Rational Synthesis

Dana Fisman
Hebrew University and IBM Haifa

Orna Kupferman
Hebrew University

Yoad Lustig
Rice University

Abstract

Synthesis is the automated construction of a system from its specification. The system has to satisfy its specification in all possible environments. Modern systems often interact with other systems, or agents. Many times these agents have objectives of their own, other than to fail the system. Thus, it makes sense to model system environments not as hostile, but as composed of *rational agents*; i.e., agents that act to achieve their own objectives.

We introduce the problem of synthesis in the context of rational agents (*rational synthesis*, for short). The input consists of a temporal-logic formula specifying the system and temporal-logic formulas specifying the objectives of the agents. The output is an implementation T of the system and a profile of strategies, suggesting a behavior for each of the agents. The output should satisfy two conditions. First, the composition of T with the strategy profile should satisfy the specification. Second, the strategy profile should be an equilibria in the sense that, in view of their objectives, agents have no incentive to deviate from the strategies assigned to them. We solve the rational-synthesis problem for various definitions of equilibria studied in game theory. We also consider the multi-valued case in which the objectives of the system and the agents are still temporal logic formulas, but involve payoffs from a finite lattice.

1 Introduction

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it adheres to its specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [10]; the modern approach to synthesis was initiated by Pnueli and Rosner, who introduced LTL (linear temporal logic) synthesis [32]. In LTL synthesis, the specification is given in LTL and the output is a reactive system modeled by a finite-state transducer. Much of today's research in formal verification is aimed at increasing the practicality of automated synthesis, and it addresses challenges like simplification of synthesis algorithms [19], compositionality and modularity [17, 23], extensions of the basic setting to richer ones (c.f., synthesis of distributed systems, concurrent systems, and on-line algorithms [1, 2, 18, 25]), and extensions of the underline techniques to further applications (c.f. automated control and repair [14, 33]).

In synthesis, there is a distinction between system outputs, controlled by the system, and system inputs, controlled by the environment. A system should be able to cope with all values of the input signals, while setting the output signals to desired values [32]. Therefore, the quantification structure on input and output signals is different. Input signals are universally quantified while output signals are existentially quantified.

Modern systems often interact with other systems. For example, the clients interacting with a server are by themselves distinct entities (which we call agents) and are many times implemented by systems. In the traditional approach to synthesis, the way in which the environment is composed of its underlying agents is abstracted. In particular, the agents can be seen as if their only objective is to conspire to fail the system. Hence the term "hostile environment" that is traditionally used in the context of synthesis. In real life, however, many times agents have goals of their own, other than to fail the system. The approach taken in the field of algorithmic game theory [29] is to assume that agents interacting with a computational system are *rational*, i.e., agents act to achieve their own goals. Assuming agents rationality is a restriction on the agents behavior and is therefore equivalent to restricting the universal quantification on the environment. Thus, the following question arises: can system synthesizers capitalize on the rationality and goals of agents interacting with the system?

Consider for example a peer-to-peer network with only two agents. Each agent is interested in downloading infinitely often, but has no incentive to upload. In order, however, for one agent to download, the other agent must upload. More formally, for each $i \in \{0,1\}$, Agent i controls the bits u_i ("Agent i tries to upload") and d_i ("Agent i tries to download"). The objective of Agent i is *always eventually* $(d_i \wedge u_{1-i})$. Assume that we are asked to synthesize the protocol for Agent 0. It is not hard to see that the objective of Agent 0 depends on his input signal, implying he cannot ensure his objective in the traditional synthesis sense. On the other hand, suppose that Agent 0, who is aware of the objective of Agent 1, declares and follows the following TIT FOR TAT strategy: I will upload at the first time step, and from that point onward I will reciprocate the actions of Agent 1. Formally, this amounts to initially setting u_0 to **True** and for every time $k > 0$, setting u_0 at time k to equal u_1 at time $k-1$. It is not hard to see that, against this strategy, Agent 1 can only ensure his objective by satisfying Agent 0 objective as well. Thus, assuming Agent 1 acts rationally, Agent 0 can ensure his objective.

The example above demonstrates that a synthesizer can capitalize on the rationality of the agents that constitute its environment. When synthesizing a protocol for rational agents, we still have no control on their actions. We would like, however, to generate a strategy for each agent (a *strategy profile*) such that once the strategy profile is given to the agents, then a rational agent would have no incentive to deviate from the strategy suggested to him and would follow it. Such a strategy profile is called in game theory a *solution* to the game. Accordingly, the *rational synthesis* problem gets as input temporal-logic formulas specifying the objective φ_0 of the system and the objectives $\varphi_1, \ldots, \varphi_n$ of the agents that constitute the environment. The desired output is a system and a strategy profile for the agents such that the following hold. First, if all agents adhere to their strategies, then the result of the interaction of the system and the agents satisfies φ_0. Second, once the system is in place, and the agent are playing a game among themselves, the strategy profile is a solution to this game.[1]

A well known solution concept is *Nash equilibrium* [27]. A strategy profile is in Nash equilibrium if no agent has an incentive to deviate from his assigned strategy, provided that the other agents adhere to the strategies assigned to them. For example, if the TIT FOR TAT strategy for Agent 0 is suggested to both agents, then the pair of strategies is a Nash equilibrium. Indeed, for all $i \in \{0,1\}$, if Agent i assumes that Agent $1-i$ adheres to his strategy, then by following the strategy, Agent i knows that his objective would be satisfied, and he has no incentive to deviate from it. The stability of a Nash equilibrium depends on the players assumption that the other players adhere to the strategy. In some cases this is a reasonable assumption. Consider, for example, a standard protocol published by some known authority such as IEEE. When a programmer writes a program implementing the standard, he tends to assume that his program is going to interact with other programs that implement the same standard. If the published standard is a Nash equilibrium, then there is no incentive to write a program that diverts from the standard. Game theory suggests several *solution concepts*, all capturing the idea that the participating agents have no incentive to deviate from the protocol (or strategy) assigned to them. We consider three well-studied solution concepts [29]: dominant-strategies solution, Nash equilibrium, and subgame-perfect Nash equilibrium.

An important facet in the task of a rational synthesizer is to synthesize a system such that once it is in place, the game played by the agents has a solution with a favorable outcome. *Mechanism design*, studied in game theory and economy [28, 29], is the study of designing a game whose outcome (assuming players rationality) achieves some goal. Rational synthesis can be viewed as a variant of mechanism design in which the game is induced by the objective of the system, and the objectives of both the system and the agents refer to their on-going interaction and are specified by temporal-logic formulas.

Having defined rational synthesis, we turn to solve it. In [7], the authors introduced *strategy logic* – an extension of temporal logic with first order quantification over strategies. The rich structure of strategy logic enables it to specify properties like the existence of a Nash-equilibrium. While [7] does not consider the synthesis problem, the technique suggested there can be used in order to solve the rational-synthesis problem for Nash equilibrium and dominant strategies. Strategy logic, however, is not sufficiently expressive in order to specify subgame-perfect-Nash equilibrium [35] which, as advocated in [37] (see also Section 3), is the most suited for infinite multiplayer games — those induced by rational synthesis. The weakness of strategy logic is its inability to quantify over game histories. We extend strategy logic with history variables, and show that the extended logic is sufficiently expressive to express rational synthesis for the three solution concepts we study. Technically, adding history variables to strategy logic results in a *memoryful logic* [21], in which temporal logic formulas have to be evaluated not along paths that start at the present, but along paths that start at the root and go through the present.

[1]For a formal definition of *rational synthesis*, see Definition 3.1.

Classical applications of game theory consider games with real-valued payoffs. For example, agents may bid on goods or grade candidates. In the peer-to-peer network example, one may want to refer to the amount of data uploaded by each agent, or one may want to add the possibility of pricing downloads. The full quantitative setting is undecidable already in the context of model checking [3]. Yet, several special cases for which the problem is decidable have been studied [4]. We can distinguish between cases in which decidability is achieved by restricting the type of systems [3], and cases in which it is achieved by restricting the domain of values [13]. We solve the quantitative rational synthesis problem for the case the domain of values is a finite distributive De Morgan lattice. The lattice setting is a good starting point to the quantitative setting. First, lattices have been successfully handled for easier problems, and in particular, multi-valued synthesis [15, 16]. In addition, lattices are sufficiently rich to express interesting quantitative properties. This is sometime immediate (for example, in the peer-to-peer network, one can refer to the different attributions of the communication channels, giving rise to the lattice of the subsets of the attributions), and sometimes thanks to the fact that real values can often be abstracted to finite linear orders. From a technical point of view, our contribution here is a solution of a latticed game in which the value of the game cannot be obtained by joining values obtained by different strategies, which is unacceptable in synthesis.

1.1 Related Work

Already early work on synthesis has realized that working with a hostile environment is often too restrictive. The way to address this point, however, has been by adding assumptions on the environment, which can be part of the specification (c.f., [5]). The first to consider the game-theoretic approach to dealing with rationality of the environment in the context of LTL synthesis were Chatteerjee and Henzinger [8]. The setting in [8], however, is quite restricted; it considers exactly three players, where the third player is a fair scheduler, and the notion of *secure equilibria* [6]. Secure equilibria, introduced in [6], is a Nash equilibria in which each of the two players prefers outcomes in which only his objective is achieved over outcomes in which both objectives are achieved, which he still prefers over outcomes in which his objective is not achieved. It is not clear how this notion can be extended to multiplayer games, and to the distinction we make here between controllable agents that induce the game (the system) and rational agents (the environment). Also, the set of solution concepts we consider is richer.

Ummels [37] was the first to consider subgame perfect equilibria in the context of infinite multiplayer games. The setting there is of turn-based games and the solution goes via a reduction to 2-player games. Here, we consider concurrent games and therefore cannot use such a reduction. Another difference is that [37] considers parity winning conditions whereas we use LTL objectives. In addition, the fact that the input to the rational synthesis problem does not include a game makes the memoryful nature of subgame perfect equilibria more challenging, as we cannot easily reduce the LTL formulas to memoryless parity games.

To the best of our knowledge, we are the first to handle the multi-valued setting. As we show, while the lattice case is decidable, its handling required a nontrivial extension of both the Boolean setting and the algorithms known for solving latticed games [16].

2 Preliminaries

We consider *infinite concurrent multiplayer games* (in short, *games*) defined as follows. A *game arena* is a tuple $\mathcal{G} = \langle V, v_0, I, (\Sigma_i)_{i\in I}, (\Gamma_i)_{i\in I}, \delta\rangle$, where V is a set of nodes, v_0 is an initial node, I is a set of players, and for $i \in I$, the set Σ_i is the set of actions of Player i and $\Gamma_i : V \to 2^{\Sigma_i}$ specifies the actions that Player i can take at each node. Let $I = \{1, \ldots, n\}$. Then, the transition relation $\delta : V \times \Sigma_1 \times \cdots \times \Sigma_n \to V$ is a deterministic function mapping the current node and the current choices of the agents to the successor node. The transition function may be restricted to its relevant domain. Thus, $\delta(v, \sigma_1, \ldots, \sigma_n)$ is defined for $v \in V$ and $\langle \sigma_1, \ldots, \sigma_n\rangle \in \Gamma_1(v) \times \cdots \times \Gamma_n(v)$.

A *position* in the game is a tuple $\langle v, \sigma_1, \sigma_2, \ldots, \sigma_n\rangle$ with $v \in V$ and $\sigma_i \in \Gamma_i(v)$ for every $i \in I$. Thus, a position describes a state along with possible choices of actions for the players in this state. Consider a sequence $p = p_0 \cdot p_1 \cdot p_2 \cdots$ of positions. For $k \geq 0$, we use *node*(p_k) to denote the state component of p_k, and use $p_k[i]$, for $i \in I$, to denote the action of Player i in p_k. The notations extend to p in the straightforward way. Thus, *node*(p) is the projection of p on the first component. We say that p is a *play* if the transitions between positions is consistent with δ. Formally, p is a *play starting at node* v if *node*$(p_0) = v$ and for all $k \geq 0$, we have *node*$(p_{k+1}) = \delta(p_k)$. We use $\mathcal{P}_\mathcal{G}$ (or simply $\mathcal{P}$ when $\mathcal{G}$ is clear from the context) to denote all possible plays of $\mathcal{G}$.

Note that at every node $v \in V$, each player i chooses an action $\sigma_i \in \Gamma_i(v)$ simultaneously and independently of the other players. The game then proceeds to the successor node $\delta(v, \sigma_1, \ldots, \sigma_n)$. A *strategy* for Player i is a function $\pi_i : V^+ \mapsto \Sigma_i$ that maps histories of the game to an action suggested to Player i. The suggestion has to be consistent with Γ_i. Thus, for every $v_0 v_1 \cdots v_k \in V^+$, we have $\pi_i(v_0 v_1 \cdots v_k) \in \Gamma_i(v_k)$. Let Π_i denote the set of possible strategies for Player i. For a set of players $I = \{1, \ldots, n\}$, a *strategy profile* is a tuple of strategies $\langle \pi_1, \pi_2, \ldots, \pi_n \rangle \in \Pi_1 \times \Pi_2 \times \cdots \times \Pi_n$. We denote the strategy profile by $(\pi_i)_{i \in I}$ (or simply π, when I is clear from the context). We say that p is an *outcome* of the profile π if for all $k \geq 0$ and $i \in I$, we have $p_k[i] = \pi_i(\textit{node}(p_0) \cdot \textit{node}(p_1) \cdots \textit{node}(p_k))$. Thus, p is an outcome of π if all the players adhere to their strategies in π. Note that since δ is deterministic, π fixes a single play from each state of the game. Given a profile π we denote by *outcome*$(\pi)^\mathcal{G}$ (or simply *outcome*(π)) the one play in $\mathcal{G}$ that is the outcome of π when starting in v_0. Given a strategy profile π and a nonempty sequence of nodes $h = v_0 v_1 \ldots v_k$, we define the *shift of* π *by* h as the strategy profile $(\pi_i^h)_{i \in I}$ in which for all $i \in I$ and all histories $w \in V^*$, we have $\pi_i^h(w) = \pi_i(h \cdot w)$. We denote by *outcome*$(\pi)_h^\mathcal{G}$ (or simply *outcome*$(\pi)_h$) the concatenation of $v_0 v_1 \ldots v_{k-1}$ with the one play in $\mathcal{G}$ that is the outcome of π^h when starting in v_k. Thus, *outcome*$(\pi)_h$ describes the outcome of a game that has somehow found itself with history h, and from that point, the players behave if the history had been h. Given a profile $(\pi_i)_{i \in I}$, an index $j \in I$, and a strategy π'_j for Player j, we use (π_{-j}, π'_j) to refer to the profile of strategies in which the strategy for all players but j is as in π, and the strategy for Player j is π'_j. Thus, $(\pi_{-j}, \pi'_j) = \langle \pi_1, \pi_2, \ldots, \pi_{j-1}, \pi'_j, \pi_{j+1}, \ldots, \pi_n \rangle$.

3 Rational Synthesis

In this section we define the problem of rational synthesis. We work with the following model: the world consists of the *system* and a set of n agents *Agent 1*$, \ldots ,$*Agent n*. For uniformity we refer to the system as *Agent 0*. We assume that Agent i controls a set X_i of variables, and the different sets are pairwise disjoint. At each point in time, each agent sets his variables to certain values. Thus, an action of *Agent i* amounts to assigning values to his variables. Accordingly, the set of actions of *Agent i* is given by 2^{X_i}. We use X to denote $\bigcup_{0 \leq i \leq n} X_i$. We use X_{-i} to denote $X \setminus X_i$ for $0 \leq i \leq n$. Each of the agents (including the system) has an objective. The objective of an agent is formulated using a linear temporal logic formula (LTL [31]) over the set of variables of all agents.[2] We use φ_i to denote the objective of *Agent i*.

This setting induces the game arena $\mathcal{G} = \langle V, v_0, I, (\Sigma_i)_{i \in I}, (\Gamma_i)_{i \in I}, \delta \rangle$ defined as follows. The set of players $I = \{0, 1, \ldots, n\}$ consists of the system and the agents. The moves of agent i are all the possible assignments to its variables. Thus, $\Sigma_i = 2^{X_i}$. We use Σ, Σ_i, and Σ_{-i} to denote the sets 2^X, 2^{X_i}, and $2^{X_{-i}}$, respectively. An agent can set his variables as he wishes throughout the game. Thus $\Gamma_i(v) = \Sigma_i$ for every $v \in V$. The game records in its vertices all the actions taken by the agents so far. Hence, $V = \Sigma^*$ and for all $v \in \Sigma^*$ and $\langle \sigma_0, \ldots, \sigma_n \rangle \in \Sigma$, we have $\delta(v, \sigma_0, \ldots, \sigma_n) = v \cdot \langle \sigma_0, \ldots, \sigma_n \rangle$.

At each moment in time, the system gets as input an assignment in Σ_{-0} and it generates as output an assignment in Σ_0. For every possible history $h \in (\Sigma_{-0} \cup \Sigma_0)^*$ the system should decide what $\sigma_0 \in \Sigma_0$ it outputs next. Thus, a strategy for the system is a function $\pi_0 : \Sigma^* \to \Sigma_0$ (recall that $\Sigma = \Sigma_{-0} \cup \Sigma_0$ and note that indeed $V^+ = \Sigma^*$). In the standard synthesis problem, we say that π_0 realizes φ_0 if all the computations that π_0 generates satisfy φ_0. In rational synthesis, on the other hand, we also generate strategies for the other agents, and the single computation that is the outcome of all the strategies should satisfy φ_0. That is, we require *outcome*$(\pi)^\mathcal{G} \models \varphi_0$ where $\mathcal{G}$ is as defined above. In addition, we should generate the strategies for the other agents in a way that would guarantee that they indeed adhere to their strategies.

Recall that while we control the system, we have no control on the behaviors of *Agent 1*$, \ldots ,$*Agent n*. Let $\pi_0 : \Sigma^* \to \Sigma_0$ be a strategy for the system in $\mathcal{G}$. Then, π_0 induces the game $\mathcal{G}_{\pi_0} = \langle \Sigma^*, \epsilon, I, (\Sigma_i)_{i \in I}, (\Gamma'_i)_{i \in I}, \delta \rangle$, where for $i \in I \setminus \{0\}$, we have $\Gamma'_i = \Gamma_i$, and $\Gamma'_0(w) = \{\pi_0(w_{-0})\}$, where w_{-0} is obtained form w by projecting its letters on Σ_{-0}. Recall that δ is restricted to the relevant domain. Thus, as Γ'_0 is deterministic, we can

[2] We could have worked with any other ω-regular formalism for specifying the objectives. We chose LTL for simplicity of the presentation.

regard $\mathcal{G}_{\pi_0}$ as an n-player (rather than $n+1$-player) game. Note that $\mathcal{G}_{\pi_0}$ contains all the possible behaviors of *Agent 1*, $\ldots$, *Agent n*, when the system adheres to π_0.

Definition 3.1 (Rational Synthesis) Consider a solution concept γ. The problem of rational synthesis (with solution concept γ) is to return, given LTL formulas $\varphi_0, \varphi_1, \ldots, \varphi_n$, specifying the objectives of the system and the agents constituting its environment, a strategy profile $\pi = \langle \pi_0, \pi_1, \ldots, \pi_n \rangle \in \Pi_0 \times \Pi_1 \times \cdots \times \Pi_n$ such that both (a) $\mathit{outcome}(\pi)^{\mathcal{G}} \models \varphi_0$ and (b) the strategy profile $\langle \pi_1, \ldots, \pi_n \rangle$ is a solution in the game $\mathcal{G}_{\pi_0}$ with respect to the solution concept γ. ⌟

The rational-synthesis problem gets a solution concept as a parameter. As discussed in Section 1, the fact $\langle \pi_1, \ldots, \pi_n \rangle$ is a solution with respect to the concept guarantees that it is not worthwhile for the agents constituting the environment to deviate from the strategies assigned to them. Several solution concepts are studied and motivated in game theory. We focus on three leading concepts, and we first recall their definitions and motivations in game theory. The common setting in game theory is that the objective for each player is to maximize his *payoff* – a real number that is a function of the play. We use $\mathit{payoff}_i : \mathcal{P} \to \mathbb{R}$ to denote the payoff function of player i. That is, payoff_i assigns to each possible play p a real number $\mathit{payoff}_i(p)$ expressing the payoff of i on p. For a strategy profile π we use (with a slight abuse of notation) $\mathit{payoff}_i(\pi)$ to abbreviate $\mathit{payoff}_i(\mathit{outcome}(\pi))$.

The simplest and most appealing solution concept is dominant-strategies solution. A *dominant strategy* is a strategy that a player can never lose by adhering to, regardless of the strategies of the other players. Therefore, if there is a profile of strategies π in which all strategies π_i are dominant, then no player has an incentive to deviate from the strategy assigned to him in π. Formally, π is a *dominant strategy profile* if for every $1 \leq i \leq n$ and for every profile π' with $\pi'_i \neq \pi_i$, we have that $\mathit{payoff}_i(\pi') \leq \mathit{payoff}_i(\pi'_{-i}, \pi_i)$. Consider, for example, a game played by three players: Alice, Bob and Charlie whose actions are $\{a_1, a_2\}$, $\{b_1, b_2\}$ and $\{c_1, c_2\}$, respectively. The game is played on the game arena depicted in the left of Figure 1. The labels on the edges are marked by the possible action moves. Each player wants to visit infinitely often a node marked by his initial letter. In this game, Bob's strategy of choosing b_1 from Node 2 is a dominant strategy. All of the strategies of Charlie are dominating. Alice, though, has no dominating strategy. Unfortunately, in many games some agents do not have dominant strategies, thus no dominant-strategy solution exists. Naturally, if no dominant strategy solution exists, one would still like to consider other solution concepts.

Another well known solution concept is Nash equilibrium [27]. A strategy profile is *Nash equilibrium* if no player has an incentive to deviate from his strategy in π provided he assumes the other players adhere to the strategies assigned to them in π. Formally, π is a *Nash equilibrium profile* if for every $1 \leq i \leq n$ and for every strategy $\pi'_i \neq \pi_i$, we have that $\mathit{payoff}_i(\pi_{-i}, \pi'_i) \leq \mathit{payoff}_i(\pi)$. For example, the strategy profile depicted in the middle of Figure 1 by dotted edges is a Nash equilibrium of the game to its left. Knowing the strategy of the other players, each player cannot gain by deviating from his strategy.

An important advantage of Nash equilibrium is that a Nash equilibrium exists in almost every game [30].[3] A weakness of Nash equilibrium is that it is not nearly as stable as a dominant-strategy solution: if one of the other players deviates from his assigned strategy, nothing is guaranteed.

Nash equilibrium is suited to a type of games in which the players make all their decisions without knowledge of other players choices. The type of games considered in rational synthesis, however, are different, as players do have knowledge about the choices of the other players in earlier rounds of the game. To see the problem that this setting poses for Nash equilibrium, let us consider the ULTIMATUM game. In ULTIMATUM, Player 1 chooses a value $x \in [0, 1]$, and then Player 2 chooses whether to accept the choice, in which case the payoff of Player 1 is x and the payoff of Player 2 is $1 - x$, or to reject the choice, in which case the payoff of both players is 0. One Nash equilibrium in ULTIMATUM is $\pi = \langle \pi_1, \pi_2 \rangle$ in which π_1 advises Player 1 to always choose $x = 1$ and π_2 advises Player 2 to always reject. It is not hard to see that π is indeed a Nash equilibrium. In particular, if Player 2 assumes that Player 1 follows π_1, he has no incentive to deviate from π_2. Still, the equilibrium is unstable. The reason is that π_2 is inherently not credible. If Player 1 chooses x smaller than 1, it is irrational for Player 2 to reject, and Player 1 has no reason to assume that Player 2 adheres to π_2. This instability of a Nash equilibrium is especially true in a setting in which the players have information about the choices made by the other players. In particular, in ULTIMATUM, Player 1 knows that Player 2 would make his choice after knowing what x is.

[3] In particular, all n-player turn-based games with ω-regular objectives have Nash equilibrium [9].

To see this problem in the setting of infinite games, consider the strategy profile depicted in the right of Figure 1 by dashed edges. This profile is also a Nash equilibrium of the game in the left of the figure. It is, however, not very rational. The reason is that if Alice deviates from her strategy by choosing a_2 rather than a_1 then it is irrational for Bob to stick to his strategy. Indeed, if he sticks to his strategy he does not meet his objective, yet if he deviates and chooses b_1 he does meet his objective.

This instability of Nash equilibrium has been addressed in the definition of subgame-perfect equilibrium [35]. A strategy profile π is in *subgame-perfect equilibrium (SPE)* if for every possible history of the game, no player has an incentive to deviate from his strategy in π provided he assumes the other players adhere to the strategies assigned to them in π. Formally, π is an SPE profile if for every possible history h of the game, player $1 \leq i \leq n$, and strategy $\pi'_i \neq \pi_i$, we have that $\mathit{payoff}_i(\pi_{-i}, \pi'_i)_h \leq \mathit{payoff}_i(\pi)_h$. The dotted strategy depicted in the middle of Figure 1 is a subgame-perfect equilibrium. Indeed, it is a Nash equilibrium from every possible node of the arena, including non-reachable ones.

In the context of on-going behaviors, real-valued payoffs are a big challenge and most works on reactive systems use Boolean temporal-logic as a specification language. Below we adjust the definition of the three solution concepts to the case the objectives are LTL formulas.[4] Essentially, the adjustment is done by assuming the following simple payoffs: If the objective φ_i of Agent i holds, then his payoff is 1; otherwise his payoff is 0. The induced solution concepts are then as followed. Consider a strategy profile $\pi = \langle \pi_1, \ldots, \pi_n \rangle$.

- We say that π is a *dominant strategy profile* if for every $1 \leq i \leq n$ and profile π' with $\pi'_i \neq \pi_i$, if $\mathit{outcome}(\pi') \models \varphi_i$, then $\mathit{outcome}(\pi'_{-i}, \pi_i) \models \varphi_i$.
- We say that π is a *Nash equilibrium profile* if for every $1 \leq i \leq n$ and profile π' with $\pi'_i \neq \pi_i$, if $\mathit{outcome}(\pi_{-i}, \pi'_i) \models \varphi_i$, then $\mathit{outcome}(\pi) \models \varphi_i$.
- We say that π is a *subgame-perfect equilibrium profile* if for every history $h \in \Sigma^*$, $1 \leq i \leq n$, and profile π' with $\pi'_i \neq \pi_i$, if $\mathit{outcome}(\pi_{-i}, \pi'_i)_h \models \varphi_i$, then $\mathit{outcome}(\pi)_h \models \varphi_i$.

4 Solution in the Boolean Setting

In this section we solve the rational-synthesis problem. Let $I = \{0, 1, \ldots, n\}$ denote the set of agents. Recall that $\Sigma_i = 2^{X_i}$ and $\Sigma = 2^X$, where $X = \cup_{i \in I} X_i$, and that the partition of the variables among the agents induces a game arena with states in Σ^*. Expressing rational synthesis involves properties of strategies and histories. *Strategy Logic* [7] is a logic that treats strategies in games as explicit first-order objects. Given an LTL formula ψ and strategy variables $z_0, \ldots, z_n$ ranging over strategies of the agents, the strategy logic formula $\psi(z_0, \ldots, z_n)$ states that ψ holds in the outcome of the game in which Agent i adheres to the strategy z_i. The use of existential and universal quantifiers on strategy variables enables strategy logic to state that a given profile consists of dominant strategies or is a Nash equilibrium. However, strategy logic is not strong enough to state the existence of a subgame perfect equilibrium. The reason is that a formula $\varphi(z_0, \ldots, z_n)$ in strategy logic assumes that the strategies $z_0, \ldots,$ z_n are computed from the initial vertex of the game, and it cannot refer to histories that diverge from the strategies. We therefore extend strategy logic with first order variables that range over arbitrary histories of the game.

4.1 Extended Strategy Logic

Formulas of *Extended Strategy Logic* (ESL) are defined with respect to a game $\mathcal{G} = \langle V, v_0, I, (\Sigma_i)_{i \in I}, (\Gamma_i)_{i \in I}, \delta \rangle$, a set $\mathbb{H}$ of history variables, and sets $\mathbb{Z}_i$ of strategy variables for $i \in I$. Let $I = \{0, \ldots, n\}$, $\Sigma = \Sigma_0 \times \cdots \times \Sigma_n$, and let ψ be an LTL formula over Σ. Let h be a history variable in $\mathbb{H}$, and let $z_0, ..., z_n$ be strategy variables in $\mathbb{Z}_0, \ldots, \mathbb{Z}_n$, respectively. We use z as an abbreviation for $z_0, ..., z_n$. The set of ESL formulas is defined inductively as follows.[5]

$$\Psi ::= \quad \psi(z) \mid \psi(z; h) \mid \Psi \vee \Psi \mid \neg \Psi \mid \exists z_i . \Psi \mid \exists h . \Psi$$

[4] In Section 5, we make a step towards generalizing the framework to the multi-valued setting and consider the case the payoffs are taken from a finite distributive lattice.

[5] We note that strategy logic as defined in [7] allows the application of LTL path operators ($\bigcirc$ and $\mathcal{U}$) on strategy logic closed formulas. Since we could not come up with a meaningful specification that uses such applications, we chose to ease the presentation and do not allow them in ESL. Technically, it is easy to extend ESL and allow such applications.

We use the usual abbreviations $\wedge, \rightarrow$, and $\forall$. We denote by *free*(Ψ) the set of strategy and history variables that are *free* (not in a scope of a quantifier) in Ψ. A formula Ψ is *closed* if *free*$(\Psi) = \emptyset$. The *alternation depth* of a variable of a closed formula is the number of quantifier switches ($\exists\forall$ or $\forall\exists$, in case the formula is in positive normal form) that bind the variable. The *alternation depth* of closed formula Ψ is the maximum alternation depth of a variable occurring in the formula.

We now define the semantics of ESL. Intuitively, an ESL formula of the form $\psi(z;h)$ is interpreted over the game whose prefix matches the history h and the suffix starting where h ends is the outcome of the game that starts at the last vertex of h and along which each agent $i \in I$ adheres to his strategy in z. Let $\mathbb{X} \subseteq \mathbb{H} \cup \bigcup_{i\in I} \mathbb{Z}_i$ be a set of variables. An assignment $\mathcal{A}_{\mathbb{X}}$ assigns to every history variable $h \in \mathbb{X} \cap \mathbb{H}$, a history $\mathcal{A}_{\mathbb{X}}(h) \in V^+$ and assigns to every strategy variable $z_i \in \mathbb{X} \cap \mathbb{Z}_i$, a strategy $\mathcal{A}_{\mathbb{X}}(z_i) \in \Pi_i$. Given an assignment $\mathcal{A}_{\mathbb{X}}$ and a strategy $\pi_i \in \Pi_i$, we denote by $\mathcal{A}_{\mathbb{X}}[z_i \leftarrow \pi_i]$ the assignment $\mathcal{A}'_{\mathbb{X}\cup\{z_i\}}$ in which $\mathcal{A}'_{\mathbb{X}\cup\{z_i\}}(z_i) = \pi_i$ and for a variable $x \neq z_i$ we have $\mathcal{A}'_{\mathbb{X}\cup\{z_i\}}(x) = \mathcal{A}_{\mathbb{X}}(x)$. For histories of the game $w \in V^+$ we define $\mathcal{A}_{\mathbb{X}}[h \leftarrow w]$ similarly.

We now describe when a given game $\mathcal{G}$ and a given assignment $\mathcal{A}_{\mathbb{X}}$ satisfy an ESL formula Ψ, where $\mathbb{X}$ is such that *free*$(\Psi) \subseteq \mathbb{X}$. For LTL, the semantics is as usual [24].

$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \psi(z)$	iff	*outcome*$(\mathcal{A}_{\mathbb{X}}(z))^{\mathcal{G}} \models \psi$	$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \Psi_1 \vee \Psi_2$	iff	$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \Psi_1$ or $(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \Psi_2$
$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \psi(z;h)$	iff	*outcome*$(\mathcal{A}_{\mathbb{X}}(z))^{\mathcal{G}}_{\mathcal{A}_{\mathbb{X}}(h)} \models \psi$	$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \exists z_i.\Psi$	iff	$\exists \pi_i {\in} \Pi_i.(\mathcal{G},\mathcal{A}_{\mathbb{X}}[z_i \leftarrow \pi_i]) \models \Psi$
$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \neg\Psi$	iff	$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \not\models \Psi$	$(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \exists h.\Psi$	iff	$\exists w {\in} V^+.(\mathcal{G},\mathcal{A}_{\mathbb{X}}[h \leftarrow w]) \models \Psi$

For an ESL formula Ψ we use $[\![\Psi]\!]$ to denote its set of satisfying assignments; that is, $[\![\Psi]\!] = \{(\mathcal{G}, \mathcal{A}_{\mathbb{X}}) \mid \mathbb{X} =$ *free*(Ψ) and $(\mathcal{G},\mathcal{A}_{\mathbb{X}}) \models \Psi\}$. Given an ESL formula Ψ and a game graph $\mathcal{G}$, we denote by $[\![\Psi]\!]_{\mathcal{G}}$ the assignment $\mathcal{A}_{\mathbb{X}}$ to the free variables in Ψ such that $(\mathcal{G}, \mathcal{A}_{\mathbb{X}}) \in [\![\Psi]\!]$.

Before we show how $[\![\Psi]\!]_{\mathcal{G}}$ can be computed we show that ESL is strong enough to express the solution to the rational-synthesis problems for the three solution concepts we study.

4.2 Expressing Rational Synthesis

We now show that the rational synthesis problem for the three solution concepts we study can be stated in ESL. We first state that a given strategy profile $y = (y_i)_{i\in I}$ is a solution concept on the game $\mathcal{G}_{y_0}$, that is, the game induced by $\mathcal{G}$ when Agent 0 adheres to his strategy in y. We use I_{-0} to denote the set $\{1,\ldots,n\}$, that is, the set of all agents except for the system, which is Agent 0. Given a strategy profile $z = (z_i)_{i\in I}$, we use $(z_{-\{i,0\}}, y_i, y_0)$ to denote the strategy profile where all agents but i and 0 follow z and agents i and 0 follow y_i and y_0, respectively. For $i \in I$, let φ_i be the objective of Agent i. For a solution concept $\gamma \in \{\mathrm{DS}, \mathrm{NASH}, \mathrm{SPE}\}$ and a strategy profile $y = (y_i)_{i\in I}$, the formula $\Psi^\gamma(y)$, expressing that the profile $(y_i)_{i\in I_{-0}}$ is a solution with respect to γ in $\mathcal{G}_{y_0}$, is defined as follows.

- $\Psi^{\mathrm{DS}}(y) := \bigwedge_{i\in I_{-0}} \forall z.\ (\varphi_i(z_{-0}, y_0) \rightarrow \varphi_i(z_{-\{i,0\}}, y_i, y_0))$.
- $\Psi^{\mathrm{NASH}}(y) := \bigwedge_{i\in I_{-0}} \forall z_i.\ (\varphi_i(y_{-i}, z_i) \rightarrow \varphi_i(y))$.
- $\Psi^{\mathrm{SPE}}(y) := \forall h. \bigwedge_{i\in I_{-0}} \forall z_i.\ ((\varphi_i(y_{-i}, z_i, h) \rightarrow (\varphi_i(y, h))$.

We can now state the existence of a solution to the rational-synthesis problem with input $\varphi_0, \ldots, \varphi_n$ by the closed formula $\Phi^\gamma := \exists (y_i)_{i\in I}.(\varphi_0((y_i)_{i\in I}) \wedge \Psi^\gamma((y_i)_{i\in I}))$. Indeed, the formula specifies the existence of a strategy profile whose outcome satisfies φ_0 and for which the strategies for the agents in I_{-0} constitute a solution with respect to γ in the game induced by y_0.

4.3 ESL Decidability

In order to solve the rational-synthesis problem we are going to use automata on infinite trees. Given a set D of directions, a D-*tree* is the set D^*. The elements in D^* are the *nodes* of the tree. The node ϵ is the root of the tree. For a node $u \in D^*$ and a direction $d \in D$, the node $u \cdot d$ is the *successor* of u with *direction* d. Given D and an alphabet Σ, a Σ-labeled D-tree is a pair $\langle D^*, \tau\rangle$ such that $\tau : D^* \rightarrow \Sigma$ maps each node of D^* to a letter in Σ.

An *alternating parity tree automaton (APT)* is a tuple $\mathcal{A} = \langle \Sigma, D, Q, \delta_0, \delta, \chi \rangle$, where Σ is the input alphabet, D is the directions set, Q is a finite set of states, δ_0 is the initial condition, δ is the transition relation and $\chi : Q \mapsto \{1, \ldots, k\}$ is the parity condition. The initial condition δ_0 is a positive boolean formula over Q specifying the initial condition. For example, $(q_1 \vee q_2) \wedge q_3$ specifies that the APT accepts the input tree if it accepts it from state q_3 as well as from q_1 or q_2. The transition function δ maps each state and letter to a boolean formula over $D \times Q$. Thus, as with δ_0, the idea is to allow the automaton to send copies of itself in different states. In δ, the copies are sent to the successors of the current node, thus each state is paired with the direction to which the copy should proceed. Due to the lack of space, we refer the reader to [11] for the definition of runs and acceptance.

Base ESL formulas, of the form $\psi(z, h)$, refer to exactly one strategy variable for each agent, and one history variable. The assignment for these variables can be described by a $(\Sigma \times \{\bot, \top\})$-labeled Σ-tree, where the Σ-component of the labels is used in order to describe the strategy profile π assigned to the strategy variable, and the $\{\bot, \top\}$-component of the labels is used in order to label the tree by a unique finite path corresponding to the history variable. We refer to a $(\Sigma \times \{\bot, \top\})$-labeled Σ-tree as a *strategy-history tree*. A node $u = d_0 d_1 \ldots d_k$ in a strategy-history tree $\langle \Sigma^*, \tau \rangle$ corresponds to a history of the play in which at time $0 \leq j \leq k$, the agents played as recorded in d_j. A label $\tau(u) = (\sigma_0, \ldots, \sigma_n, \dashv)$ of node u describes (1) for each agent i, an action σ_i that the strategy π_i advises Agent i to take when the history of the game so far is u, and (2) whether the node is along the path corresponding to the history. Among the $|\Sigma|$ successors of u in the strategy-history tree, only the successor $u \cdot \tau(u)$ corresponds to a scenario in which all the agents adhere to their strategies in the strategy profile described in $\langle \Sigma^*, \tau \rangle$. We say that a path ρ in $\langle \Sigma^*, \tau \rangle$ is *obedient* if for all nodes $u \cdot d \in \rho$, for $u \in \Sigma^*$ and $d \in \Sigma$, we have $d = \tau(u)$. Note that there is a single obedient path in every strategy tree. This path corresponds to the single play in which all agents adhere to their strategies. The $\{\bot, \top\}$ labeling is legal if there is a unique finite path starting at the root, all of whose node are marked with $\top$. Note that there is a single path in the tree whose prefix is marked by $\top$'s and whose suffix is obedient.

An ESL formula Ψ may contain several base formulas. Therefore, Ψ may contain, for each $i \in I$, several strategy variables in $\mathbb{Z}_i$ and several history variables in $\mathbb{H}$. For $i \in I$, let $\{z_i^1, \ldots, z_i^{m_i}\}$ be the set of strategy variables in $\Psi \cap \mathbb{Z}_i$. Recall that each strategy variable $z_i^j \in \mathbb{Z}_i$ corresponds to a strategy $\pi_i^j : \Sigma^* \to \Sigma_i$. Let $\{h_1, \ldots, h_m\}$ be the set of history variables in Ψ. Recall that each history variable h corresponds to a word in Σ^*, which can be seen as a function $w_h : \Sigma^* \to \{\top, \bot\}$ labeling only that word with $\top$'s. Thus, we can describe an assignment to all the variables in Ψ by a Υ-labeled Σ-tree, with $\Upsilon = \Sigma_0^{m_0} \times \Sigma_1^{m_1} \times \cdots \times \Sigma_n^{m_n} \times \{\bot, \top\}^m$.

We solve the rational synthesis problem using tree automata that run on Υ-labeled Σ-trees. Note that the specification of rational synthesis involves an external quantification of a strategy profile. We construct an automaton $\mathcal{U}$ that accepts all trees that describe a strategy profile that meets the desired solution. A witness to the nonemptiness of the automaton then induces the desired strategies.

We define $\mathcal{U}$ as an APT. Consider an ESL formula $\psi(z, h)$. Consider a strategy tree $\langle \Sigma^*, \tau \rangle$. Recall that ψ should hold along the path that starts at the root of the tree, goes through h, and then continues to *outcome*$(z)_h$. Thus, adding history variables to strategy logic results in a *memoryful logic* [21], in which LTL formulas have to be evaluated not along a path that starts at the present, but along a path that starts at the root and goes through the present. The memoryful semantics imposes a real challenge on the decidability problem, as one has to follow all the possible runs of a nondeterministic automaton for ψ, which involves a satellite implementing the subset construction of this automaton [21]. Here, we use instead the $\{\bot, \top\}$-component of the label of τ.

The definition of the APT$\mathcal{A}_\Psi$ for $[\![\Psi]\!]_\mathcal{G}$ works by induction on the structure of Ψ. At the base level, we have formulas of the form $\psi(z, h)$, where ψ is an LTL formula, z is a strategy profile, and h is a history variable. The constructed automaton then has three tasks. The first task is to check that the $\{\bot, \top\}$ labeling is legal; i.e. there is a unique path in the tree marked by $\top$'s. The second task is to detect the single path that goes through h and continues from h according to the strategy profile z. The third task is to check that this path satisfies ψ. The inductive steps then built on APT complementation, intersection, union and projection [26]. In particular, as in strategy logic, quantification over a strategy variable for agent i is done by "projecting out" the corresponding Σ_i label from the tree. That is, given an automaton $\mathcal{A}$ for Ψ, the automaton for $\exists z_i.\Psi$ ignores the Σ_i component that refers to z_i and checks $\mathcal{A}$ on a tree where this component is guessed. The quantification over history variables is similar. Given an automaton $\mathcal{A}$ for Ψ the automaton for $\exists h.\Psi$ ignores the $\{\bot, \top\}$ part of the label that corresponds to h and checks $\mathcal{A}$ on a tree where the $\{\bot, \top\}$ part of the label is guessed.

Theorem 4.1 *Let Ψ be an ESL formula over $\mathcal{G}$. Let d be the alternation depth of Ψ. We can construct an APT $\mathcal{A}_\Psi$ such that $\mathcal{A}_\Psi$ accepts $[\![\Psi]\!]_\mathcal{G}$ and its emptiness can be checked in time $(d+1)$-EXPTIME in the size of Ψ.*

4.4 Solving Rational Synthesis

We can now reduce rational-synthesis to APT emptiness.

Theorem 4.2 *The LTL rational-synthesis problem is 2EXPTIME-complete for the solution concepts of dominant strategy, Nash equilibrium, and subgame-perfect equilibrium.*

Proof: We have shown in Section 4.2 that the rational-synthesis problem for $\gamma \in \{\text{DS}, \text{NASH}, \text{SPE}\}$ can be specified by an ESL formula Φ^γ with one alternation. It follows from Theorem 4.1 that we can construct an APT accepting $[\![\Phi^\gamma]\!]_\mathcal{G}$ (where $\mathcal{G}$ is as defined in Section 3) whose emptiness can be solved in 2EXPTIME. Hence, the problem is in 2EXPTIME.

Hardness in 2EXPTIME follows easily from the 2EXPTIME-hardness of LTL synthesis [34]. Indeed, synthesis against a hostile environment can be reduced to rational synthesis against an agent whose objective is *true*. □

Remark 4.3 In the above we have shown how to solve the problem of rational synthesis. It is easy to extend our algorithm to solve the problem of *rational control*, where one needs to control a system in a way it would satisfy its specification assuming its environment consists of rational agents whose objectives are given. Technically, the control setting induces the game to start with, thus the strategy trees are no longer Σ-trees, and rather they are $(S \times \Sigma)$-trees, where S is the state space of the system we wish to control. ⌟

5 Solution in the Multi-Valued Setting

As discussed in Section 1, classical applications of game theory consider games with quantitative payoffs. The extension of the synthesis problem to the rational setting calls also for an extension to the quantitative setting. Unfortunately, the full quantitative setting is undecidable already in the context of model checking [3]. In this section we study a decidable fragment of the quantitative rational synthesis problem: the payoffs are taken from *finite De-Morgan lattices*. A lattice $\langle A, \leq \rangle$ is a partially ordered set in which every two elements $a, b \in A$ have a least upper bound (a *join* b, denoted $a \vee b$) and a greatest lower bound (a *meet* b, denoted $a \wedge b$). A lattice is *distributive* if for every $a, b, c \in A$, we have $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$. De-Morgan lattices are distributive lattices in which every element a has a unique complement element $\neg a$ such that $\neg\neg a = a$, De-Morgan rules hold, and $a \leq b$ implies $\neg b \leq \neg a$. Many useful payoffs are taken from finite De-Morgan lattices: all payoffs that are linearly ordered, payoffs corresponding to subsets of some set, payoffs corresponding to multiple view-points, and more [15, 16].

We specify qualitative specifications using the temporal logic *latticed LTL* (LLTL, for short), where the truth value of a specification is an element in a lattice. For a strategy profile π and an LLTL objective φ_i of Agent i, the payoff of Agent i in π is the truth value of φ_i in *outcome*(π). A synthesizer would like to find a profile π in which *payoff*$_0(\pi)$ is as high as possible. Accordingly, we define the latticed rational synthesis as follows.

Definition 5.1 (Latticed Rational Synthesis) Consider a solution concept γ. The problem of latticed rational synthesis (with solution concept γ) is to return, given LLTL formulas $\varphi_0, \ldots, \varphi_n$ and a lattice value $v \in$ Ł, a strategy profile $\pi = \langle \pi_0, \pi_1, \ldots, \pi_n \rangle \in \Pi_0 \times \Pi_1 \times \cdots \times \Pi_n$ such that (a) *payoff*$_0(\pi) \geq v$ and (b) the strategy profile $\langle \pi_1, \ldots, \pi_n \rangle$ is a solution in the game $\mathcal{G}_{\pi_0}$ with respect to the solution concept γ. ⌟

In the Boolean setting, we reduced the rational-synthesis problem to decidability of ESL. The decision procedure for ESL is based on the automata-theoretic approach, and specifically on APT's. In the lattice setting, automata-theoretic machinery is not as developed as in the Boolean case. Consequently, we restrict attention to LLTL specifications that can be translated to deterministic latticed Büchi word automata (LDBW), and to the solution concept of Nash equilibrium.[6]

[6] A *Büchi* acceptance conditions specifies a subset F of the states, and an infinite sequence of states satisfies the condition if it visits F infinitely often. A *generalized Büchi condition* specifies several such sets, all of which should be visited infinitely often.

An LDBW can be expanded into a deterministic latticed Büchi tree automata (LDBT), which is the key behind the analysis of strategy trees. It is not hard to lift to the latticed setting almost all the other operations on tree automata that are needed in order to solve rational synthesis. An exception is the problem of emptiness. In the Boolean case, tree-automata emptiness is reduced to deciding a two-player game [12]. Such games are played between an $\vee$-player, who has a winning strategy iff the automaton is not empty (essentially, the $\vee$-player chooses the transitions with which the automaton accepts a witness tree), and a $\wedge$-player, who has a winning strategy otherwise (essentially, the $\wedge$-player chooses a path in the tree that does not satisfy the acceptance condition). A winning strategy for the $\vee$-player induces a labeled tree accepted by the tree automaton.

In latticed games, deciding a game amounts to finding a lattice value l such that the $\vee$-player can force the game to computations in which his payoff is at least l. The value of the game need not be achieved by a single strategy and algorithms for analyzing latticed games consider values that emerge as the join of values obtained by following different strategies [16, 36]. A labeled tree, however, relates to a single strategy. Therefore, the emptiness problem for latticed tree automata, to which the latticed rational synthesis is reduced, cannot be reduced to solving latticed games. Instead, one has to consider the *single-strategy* variant of latticed games, namely the problem of finding values that the $\vee$-player can ensure by a single strategy. We address this problem below.

Theorem 5.2 *Consider a latticed Büchi game G. Given a lattice element l, we can construct a Boolean generalized-Büchi game G_l such that the $\vee$-player can achieve value greater or equal l in G using a single strategy iff the $\vee$-player wins in G_l. The size of G_l is bounded by $|G| \cdot |Ł|^2$ and G_1 has at most $|Ł|$ acceptance sets.*

Using Theorem 5.2, we can solve the latticed rational synthesis problem in a fashion similar to the one we used in the Boolean case. We represent strategy profiles by Σ-labeled Σ-trees, and sets of profiles by tree automata. We construct two Boolean generalized-Büchi tree automata. The first, denoted $\mathcal{A}_0$, for the language of all profiles π in which $payoff_0(\pi) \geq v$, and the second, denoted $\mathcal{A}_N$, for the language of all Nash equilibria. The intersection of $\mathcal{A}_0$ and $\mathcal{A}_N$ then contains all the solutions to the latticed rational synthesis problem. Thus, solving the problem amounts to returning a witness to the nonemptiness of the intersection, and we have the following.

Theorem 5.3 *The latticed rational-synthesis problem for objectives in LDBW and the solution concept of Nash equilibrium is in EXPTIME.*

We note that the lower complexity with respect to the Boolean setting (Theorem 4.2) is only apparent, as the objectives are given in LDBWs, which are less succinct than LLTL formulas [15, 20].

6 Discussion

We introduced *rational synthesis* — synthesizing a system that functions in a rational environment. As in traditional synthesis, one cannot control the agents that constitute the environment. Unlike traditional synthesis, the agents have objectives, we can suggest a strategy for each agent, and we can assume that rational agents follow strategies they have no incentive to deviate from.

The solution of the rational synthesis problem relies on an extension of strategy logic [7]. The modularity of our solution separates the game-theoretic considerations and the synthesis technique. Indeed our technique can be applied to any solution concept that can be expressed in extended strategy logic. We show that for the common solution concepts of dominant strategies equilibrium, Nash equilibrium, and subgame perfect equilibrium, rational synthesis has the same complexity as traditional synthesis The versatility of the extended logic enables many extensions of the setting. For example, one can associate different solutions concepts with different sub-specifications. In particular, it is often desirable in practice to ensure that some properties of the system hold regardless of the rationality of the agents. This can be done by letting the specifier specify, in addition to φ_0, also an LTL formula φ'_0 (typically $\varphi_0 \rightarrow \varphi'_0$) that should be satisfied in the traditional synthesis interpretations, namely in all environments.

References

[1] B. Aminof, O. Kupferman, and R. Lampert. Reasoning about online algorithms with weighted automata. In *Proc. 20th SODA*, pages 835–844, 2009.

[2] P.C. Attie, A. Arora, and E.A. Emerson. Synthesis of fault-tolerant concurrent programs. *TOPLAS*, 26:128–185, 2004.

[3] A. Chakrabarti, K. Chatterjee, T.A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *Proc. 13th CHARME*, LNCS 3725, pages 50–64, 2005.

[4] K. Chatterjee, L. Doyen, and T. Henzinger. Quantative languages. In *Proc. 17th CSL*, LNCS 5213, pages 385-400, 2008.

[5] K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th CONCUR*, LNCS 5201, pages 147–161, 2008.

[6] K. Chatterjee, T. Henzinger, and M. Jurdzinski. Games with secure equilibria. *Theoretical Computer Science*, 2006.

[7] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *18th CONCUR*, LNCS, pages 59–73, 2007.

[8] K. Chatterjee and T.A. Henzinger. Assume-guarantee synthesis. In *Proc. 13th TACAS*, LNCS 4424, pages 261–275, 2007.

[9] K. Chatterjee, R. Majumdar, and M. Jurdzinski. On Nash equilibria in stochastic games. In *Proc. 13th CSL*, LNCS 3210, pages 26–40, 2004.

[10] A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35, 1963.

[11] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, LNCS 2500, 2002.

[12] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th STOC*, pages 60–65, 1982.

[13] A. Gurfinkel and M. Chechik. Multi-valued model-checking via classical model-checking. In *14th CONCUR*, LNCS 2761, pages 263–277, 2003.

[14] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *Proc 17th CAV*, LNCS 3576, pages 226–238, 2005.

[15] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th VMCAI*, LNCS 4349, pages 199 – 213, 2007.

[16] O. Kupferman and Y. Lustig. Latticed simulation relations and games. In *5th ATVA*, LNCS 4762, pages 316–330, 2007.

[17] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc 18th CAV*, LNCS 4144, pages 31–44, 2006.

[18] O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *Proc. 16th LICS*, pages 389–398, 2001.

[19] O. Kupferman and M.Y. Vardi. Safraless decision procedures, In *Proc. 46th FOCS*, pages 531–540, 2005.

[20] O. Kupferman and M.Y. Vardi. From linear time to branching time. *TOCL*, 6(2):273–294, 2005.

[21] O. Kupferman and M.Y. Vardi. Memoryful branching-time logics. In *Proc. 21st LICS*, pages 265–274, 2006.

[22] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th STOC*, pages 224–233, 1998.

[23] Y. Lustig and M.Y. Vardi. Synthesis from component libraries. In *Proc. 12th FOSSACS*, LNCS 5504, pages 395–409, 2009.

[24] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[25] R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. In *16th CONCUR*, LNCS 3653, pages 562–576, 2005.

[26] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[27] J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, 1950.

[28] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st STOC*, pages 129–140, 1999.

[29] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[30] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

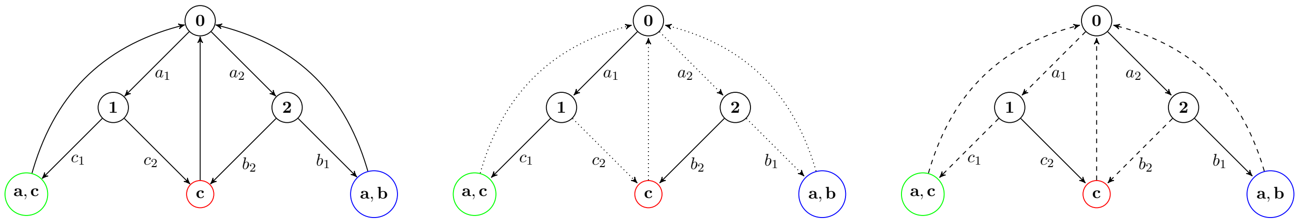

Figure 1: A game, two Nash equilibria and one subgame-perfect equilibrium.

[31] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57, 1977.

[32] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

[33] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[34] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.

[35] R. Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4(1):25–55, March 1975.

[36] S. Shoham and O. Grumberg. Multi-valued model checking games. In *Proc. 3rd ATVA*, LNCS 3707, pages 354–369, 2005.

[37] M. Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *Proc. 26th FSTTCS*, LNCS 4337, pages 212–223, 2006.

[38] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

A Proofs

A.1 Proof of Theorem 4.1

The construction proceeds by induction on the structure of Ψ. Note that while the APT is defined with respect to Υ-labeled Σ-trees, a base formula $\psi(z, h)$ focuses on a $(\Sigma \times \{\bot, \top\})$ projection of the label (the one assigning values to the variables in z and h). We describe here in detail the base case, where $\Psi = \psi(z, h)$. The case where $\Psi = \psi(h)$ can be derived from the case $\Psi = \psi(z, h)$ by checking in addition that only the root is labeled $\top$. The cases Ψ is of the form $\Psi_1 \vee \Psi_2, \neg\Psi_1, \exists z_i.\Psi_1$, and $\exists h.\Psi_1$ follow from the closure of APTs to union, complementation, and projection.

The complexity analysis follows from the fact that the automaton for $\psi(z, h)$ is exponential in ψ, and each sequence of quantifiers that increases the alternation depth by one, involves an exponential blow up in the state space and a polynomial blow up in the index [26]. Thus, the number of states in $\mathcal{A}_\Psi$ is $(d+1)$-exponential in Ψ and the index of $\mathcal{A}_\Psi$ is polynomial (of degree d) in Ψ, where d is the alternation depth of Ψ. Since the projection operation results in a nondeterministic (rather than an alternating) tree automaton, the emptiness check when the last operation is projection does not involve an additional exponential blow up.

Let $\Psi = \psi(z, h)$. Given an LTL formula ψ, one can construct an APT $\mathcal{U}_\psi$ with $2^{O(|\psi|)}$ states and index 3 such that $\mathcal{U}_\psi$ accepts all trees all of whose paths satisfy ψ [38]. Let $\mathcal{U}_\psi = \langle \Sigma, \Sigma, Q, \delta^0, \delta, \chi \rangle$. For the first and second tasks we use four states q_{his}, q_{fut}, q_{acc}, and q_{rej}. The automaton $\mathcal{A}_\Psi$ starts by sending two copies, one at the initial state of $\mathcal{U}_\psi$ and one at q_{his}. The copy in state q_{his} follows the *history*, i.e. the path marked with $\top$ labels. When it reads a node with a $\bot$ label, marking that the history ends and the *future* begins, it moves to the state q_{fut}. From the state q_{fut}, this copy checks that the agents adhere to the strategy. If a violation of the strategy is detected, the copy concludes that ψ need not be evaluated along the path it traversed and moves to q_{acc}. If another $\top$ has been read, the copy conclude that the $\{\top, \bot\}$-component is illegal and moves to q_{rej}. Formally, $\mathcal{A}_\Psi = \langle \Sigma \times \{\bot, \top\}, \Sigma, Q \cup \{q_{his}, q_{fut}, q_{acc}, q_{rej}\}, \delta^0 \wedge q_{his}, \nu, \chi' \rangle$, where for every $\sigma \in \Sigma$, $\dashv \in \{\bot, \top\}$, the transition function ν is defined as follows. Note that the alphabet of $\mathcal{A}_\Psi$ is Υ, rather than $\Sigma \times \{\bot, \top\}$. Since, however, base formulas refer to a single strategy profile and history variable, we restrict attention to the relevant components of the input alphabet.

- $\nu(q_{acc}, \langle\sigma, \dashv\rangle) = q_{acc}$ and $\nu(q_{rej}, \langle\sigma, \dashv\rangle) = q_{rej}$.
- For every $q \in Q$, we have $\nu(q, \langle\sigma, \dashv\rangle) = \delta(q, \sigma)$.
- $\nu(q_{his}, \langle\sigma, \bot\rangle) = \bigwedge_{d\in\Sigma}(d, q_{fut})$.
- $\nu(q_{his}, \langle\sigma, \top\rangle) = \bigvee_{d\in\Sigma}((d, q_{his}) \wedge \bigwedge_{d'\in\Sigma\setminus\{d\}}(d', q_{acc}))$.
- $\nu(q_{fut}, \langle\sigma, \top\rangle) = \bigwedge_{d\in\Sigma}(d, q_{rej})$.
- $\nu(q_{fut}, \langle\sigma, \bot\rangle) = \bigwedge_{d\in\Sigma}(\bigwedge_{d=\sigma}(d, q_{fut}) \wedge \bigwedge_{d\neq\sigma}(d, q_{acc}))$.

The parity condition χ' is such that $\chi'(q) = \chi(q)$ for every $q \in Q$ and for the other states we have $\chi'(q_{acc}) = 0$, $\chi'(q_{rej}) = 1$, $\chi'(q_{his}) = 1$, and $\chi'(q_{fut}) = 0$.

It is easy to see that a tree $\langle\Sigma^*, \tau\rangle$ is accepted by $\mathcal{A}_\Psi$ iff there is a word $w \in \Sigma^*$ such that for every prefix u of w the node u is labeled $\langle\sigma, \top\rangle$ for some $\sigma \in \Sigma$ and *outcome*$(\tau)_w \models \psi$. The number of states of $\mathcal{A}_\Psi$ is exponential in φ and its index is 3.

A.2 Proof of Theorem 5.2

Consider a lattice Ł. An element $x \in$ Ł is *join irreducible* if for all $y, z \in$ Ł we have $x \leq y \vee z$ implies $x \leq y$ or $x \leq z$. Given l, we define the game G_l as follows. Let $X_l = \{x \in JI(\text{Ł}) \mid x \leq l\}$ be the set of join irreducible elements smaller then l. By Birkhoff's representation theorem, a strategy ensures a value greater or equal l iff for every $x \in X_l$ the strategy ensures a value greater or equal x.

By the analysis in [16], the value of a latticed play p in a game G can be decomposed into three values: the acceptance value $acc(p)$, and two values $r^\vee$ and $r^\wedge$ that have to do with value relinquished by the $\vee$-player and the $\wedge$-player during the play, respectively. Furthermore, the values $r^\vee$ and $r^\wedge$ are the limits of the sequences $\{r_i^\vee\}_{i=0}^\infty$ and $\{r_i^\wedge\}_{i=0}^\infty$ where for every $i \geq 0$ the values of $r_i^\vee$ and $r_i^\wedge$ depend on the i-long prefix of the play p.

The idea underlying the reduction is to consider a Boolean game in which the values from the latticed game are made explicit by the structure of the game graph. Formally, for a latticed game $G = \{V, E\}$ with $V = V_\vee \cup V_\wedge$ and an Ł-Büchi condition $F \in \text{Ł}^V$, we define a Boolean generalized-Büchi game $G'_l = \{V', E'\}$ as follows. The state space $V' = V \times \text{Ł} \times \text{Ł}$ is such that in a state $(u, x, y) \in V \times \text{Ł} \times \text{Ł}$, we have that u stands for a state in G, the value x stands for the $\vee$-relinquished value $r_i^\vee$, and the value y stands for the $\wedge$-relinquished value $r_i^\wedge$.

Let $G = \{V, E\}$ be a latticed game with an Ł-Büchi condition $F \in \text{Ł}^V$ and initial vertex $v_0 \in V$. The *simplification* of G for $l \in$ Ł, denoted G'_l, is the Boolean game $G'_l = \{V', E'\}$ where $V' = V \times \text{Ł} \times \text{Ł}$, and the partition of V' and E' is defined as follows. First, $V'_\vee = V_\vee \times \text{Ł} \times \text{Ł}$ and $V'_\wedge = V_\wedge \times \text{Ł} \times \text{Ł}$ (note that even though G'_l is Boolean, we keep the names $\vee$-player and $\wedge$-player). The initial vertex is $\langle v_0, \top, \bot\rangle$. In order to define the edges we introduce the following notation. For $u, u' \in V$ and $x, y \in$ Ł the u'*-successor of* $\langle u, x, y\rangle$ is $\langle u', x', y'\rangle$, where either $u \in V_\vee$ in which case $x' = x \wedge (E(u, v) \vee y)$ and $y' = y$, or $u \in V_\wedge$ in which case $x' = x$ and $y' = y \vee (E(u, v) \wedge x)$. Now, $E' = \{(\langle u, x, y\rangle, \langle u', x', y'\rangle) \mid \langle u', x', y'\rangle$ is the u'-successor of $\langle u, x, y\rangle\}$.

It is left to define the generalized-Büchi condition. In order to ensure the value $l \in$ Ł, the $\vee$-player must "collect" every value $x \in X_l$ either as a value relinquished by the $\wedge$-player or by the acceptance value acc. For that, we define, for each $x \in X_l$ a set F_x in the generalized-Büchi condition. We define $F_x = (V \times \text{Ł} \times \{y \in \text{Ł} \mid y \geq x\}) \cup (\{u \in V \mid F(u) \geq x) \setminus V \times \{y \in \text{Ł} \mid y \not\geq x\} \times \text{Ł})$. The first component states for states in which the $\wedge$-player relinquished x, and the second component stands for states in which both the acceptance value is greater then x and x was not relinquished by the $\vee$-player in the past. Now, the generalized-Büchi acceptance condition is $F' = \{F_x \mid x \in X_l\}$.

Assume first there exists a single strategy π in G ensuring value greater or equal l. Every strategy π for G (for either player) induces a strategy π' in G'_l in which $\pi'(\langle u_0, x_0, y_0\rangle, \ldots, \langle u_n, x_n, y_n\rangle)$ is the $\pi(u_0, \ldots, u_n)$-successor of $\langle u_n, x_n, y_n\rangle$. Consider a $\vee$-player strategy π that ensures value greater or equal l. We show that π' is winning in G'_l. It is not hard to see that a play $p' = \langle u_0, x_0, y_0\rangle \ldots \langle u_n, x_n, y_n\rangle \ldots$ consistent with π' corresponds to a play $p = u_0 \ldots u_n \ldots$ consistent with π. Furthermore, for every $i \geq 0$, we have $x_i = r_i^\vee$ and $y_i = r_i^\wedge$. Since π ensures value l in G, the value of p is greater or equal l, and therefore, for every join irreducible $x \in V_x$ we have $val(p) \geq x$. Thus, either there exists an index i from which $r_i^\wedge \leq x$ or for infinitely many i's we have $F(u_i) \geq x$ and $r_i^\vee \geq x$. Both cases imply that the set F_x is traversed infinitely often. Thus the play p' is winning for the $\vee$-player in G'_l.

Assume now that π' is a winning strategy for the $\vee$-player in G'_l. The strategy π' induces a $\vee$-player strategy in G in the following way: Every prefix of a play $p = u_0, u_1, \ldots, u_n$ in G induces the prefix of a play $p' =$

$\langle u_0, \top, \bot\rangle, \langle u_0, x_1, y_1\rangle, \ldots, \langle u_n, x_n, y_n\rangle$, where for every $i > 0$, we have that $\langle u_i, x_i, y_i\rangle$ is the u_i-successor of $\langle u_{i-1}, x_{i-1}, y_{i-1}\rangle$. We define $\pi(p)$ to be the state u for which $\pi'(p')$ is $\langle u, x, y\rangle$. It is not hard to see that for a play p in G consistent with π, and for every $i \geq 0$, we have $x_i = r_i^\vee$ and $y_i = r_i^\wedge$. As π' is winning in G'_l, we get that for every $x \in X_l$ we have $val(p) \geq x$, and therefore $val(p) \geq l$.

A.3 Proof of Theorem 5.3

Approaching the problem in a fashion similar to the one we used in the Boolean case, we represent strategy profiles by Σ-labeled Σ-trees, and sets of profiles by tree automata. We construct two Boolean tree automata. The first, denoted $\mathcal{A}_0$, for the language of all profiles π in which $\textit{payoff}_0(\pi) \geq v$, and the second, denoted $\mathcal{A}_N$, for the language of all Nash equilibria. It is not hard to see that the intersection of $\mathcal{A}_0$ and $\mathcal{A}_N$ contains all the solutions to the latticed rational synthesis problem. Thus, solving the problem amounts to returning a witness to the nonemptiness of the intersection.

For the purposes of complexity analysis, we denote by s_i the size of the LDBW for the i-th agent specification, by $s = max\{s_i\}$ the maximal s_i, and by $m = |\text{Ł}|$ the size of the lattice.

We first construct $\mathcal{A}_0$. As in the Boolean case, we first construct an LDBT $\mathcal{A}'_0$ that maps a strategy profile π to $\textit{payoff}_0(\pi)$. Using Theorem 5.2, we can construct from $\mathcal{A}'_0$ the required Boolean tree automaton $\mathcal{A}_0$. To see how, note that the generalized-Büchi game involved has a very uniform structure. From every $\vee$-vertex, the $\vee$-player has exactly one choice associated with each $\sigma \in \Sigma$. (This property is inherited from the latticed game which in turn inherits it from the fact that the alphabet of $\mathcal{A}'_0$ is Σ.) A similar property holds for the $\wedge$-player (this property is inherited from the fact that $\mathcal{A}'_0$ runs on Σ-trees). Therefore, the generalized-Büchi game can be reduced, using standard techniques, to a generalized-Büchi tree automaton $\mathcal{A}_0$. The size of $\mathcal{A}'_0$ is $s_0 \cdot m^2$ and the number of acceptance sets in its generalized Büchi condition is bounded by m.

We now turn to build an automaton for Nash equilibria $\mathcal{A}_N$. We construct $\mathcal{A}_N$ as an intersection of n automata $\{\mathcal{A}^i_N\}_{i=1}^n$, where the language of $\mathcal{A}^i_N$ is the set of the profiles that satisfy $\textit{payoff}_i(\pi_{-i}, \pi'_i) \leq \textit{payoff}_i(\pi)$. By Birkhoff's representation theorem, an equivalent criteria would be that for every join irreducible element $j \in JI(\text{Ł})$, we have $\textit{payoff}_i(\pi_{-i}, \pi'_i) \geq j \rightarrow \textit{payoff}_i(\pi, \varphi_i) \geq j$. Given LDBW for φ_i, it is not hard to construct LDBTs for $\textit{payoff}_i(\pi_{-i}, \pi'_i)$ and $\textit{payoff}_i(\pi)$. For every join irreducible element $j \in JI(\text{Ł})$ we would like to make sure that $\textit{payoff}_i(\pi_{-i}, \pi'_i) \geq j \rightarrow \textit{payoff}_i(\pi, \varphi_i) \geq j$. To that end, we use the construction of the Boolean game $\mathcal{G}_\top$ in the proof of Theorem 5.2. Recall that in the game $\mathcal{G}_\top$, the value x is obtained by a single strategy iff the acceptance set F_x is visited infinitely often. Thus, for a specific agent $i \leq n$, and a join irreducible element $j \in JI(L)$, we can construct a Boolean Büchi tree automaton $\mathcal{B}^i_j$, of size $O(s_i \cdot m^2)$, that accepts exactly the trees encoding profiles for which $\textit{payoff}_i(\pi, \varphi_i) \geq j$. In a similar way, we can construct a tree automaton C^i_j, of similar size, that accepts trees encoding profiles for which $\textit{payoff}_i(\pi_{-i}, \pi'_i) \geq j$. Combining B^i_j and C^i_j we can get a Streett automaton A^i_j that accepts profiles for which $\textit{payoff}_i(\pi_{-i}, \pi'_i) \geq j \rightarrow \textit{payoff}_i(\pi, \varphi_i) \geq j$. The size of A^i_j is $O(s_i^2 \times m^4)$, and it has one Streett pair. Note that for a fixed i, the automata A^i_j share their structure and only differ in the acceptance condition. Therefore, for a fixed $i \leq n$, we can construct an automaton A^i_N, of size $O(s_i^2 \cdot m^4)$ and with $O(m)$ pairs, that accepts profiles for which $\textit{payoff}_i(\pi_{-i}, \pi'_i) \geq j \rightarrow \textit{payoff}_i(\pi, \varphi_i) \geq j$ for every join irreducible element $j \in JI(\text{Ł})$. By intersecting the automata $\mathcal{A}^i_N$ we get an automaton $\mathcal{A}_N$ of size $(s \cdot m)^{O(n)}$, with $O(m \cdot n)$ pairs.

The intersection of $\mathcal{A}_0$ and $\mathcal{A}_N$ is a Streett automaton of size $(s \cdot m)^{O(n)}$ and with $O(m \cdot n)$ pairs. Its emptiness can then be checked in time $(s \cdot m)^{O(m \cdot n^2)}$ [22], and we are done.