2: Jurdzinski Trends.

tables below. The trends are drawn in Table 2.2.

2.5 Discussion

PGSolver is a well-stablished framework that collects multiple algorithms to decide parity games. For several years now this platform has been the only one available to solve and benchmark in practice. Given PGSolver's limitations addressing huge graphs, several attempts of improvement have been carried out recently. Some of them have been implemented as preprocessing steps in the tool itself (such as priority compression or SCC decomposition and the like), while others chose parallelism techniques, such as Cuda [**?**], applied to the algorithms. However these improvements often do not show the desired performance.

In this chapter we started from scratch by revisiting the Zielonka Recursive Algorithm, implemented an improved and the classic versions in Scala and OCaml, comparing among them. The choice of Scala as a programming language has been not casual, but rather it comes out from a deep study focused on performance and simplicity. Scala is interoperable with Java libraries, has a concise and clear syntax, functional and object oriented features, runs on the Java Virtual Machine and has been proven to be high performing. Our main result is a new and fast tool for solving parity games capable of gaining up to two orders of magnitude in running time. In conclusion we state that there is place for a faster and better framework to solve parity games and this work is a starting point raising several interesting questions. For

example, what if one implements the other known algorithms to solve parity games in Scala? PGSolver showed that Zielonka's algorithm is the best performing. Can one reproduce the same results in Scala? We leave all these questions as future work.

CHAPTER 3

Graded Strategy Logic

Contents

In the game-theoretic setting modeling critical scenarios, knowing the existence of a strategy to achieve a certain goal is sometimes not sufficient. It may be vital, indeed, to ensure that some redundant strategy to play exists in case of some fault. Establishing how many different strategies a game admits for its agents allows to grade its resilience as well. In this chapter, we overcome the above limitations by introducing and studying *Graded Strategy Logic* (GSL) as an extension, along with *graded quantifiers*, of the recently introduced framework of *Strategy Logic* [MMV10]. Precisely, in GSL we make use of the existential $\langle\langle x \geq g \rangle\rangle \varphi$ and universal $[\![x < g]\!]\varphi$ graded strategy quantifiers to require that *there are at least* g or *all but less than* g strategies x satisfying φ, respectively. Then, by using the classical binding operator of SL, it is possible to associate these strategies to specific agents.

GSL can have useful applications in several multi-agent game scenarios. For example, in safety-critical systems, it may be worth knowing whether a controller agent has a redundant winning strategy to play in case of some fault. Having more than a strategy may increase the chances for a success [ATO$^+$09]. Such a redundancy can be easily expressed in GSL by requiring that at least two different strategies exist for the achievement of the safety goal. Another useful example concerns *Nash Equilibria*. With GSL one can determine whether there exists more than a winning strategy and so derive important game-theoretic properties about the game such as *uniqueness* of the equilibrium.

On dealing with GSL formulas, an aspect that requires some attention is the way strategies are counted. Indeed there may be strategies that look different but produce the same outcome, which therefore need to be count as one. To this aim, we introduce a suitable equivalence relation over profiles based on the strategic behavior they induce and this represents by itself an important contributions of this chapter. Other contributions relate to the investigation of basic game-theoretic and verification questions over a fragment of GSL. Recall that model checking is non-elementary-complete for SL and thus there is no hope for a better complexity for GSL. For this reason we have concentrated on the *vanilla* version of the SL$[1\mathrm{G}]$ fragment of SL. We recall that SL$[1\mathrm{G}]$ has been introduced in [MMPV12]. As for ATL, the vanilla version of SL$[1\mathrm{G}]$ further requires that two temporal operators in a formula are always interleaved by a strategy quantifier. As main result, we prove that the model-checking problem is PTIME-COMPLETE. Moreover, we have studied and obtained positive results about the determinacy of turn-based games.

3.1 Graded Strategy Logic

In this section, we introduce syntax and semantics of *Graded Strategy Logic* (GSL, for short), an extension of *Strategy Logic* (SL, for short) [MMV10], that allows to reason about the number of strategies that an agent may exploit in order to satisfy a given temporal goal. We recall that SL simply extends LTL with two strategy quantifiers and a binding construct used

to associate an agent to a strategy.

3.1.1 Model

Similarly to SL, as semantic framework we use a *game structure* [AHK02], *i.e.*, a generalization of both *Kripke structures* [Kri63] and *labeled transition systems* [Kel76], in which the system is modeled as a game where players perform actions chosen strategically as a function on the history of the play.

Definition 3.1.1 (Game Structure) *A* game structure *is a tuple* $\mathcal{G} \triangleq \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \mathsf{tr}, \mathsf{ap}, s_{\mathrm{o}} \rangle$*, where* AP *and* Ag *are finite non-empty sets of* atomic propositions *and* agents*,* Ac *and* St *are enumerable non-empty sets of* actions *and* states*,* $s_I \in \mathrm{St}$ *is a designated* initial state*, and* $\mathsf{ap} : \mathrm{St} \to 2^{\mathrm{AP}}$ *is a* labeling function *that maps each state to the set of atomic propositions true in that state. Let* $\mathrm{Dc} \triangleq \mathrm{Ag} \rightharpoonup \mathrm{Ac}$ *to be the set of* decisions*,* i.e.*, partial functions describing the choices of an action by some agent. Then,* $\mathsf{tr} : \mathrm{Dc} \to (\mathrm{St} \rightharpoonup \mathrm{St})$ *denotes the* transition function *mapping every decision* $\delta \in \mathrm{Dc}$ *to a partial function* $\mathsf{tr}(\delta) \subseteq \mathrm{St} \times \mathrm{St}$ *representing a deterministic graph over the states.*

The set of decisions in a state $s \in \mathrm{St}$ is $\mathsf{dc}(s) \triangleq \{\delta \in \mathrm{Dc} : s \in \mathsf{dom}(\mathsf{tr}(\delta))\}$. We require the absence of end-states, *i.e.*, $\mathsf{dc}(s) \neq \emptyset$. Also, we define the active agents in s as $\mathsf{ag}(s) \triangleq \{a \in \mathrm{Ag} : \exists \delta \in \mathsf{dc}(s) \,.\, a \in \mathsf{dom}(\delta)\}$ and the related associated actions as $\mathsf{ac}(s, a) \triangleq \{\delta(a) \in \mathrm{Ac} : \delta \in \mathsf{dc}(s) \wedge a \in \mathsf{dom}(\delta)\}$. A game structure $\mathcal{G}$ naturally induces a graph $\langle \mathrm{St}, Ed \rangle$ with $Ed = \bigcup_{\delta \in \mathrm{Dc}} \mathsf{tr}(\delta)$, where the infinite paths starting at the initial state s_I represent all possible *plays* (whose set is denoted by Pth) and its finite paths are called *histories* (whose set is denoted by Hst). A *strategy* is a function $\sigma \in \mathrm{Str} \triangleq \mathrm{Hst} \to \mathrm{Ac}$ prescribing which action has to be performed given a certain history. In particular, we say that $\sigma \in \mathrm{Str}(\mathrm{A}) \subseteq \mathrm{Str}$ is A-*coherent w.r.t.* a set of agents $\mathrm{A} \subseteq \mathrm{Ag}$ if $\sigma(\rho \cdot s) \in \mathsf{ac}(s, a)$, for all histories $\rho \cdot s \in \mathrm{Hst}$ and agents $a \in \mathsf{ag}(s) \cap \mathrm{A}$. Intuitively, this means that σ only prescribes actions that can be used by all agents in A. We also use of the classic concepts of *profile* $\xi \in \mathrm{Prf} \subseteq \mathrm{Ag} \to \mathrm{Str}$, which specifies an a-coherent strategy $\xi(a)$, for each agent $a \in \mathrm{Ag}$, and of the associated *play* $\pi = play(\xi)$.

As a running example, consider the game structure $\mathcal{G}_S$ depicted in Figure 3.1. It represents a model of a *scheduler system* in which two *processes*, $\mathtt{P}_1$ and $\mathtt{P}_2$, can require the access to a *shared resource*, like a processor, and an *arbiter* `A` is used to solve all conflicts that may arise when contending requests are made. The processes can use four actions: `i` for idle, `r` for request, `f` for free/release, and `a` for abandon/relinquish. The first means that the process does not want to change the current situation in which the entire system reside. The second is used to ask for the resource, when this is not yet owned, while the third releases it. Finally, the last is asserted by a process that, although has asked for the resource, did not obtain it and so it decides to relinquish the request. The whole scheduler system can reside in the

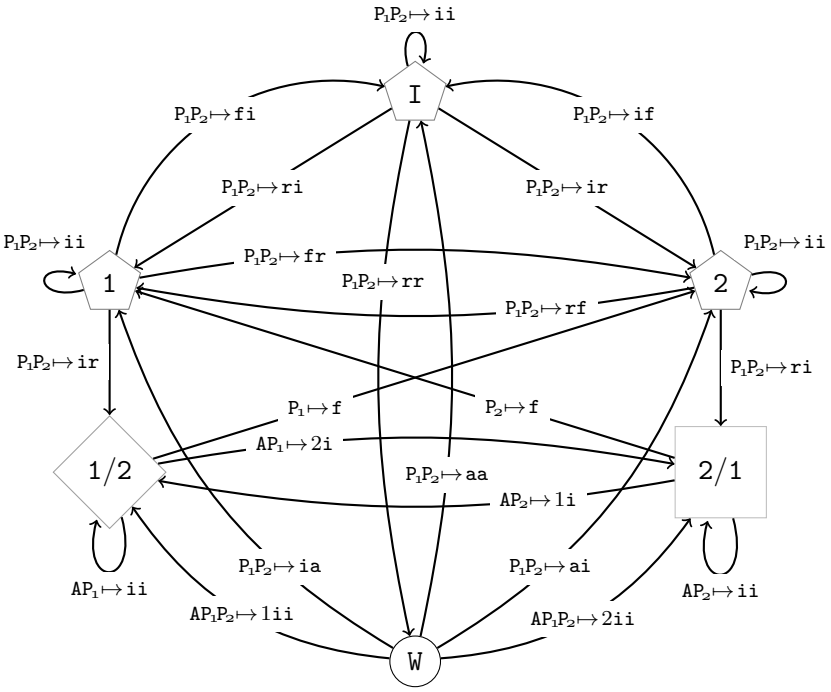

Figure 3.1: A scheduler system $\mathcal{G}_S$.

states I, 1, 2, 1/2, 2/1 and W, where the first three are ruled by the processes, the last by all the agents, and 1/2 (resp, 2/1) by $\mathtt{P}_1$ (resp., $\mathtt{P}_2$) and A. The idle state I indicates that none of the processes owns the resource, while a state $k \in \{1, 2\}$ asserts that process $\mathtt{P}_k$ is using it. The state 1/2 (resp. 2/1) indicates that the process $\mathtt{P}_1$ (resp., $\mathtt{P}_2$) has the resource, while its competitor requires it. Finally, the waiting state W represents the situation in which an action from the arbiter is required in order to solve a conflict between contending requests. To denote who is the owner of the resource, we label 1 and 1/2 (resp., 2 and 2/1) with the atomic proposition $\mathtt{r}_1$ (resp., $\mathtt{r}_2$). A decision is graphically represented by $\vec{\mathtt{a}} \mapsto \vec{\mathtt{c}}$, where $\vec{\mathtt{a}}$ is a sequence of agents and $\vec{\mathtt{c}}$ is a sequence of corresponding actions. For example $\mathtt{P}_1\mathtt{P}_2 \to \mathtt{ir}$ indicates that agents $\mathtt{P}_1$ and $\mathtt{P}_2$ take actions i and r, respectively.

3.1.2 Syntax

GSL extends SL by replacing the classic universal and existential *strategy quantifiers* $\langle\!\langle x \rangle\!\rangle$ and $[\![x]\!]$, where x belongs to a countable set of variables Vr, with their graded version $\langle\!\langle x \geq g \rangle\!\rangle$ and $[\![x < g]\!]$, in which the finite number $g \in \mathbb{N}$ denotes the corresponding *degree*. Intuitively, these quantifiers are read as *"there exist at least g strategies"* and *"all but less than g strategies"*, respectively.

Definition 3.1.2 (GSL **Syntax)** GSL formulas *are built inductively by means of the following context-free grammar, where $a \in$ Ag, $x \in$ Vr, and $g \in \mathbb{N}$:*

$$\varphi := \mathrm{LTL}(\varphi) \mid \langle\!\langle x \geq g \rangle\!\rangle \varphi \mid [\![x < g]\!] \varphi \mid (a, x)\varphi.$$

As usual, to provide the semantics of a predicative logic, it is necessary to define the concept of free and bound *placeholders* of a formula. As for SL, since strategies can be associated to both agents and variables, we need the set of *free agents/variables* $\mathsf{free}(\varphi)$ as the subset of $\mathrm{Ag} \cup \mathrm{Vr}$ containing *(i)* all agents a for which there is no binding (a, x) before the occurrence of a temporal operator and *(ii)* all variables x for which there is a binding (a, x) but no quantification $\langle\!\langle x \geq g \rangle\!\rangle$ or $[\![x < g]\!]$. A detailed definition can be found in [MMPV14]. In case $\mathsf{free}(\varphi) = \emptyset$ the formula φ is named *sentence*. Since a variable x may be bound to more than one agent at the time, we also need the subset $\mathsf{shr}(\varphi, x)$ of Ag containing those agents for which a binding (a, x) occurs in φ.

In this chapter, we prefer to focus on the *One-Goal* fragment of GSL (GSL[1G], for short) [MMPV12, MMPV14] that is already able to describe interesting properties that are not expressible in graded ATL. To formalize its syntax, we first need to introduce some notions. A *quantification prefix* over a set $\mathrm{V} \subseteq \mathrm{Vr}$ of variables is a word $\wp \in \{\langle\!\langle x \geq g \rangle\!\rangle, [\![x < g]\!] : x \in \mathrm{V} \wedge g \in \mathbb{N}\}^{|\mathrm{V}|}$ of length $|\mathrm{V}|$ such that each $x \in \mathrm{V}$ occurs just once in $\wp$. A *binding prefix* over $\mathrm{A} \subseteq \mathrm{Ag}$ is a word $\flat \in \{(a, x) : a \in \mathrm{A} \wedge x \in \mathrm{Vr}\}^{|\mathrm{A}|}$ such that each $a \in \mathrm{A}$ occurs exactly once in $\flat$. We now have all tools to define the syntactic fragment we want to analyze. The idea behind GSL[1G] is that, after a quantification prefix, we can just have a single goal, *i.e.*, a formula of the kind $\flat\psi$, where $\flat$ is a binding prefix.

Definition 3.1.3 (GSL[1G] **Syntax)** GSL[1G] formulas *are built inductively through the following context-free grammar, where $\wp$ and $\flat$ are quantification and binding prefixes:*

$$\varphi := \mathrm{LTL}(\varphi) \mid \wp\flat\varphi.$$

An example of GSL[1G] property, in the context of the scheduler system, is given by the sentence $\varphi = \wp\flat\psi$, with $\wp = \langle\!\langle \mathtt{x} \geq k \rangle\!\rangle [\![\mathtt{y}_1 < g_1]\!] [\![\mathtt{y}_2 < g_2]\!]$, $\flat = (\mathtt{A}, \mathtt{x})(\mathtt{P}_1, \mathtt{y}_1)(\mathtt{P}_2, \mathtt{y}_2)$, and $\psi = \mathtt{F}(\mathtt{r}_1 \vee \mathtt{r}_2)$, which asserts the existence of at least k strategies for the arbiter $\mathtt{A}$ to ensure that one of the two processes $\mathtt{P}_1$ and $\mathtt{P}_2$ receives the resource, once less that g_1 and g_2 strategies, can be avoided by them, respectively.

3.1.3 Semantics

Similarly to SL, the interpretation of a GSL formula requires a valuation for its free placeholders. This is done via *assignments*, *i.e.*, partial functions $\chi \in \mathrm{Asg} \triangleq (\mathrm{Vr} \cup \mathrm{Ag}) \rightharpoonup \mathrm{Str}$ mapping variables and agents to strategies. An assignment χ is *complete* iff it is defined on all agents, *i.e.*, $\chi(a) \in \mathrm{Str}(a)$, for all $a \in \mathrm{Ag} \subseteq \mathsf{dom}(\chi)$. In this case, it directly identifies the profile $\chi_{\restriction \mathrm{Ag}}$ given by the restriction of χ to Ag. In addition, $\chi[e \mapsto \sigma]$, with $e \in \mathrm{Vr} \cup \mathrm{Ag}$ and $\sigma \in \mathrm{Str}$, is the assignment defined on $\mathsf{dom}(\chi[e \mapsto \sigma]) \triangleq \mathsf{dom}(\chi) \cup \{e\}$

that differs from χ only on the fact that e is associated with σ. Formally, $\chi[e \mapsto \sigma](e) = \sigma$ and $\chi[e \mapsto \sigma](e') = \chi(e')$, for all $e' \in \mathsf{dom}(\chi) \setminus \{e\}$. Finally, given a formula φ, we say that χ is φ-*coherent* iff *(i)* $\mathsf{free}(\varphi) \subseteq \mathsf{dom}(\chi)$, *(ii)* $\chi(a) \in \mathrm{Str}(a)$, for all $a \in \mathsf{dom}(\chi) \cap \mathrm{Ag}$, and *(iii)* $\chi(x) \in \mathrm{Str}(a)$, for all $x \in \mathsf{dom}(\chi) \cap \mathrm{Vr}$ and $a \in \mathsf{shr}(\varphi, x)$.

We now define the semantics of a GSL formula φ *w.r.t.* a game structure $\mathcal{G}$ and a φ-coherent assignment χ. In particular, we write $\mathcal{G}, \chi \models \varphi$ to indicate that φ holds in $\mathcal{G}$ under χ. The semantics of LTL formulas and agent bindings are defined as in SL. The definition of graded strategy quantifiers, instead, makes use of a generic equivalence relation $\equiv^{\varphi}_{\mathcal{G}}$ on assignments that depends on the structure and the formula under exam. This equivalence is used to reasonably count the number of strategies that satisfy a formula *w.r.t.* an *a priori* fixed criterion. Observe that we use a relation on assignments instead of a more direct one on strategies, since the classification may also depend on the context determined by the strategies previously quantified. In Section 3.2, we will come back on the properties the equivalence has to satisfy in order to be used in the semantics of GSL.

Definition 3.1.4 (GSL **Semantics)** *Let $\mathcal{G}$ be a* CGS *and φ a* GSL *formula. For all φ-coherent assignments $\chi \in \mathrm{Asg}$, the relation $\mathcal{G}, \chi \models \varphi$ is inductively defined as follows.*

1. *All* LTL *operators are interpreted as usual.*
2. *For each $x \in \mathrm{Vr}$, $g \in \mathbb{N}$, and $\varphi \in$ GSL, it holds that:*
   - *(a)* $\mathcal{G}, \chi \models \langle\!\langle x \geq g \rangle\!\rangle \varphi$ *iff*
     $|(\{\chi[x \mapsto \sigma] : \sigma \in \varphi[\mathcal{G}, \chi](x)\}/\equiv^{\varphi}_{\mathcal{G}})| \geq g$;
   - *(b)* $\mathcal{G}, \chi \models [\![x < g]\!] \varphi$ *iff*
     $|(\{\chi[x \mapsto \sigma] : \sigma \in \neg\varphi[\mathcal{G}, \chi](x)\}/\equiv^{\neg\varphi}_{\mathcal{G}})| < g$;

   where $\eta[\mathcal{G}, \chi](x) \triangleq \{\sigma \in \mathrm{Str}(\mathsf{shr}(\eta, x)) : \mathcal{G}, \chi[x \mapsto \sigma] \models \eta\}$ is the set of $\mathsf{shr}(\eta, x)$-coherent strategies that, being assigned to x in χ, satisfy η.
3. *For each $a \in \mathrm{Ag}$, $x \in \mathrm{Vr}$, and $\varphi \in$ GSL, it holds that $\mathcal{G}, \chi \models (a, x)\varphi$ iff $\mathcal{G}, \chi[a \mapsto \chi(x)] \models \varphi$.*

Intuitively, by using the existential quantifier $\langle\!\langle x \geq g \rangle\!\rangle \varphi$, we can count how many equivalence classes *w.r.t.* $\equiv^{\varphi}_{\mathcal{G}}$ there are over the set of assignments $\{\chi[x \mapsto \sigma] : \sigma \in \varphi[\mathcal{G}, \chi](x)\}$ that, extending χ, satisfy φ. The universal quantifier $[\![x < g]\!] \varphi$ is simply the dual of $\langle\!\langle x \geq g \rangle\!\rangle \varphi$ and it allows to count how many classes *w.r.t.* $\equiv^{\neg\varphi}_{\mathcal{G}}$ there are over the set of assignments $\{\chi[x \mapsto \sigma] : \sigma \in \neg\varphi[\mathcal{G}, \chi](x)\}$ that, extending χ, do not satisfy φ. It is worth noting that, all GSL formulas with degree 1 are SL formulas. Also, the verification of a sentence φ does not depend on assignments, so, we just write $\mathcal{G} \models \varphi$.

Consider again the sentence φ of the scheduler example. Once a reasonable equivalence relation on assignments is fixed (see Section 3.2), one can see that $\mathcal{G}_S \models \varphi$ with $\mathtt{k} \geq 0$ and $(g_1, g_2) = (1, 2)$ but $\mathcal{G}_S \not\models \varphi$ with $(k, g_1, g_2) = (1, 1, 1)$. Indeed, if the processes use the

same strategy, they may force the play to be in $(\mathtt{I}^+ \cdot \mathtt{W})^* \cdot \mathtt{I}^\omega + (\mathtt{I}^+ \cdot \mathtt{W})^\omega$, so they either avoid to do a request or relinquish a request that is not immediately served. Consequently, to satisfy φ, we need to verify the property against all but one strategy of $\mathtt{P}_2$, *i.e.*, the one used by $\mathtt{P}_1$. Under these assumptions, we can see that the arbiter $\mathtt{A}$ has an infinite number of different strategies by suitably choosing the actions on all histories ending in the state $\mathtt{W}$.

Before continuing, we show how GATL can be embedded into GSL[1G]. In [FNP09a], the authors introduce two different semantics for their logic, called *off-line* and *on-line*. Under the first one, over a game structure with agents α and $\overline{\alpha}$, the GATL formula $\langle\!\langle\alpha\rangle\!\rangle^g\psi$ is equivalent to the GSL[1G] sentence $\langle\!\langle \mathtt{x} \geq g\rangle\!\rangle[\![\overline{\mathtt{x}} < 1]\!](\alpha, \mathtt{x})(\overline{\alpha}, \overline{\mathtt{x}})\psi$. Under the second semantics, instead, it is equivalence to the sentence $[\![\overline{\mathtt{x}} < 1]\!]\langle\!\langle \mathtt{x} \geq g\rangle\!\rangle(\alpha, \mathtt{x})(\overline{\alpha}, \overline{\mathtt{x}})\psi$. It is evident than that the counting over strategies in GATL is limited to the existential agent only. Moreover, we want to note that, the criteria at the base of the strategy classification is strictly coupled with the three temporal operators $\mathtt{X}\varphi$, $\varphi_1\mathtt{U}\varphi_2$, and $\mathtt{G}\varphi$ admitted in the syntax and cannot be easily extended to the whole LTL.

3.2 Strategy Equivalence

Our definition of GSL semantics makes use of an arbitrary equivalence relation on assignments. This choice introduces flexibility in its description, since one can come up with different logics by opportunely choosing different equivalences. In this section, we focus on a particular relation whose key feature is to classify as equivalent all assignments that reflect the same "*strategic reasoning*", although they may have completely different structures. Just to get an intuition about what we mean, consider two assignments χ_1 and χ_2 and the corresponding involved strategies associated with the agents $\mathtt{a}_1$ and $\mathtt{a}_2$. Assume now that, for each $i \in \{1, 2\}$, the homologous strategies $\chi_1(\mathtt{a}_i)$ and $\chi_2(\mathtt{a}_i)$ only differ on histories never met by a play because of a specific combination of their actions. Clearly, χ_1 and χ_2 induce the same agent behaviors, which means to reflect the same strategic reasoning. Therefore, it is natural to set them as equivalent, as we do.

In the sequel, in order to illustrate the introduced concepts, we analyze subformulas of the above described sentence $\langle\!\langle \mathtt{x} \geq k\rangle\!\rangle[\![\mathtt{y}_1 < 1]\!][\![\mathtt{y}_2 < 2]\!](\mathtt{A},\mathtt{x})(\mathtt{P}_1,\mathtt{y}_1)(\mathtt{P}_2,\mathtt{y}_2)\mathtt{F}(\mathtt{r}_1 \vee \mathtt{r}_2)$, together with their negations, over the game structure $\mathcal{G}_S$ of Figure 3.1.

3.2.1 Elementary Requirements

Logics usually admit syntactic redundancy. For example, in LTL we have $\neg\mathtt{X}(\mathtt{p} \wedge \mathtt{q}) \equiv \mathtt{X}\neg(\mathtt{p} \wedge \mathtt{q}) \equiv \mathtt{X}(\neg\mathtt{p} \vee \neg\mathtt{q})$. Also, the semantics is normally closed under substitution. Yet for LTL, this means that $\neg\mathtt{X}(\mathtt{p} \wedge \mathtt{q})$ can be replaced with $\mathtt{X}\neg(\mathtt{p} \wedge \mathtt{q})$ or $\mathtt{X}(\neg\mathtt{p} \vee \neg\mathtt{q})$, without changing the meaning of a formula. GSL should not be an exception. To ensure this, we

require the invariance of the equivalence relation on assignments *w.r.t.* the syntax of the involved formulas.

Definition 3.2.1 (Syntax Independence) *An equivalence relation on assignments* $\equiv^{\cdot}_{\mathcal{G}}$ *is* syntax independent *if, for any pair of equivalent formulas* φ_1 *and* φ_2 *and* $(\mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2))$*-coherent assignments* $\chi_1, \chi_2 \in \mathrm{Asg}$*, we have that* $\chi_1 \equiv^{\varphi_1}_{\mathcal{G}} \chi_2$ *iff* $\chi_1 \equiv^{\varphi_2}_{\mathcal{G}} \chi_2$*.*

As declared above, our aim is to classify as equivalent *w.r.t.* a formula φ all assignments that induce the same strategic reasoning. Therefore, we cannot distinguish them *w.r.t.* the satisfiability of φ itself.

Definition 3.2.2 (Semantic Consistency) *An equivalence relation on assignments* $\equiv^{\cdot}_{\mathcal{G}}$ *is* semantically consistent *if, for any formula* φ *and* φ*-coherent assignments* $\chi_1, \chi_2 \in \mathrm{Asg}$*, we have that if* $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$ *then either* $\mathcal{G}, \chi_1 \models \varphi$ *and* $\mathcal{G}, \chi_2 \models \varphi$ *or* $\mathcal{G}, \chi_1 \not\models \varphi$ *and* $\mathcal{G}, \chi_2 \not\models \varphi$*.*

3.2.2 Play Requirement

We now deal with the equivalence relation for the basic case of temporal properties. Before disclosing the formalization, we would like to give an intuition on how to evaluate the equivalence of two complete assignments χ_1 and χ_2 *w.r.t.* their agreement on the verification of a generic LTL property ψ. Let π_1 and π_2 with $\pi_1 \neq \pi_2$ be the plays satisfying ψ induced by χ_1 and χ_2, respectively. Also, consider their maximal common prefix $\rho = \mathsf{prf}(\pi_1, \pi_2) \in \mathrm{Hst}$. If the latter history can be extended to a play in such a way that ψ does not hold, we are sure that the reasons why both the assignments satisfy the property are different, as they reside in the parts the two plays diverge. Consequently, we can assume χ_1 and χ_2 to be non-equivalent *w.r.t.* ψ. Conversely, if all infinite extensions of ρ necessarily satisfy ψ, we may affirm that this is already a witness of the verification of the property by the two plays and, so, by the two assignments. Hence, we can assume χ_1 and χ_2 to be equivalent *w.r.t.* ψ.

In the following, we make often use of the concept of witness of an LTL formula ψ as the set $\mathrm{W}_\psi \triangleq \{\rho \in \mathrm{Hst} : \forall \pi \in \mathrm{Pth} \,.\, \rho < \pi \Rightarrow \pi \models \psi\}$ containing all those histories that cannot be extended to a play that violates the property.

Definition 3.2.3 (Play Consistency) *An equivalence relation on assignments* $\equiv^{\cdot}_{\mathcal{G}}$ *is* play consistent *if, for any* LTL *formula* ψ *and* ψ*-coherent assignments* $\chi_1, \chi_2 \in \mathrm{Asg}$*, we have that* $\chi_1 \equiv^{\psi}_{\mathcal{G}} \chi_2$ *iff either* $\pi_1 = \pi_2$ *or* $\mathsf{prf}(\pi_1, \pi_2) \in \mathrm{W}_\psi$*, where* $\pi_1 = \mathsf{play}(\chi_{1\restriction \mathrm{Ag}})$ *and* $\pi_2 = \mathsf{play}(\chi_{2\restriction \mathrm{Ag}})$ *are the plays induced by* χ_1 *and* χ_2*, respectively, and* $\mathrm{W}_\psi \subseteq \mathrm{Hst}$ *represents the witness set of* ψ*.*

To see how to apply the above definition, consider the formula $\psi = \mathtt{F}(\mathtt{r}_1 \vee \mathtt{r}_2)$ and let W_ψ be the corresponding witness set, whose minimal histories can be represented by

the regular expression $\mathtt{I}^+ \cdot (1+2) + (\mathtt{I}^+ \cdot \mathtt{W})^+ \cdot (1+2+1/2+2/1)$. Moreover, let $\chi_1, \chi_2, \chi_3 \in \mathrm{Asg}(\{\mathtt{A}, \mathtt{P}_1, \mathtt{P}_2\})$ be three complete assignments on which we want to check the play consistency. We assume that each χ_i associates a strategy $\chi_i(\mathtt{a}) = \sigma_i^{\mathtt{a}}$ with the agent $\mathtt{a} \in \{\mathtt{A}, \mathtt{P}_1, \mathtt{P}_2\}$ as defined in the following, where $\rho, \rho_{\mathtt{s}} \in \mathrm{Hst}$ with $\mathsf{lst}(\rho) \neq \mathtt{I}$ and $\rho_{\mathtt{s}} \cdot \mathtt{s} \in \mathrm{Hst}$: for the arbiter $\mathtt{A}$, we set $\sigma^{\mathtt{A}}_{1/2}(\rho_{\mathtt{W}} \cdot \mathtt{W}) \triangleq 2$, $\sigma^{\mathtt{A}}_{1/2/3}(\rho_{1/2} \cdot 1/2) = \sigma^{\mathtt{A}}_{2}(\rho_{2/1} \cdot 2/1) \triangleq \mathtt{i}$, and $\sigma^{\mathtt{A}}_{3}(\rho_{\mathtt{W}} \cdot \mathtt{W}) = \sigma^{\mathtt{A}}_{1/3}(\rho_{2/1} \cdot 2/1) \triangleq 1$; for the processes, instead, we set $\sigma^{\mathtt{P}_1}_{1/2/3}(\rho) = \sigma^{\mathtt{P}_2}_{1/2/3}(\rho) \triangleq \mathtt{i}$, $\sigma^{\mathtt{P}_1}_{1/2}(\rho_{\mathtt{I}} \cdot \mathtt{I}) = \sigma^{\mathtt{P}_2}_{1/2/3}(\rho_{\mathtt{I}} \cdot \mathtt{I}) \triangleq \mathtt{r}$, and $\sigma^{\mathtt{P}_1}_{3}(\rho_{\mathtt{I}} \cdot \mathtt{I}) \triangleq \mathtt{i}$. Now, one can see that $\chi_1 \equiv^{\psi}_{\mathcal{G}} \chi_2$, but $\chi_1 \not\equiv^{\psi}_{\mathcal{G}} \chi_3$. Indeed, χ_1, χ_2, and χ_3 induce the plays $\pi_1 = \mathtt{I} \cdot \mathtt{W} \cdot 2/1 \cdot 1/2^{\omega}$, $\pi_2 = \mathtt{I} \cdot \mathtt{W} \cdot 2/1^{\omega}$, and $\pi_3 = \mathtt{I} \cdot 1^{\omega}$, respectively, where $\rho_{12} = \mathsf{prf}(\pi_1, \pi_2) = \mathtt{I} \cdot \mathtt{W} \cdot 2/1$ and $\rho_{13} = \mathsf{prf}(\pi_1, \pi_3) = \mathtt{I}$ are the corresponding common prefixes. Thus, ρ_{12} belongs to the witness W_{ψ}, while ρ_{13} does not. As another example, consider the formula $\overline{\psi} = \mathtt{G}(\neg \mathtt{r}_1 \wedge \neg \mathtt{r}_2)$, which is equivalent to the negation of the previous one, and observe that its witness set $\mathrm{W}_{\overline{\psi}}$ is empty. Moreover, let $\chi_1, \chi_2, \chi_3 \in \mathrm{Asg}(\{\mathtt{A}, \mathtt{P}_1, \mathtt{P}_2\})$ be the three complete assignments we want to analyze. The strategies for the the arbiter $\mathtt{A}$ are defined as above, while those of the processes follows: $\overline{\sigma}^{\mathtt{P}_i}_{1/2/3}(\rho) \triangleq \mathtt{i}$, $\overline{\sigma}^{\mathtt{P}_i}_{1/2}(\rho_{\mathtt{I}} \cdot \mathtt{I}) \triangleq \mathtt{r}$, $\overline{\sigma}^{\mathtt{P}_i}_{1/2}(\rho_{\mathtt{W}} \cdot \mathtt{W}) \triangleq \mathtt{a}$, and $\overline{\sigma}^{\mathtt{P}_i}_{3}(\rho_{\mathtt{I}} \cdot \mathtt{I}) = \overline{\sigma}^{\mathtt{P}_i}_{3}(\rho_{\mathtt{W}} \cdot \mathtt{W}) \triangleq \mathtt{i}$, for all $i \in \{1, 2\}$ and $\rho, \rho_{\mathtt{s}} \in \mathrm{Hst}$ with $\mathsf{lst}(\rho) \notin \{\mathtt{I}, \mathtt{W}\}$ and $\rho_{\mathtt{s}} \cdot \mathtt{s} \in \mathrm{Hst}$. Now, one can see that $\chi_1 \equiv^{\overline{\psi}}_{\mathcal{G}} \chi_2$, but $\chi_1 \not\equiv^{\overline{\psi}}_{\mathcal{G}} \chi_3$. Indeed, χ_1 and χ_2 induce the same play $(\mathtt{I} \cdot \mathtt{W})^{\omega}$, while χ_3 runs along $\mathtt{I}^{\omega}$. Thus, χ_1 and χ_2 are equivalent, but χ_1 and χ_3 are not.

3.2.3 Strategy Requirements

The semantics of a binding construct $\varphi = (a, x)\eta$ just involves a redefinition of the underlying assignment χ, since it asserts that φ holds under χ once the inner part η can be satisfied by associating the agent a to the strategy $\chi(x)$. Thus, the equivalence of two assignments χ_1 and χ_2 *w.r.t.* φ necessarily depends on that of their extensions on a *w.r.t.* η.

Definition 3.2.4 (Binding Consistency) *An equivalence relation on assignments $\equiv^{\cdot}_{\mathcal{G}}$ is* binding consistent *if, for any formula $\varphi = (\mathtt{a}, \mathtt{x})\eta$ and φ-coherent assignments $\chi_1, \chi_2 \in \mathrm{Asg}$, we have that $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$ iff $\chi_1[\mathtt{a} \mapsto \chi_1(\mathtt{x})] \equiv^{\eta}_{\mathcal{G}} \chi_2[\mathtt{a} \mapsto \chi_2(\mathtt{x})]$.*

To get familiar with the above concept, consider the formula $\flat\psi$, where $\flat \triangleq (\mathtt{A}, \mathtt{x})(\mathtt{P}_1, \mathtt{y}_1)(\mathtt{P}_2, \mathtt{y}_2)$, and let $\chi_1, \chi_2, \chi_3 \in \mathrm{Asg}(\{\mathtt{x}, \mathtt{y}_1, \mathtt{y}_2\})$ be the assignments assuming as values the strategies $\chi_i(\mathtt{x}) \triangleq \sigma_i^{\mathtt{A}}$ and $\chi_i(\mathtt{y}_j) \triangleq \sigma_i^{\mathtt{P}_j}$ previously defined, where $i \in \{1, 2, 3\}$ and $j \in \{1, 2\}$. Then, by definition, it is immediate to see that $\chi_1 \equiv^{\flat\psi}_{\mathcal{G}} \chi_2$, but $\chi_1 \not\equiv^{\flat\psi}_{\mathcal{G}} \chi_3$.

Before continuing with the analysis of the equivalence, it is important to make an observation about the dual nature of the existential and universal quantifiers *w.r.t.* the counting of strategies. We do this by exploiting the classic game-semantics metaphor originally proposed for first-order logic by Lorenzen and Hintikka, where the choice of an existential variable is done by a player called $\exists$ and that of the universal ones by its opponent $\forall$. Consider a sentence

$\langle\langle x_1 \geq g_1 \rangle\rangle [\![x_2 < g_2]\!] \eta$, having $\langle\langle \mathtt{y}_1 \geq h_1 \rangle\rangle \eta_1$ and $[\![\mathtt{y}_2 < h_2]\!] \eta_2$ as two subformulas in η. When player $\exists$ tries to choose h_1 different strategies $\mathtt{y}_1$ to satisfy η_1, it has also to maximize the number of strategies x_1 verifying $[\![x_2 < g_2]\!] \eta$ to be sure that the constraint $\geq g_1$ of the first quantifier is not violated. At the same time, player $\forall$ tries to do the opposite while choosing h_2 different strategies y_2 not satisfying η_2, *i.e.*, it needs to maximize the number of strategies x_2 falsifying η in order to violate the constraint $< g_2$ of the second quantifier.

With this observation in mind, we can now treat the equivalence for the existential quantifier. Two assignments χ_1 and χ_2 are equivalent *w.r.t.* a formula $\varphi = \langle\langle x \geq g \rangle\rangle \eta$ if player $\exists$ is not able to find a strategy σ among those satisfying η, to associate with the variable x, that allows the corresponding extensions of χ_1 and χ_2 on x to induce different behaviors *w.r.t.* η itself. In other words, $\exists$ cannot distinguish between the two assignments, as they behave the same independently from the way they are extended.

Definition 3.2.5 (Existential Consistency) *An equivalence relation on assignments* $\equiv^{\cdot}_{\mathcal{G}}$ *is* existentially consistent *if, for any formula* $\varphi = \langle\langle \mathtt{x} \geq g \rangle\rangle \eta$ *and* φ*-coherent assignments* $\chi_1, \chi_2 \in \mathrm{Asg}$*, we have that* $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$ *iff, for each strategy* $\sigma \in \eta[\mathcal{G}, \chi_1](\mathtt{x}) \cup \eta[\mathcal{G}, \chi_2](\mathtt{x})$*, it holds that* $\chi_1[\mathtt{x} \mapsto \sigma] \equiv^{\eta}_{\mathcal{G}} \chi_2[\mathtt{x} \mapsto \sigma]$.

To clarify the above definition, consider the formula $\varphi = \langle\langle \mathtt{y}_2 \geq 2 \rangle\rangle \flat \overline{\psi}$ and let $\chi_1, \chi_2, \chi_3 \in \mathrm{Asg}(\{\mathtt{x}, \mathtt{y}_1\})$ be the three assignments assuming as values the strategies $\chi_i(\mathtt{x}) \triangleq \overline{\sigma}^{\mathtt{A}}_i$ and $\chi_i(\mathtt{y}_1) \triangleq \overline{\sigma}^{\mathtt{P}_1}_i$ previously defined, where $i \in \{1, 2, 3\}$. It is possible to see that $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$, but $\chi_1 \not\equiv^{\varphi}_{\mathcal{G}} \chi_3$. By definition, $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$ iff, for each strategy $\overline{\sigma} \in (\flat \overline{\psi})[\mathcal{G}, \chi_1](\mathtt{y}_2) \cup (\flat \overline{\psi})[\mathcal{G}, \chi_2](\mathtt{y}_2)$, it holds that $\chi_1[\mathtt{y}_2 \mapsto \overline{\sigma}] \equiv^{\flat\overline{\psi}}_{\mathcal{G}} \chi_2[\mathtt{y}_2 \mapsto \overline{\sigma}]$. Now, observe that the strategy $\overline{\sigma}^{\mathtt{P}_2}_1$ introduced above is the unique one that allows χ_1 and χ_2 to satisfy $\flat\overline{\psi}$ once extended on $\mathtt{y}_2$. At this point, we can easily show that $\chi_1[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1] \equiv^{\flat\overline{\psi}}_{\mathcal{G}} \chi_2[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1]$, as the the derived complete assignments $\chi_1[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1] \circ \flat$ and $\chi_2[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1] \circ \flat$ induce the same play $(\mathtt{I} \cdot \mathtt{W})^{\omega}$. The non-equivalence of χ_1 and χ_3 easily follows from the fact that $\overline{\sigma}^{\mathtt{P}_2}_1 \notin (\flat\overline{\psi})[\mathcal{G}, \chi_3](\mathtt{y}_2)$, as $\chi_3[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1] \circ \flat$ induces the play $\mathtt{I} \cdot 2^{\omega}$ that does not satisfy $\overline{\psi}$. Thus, $\chi_1[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1] \not\equiv^{\flat\overline{\psi}}_{\mathcal{G}} \chi_3[\mathtt{y}_2 \mapsto \overline{\sigma}^{\mathtt{P}_2}_1]$.

We conclude with the equivalence for the universal quantifier. Two assignments χ_1 and χ_2 are equivalent *w.r.t.* a formula $\varphi = [\![x < g]\!] \eta$ if, for each index $i \in \{1, 2\}$ and strategy σ_i player $\forall$ chooses among those satisfying η under χ_i, there is a strategy σ_{3-i} this player can choose among those satisfying η under χ_{3-i} such that, once the two strategies are associated with the variable x, they make the corresponding extensions of assignments equivalent *w.r.t.* η. This means that the parts of the game structure that are reachable under χ_1 and χ_2 contain exactly the same information *w.r.t.* the verification of the inner formula. In other words, $\forall$ cannot distinguish between the two assignments, as the induced subtrees of possible plays are practically the same.

Definition 3.2.6 (Universal Consistency) *An equivalence relation on assignments* $\equiv^{\cdot}_{\mathcal{G}}$ *is* universally consistent *if, for any formula* $\varphi = [\![\mathtt{x} < g]\!] \eta$ *and* φ*-coherent assignments*

$\chi_1, \chi_2 \in \mathrm{Asg}$*, we have that* $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$ *iff, for each* $i \in \{1,2\}$ *and strategy* $\sigma_i \in \eta[\mathcal{G}, \chi_i](\mathtt{x})$*, there is a strategy* $\sigma_{3-i} \in \eta[\mathcal{G}, \chi_{3-i}](\mathtt{x})$ *such that* $\chi_1[\mathtt{x} \mapsto \sigma_1] \equiv^{\eta}_{\mathcal{G}} \chi_2[\mathtt{x} \mapsto \sigma_2]$.

Finally, to better understand the above definition, consider the formula $\varphi = [\![\mathtt{y}_1 < 1]\!]\eta$, where $\eta = [\![\mathtt{y}_2 < 2]\!]\flat\psi$, and let $\chi_1, \chi_2, \chi_3 \in \mathrm{Asg}(\{\mathtt{x}\})$ be the three assignments assuming as values the strategies $\chi_i(\mathtt{x}) \triangleq \sigma_i^{\mathtt{A}}$ previously defined, where $i \in \{1,2,3\}$. We can now see that $\chi_1 \equiv^{\varphi}_{\mathcal{G}} \chi_2$, but $\chi_1 \not\equiv^{\varphi}_{\mathcal{G}} \chi_3$. First, observe that $\eta[\mathcal{G}, \chi_1](\mathtt{y}_1) = \eta[\mathcal{G}, \chi_2](\mathtt{y}_1) = \mathrm{Str}$. Indeed, for all strategies $\sigma \in \mathrm{Str}$, we have that $\mathcal{G}, \chi_1[\mathtt{y}_1 \mapsto \sigma] \models \eta$ and $\mathcal{G}, \chi_2[\mathtt{y}_1 \mapsto \sigma] \models \eta$, since $\mathcal{G}, \chi_1[\mathtt{y}_1 \mapsto \sigma, \mathtt{y}_2 \mapsto \sigma'] \models \flat\psi$ and $\mathcal{G}, \chi_2[\mathtt{y}_1 \mapsto \sigma, \mathtt{y}_2 \mapsto \sigma'] \models \flat\psi$, for all $\sigma' \in \mathrm{Str}$ such that $\sigma \neq \sigma'$. This is due to the fact that the plays π_1 and π_2 induced by the two complete assignments $\chi_1[\mathtt{y}_1 \mapsto \sigma, \mathtt{y}_2 \mapsto \sigma'] \circ \flat$ and $\chi_2[\mathtt{y}_1 \mapsto \sigma, \mathtt{y}_2 \mapsto \sigma'] \circ \flat$ differ from $(\mathtt{I}^+ \cdot \mathtt{W})^* \cdot \mathtt{I}^\omega$ and $(\mathtt{I}^+ \cdot \mathtt{W})^\omega$, as the strategies of the two processes are different. Also, they share a common prefix $\rho = \mathsf{prf}(\pi_1, \pi_2)$ belonging to W_ψ, since the strategies of the arbiter only differ on the histories ending in the state $2/1$. We can now show that χ_1 and χ_2 are equivalent, by applying the above definition in which we assume that $\sigma_i = \sigma_{3-i}$. To prove that χ_1 and χ_3 are non-equivalent, we show that there is a strategy $\sigma \in \eta[\mathcal{G}, \chi_1](\mathtt{y}_1)$ for χ_1 such that, for all strategies $\sigma' \in \eta[\mathcal{G}, \chi_3](\mathtt{y}_1)$ for χ_3, it holds that $\chi_1[\mathtt{y}_1 \mapsto \sigma] \not\equiv^{\eta}_{\mathcal{G}} \chi_3[\mathtt{y}_1 \mapsto \sigma']$. As before, observe that $\eta[\mathcal{G}, \chi_1](\mathtt{y}_1) = \eta[\mathcal{G}, \chi_3](\mathtt{y}_1) = \mathrm{Str}$ and choose $\sigma \in \mathrm{Str}$ as the strategy $\sigma_1^{\mathtt{P}_1}$ previously defined. At this point, one can easily see that all plays compatible with $\chi_1[\mathtt{y}_1 \mapsto \sigma] \circ \flat$ pass through either $\mathtt{I} \cdot 1$ or $\mathtt{I} \cdot \mathtt{W} \cdot 2/1$, while a play compatible with $\chi_3 \circ \flat$ cannot pass through the latter history. Thus, the non-equivalence of the two assignments immediately follows.

3.3 Main Results

In this section, we address two fundamental questions about $\mathrm{GSL}[1\mathrm{G}]$ over turn-based game structures, namely, determinacy and model checking. For the sake of clarity of exposition, we restrict our attention to the case of 2 agents only. Also, for complexity reasons, we provide a procedure for a vanilla fragment of the logic in which all temporal properties are used as in CTL and ATL. The whole logic will be then object of the extended version of this work. Observe that, by applying a conversion from concurrent to turn-based structures similar to the one described in [MMS14], we can lift our model-checking procedure to the more complex context of concurrent games.

3.3.1 Determinacy

Recall that the determinacy has been first proved for classic Borelian turn-based two-player games in [Mar75]. However, the proof used there does not directly apply to our graded setting. To give an evidence of the differences between the two frameworks, observe that

in $\mathrm{SL}[1\mathrm{G}, 2\mathrm{AG}]$ sentences of the kind $\langle\!\langle x \rangle\!\rangle [\![\overline{x}]\!] \eta$ imply $[\![\overline{x}]\!] \langle\!\langle x \rangle\!\rangle \eta$, while in $\mathrm{GSL}[1\mathrm{G}, 2\mathrm{AG}]$ the corresponding implication $\langle\!\langle x \geq i \rangle\!\rangle [\![\overline{x} < j]\!] \eta \Rightarrow [\![\overline{x} < j]\!] \langle\!\langle x \geq i \rangle\!\rangle \eta$ does not hold. The determinacy property we are interested in is exactly the converse direction, *i.e.*, $[\![\overline{x} < j]\!] \langle\!\langle x \geq i \rangle\!\rangle \eta \Rightarrow \langle\!\langle x \geq i \rangle\!\rangle [\![\overline{x} < j]\!] \eta$. In particular, we extend the Gale-Stewart Theorem [PP04], by exploiting a deep generalization of the technique used in [FNP09a]. The idea consists of a fixed-point calculation over the number of winning strategies an agent can select against all but a fixed number of those of its opponent. Regarding this approach, we observe that the simpler counting considered in [FNP09a] is restricted to existential quantifications.

Construction 3.3.1 (Grading Function) *Consider a two-agent turn-based game structure $\mathcal{G}$ with $\mathrm{Ag} = \{\alpha, \overline{\alpha}\}$. Moreover, let ψ be an* LTL *formula, where $\mathrm{W}_{\psi}, \mathrm{W}_{\neg\psi} \subseteq \mathrm{Hst}$ denotes the witness sets for ψ and $\neg\psi$, respectively. It is immediate to see that, in case $s_I \in \mathrm{W}_{\psi}$ (resp., $s_I \in \mathrm{W}_{\neg\psi}$), all strategy profiles are equivalent* w.r.t. *the temporal property ψ (resp., $\neg\psi$). If $s_I \in \mathrm{X} \triangleq \mathrm{Hst} \setminus (\mathrm{W}_{\psi} \cup \mathrm{W}_{\neg\psi})$, instead, we need to introduce a* grading function $\mathsf{G}^{\alpha}_{\psi} : \mathrm{X} \to \Gamma$, *where $\Gamma \triangleq \mathbb{N} \to (\mathbb{N} \cup \{\omega\})$, that allows to determine how many different strategies the agent α (resp., $\overline{\alpha}$) owns* w.r.t. *ψ (resp., $\neg\psi$). Informally, $\mathsf{G}^{\alpha}_{\psi}(\rho)(j)$ represents the number of winning strategies player α can put up against all but at most j strategies of its adversary $\overline{\alpha}$, once the current play has already reached the history $\rho \in \mathrm{X}$. Before continuing, observe that α has sometimes the possibility to commit a suicide,* i.e.*, to choose a strategy leading directly to a history in $\mathrm{W}_{\neg\psi}$, with the hope to win the game by collapsing all strategies of its opponent into a unique class. The set of histories enabling this possibility is defined as follows: $\mathrm{S} \triangleq \{\rho \in \mathrm{X} : \exists \rho' \in \mathrm{W}_{\neg\psi} \,.\, \rho < \rho' \wedge \forall \rho'' \in \mathrm{Hst} \,.\, \rho \leq \rho'' < \rho' \Rightarrow \rho'' \in \mathrm{Hst}_{\alpha}\}$, where $\mathrm{Hst}_{\alpha} = \{\forall \rho \in \mathrm{Hst} : \mathsf{ag}(\mathsf{lst}(\rho)) = \{\alpha\}\}$ is the set of histories ending in a state controlled by α. Intuitively, this agent can autonomously extend a history $\rho \in \mathrm{S}$ into one $\rho' \in \mathrm{W}_{\neg\psi}$ that is surely loosing, independently from the behavior of $\overline{\alpha}$. Note that there may be several suicide strategies, but all of them are equivalent* w.r.t. *the property ψ. Also, against them, all counter strategies of $\overline{\alpha}$ are equivalent as well. At this point, to define the function $\mathsf{G}^{\alpha}_{\psi}$, we introduce the auxiliary functor $\mathsf{F}^{\alpha}_{\psi} : (\mathrm{X} \to \Gamma) \to (\mathrm{X} \to \Gamma)$, whose least fixpoint represents a function returning the maximum number of different strategies α can use against all but a precise fixed number of counter strategies of $\overline{\alpha}$. Formally, we have that:*

$$\mathsf{F}^{\alpha}_{\psi}(\mathsf{f})(\rho)(j) \triangleq \begin{cases} \sum_{\rho' \in \mathsf{suc}(\rho) \cap \mathrm{X}} \mathsf{f}(\rho')(j) + |\mathsf{suc}(\rho) \cap \mathrm{W}_{\psi}|, & \text{if } \rho \in \mathrm{Hst}_{\alpha} \text{ and } j = 0; \\ \sum_{\rho' \in \mathsf{suc}(\rho) \cap \mathrm{X}} \mathsf{f}(\rho')(j), & \text{if } \rho \in \mathrm{Hst}_{\alpha} \text{ and } j > 0; \\ \sum_{\mathsf{c} \in \mathrm{C}(\rho)(j)} \prod_{\rho' \in \mathsf{dom}(\mathsf{c})} \mathsf{f}(\rho')(\mathsf{c}(\rho')), & \text{otherwise}; \end{cases}$$

where $\mathsf{suc}(\rho) = \{\rho' \in \mathrm{Hst} : \exists s \in \mathrm{St}. \ \rho s = \rho'\}$ and $\mathsf{C}(\rho)(i) \subseteq (\mathsf{suc}(\rho) \cap \mathrm{Z}) \rightharpoonup \mathbb{N}$ contains all partial functions $\mathsf{c} \in \mathsf{C}(\rho)(i)$ for which α owns a suicide strategy on the histories not in their domains, i.e.*, $(\mathsf{suc}(\rho) \cap \mathrm{Z}) \setminus \mathsf{dom}(\mathsf{c}) \subseteq \mathrm{S}$, and the sum of all values assumed by c plus the number of successor histories that are neither surely winning nor contained in the domain*

of c *equals to* i*,* i.e.*,* $i = \sum_{\rho' \in \mathsf{dom(c)}} \mathsf{c}(\rho') + |\mathsf{suc}(\rho) \setminus (\mathrm{X} \cup \mathsf{dom(c)})|$*. Intuitively, the first item of the definition simply asserts that the number of strategies* $\mathsf{F(f)}(\rho)(0)$ *that agent* α *has on the* α*-history* ρ*, without excluding any counter strategy of its adversary, is obtainable as the sum of the* $\mathsf{f}(\rho')(0)$ *strategies on the successor histories* $\rho' \in \mathrm{X}$ *plus a single strategy for each successor history that is surely winning. Similarly, the second item takes into account the case in which we can avoid exactly* j *counter strategies. The last item, instead, computes the number of strategies for* α *on the* $\overline{\alpha}$*-histories. In particular, through the set* $\mathsf{C}(\rho)(j)$*, it first determines in how many ways it is possible to split the number* j *of counter strategies to avoid among all successor histories of* ρ*. Then, for each of these splittings, it calculates the product of the corresponding numbers* $\mathsf{f}(\rho')(\mathsf{c}(\rho'))$ *of strategies for* α*. We are finally able to define the grading function* $\mathsf{G}^{\alpha}_{\psi}$ *by means of the least fixpoint* $\mathsf{f}^{\star} = \mathsf{F}^{\alpha}_{\psi}(\mathsf{f}^{\star})$ *of the functor* $\mathsf{F}^{\alpha}_{\psi}$ *as follows:* $\mathsf{G}^{\alpha}_{\psi}(\rho)(j) \triangleq \sum_{h=0}^{j} \mathsf{f}^{\star}(\rho)(h) + [\rho \in \mathrm{S} \wedge j \geq 1]$*. Intuitively,* $\mathsf{G}^{\alpha}_{\psi}(\rho)(j)$ *is the sum of the numbers* $\mathsf{f}^{\star}(\rho)(h)$ *of winning strategies the agent* α *can exploit against all but exactly* h *strategies of its adversary* $\overline{\alpha}$*, for each* $h \in [0, j]$*. Moreover, if* $\rho \in \mathrm{S}$*, we need to add to this counting the suicide strategy that* α *can use once* $\overline{\alpha}$ *avoids to apply his unique counter strategy. This is formalized through the standard notation* $[\eth]$ *[GKP94] that is evaluated to* 1*, if the condition* $\eth$ *is true, and to* 0*, otherwise.*

Thanks to the above construction, one can compute the maximum number of strategies that a player has at its disposal against all but a fixed number of strategies of the opponent. Next lemma, whose statement can be constructively proved by transfinite induction on the recursions of the functor $\mathsf{F}^{\alpha}_{\psi}$, precisely describes this fact. Indeed, we show how the satisfiability of a GSL[1G, 2AG] sentence $\langle\!\langle x \geq i \rangle\!\rangle [\![\overline{x} \leq j]\!] (\alpha, x)(\overline{\alpha}, \overline{x})\psi$ can be decided via the computation of the associated grading function $\mathsf{G}^{\alpha}_{\psi}$.

Lemma 3.3.1 (Grading Function) *Let* $\mathcal{G}$ *be a two-agent turn-based game structure, where* $\mathrm{Ag} = \{\alpha, \overline{\alpha}\}$*, and* $\varphi = \langle\!\langle x \geq i \rangle\!\rangle [\![\overline{x} \leq j]\!] (\alpha, x)(\overline{\alpha}, \overline{x})\psi$ *a* GSL[1G, 2AG] *sentence. Moreover, let* $\mathsf{G}^{\alpha}_{\psi}$ *be the grading function and* $\mathrm{W}_{\psi}, \mathrm{W}_{\neg\psi}, \mathrm{X} \subseteq \mathrm{Hst}$ *the sets of histories obtained in Construction 3.3.1. Then,* $\mathcal{G} \models \varphi$ *iff one of the following three conditions hold:* (i) $i \leq 1$*,* $j \geq 0$*, and* $s_I \in \mathrm{W}_{\psi}$*;* (ii) $i \leq 1$*,* $j \geq 1$*, and* $s_I \in \mathrm{W}_{\neg\psi}$*;* (iii) $i \leq \mathsf{G}^{\alpha}_{\psi}(s_I)(j)$ *and* $s_I \in \mathrm{X}$*.*

Again by transfinite induction on its recursive structure, we can prove a quite natural but fundamental property of the grading function, *i.e.*, its duality in the form described in the next lemma. To give an intuition, assume that agent $\overline{\alpha}$ has at most j strategies to satisfy the temporal property $\neg\psi$ against all but at most i strategies of its adversary α. Then, it can be shown that the latter has more than i strategies to satisfy ψ against all but at most j strategies of the former.

Lemma 3.3.2 (Grading Duality) *Let* $\mathsf{G}^{\alpha}_{\psi}$ *and* $\mathsf{G}^{\overline{\alpha}}_{\neg\psi}$ *be the grading functions and* $\mathrm{X} \subseteq \mathrm{Hst}$ *the set of histories obtainable by Construction 3.3.1. For all histories* $\rho \in \mathrm{X}$ *and indexes* $i, j \in \mathbb{N}$*, it holds that if* $\mathsf{G}^{\overline{\alpha}}_{\neg\psi}(\rho)(i) \leq j$ *then* $i < \mathsf{G}^{\alpha}_{\psi}(\rho)(j)$.

Summing up the above two results, we can easily prove that, on turn-based game structures, GSL[1G, 2AG] is determined. Indeed, suppose that $s_I \in \mathrm{X}$ and $\mathcal{G} \models [\![\overline{x} \leq j]\!]\langle\!\langle x \geq i \rangle\!\rangle \flat\psi$, where $\flat = (\alpha, x)(\overline{\alpha}, \overline{x})$ (the case with $s_I \in \mathrm{W}_{\psi}$ immediately follows from classic Martin's Determinacy Theorem [Mar75, Mar85]). Obviously, $\mathcal{G}$ does not satisfy the negation of this sentence, *i.e.*, $\mathcal{G} \not\models \langle\!\langle \overline{x} \geq j+1 \rangle\!\rangle [\![x \leq i-1]\!] \flat\neg\psi$. Consequently, by Lemma 3.3.1, we have that $\mathsf{G}^{\overline{\alpha}}_{\neg\psi}(s_I)(i-1) \leq j$. Hence, by Lemma 3.3.2, it follows that $i \leq \mathsf{G}^{\alpha}_{\psi}(s_I)(j)$. Finally, again by Lemma 3.3.1, we obtain that $\mathcal{G} \models \langle\!\langle x \geq i \rangle\!\rangle [\![\overline{x} \leq j]\!] \flat\psi$, as required by the definition of determinacy.

Theorem 3.3.1 (Determinacy) GSL[1G, 2AG] *on turn-based game structures is determined.*

3.3.2 Model Checking

We finally describe a solution of the model-checking problem for the above mentioned fragment of GSL[1G, 2AG], which only admits *simple temporal properties*, *i.e.*, $\varphi_1 \mathtt{U} \varphi_2$, $\varphi_1 \mathtt{R} \varphi_2$, and $\mathtt{X}\varphi$, where φ_1, φ_2, and φ are sentences. This fragment, called Vanilla GSL[1G, 2AG], is in relation with GSL[1G, 2AG], as CTL and ATL are for CTL* and ATL*, respectively.

The idea here is to exploit the characterization of the grading function stated in Lemma 3.3.1 in order to verify whether a game structure $\mathcal{G}$ satisfies a sentence $\varphi = \langle\!\langle \mathtt{x} \geq i \rangle\!\rangle [\![\overline{\mathtt{x}} \leq j]\!](\alpha, \mathtt{x})(\overline{\alpha}, \overline{\mathtt{x}})\psi$, while avoiding the naive infinite calculation of $\mathsf{F}^{\alpha}_{\psi}$ least fixpoint. Fortunately, due to the simplicity of the temporal property ψ, we have that the four sets W_{ψ}, $\mathrm{W}_{\neg\psi}$, X, and S previously introduced are memoryless, *i.e.*, if a history belongs to them, every other history ending in the same state is also a member of these sets. Therefore, we can focus only on states by defining $\mathrm{W}_{\psi} \triangleq \{s \in \mathrm{St} : \mathcal{G}, s \models \mathtt{A}\psi\}$, $\mathrm{W}_{\neg\psi} \triangleq \{s \in \mathrm{St} : \mathcal{G}, s \models \mathtt{A}\neg\psi\}$, $\mathrm{X} \triangleq \mathrm{St} \setminus (\mathrm{W}_{\psi} \cup \mathrm{W}_{\neg\psi})$, and $\mathrm{S} \triangleq \{s \in \mathrm{St} : \mathcal{G}, s \models \mathtt{E}(\alpha \mathtt{UA} \neg\psi)\}$ via very simple CTL properties. Intuitively, W_{ψ} and $\mathrm{W}_{\neg\psi}$ contain the states from which agents α and $\overline{\alpha}$ can ensure, independently from the adversary, the properties ψ and $\neg\psi$, respectively. The set X, instead, contains the states on which we have still to determine the number of strategies at disposal of the two agents. Finally, S maintains the suicide states, *i.e.*, those states from which α can commit suicide by autonomously reaching $\mathrm{W}_{\neg\psi}$. In addition, since at most j strategies of $\overline{\alpha}$ can be avoided while reasoning on the sentence φ, we need just to deal with functions in the set $\Gamma \triangleq [0, j] \to (\mathbb{N} \cup \{\omega\})$ instead of $\Gamma \triangleq \mathbb{N} \to (\mathbb{N} \cup \{\omega\})$. Consequently, the functor $\mathsf{F}^{\alpha}_{\psi} : (\mathrm{X} \to \Gamma) \to (\mathrm{X} \to \Gamma)$ can be redefined as follows:

$$\mathsf{F}(\mathsf{f})(s)(h) \triangleq \begin{cases} \sum_{s' \in \mathsf{suc}(s) \cap \mathrm{X}} \mathsf{f}(s')(h) + |\mathsf{suc}(s) \cap \mathrm{W}_{\psi}|, & \text{if } s \in \mathrm{St}_{\alpha} \text{ and } h = 0; \\ \sum_{s' \in \mathsf{suc}(s) \cap \mathrm{X}} \mathsf{f}(s')(h), & \text{if } s \in \mathrm{St}_{\alpha} \text{ and } h > 0; \\ \sum_{\mathsf{c} \in \mathrm{C}(s)(h)} \prod_{s' \in \mathsf{dom}(\mathsf{c})} \mathsf{f}(s')(\mathsf{c}(s')), & \text{otherwise;} \end{cases}$$

where $\mathsf{suc}(s) = \{s' \in \mathrm{St} : (s, s') \in Ed\}$ and $\mathsf{C}(s)(i) \subseteq (\mathsf{suc}(s) \cap \mathrm{Z}) \rightharpoonup \mathbb{N}$ contains all partial functions $\mathsf{c} \in \mathsf{C}(s)(i)$ for which α owns a suicide strategy on the states not in their domains, *i.e.*, $(\mathsf{suc}(s) \cap \mathrm{Z}) \setminus \mathsf{dom}(\mathsf{c}) \subseteq \mathrm{S}$, and the sum of all values assumed by c plus the number of successors that are neither surely winning nor contained in the domain of c equals to i, *i.e.*, $i = \sum_{s' \in \mathsf{dom}(\mathsf{c})} \mathsf{c}(s') + |\mathsf{suc}(s) \setminus (\mathrm{X} \cup \mathsf{dom}(\mathsf{c}))|$. Similarly, the grading function $\mathsf{G}^{\alpha}_{\psi} : \mathrm{X} \to \Gamma$ becomes $\mathsf{G}^{\alpha}_{\psi}(s)(h) \triangleq \sum_{l=0}^{h} \mathsf{f}^{\star}(s)(l) + [s \in \mathrm{S} \wedge h \geq 1]$, where $\mathsf{f}^{\star}$ is the least fixpoint of $\mathsf{F}^{\alpha}_{\psi}$.

Unfortunately, these redefinitions are not enough by their own to ensure that the fixpoint calculation can be done in a finite, possibly small, number of iterations of the functor. This is due to two facts: the functions in Γ have an infinite codomain and the game structure $\mathcal{G}$ have cycles inside. In order to solve such a problem, we make use of the following observation. Suppose that agent α has at least one strategy on one of its states $s \in \mathrm{St}_{\alpha}$ that is also part of a cycle in which no state of its opponent $\overline{\alpha}$ is adjacent to the set $\mathrm{W}_{\neg\psi}$. Then, α can use this cycle from s to construct an infinite number of nonequivalent strategies, by simply pumping-up the number of time he decides to traverse it before following the previously found strategy. Therefore, in this case, we avoid to compute the infinite number of iterations required to reach the fixpoint, by directly assuming ω as value. Formally, we introduce the functor $\mathsf{I} : (\mathrm{X} \to \Gamma) \to (\mathrm{X} \to \Gamma)$ defined as follows, where $\mathrm{L} \subseteq \mathrm{St}_{\alpha}$ denotes the set of α-states belonging to a cycle of the above kind: $\mathsf{I}(\mathsf{f})(s)(h) = \omega$, if $s \in \mathrm{L}$ and $\mathsf{f}(s)(h) > 0$, and $\mathsf{I}(\mathsf{f})(s)(h) = \mathsf{f}(s)(h)$, otherwise, for all $s \in \mathrm{St}$ and $h \in [0, j]$. At this point, by induction on the ordering and topology of the strong connected components of the underlying game structure, we can prove that $\mathsf{f}^{\star} = (\mathsf{I} \circ \mathsf{F}^{\alpha}_{\psi})(\mathsf{f}^{\star})$ iff $\mathsf{f}^{\star} = \mathsf{F}^{\alpha}_{\psi}(\mathsf{f}^{\star})$, *i.e.*, the functor obtained by composing I and $\mathsf{F}^{\alpha}_{\psi}$ have exactly the same fixpoint of $\mathsf{F}^{\alpha}_{\psi}$ alone. Moreover, $\mathsf{f}^{\star} = (\mathsf{I} \circ \mathsf{F}^{\alpha}_{\psi})^{n}(\mathsf{f}_{\mathrm{o}})$ where $j \cdot |\mathcal{G}| \leq n$ and f_{o} is the zero function, *i.e.*, $\mathsf{f}_{\mathrm{o}}(s)(h) = 0$, for all $s \in \mathrm{St}$ and $h \in [0, j]$. Hence, we can compute $\mathsf{f}^{\star}$ in a number of iterations of $\mathsf{I} \circ \mathsf{F}^{\alpha}_{\psi}$ that is linear in both the degree j and the size of $\mathcal{G}$. We want to finally observe that the computation of the set L can be done in quadratic time by using a classic Büchi procedure.

As an example of application of the model-checking procedure, consider the two-agent turn-based game structure $\mathcal{G}$ depicted in Figure 3.2, with the circle states ruled by α, the square ones by its opponent $\overline{\alpha}$, and where $\mathtt{s}_5$ and $\mathtt{s}_8$ are labeled by the atomic proposition $\mathtt{p}$. Also, consider the vanilla GSL[1G, 2AG] sentence $\varphi = \langle\!\langle \mathtt{x} \geq i \rangle\!\rangle [\![\overline{\mathtt{x}} \leq j]\!] (\alpha, \mathtt{x})(\overline{\alpha}, \overline{\mathtt{x}})\mathtt{Fp}$. As first thing, we need to compute the five preliminary sets of states $\mathrm{W}_{\mathtt{Fp}} = \{\mathtt{s}_5, \mathtt{s}_8\}$ (the light-gray area), $\mathrm{W}_{\neg\mathtt{Fp}} = \{\mathtt{s}_3, \mathtt{s}_6\}$ (the dark-gray area), $\mathrm{X} = \{\mathtt{s}_0, \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_4, \mathtt{s}_7\}$ (the white area partitioned into strong-connected components), $\mathrm{S} = \{\mathtt{s}_0, \mathtt{s}_2\}$, and $\mathrm{L} = \{\mathtt{s}_7\}$. At this point, we can evaluate the fixpoint $\mathsf{f}^{\star}$ of the functor $\mathsf{I} \circ \mathsf{F}^{\alpha}_{\psi}$ that can be obtained, due to the topology of

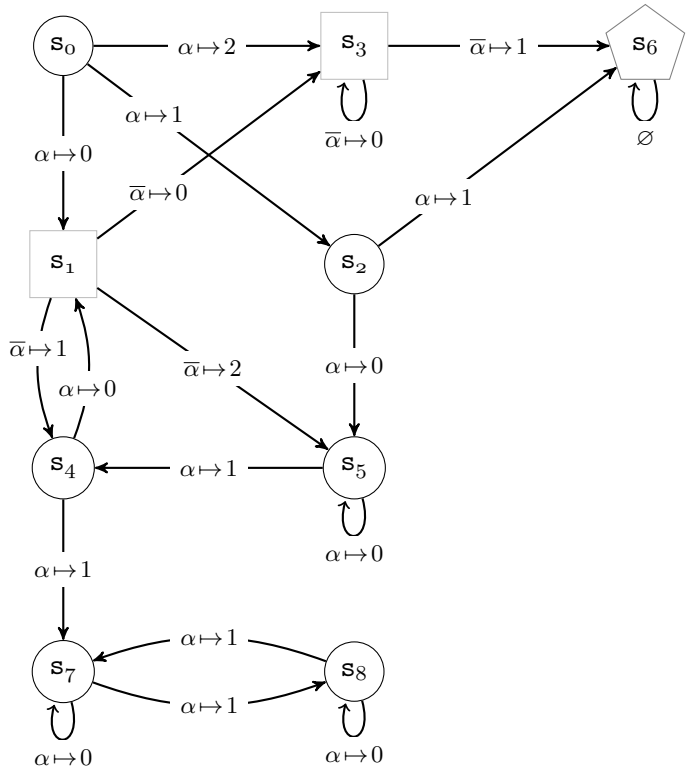

Figure 3.2: A two-player turn-based game structure.

$\mathcal{G}$, after seven iterations, *i.e.*, $\mathsf{f}^\star = (\mathsf{I} \circ \mathsf{F}^\alpha_\psi)^7(\mathsf{f}_0)$. Indeed, at the first one, the values on the states $\mathtt{s}_2$ and $\mathtt{s}_7$ are stabilized to $\mathsf{f}^\star(\mathtt{s}_2)(0) = 1$, $\mathsf{f}^\star(\mathtt{s}_7)(0) = \omega$, and $\mathsf{f}^\star(\mathtt{s}_2)(h) = \mathsf{f}^\star(\mathtt{s}_7)(h) = 0$, for all $h \in [1, j]$. After six iterations, we obtain $\mathsf{f}^\star(\mathtt{s}_1)(0) = 0$, $\mathsf{f}^\star(\mathtt{s}_1)(h) = \omega$, for all $h \in [1, j]$, and $\mathsf{f}^\star(\mathtt{s}_4)(h) = \omega$, for all $h \in [0, j]$. By computing the last iteration, we derive $\mathsf{f}^\star(\mathtt{s}_0)(0) = 1$ and $\mathsf{f}^\star(\mathtt{s}_0)(h) = \omega$, for all $h \in [1, j]$. Note that 7 is exactly the sum $1 + 5 + 1$ of iterations that the components of the longest chain $\{\mathtt{s}_7\} < \{\mathtt{s}_1, \mathtt{s}_4\} < \{\mathtt{s}_0\}$ need in order to stabilize the values on their states. Finally, we can verify whether $\mathcal{G} \models \varphi$, by computing the grading function $\mathsf{G}^\alpha_{\mathtt{Fp}}$ at s_0, whose values are $\mathsf{G}^\alpha_{\mathtt{Fp}}(s_0)(0) = 1$ and $\mathsf{G}^\alpha_{\mathtt{Fp}}(s_0)(h) = \omega$, for all $h \in [1, j]$. Thus, we have that $\mathcal{G} \models \varphi$ iff $i = 1$ or $j > 0$.

In order to obtain a PTIME procedure, we have also to ensure that each evaluation of the composed functor $\mathsf{I} \circ \mathsf{F}^\alpha_\psi$ can be computed in PTIME *w.r.t.* the above mentioned parameters. Actually, the whole I and the first two items of F^α_ψ can be easily calculated in linear time. On the contrary, the last item may in general require a sum of an exponential number of elements. Indeed, due to all possible ways a degree j can be split among the successors of a state s, we have that the corresponding set $\mathrm{C}(s)(j)$ may contain an exponential number of functions. To avoid such a problem, exploiting a technique similar to the one proposed in [BMM10, BMM12], we can linearly transform every game structure into an equivalent one, where all states ruled by $\overline{\alpha}$ have degree at most 2. In this way, the cardinality of $\mathrm{C}(s)(j)$ is

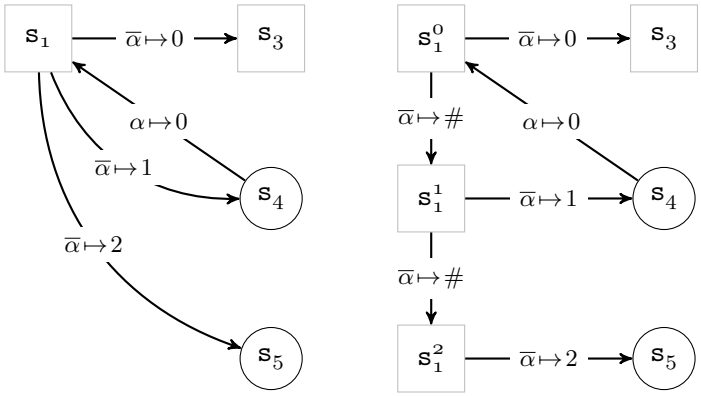

Figure 3.3: Degree transformation.

bounded by j. For example, consider the left part of Figure 3.3 representing the substructure of the previous game structure $\mathcal{G}$ induced by the state s_1 together with its three successors. It is not hard to see that we can replace it, in $\mathcal{G}$, by the binary graph at its right, without changing the number of strategies that the two agents have at their disposal.

Theorem 3.3.2 (Model Checking) *The model-checking problem for* Vanilla GSL[1G, 2AG] *is* PTIME-COMPLETE w.r.t. *both the size of the game structure and the sentence.*

3.4 Discussion

In many multi-agent systems several agents simultaneously compete for the achievement of an individual or conjoint goal. General questions often investigated in these settings are: *is there a winning strategy?* Or, *is the game surely winning?*. Recently, opportune logics for the strategic reasoning have been introduced for the specification of goals and through a suitable application of classical existential and universal modalities it has been possible to address positively the above questions [AHK02].

In game settings, however, questions of equal interest are: *is the winning strategy unique? what is the success rate of the game? is it true that all but k strategies for an agent are winning?* These questions are critical in addressing questions related to fundamental solution concepts. For example in *Nash Equilibrium* uniqueness may provide further insights about the properties of the equilibrium itself[Mye91]. Similarly, knowing that k strategies are not winning at a certain round of a game while they become $k-1$ at the successive round can give insights about the way the competitive agents play. Unfortunately, standard methods to investigate both non-uniqueness and non-universality of strategies are either very restrictive or difficult to evaluate due to the complex combination of moves the agents can take [Mye91].

To answer all above questions we have introduced in this chapter GSL, an extension of Strategy Logic along with graded modalities and investigated basic game-theoretic questions along it. The use of a powerful formalism such as Strategy Logic ensures the ability of

dealing with very intricate game scenarios [MMPV14]. The obvious drawback of this is a considerable amount of work on solving any related question [MMPV12]. One of the main difficulties we have faced in GSL has been the definition of the right methodology to count strategies. To this aim, we have introduced a suitable equivalence relation over strategy profiles based on the strategic behavior they induce and studied its robustness. Also, we have provided arguments and examples along this chapter to give an evidence of the usefulness of GSL and the suitability of the proposed counting.

In order to provide results of practical use, we have investigated basic questions over a restricted fragment of GSL. Precisely we have considered the case in which the graded modalities are applied to the *vanilla* restriction of the one-goal fragment of SL [MMPV12]. The resulting logic, named *Vanilla* SL[1G], has been investigated in the turn-based setting. We have obtained positive results about determinacy and showed that the related model-checking problem is PTIME-COMPLETE.

The framework and the results presented in this chapter open for several future work questions. First, it would be worth investigating the extension of existing formal verification tools such as MCMAS [LR06] along with our results. We recall that MCMAS, originally developed for the verification for multi-agent models with respect to specification given in ATL [LR06], has been recently extended to handle Strategy Logic specifications [ČLMM14]. Under our formalism it is possible, in one round, to report that more than a strategy gives a fault and possibly correct all of them. This in a way similar as the verification tool NuSMV has been extended to deal with *graded-CTL* verification [FNP09a]. Another research direction regards investigating the graded extension of other formalism for the strategic reasoning such as ATL *with context* [BLLM09, LLM10], as well as, for the sake of completeness, to determine the complexity of the model checking problem with respect other fragments of Strategy Logic[MMPV14].

Conclusion

This thesis reports the results of three years of a continuous research work made under the supervision of Prof. Aniello Murano and in collaboration with other colleagues of the university of Napoli . The work has mainly concerned with quantitative aspects in open-system specification, that has allowed us to introduce different new formalisms and algorithms in this setting. Precisely, we have considered both the case in which the specification is glued with the system model (the first part of the thesis) and the case in which it comes as an external formalism (the second part of the thesis).

In the former case, the quantitative requirements we have considered regard the timing aspects of the system, *i.e.*, the elapsing of the time between the start of a particular task and its accomplishment. Specifically, we have worked on several variants of parity games in which the quantitative requirements was added to the classic qualitative one. The solution of the considered specification (properly, winning conditions) have required ad hoc polynomial reductions to either a Büchi or a Parity Game. Therefore, all the solutions we propose, are optimal with respect to the known complexities to solve Büchi and Parity Games.

In the second part of the thesis we have concentrate on multi-agent (open) systems with external quantitative specification. Precisely, we have introduced *Graded Strategy Logic*, an extension of *Strategy Logic*, in order to count different strategies that an agent has available to verify a given formula. The main difficulty in this setting has been to come out with a suitable "semantic" counting over strategies. To this aim we have introduced an ad hoc equivalence relation over strategies and proved to be robust and efficient for our aim. To give an evidence of the usefulness of the introduced framework we have considered a fragment of Graded Strategy Logic and proved that its model checking is solvable in PTIME.

Both the internal and external quantitative specifications we have considered are new and open two different lines of research that, in our opinion, are prone to significant and future improvements. In the internal case setting, one can consider additional or more sophisticated values along the model. For example, one can consider to enrich the model with values that record some energy consumption (as in energy parity game [CD10]) or to use multiple values. Also, another interesting developments concerns to extend the prompt reasoning to infinite state systems by considering, for example, Pushdown Parity Games [Wal01, ALM$^+$13, BSW03] or inject a prompt μ-calculus modal logic (instead of LTL) to have a proper prompt parity extension of Strategy Logic.

In the external specification setting, the work done can be considered as a solid and robust core-engine machinery to deal with sophisticated solution concepts. As next natural step one may would extend our framework to reacher fragments of Graded Strategy Logic. Also it would be useful to implement it in some well-known tools, such as MCMAS [ČLMM14,

LR06] that so far are only able to evaluate qualitative aspects of Strategy Logic. This is just left as future work.

Bibliography

[Ada09] Adam Antonik and Nathaniel Charlton and Michael Huth. Polynomial-time under-approximation of winning regions in parity games. *ENTCS*, 225:115–139, 2009.

[AH98] R. Alur and T. A. Henzinger. Finitary fairness. *ACM Trans. Program. Lang. Syst.*, 20(6), 1998.

[AHK02] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49(5):672–713, 2002.

[AHK10] S. Almagor, Y. Hirshfeld, and O. Kupferman. Promptness in omega-Regular Automata. In *ATVA'10*, LNCS 7388, pages 22–36, 2010.

[AKM12] B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. *Inf. Comput.*, pages 68–86, 2012.

[ALM+13] B. Aminof, A. Legay, A. Murano, O. Serre, and M. Y. Vardi. Pushdown module checking with imperfect information. *Inf. Comput.*, 223:1–17, 2013.

[AMM11] B. Aminof, F. Mogavero, and A. Murano. Synthesis of hierarchical systems. In *FACS '11*, LNCS, pages 42–60. Springer, 2011.

[AMM13] B. Aminof, F. Mogavero, and A. Murano. Synthesis of hierarchical systems. *Science of Comp. Program.*, 83:56–79, 2013.

[Ant14] Antonio Di Stasio and Aniello Murano and Vincenzo Prignano and Loredana Sorrentino. Solving Parity Games in Scala. pages 145–161, 2014.

[ATO+09] T. Antal, A. Traulsen, H. Ohtsuki, C.E. Tarnita, and M.A. Nowak. Mutation-Selection Equilibrium in Games with Multiple Strategies. 258(4):614–622, 2009.

[Bar11] Barringer, Howard and Havelund, Klaus. *TraceContract: A Scala DSL for trace analysis*. Springer, 2011.

[Ber01] Berger, Emery D and Zorn, Benjamin G and McKinley, Kathryn S. Composing high-performance memory allocators. 36(5):114–124, 2001.

[Ber07] D. Berwanger. Admissibility in infinite games. In *STACS'07*, pages 188–199, 2007.

[Ber13] Berger, Emery D and Zorn, Benjamin G and McKinley, Kathryn S. OOPSLA 2002: Reconsidering custom memory allocation. *ACM SIGPLAN Notices*, 48(4):46–57, 2013.

[BLLM09] T. Brihaye, A. Da Costa Lopes, F. Laroussinie, and N. Markey. Atl with strategy contexts and bounded memory. In *LFCS '09*, LNCS 5407, pages 92–106. Springer, 2009.

[BMM09] A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. In *LICS'09*, pages 342–351. IEEE Computer Society, 2009.

[BMM10] A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic with Binary Coding. In *CSL'10*, LNCS 6247, pages 125–139. Springer, 2010.

[BMM12] A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. Association for Computing Machinery, 2012.

[BSW03] A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *FSTTCS*, pages 88–99, 2003.

[CD10] K. Chatterjee and L. Doyen. Energy Parity Games. *CoRR*, abs/1001.5183, 2010.

[CD12] K. Chatterjee and L. Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012.

[CDHR10] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *FSTTCS'10*, LIPIcs 8, pages 505–516, 2010.

[CE81] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71, 1981.

[CGP02] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2002.

[CHH09] K. Chatterjee, T. A. Henzinger, and F. Horn. Finitary winning in ω-regular games. *ACM Trans. Comput. Logic*, 11(1), 2009.

[CHJ05] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *LICS'05*, pages 178–187, 2005.

[CHP07] K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. In *CONCUR'07*, LNCS 4703, pages 59–73. Springer, 2007.

[CJH04] K. Chatterjee, M. Jurdzinski, and T. A. Henzinger. Quantitative stochastic parity games. In *SODA'04*, pages 121–130, 2004.

[CLM15] P. Cermák, A. Lomuscio, and A. Murano. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. 2015.

[ČLMM14] P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV'14*, LNCS 8559, pages 524–531. Springer, 2014.

[DL+08] S. N. Durlauf, B. Lawrence, et al. The new Palgrave dictionary of economics. 2008.

[EJ88] E.A. Emerson and C.S. Jutla. The Complexity of Tree Automata and Logics of Programs (Extended Abstract). pages 328–337, 1988.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *FOCS'91*, pages 368–377, 1991.

[Fin72] K. Fine. In So Many Possible Worlds. *NDJFL*, 13:516–520, 1972.

[FMP08] A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *LMCS*, 4(3):1–21, 2008.

[FNP09a] A. Ferrante, M. Napoli, and M. Parente. Graded-CTL: Satisfiability and Symbolic Model Checking. In *ICFEM'10*, LNCS 5885, pages 306–325. Springer, 2009.

[FNP09b] A. Ferrante, M. Napoli, and M. Parente. Model Checking for Graded CTL. *FI*, 96(3):323–339, 2009.

[Fri09] Friedmann, Oliver and Lange, Martin. The PGSolver collection of parity game solvers. *University of Munich*, 2009.

[FZ12] N. Fijalkow and M. Zimmermann. Cost-parity and cost-streett games. In *FSTTCS'12*, pages 124–135, 2012.

[Gay98] Gay, David and Aiken, Alex. Memory management with explicit regions. 33(5), 1998.

[GHJV94] E. Gammaand, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[GKP94] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics - A Foundation for Computer Science (2nd ed.)*. 1994.

[GOR97] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *LICS'97*, pages 306–317. IEEE Computer Society, 1997.

[HB91] B. Hollunder and F. Baader. Qualifying Number Restrictions in Concept Languages. In *91*, pages 335–346, 1991.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and models of concurrent systems. NATO Advanced Summer Institutes vol. F-13.*, pages 477–498. Springer, New York, NY, USA, 1985.

[HTW08] F. Horn, W. Thomas, and N. Wallmeier. Optimal strategy synthesis in request-response games. In *ATVA'08*, LNCS 5311, pages 361–373, 2008.

[Hun11] Hundt, Robert. Loop recognition in c++/java/go/scala. *Proceedings of Scala Days*, 2011, 2011.

[Jen00] Jens Vöge and Marcin Jurdzinski. A Discrete Strategy Improvement Algorithm for Solving Parity Games. pages 202–215, 2000.

[Jur98] M. Jurdzinski. Deciding the winner in parity games is in up $\cap$ co-up. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[Kel76] R.M. Keller. Formal verification of parallel programs. *CACM*, 19(7):371–384, 1976.

[KMM06] O. Kupferman, G. Morgenstern, and A. Murano. Typeness for omega-regular automata. *Int. J. Found. Comput. Sci.*, 17(4):869–884, 2006.

[Koz83] D. Kozen. Results on the Propositional mu-Calculus. *TCS*, 27(3):333–354, 1983.

[KPV09] O. Kupferman, N. Piterman, and M. Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.

[Kri63] S.A. Kripke. Semantical Considerations on Modal Logic. *APF*, 16:83–94, 1963.

[KSV02] O. Kupferman, U. Sattler, and M.Y. Vardi. The Complexity of the Graded μ-Calculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.

[KV97] O. Kupferman and M. Y. Vardi. Module checking revisited. In *CAV'97*, volume 1254 of *LNCS*, pages 36–47, 1997.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.

[KVW01] O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *IC*,164(2):322–344, 2001.

[LLM10] A.D.C. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS'10*, LIPIcs 8, pages 120–132, 2010.

[LR06] A. Lomuscio and F. Raimondi. Model Checking Knowledge, Strategies, and Games in Multi-Agent Systems. In *AAMAS'06*, pages 161–168, 2006.

[Luc00] Luca de Alfaro and Thomas A. Henzinger and Freddy Y. C. Mang. The Control of Synchronous Systems. pages 458–473, 2000.

[Mar75] A.D. Martin. Borel Determinacy. *AM*, 102(2):363–371, 1975.

[Mar85] A.D. Martin. A Purely Inductive Proof of Borel Determinacy. In *SPM'82*, Recursion Theory, pages 303–308. American Mathematical Society and Association for Symbolic Logic, 1985.

[Mar93] Martín Abadi and Leslie Lamport. Composing Specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

[Mar00] Marcin Jurdzinski. Small Progress Measures for Solving Parity Games. pages 290–301, 2000.

[Mar08] Marcin Jurdzinski and Mike Paterson and Uri Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[McN93] McNaughton, Robert. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[MMPV12] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.

[MMPV14] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.*, 15(4):34, 2014.

[MMS13] F. Mogavero, A. Murano, and L. Sorrentino. On promptness in parity games. In *LPAR*, pages 601–618, 2013.

[MMS14] F. Mogavero, A. Murano, and L. Sauro. Strategy Games: A Renewed Framework. In *AAMAS'14*, pages 869–876, 2014.

[MMV10] F. Mogavero, A. Murano, and M.Y. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144, 2010.

[Mos87] Andrzej Wlodzimierz Mostowski. Hierarchies of weak monadic formulas for two successors arithmetic. *Elektronische Informationsverarbeitung und Kybernetik*, 23(10/11):509–515, 1987.

[Mye91] R.B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1991.

[Ode04] Odersky, Martin and Altherr, Philippe and Cremet, Vincent and Emir, Burak and Maneth, Sebastian and Micheloud, Stéphane and Mihaylov, Nikolay and Schinz, Michel and Stenman, Erik and Zenger, Matthias. An overview of the Scala programming language. 2004.

[Ode08] Odersky, Martin and Spoon, Lex and Venners, Bill. *Programming in scala*. Artima Inc, 2008.

[Oli09] Oliver Friedmann and Martin Lange. Solving Parity Games in Practice. pages 182–196, 2009.

[Orn00] Orna Kupferman and P. Madhusudan and P. S. Thiagarajan and Moshe Y. Vardi. Open Systems in Reactive Environments: Control and Synthesis. pages 92–107, 2000.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*, pages 46–57, 1977.

[PP04] D. Perrin and J. Pin. *Infinite Words.*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[PR89] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. pages 179–190, 1989.

[QS81] J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Programs in Cesar. In *SP'81*, LNCS 137, pages 337–351, 1981.

[Raj97] Rajeev Alur and Thomas A. Henzinger and Orna Kupferman. Alternating-Time Temporal Logic. pages 23–60, 1997.

[RB94] E. Rasmusen and B. Blackwell. Games and information. *Cambridge, MA*, 15, 1994.

[Sve07] Sven Schewe. Solving Parity Games in Big Steps. pages 449–460, 2007.

[Tho90] W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science (vol. B)*, pages 133–191. MIT Press, 1990.

[Wal01] Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.