Strategy Composition in Compositional Games

Marcus Gelderie*

RWTH Aachen, Lehrstuhl für Informatik 7,
Logic and Theory of Discrete Systems,
D-52056 Aachen
gelderie@automata.rwth-aachen.de

Abstract. When studying games played on finite arenas, the arena is given explicitly, hiding the underlying structure of the arena. We study games where the global arena is a product of several smaller, constituent arenas. We investigate how these "global games" can be solved by playing "component games" on the constituent arenas. To this end, we introduce two kinds of products of arenas. Moreover, we define a suitable notion of strategy composition and show how, for the first notion of product, winning strategies in reachability games can be composed from winning strategies in games on the constituent arenas. For the second kind of product, the complexity of solving the global game shows that a general composition theorem is equivalent to proving PSPACE = EXPTIME.

1 Introduction

Infinite games with ω-regular winning conditions have been studied extensively over the past decades [1–5]. This research has been most successful in establishing results about solving ω-regular games on an "abstract" arena. A fundamental open problem, which is of intrinsic interest in the area of automated synthesis, is to exploit the compositional structure of an arena to derive a compositional representation of a winning strategy. For instance, if an arena is viewed as a product of several smaller transition systems, is it possible to lift this structure to strategies in games on this arena?

The classical results on ω-regular games depend on the representation of a winning strategy by an automaton. None of these results allows to transfer a given composition of an arena into a composition of automata in such a way that a winning strategy is implemented. Since there is no lack of methods for composing automata (for example, the cascade product), it rather seems that automata are too "coarse" a tool to capture this compositional structure.

We study the compositional nature of winning strategies in games played on products of arenas. Products of arenas can be defined in a variety of ways (see e.g. [6]). As a first step towards a compositional approach to synthesis, we restrict ourselves to two notions, *parallel* and *synchronized* product. Our notion of strategy composition relies on a Turing machine based model for strategy

* Supported by DFG research training group 1298, "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems" (AlgoSyn).

F.V. Fomin et al. (Eds.): ICALP 2013, Part II, LNCS 7966, pp. 263–274, 2013.

representation, called a *strategy machine*. Using this model, we show how winning strategies in reachability games can be composed from winning strategies in games over the constituent factors of the overall product arena. We study the complexity of such a composition: its size, its runtime and the computational complexity of finding it. This entails a study of the complexity of deciding who wins the game.

Compositionality in an arena is closely linked to a succinct representation of that arena. Likewise, composing a winning strategy from smaller winning strategies may yield a much smaller representation for that strategy. Finding succinct transition systems from specifications was studied in [7]. The authors consider the problem of finding a succinct representation of a model for a given CTL formula. They show that such succinct models are unlikely to exist in general.

Transition systems which are obtained by "multiplying" smaller transition systems have also been studied in [8]. The authors consider the problem of model checking such systems. They show the model checking problem for such systems to be of high complexity for various notions of behavioral specification and model checking problem.

Strategy machines were introduced in [9] (the model has been studied in a different setting in [10]). They allow for a broader range of criteria by which to compare strategies. Being based on Turing machines, strategy machines allow, for instance, to investigate the "runtime" of a strategy and to quantify and compare "dynamic" memory (the tape content) and "static memory" (the control states).

The complexity of deciding the winner of a game has been subject to extensive research in the case of games on an abstract arena [11–13]. These complexity results depend on the size of the abstract arena. We investigate the complexity of deciding the winner based on a composite representation of the arena.

Our paper is structured as follows: We first define two notions of product of arenas, the *parallel product* and the *synchronized product*. The games we study are played on arenas that are composed from smaller arenas using these two operators. Having defined the notion of arena composition, we define strategy machines and use them to introduce our notion of strategy composition. Subsequently, we study reachability games. We do this separately for the parallel product and the synchronized product. To this end, we first introduce two natural ways of defining a reachability condition on a composite arena, *local* and *synchronized reachability*. For the parallel product we obtain a compositionality theorem for both local and synchronized reachability. For the synchronized product we show that deciding the game is EXPTIME complete. From this we deduce that finding a general composition theorem is equivalent to showing EXPTIME = PSPACE.

2 Games on Composite Arenas

An *arena* is a directed, bipartite graph $\mathcal{A} = (V, E)$. The partition of $\mathcal{A}$ is $V = V^{(0)} \uplus V^{(1)}$. Given $v \in V$ let $vE = \{v' \in V \mid (v, v') \in E\}$. We define two operators on arenas: the *parallel product* and the *synchronized product*. Let $\mathbb{B} = \{0, 1\}$.

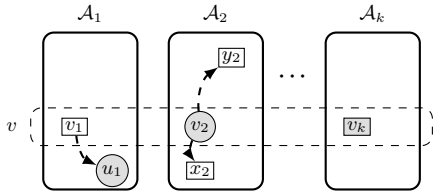
Definition 1 (Parallel Product). *Consider arenas* $\mathcal{A}_1, \dots, \mathcal{A}_k$ *with* $\mathcal{A}_i = (V_i, E_i)$. *The* parallel product $\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_k = (V, E)$ *is the arena given by*

- $V = \mathbb{B} \times \prod_{i=1}^{k} V_i$
- $E = \{\big((\sigma, v), (1-\sigma, v')\big) \mid \exists i\colon\ v_i \in V_i^{(\sigma)} \wedge (v_i, v_i') \in E_i \wedge \forall j \neq i\colon\ v_j = v_j'\}$
- $V^{(\sigma)} = \{(b, v) \in V \mid b = \sigma\}$

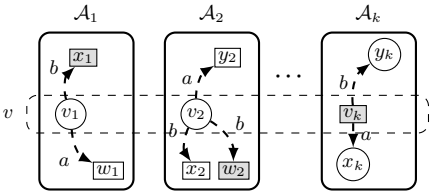
Note that the parallel product again gives a bipartite arena. Note furthermore that, given a vertex $(\sigma, v) \in \mathbb{B} \times \prod_i V_i$, the number of vertices $v_i \in V_i^{(0)}$ alternatingly increases and decreases by one along all paths starting in (σ, v). The number of components player 0 controls in each of his moves is given by:

$$\mathrm{rank}_0(\sigma, v) = |\{i \mid v_i \in V_i^{(0)}\}| + \sigma$$

Player 1 controls $\mathrm{rank}_1(\sigma, v) = k - \mathrm{rank}_0(\sigma, v) + 1$ components during his moves. For every $p \in \mathbb{B}$ we have $\mathrm{rank}_p(\sigma', v') = \mathrm{rank}_p(\sigma, v)$ for all (σ', v') reachable from (σ, v). If (σ, v) is clear from context, we thus simply write rank_p.



(a) Parallel Product: Edges are taken locally. The square player may move in, e.g., $\mathcal{A}_1$ but not in $\mathcal{A}_2$.



(b) Synchronized Product: Where transitions permit it, edges are taken globally. The circle player may choose transitions in $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_k$ is not affected.

Fig. 1. A vertex v in the parallel and synchronized product of (labeled) arenas $\mathcal{A}_1, \dots, \mathcal{A}_k$. The shaded vertices define a possible successor state of v.

To define the synchronized product we use *labeled arenas*. A labeled arena is a triple $\mathcal{A} = (V, \Delta, \Sigma)$ with $V = V^{(0)} \uplus V^{(1)}$ and $\Delta \subseteq \bigcup_{\sigma \in \mathbb{B}} V^{(\sigma)} \times \Sigma \times V^{(1-\sigma)}$ for some finite set Σ of *letters*.

Definition 2 (Synchronized Product). *Consider labeled arenas* $\mathcal{A}_1, \dots, \mathcal{A}_k$ *with* $\mathcal{A}_i = (V_i, \Delta_i, \Sigma_i)$. *The* synchronized product $\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_k = (V, \Sigma, \Delta)$ *is given by*

- $V = \mathbb{B} \times \prod_{i=1}^{k} V_i$
- $\Sigma = \bigcup_{i=1}^{k} \Sigma_i$
- $\big((\sigma, v), a, (1-\sigma, v')\big) \in \Delta$ *iff for all* i, *whenever* $a \in \Sigma_i$ *and* $v_i \in V_i^{(\sigma)}$, *then also* $(v_i, a, v_i') \in \Delta_i$ *and* $v_i = v_i'$ *otherwise.*

Remark 1. Neither the parallel product nor the synchronized product are associative in general. This is due to the fact that we absorb the information about whose turn it is into the arena. We do so for technical reasons. It is nonessential for the results.

In this paper we study ω-regular games. We assume the reader is familiar with the elementary theory of ω-regular games. For an introduction see [4, 5]. In the following, we recall some terminology. A *game* is a tuple $\mathbf{G} = (\mathcal{A}, W, v_0) = (\mathcal{A}, W)$ consisting of an arena $\mathcal{A} = (V, E)$ and a *winning condition* $W \subseteq V^\omega$ and an *initial vertex* v_0. We always assume that there is a designated initial vertex, even if we do not always list it explicitly. $\mathbf{G}$ is ω-regular if W is ω-regular. We denote the players by player 0 and player 1. A *play* in $\mathbf{G}$ is an infinite path $\pi = v_0 v_1 v_2 \cdots$ through $\mathcal{A}$, starting from v_0. On nodes in $V^{(0)}$ player 0 chooses the next vertex. Otherwise, player 1 chooses. The play is won by player 0 if $\pi \in W$. We denote the *winning set* of player σ by $\mathcal{W}^{(\sigma)} = \mathcal{W}^{(\sigma)}(\mathbf{G})$. The *attractor* for player p on a set F is denoted by $\mathrm{Attr}_p^{\mathcal{A}}(F)$ and defined as usual. It is the set of vertices from which p can enforce a visit to F.

To study games on composite arenas, we require some additional notation. Consider a game $\mathbf{G} = (\mathcal{A}, W)$ on a *composite arena* $\mathcal{A} = \mathcal{A}_1 * \cdots * \mathcal{A}_k$, with $* \in \{\|, \otimes\}$. We call $\mathcal{A}_1, \ldots, \mathcal{A}_k$ the *constituent* arenas of $\mathcal{A}$. A game $\mathbf{G}_i = (\mathcal{A}_i, W_i)$ for some $W_i \subseteq V_i^\omega$ is called a *component game*.

The winning condition W is necessarily given by means of some finite representation. In this paper we consider mainly reachability conditions, which are determined by a set $F \subseteq V$. A play π satisfies the reachability condition F if $\pi(i) \in F$ for some $i \in \mathbb{N} = \{0, 1, 2, \ldots\}$.

It is sometimes convenient to specify properties on a path in some logic. In this paper we use LTL to express temporal properties on paths. We again assume the reader is familiar with LTL (see [4, 5] for an introduction). We write $\psi \,\mathsf{U}\, \phi$ for the strict until (ϕ is true eventually, and, until then, ψ holds).

3 Strategy Machines and Strategy Composition

Classically, strategies are represented using (usually finite) automata. Automata are a state space view on a computational system. They abstract away from implementation details. This comes at the price of loosing information about important implementation aspects, such as runtime and space usage.

The abstract view of automata is sometimes coarser than required. To augment this view, in [9] *strategy machines* were introduced. Strategy machines are Turing machines with three designated tapes, the IO-tape, the computation tape and the memory tape. The semantics of a strategy machine can intuitively be described as follows. A vertex (encoded in binary) appears on the IO-tape. The strategy machine inspects the content of its memory tape and computes a new vertex (using all three tapes). The content of the memory tape is updated and the computation tape is cleared. The new vertex is written on the output tape and the process repeats. We now recall the definition from [9]. Let $\hat{\mathbb{B}} = \mathbb{B} \cup \{\#\}$.

Definition 3 (Strategy Machine). *A strategy machine is a deterministic 3-tape Turing machine* $\mathcal{M} = (Q, \mathbb{B}, \hat{\mathbb{B}}, q_I, q_O, \delta)$ *with*

- *a finite set* Q *of* states
- *tape alphabet* $\hat{\mathbb{B}}$ *and input alphabet* $\mathbb{B}$
- *two designated states,* q_I*, the* input state *and* q_O*, the* output state
- *a designated* IO-tape t_{IO}
- *a designated* memory tape t_M
- *a designated* computation tape t_C

The partial transition function $\delta\colon Q \times \hat{\mathbb{B}}^3 \dashrightarrow Q \times \hat{\mathbb{B}}^3 \times \{-1, 0, 1\}^3$ *satisfies*

- $\delta(q, b) \neq (q_I, b', d)$ *for all* $q \in Q$, $b, b' \in \hat{\mathbb{B}}^3$ *and* $d \in \{-1, 0, 1\}^3$.
- $\delta(q_O, b)$ *is undefined for all* $b \in \hat{\mathbb{B}}^3$.

Since the tape and input alphabets are always the same, we usually omit them in the list of components of a strategy machine.

We sketch the semantics of a strategy machine (a formal definition can be found in [9]). Configurations are defined as usual. An *iteration of* $\mathcal{M}$ is a sequence of configurations beginning with an *initial configuration* (with state q_I) and a *terminal configuration* (with state q_O). By definition of δ, q_I and q_O appear exactly at the beginning and at the end of an iteration. The iteration beginning in configuration c is unique (if it exists) and depends only on the input $t_{IO}(c)$ on the IO-tape and on the content $t_M(c)$ of the memory tape. We write c' for the unique terminal configuration reachable from c and we write (c, c') for the entire iteration. Its *length*, the number of computation steps, is denoted by $L(c, c')$.

Strategy machines are intended to implement functions on sequences of input. Let $\pi \in (\mathbb{B}^*)^\infty = (\mathbb{B}^*)^* \cup (\mathbb{B}^*)^\omega$, i.e. $\pi(i) \in \mathbb{B}^*$ for all $i \in \mathrm{dom}(\pi)$. Let $c_{\pi,0} = (q_I, \pi(0), \varepsilon, \varepsilon, 0, 0, 0)$. We define $c_{\pi,i+1}$ to be the configuration which inherits the memory tape content from the terminal configuration of the i-th iteration and has $\pi(i+1)$ as input on the IO-tape. Formally, $c_{\pi,i+1} = (q_I, \pi(i+1), t_M(c'_{\pi,i}), \varepsilon, 0, 0, 0)$. We also say that $(c_{\pi,i}, c'_{\pi,i})$ and $(c_{\pi,i+1}, c'_{\pi,i+1})$ are *compatible*. An iteration is *admissible* if it is of the form $(c_{\pi,i}, c'_{\pi,i})$ for some $\pi \in (\mathbb{B}^*)^\omega$. A sequence of iterations $(c_0, c'_0)(c_1, c'_1)\cdots$ which are compatible is called a *computation of* $\mathcal{M}$. Define $f_\mathcal{M}(\pi) = t_{IO}(c'_{\pi,0}) \cdot t_{IO}(c'_{\pi,1}) \cdot \cdots$. $f_\mathcal{M}$ maps strings from $(\mathbb{B}^*)^*$ to $(\mathbb{B}^*)^*$ (where, in our setting, elements from $\mathbb{B}^*$ are encoded vertices). We say $\mathcal{M}$ *implements* $f_\mathcal{M}$. We sometimes identify $\mathcal{M}$ with $f_\mathcal{M}$ and say, for instance, that $\mathcal{M}$ is a winning strategy.

One of the benefits of strategy machines is the complexity measures they offer to evaluate a strategy. We define the *latency* $T(\mathcal{M})$ and the *space requirement* $S(\mathcal{M})$ of a strategy machine $\mathcal{M}$ as follows:

$$T(\mathcal{M}) = \sup_{\pi \in (\mathbb{B}^*)^\omega} \sup_{n \in \mathbb{N}} L(c_{\pi,n}, c'_{\pi,n})$$

$$S(\mathcal{M}) = \sup_{\pi \in (\mathbb{B}^*)^\omega} \sup_{n \in \mathbb{N}} |t_M(c_{\pi,n})|$$

Finally, the *size* of $\mathcal{M}$ is the number $\|\mathcal{M}\| = |Q|$ of its control states.

In modular programming subroutines are a central concept. Let us formalize this notion. An *n-template* is a strategy machine $\mathcal{M} = (Q_M, q_{M,I}, q_{M,O}, \delta_M)$ with $2n$ distinguished states $\mathrm{sub}_1, \mathrm{ret}_1, \ldots, \mathrm{sub}_n, \mathrm{ret}_n$ such that $\delta_M(\mathrm{sub}_i, b)$ is undefined for all $1 \leq i \leq n$ and all $b \in \hat{\mathbb{B}}^3$. Let $\mathcal{M}$ be an n-template and let $\mathcal{S}_i = (Q_i, q_{i,I}, q_{i,O}, \delta_i)$, $1 \leq i \leq n$, be strategy machines. Define the strategy machine $\mathcal{M}[\mathcal{S}_1, \ldots, \mathcal{S}_n] = (Q_M \uplus \biguplus_i Q_i, q_{M,I}, q_{M,O}, \delta)$ by $\delta(\mathrm{sub}_i, b) = \delta_i(q_{i,I}, b)$ and $\delta(q_{i,O}, b) = (\mathrm{ret}_i, b, 0, 0, 0)$. In all other cases δ coincides with δ_M or δ_i, whenever this makes sense. Such a machine is called a *composition* of $\mathcal{S}_1, \ldots, \mathcal{S}_k$.

Definition 4. *Let* $* \in \{\|, \otimes\}$ *and let* $\mathbf{G} = (\mathcal{A}_1 * \cdots * \mathcal{A}_k, W)$.

1. *Let* $\mathbf{G}_1, \ldots, \mathbf{G}_k$ *be component games with winning strategies* $\mathcal{S}_1, \ldots, \mathcal{S}_k$ *for player 0. A* k*-template* $\mathcal{M}$ *such that* $\mathcal{M}[\mathcal{S}_1, \ldots, \mathcal{S}_k]$ *is a winning strategy in* $\mathbf{G}$ *is called a* winning composition of $\mathcal{S}_1, \ldots, \mathcal{S}_k$. *If* $\mathcal{M}$ *is a winning composition of any choice* $\mathcal{S}_1, \ldots, \mathcal{S}_k$ *of winning strategies for player 0 in* $\mathbf{G}_1, \ldots, \mathbf{G}_k$*, it is called a* winning composition of $\mathbf{G}_1, \ldots, \mathbf{G}_k$.
2. *A class* $\boldsymbol{\Lambda}$ *of games on composite arenas is said to* admit polynomial compositions*, if for every* $\mathbf{G} = (\mathcal{A}_1 * \cdots * \mathcal{A}_k, W) \in \boldsymbol{\Lambda}$ *there exist component games* $\mathbf{G}_1, \ldots, \mathbf{G}_k$ *and a polynomial sized winning composition* $\mathcal{M}$ *of* $\mathbf{G}_1, \ldots, \mathbf{G}_k$ *such that for some choice* $\mathcal{S}_1, \ldots, \mathcal{S}_k$ *of component winning strategies* $T(\mathcal{M}[\mathcal{S}_1, \ldots, \mathcal{S}_k]) \in \mathsf{poly}(\|\mathbf{G}\|)$*. Such a tuple* $\mathcal{M}, \mathcal{S}_1, \ldots, \mathcal{S}_k$ *is called a* polynomial composition.

Part of the appeal of polynomial compositions is their efficiency: By definition, a class of games admitting polynomial compositions enables us to find strategies with a polynomial latency (and thus a polynomial space requirement) which, depending on $\mathbf{G}_1, \ldots, \mathbf{G}_k$, have polynomial size. Note that for positionally determined component games this is always the case. The converse holds for trivial reasons. Any strategy machine $\mathcal{M}$ of polynomial size implementing a winning strategy with latency bounded polynomially can trivially be seen as a polynomial composition of any choice of machines $\mathcal{S}_1, \ldots, \mathcal{S}_k$.

We elaborate a bit on the restrictions we impose in the above definition. The requirement that all strategies in component games are interchangeable is to ensure that we compose general component games, not specific choices of strategies. In the definition of polynomial composition, the restriction on the size is to avoid templates which never call their subroutines and instead implement the entire global winning strategy on their own. Likewise, the restriction on the latency is to avoid enabling too powerful computations during the course of a single iteration (such as, for instance, solving the entire game every turn).

4 Games on Parallel Products

In this section we study reachability games over parallel products of arenas. If the arena is given as a composition of smaller arenas, it is natural to also study several compositional ways of specifying the reachability condition. Let $F_i \subseteq V_i$. We study the following formalisms:

1. *local reachability*, where $F = \mathcal{F}_{\mathrm{loc}}(F_1, \ldots, F_k)$ is the set of all $v \in \prod_i V_i$ with $v_i \in F_i$ for some i
2. *synchronized reachability*, where $F = \mathcal{F}_{\mathrm{sync}}(F_1, \ldots, F_k)$ is the set of all $v \in \prod_i V_i$ with $v_i \in F_i$ for every i

If $F = \mathcal{F}(F_1, \ldots, F_k)$, $\mathcal{F} \in \{\mathcal{F}_{\mathrm{loc}}, \mathcal{F}_{\mathrm{sync}}\}$, the set player 0 has to reach is $\mathbb{B} \times F$, i.e. the $\mathbb{B}$-component does not influence the outcome of the game.

Remark 2. One might consider *asynchronous* reachability, where all components F_i must be reached, but not necessarily at the same time. We omit this condition here, because it is not expressible as a reachability condition on the composite arena.

Theorem 1. *Games of the form* $\mathbf{G} = (\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_k, \mathcal{F}_{\mathrm{loc}}(F_1, \ldots, F_k))$ *admit polynomial compositions where the component games in def. 4 can be chosen as reachability games. In particular, component positional strategies suffice. Moreover, a polynomial composition can be computed in polynomial time.*

We omit the full proof due to space constraints. However, the idea is to show that deciding the winning set can be done by deciding conditions on the components: Player 0 wins from (σ_0, v_0) iff one of the following two applies

1. There exists i with $v_{i,0} \in V_i^{(0)}$ and $v_{i,0}E_i \cap F_i \neq \emptyset$.
2. $|\{i \mid v_{i,0} \in \mathrm{Attr}_0^{\mathcal{A}_i}(F_i)\}| \geq \mathrm{rank}_1$

The proof of this characterization gives component strategies for both players, which can be composed to a winning strategy for the respective player in $\mathbf{G}$.

Next, we consider synchronized reachability. We have:

Theorem 2. *Games of the form* $\mathbf{G} = (\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_k, \mathcal{F}_{\mathrm{sync}}(F_1, \ldots, F_k))$ *admit polynomial compositions. The component games in def. 4 can be chosen as positionally determined games. A polynomial composition can be computed in polynomial time.*

The proof idea is to characterize the winning set by considering component games. The proof of the characterization gives polynomial compositions for both players. The characterization is more involved than in thm. 1. We split the problem into subcases. For space reasons, we only state the characterization for the case $\mathrm{rank}_1 = 1$ and the case where both $\mathrm{rank}_1 > 1$ and $\mathrm{rank}_0 > 1$ and $\sigma_0 = 1$.

Lemma 1. *In thm. 2, let* (σ_0, v_0) *be such that* $\mathrm{rank}_1 = 1$. *Then player 0 wins from* (σ_0, v_0) *iff all of the following hold:*

1. *for all* i *we have* $v_{i,0} \in \mathrm{Attr}_0^{\mathcal{A}_i}(F_i)$
2. $|\{i \mid v_{i,0} \in \mathrm{Attr}_0^{\mathcal{A}_i}(F_i \cap V_i^{(0)})\}| \geq k - 1$
3. *if* $\sigma_0 = 1$ *and* $v_{j,0} \in F_j \cap V_j^{(1)}$ *for some* j, *then* $v_{i,0} \in F_i$ *for all* $i \neq j$ *or* $v_{j,0}E_j \subseteq \mathrm{Attr}_0^{\mathcal{A}_j}(F_j)$

A polynomial winning composition for player 0 (resp. player 1) with respect to positional strategies in component reachability (resp. safety) games can be computed in polynomial time.

In lem. 1 the component games are also reachability games (just like in thm. 1). In the next lemma, where $\mathrm{rank}_0 > 1$ and $\mathrm{rank}_1 > 1$, this is no longer the case. Instead, we use games with a temporal winning condition of low complexity. We call a game $\mathbf{G} = (\mathcal{A}, W)$ a *reachability game with safety constraint* if W is given by an LTL-formula $\varphi = S \,\mathsf{U}\, F$ with sets $S, F \subseteq V_{\mathcal{A}}$. We have:

Proposition 1. *Every reachability game with safety constraint is determined with positional strategies. Moreover, a winning strategy for both players can be computed in time $\mathcal{O}(|V_{\mathcal{A}}| + |E_{\mathcal{A}}|)$, if it exists.*

For simplicity, we exclude the trivial case where the initial position is already in $\mathcal{F}_{\mathrm{sync}}(F_1, \dots, F_k)$. We consider the case where player 1 moves first.

Lemma 2. *In thm. 2, let* $\mathrm{rank}_0 > 1$ *and* $\mathrm{rank}_1 > 1$. *Then player 0 wins from* $(1, v_0) \notin \mathcal{F}_{\mathrm{sync}}(F_1, \dots, F_k)$ *iff all of the following constraints are met:*

1. $v_{i,0} \in V_i^{(1)} \implies (v_{i,0} \in F_i \wedge \forall v_i' \in v_{i,0}E_i \colon v_i' \in F_i \vee v_i'E_i \cap F_i \neq \emptyset)$
2. $v_{i,0} \in V_i^{(0)} \setminus F_i \implies v_{i,0}E_i \cap F_i \neq \emptyset$
3. $|\{i \mid v_{i,0} \in \mathcal{W}^{(0)}\big(\mathcal{A}_i, (V_i^{(0)} \cup F_i)\,\mathsf{U}(V_i^{(0)} \cap F_i)\big)\}| \geq k - 1$

A polynomial winning composition for player 0 (resp. player 1) with respect to positional strategies in component reachability games with safety constraint can be computed in polynomial time.

We see that in the case of synchronized reachability we have to use a stronger notion of winning condition in the component games than we did in the global game. What kind of component games we need depends on the initial position.

The polynomial composition in thm. 2 relies on positional winning strategies in component games. This is closely tied to complexity in the following way:

Decision Problem ($\mathsf{PARALLEL\text{-}REACH}[\mathcal{F}]$)**.**
Input: Arenas $\mathcal{A}_1, \dots, \mathcal{A}_k$, a vertex $s = (\sigma_0, v) \in \mathbb{B} \times \prod_{i=1}^k V_i$ and sets $F_i \subseteq V_i$
Decide: $s \in \mathcal{W}^{(0)}$ in the game $\mathbf{G} = (\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_k, \mathcal{F}(F_1, \dots, F_k))$?

Corollary 1. *Let* $\mathcal{F} \in \{\mathcal{F}_{\mathrm{loc}}, \mathcal{F}_{\mathrm{sync}}\}$. $\mathsf{PARALLEL\text{-}REACH}[\mathcal{F}]$ *is* PTIME*-complete.*

We also have the following corollary of thm. 1 and thm. 2:

Corollary 2. *Let* $\mathbf{G} = (\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_k, \mathcal{F}(F_1, \dots, F_k))$, *where* $\mathcal{F} \in \{\mathcal{F}_{\mathrm{loc}}, \mathcal{F}_{\mathrm{sync}}\}$, *be a reachability game. There exists a winning strategy machine $\mathcal{M}$ for player 0 of size* $\|\mathcal{M}\| \in \mathsf{poly}\big(\sum_{i=1}^k \|\mathcal{A}_i\|\big)$ *and latency* $T(\mathcal{M}) \in \mathsf{poly}\big(\sum_{i=1}^k (\|\mathcal{A}_i\|)\big)$.

5 Games on Synchronized Products

In this section we investigate arenas obtained via the synchronized product. Here the situation is quite different. We first consider the complexity of deciding the winning region in those games:

Decision Problem (SYNC-REACH[$\mathcal{F}$]).
Input: Arenas $\mathcal{A}_1, \dots, \mathcal{A}_k$, a vertex $s = (\sigma_0, v) \in \mathbb{B} \times \prod_{i=1}^{k} V_i$ and sets $F_i \subseteq V_i$
Decide: $s \in \mathcal{W}^{(0)}$ in the game $\mathbf{G} = (\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_k, \mathcal{F}(F_1, \dots, F_k))$?

Theorem 3. *Let $\mathcal{F} \in \{\mathcal{F}_{\text{loc}}, \mathcal{F}_{\text{sync}}\}$. SYNC-REACH[$\mathcal{F}$] is* EXPTIME*-complete.*

Proof. We only show the claim for $\mathcal{F} = \mathcal{F}_{\text{loc}}$. The proof for $\mathcal{F} = \mathcal{F}_{\text{sync}}$ is an adaption of this proof.

Membership in EXPTIME is trivial (for instance, using a classical attractor on an in-memory explicit graph). We therefore focus on hardness.

The main idea is to reduce the acceptance problem of an APSPACE-Turing-machine. Since it is APSPACE, we have only polynomially many tape cells in use. We introduce a labeled arena $\mathcal{A}_i$ for each cell plus one additional labeled arena $\mathcal{A}_H$ storing both the state of the machine and the head position. Letters in $\mathcal{A}_i$ are indexed by i so that transitions may target a specific tape cell. The tape cell is updated by having players choose a transition label aligning the head position in $\mathcal{A}_H$ with the index of of the cell to be updated. Since the transition in $\mathcal{A}_H$ cannot observe the state of $\mathcal{A}_i$, the players may cheat with respect to the content of the tape cells. The construction below introduces a mechanism to enable players to challenge an opponent's cheating moves and win.

Let $\mathcal{M} = (Q_\exists, Q_\forall, \hat{\Gamma}, \Gamma, q_0, \Delta_\mathcal{M}, \{q_F\})$ be a bipartite APSPACE-machine. We suppose $\hat{\Gamma}$ is the input alphabet and $\Gamma \supseteq \hat{\Gamma}$ is the tape alphabet. Let $Q = Q_\exists \uplus Q_\forall$. Suppose $w \in \hat{\Gamma}^*$ is an input to $\mathcal{M}$. We assume that $\mathcal{M}$ accepts with exactly one final state $q_F \in Q_\exists$ and that q_F is never visited before termination. We may also assume that every configuration has at least one outgoing transition. Suppose p is a polynomial bounding the space of $\mathcal{M}$.

We define $p(|w|) = n$ automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ as follows. For every $i = 1, \dots, n$, let $\Gamma_i = \Gamma \times \{i\}$. Write γ_i for $(\gamma, i) \in \Gamma_i$. All n automata have the same alphabet $\Sigma = \bigcup_{i=1}^{n} \{\text{veto}_i\} \cup \bigcup_{i=1}^{n} \Gamma_i^2$. We define $\mathcal{A}_i = (A_i, \Sigma, \delta_i)$ with $A_i = (\Gamma \cup \{\bot_0, \bot_1\}) \times \mathbb{B}$ where $\delta_i \colon A_i \times \Sigma \dashrightarrow A_i$ is as follows:

$$\delta_i((\hat{\gamma}, \sigma), (\gamma_j, \gamma_j')) = \begin{cases} (\gamma', 1-\sigma) & \text{if } i = j \text{ and } \hat{\gamma} = \gamma \\ (\bot_\sigma, 1-\sigma) & \text{if } i = j \text{ but } \hat{\gamma} \neq \gamma \\ (\hat{\gamma}, 1-\sigma) & \text{if } i \neq j \end{cases}$$

for all $j \in \{1, \dots, k\}$, all $\hat{\gamma} \in \Gamma$, $\gamma_j, \gamma_j' \in \Gamma_j$ and all $\sigma \in \mathbb{B}$. We also define

$$\delta_i((\bot_p, \sigma), (\gamma_j, \gamma_j')) = (\bot_p, 1-\sigma)$$

for all $j \in \{1, \dots, k\}$, all $p, \sigma \in \mathbb{B}$ and all $\gamma_j, \gamma_j' \in \Gamma_j$.

Furthermore, a transition labeled with veto_i is defined on all player 1 states:

$$\delta_i((s, 1), \text{veto}_j) = \begin{cases} (\bot_1, 0) & \text{if } i = j \text{ and } s = \gamma_i \in \Gamma_i \\ (s, 0) & \text{if } i \neq j \text{ or } s = \bot_p,\ p \in \mathbb{B} \end{cases}$$

In particular, player 0 can never play veto_i on his components. The partition of A_i into player 0 and player 1 states is given by $A_i^{(\sigma)} = \{(s, \sigma) \mid s \in \Gamma \cup \{\bot_0, \bot_1\}\}$.

Next, we define $\mathcal{A}_H$ with states $A_H = (Q \times \{1, \dots, n\}) \uplus \{C, (\top, 0), (\bot, 0)\}$, where $A_H^{(0)} = \{(q, h) \mid q \in Q_\exists\} \cup \{(\top, 0), (\bot, 0)\}$ and $A_H^{(1)} = \{(q, h) \mid q \in Q_\forall\} \cup \{C\}$. The alphabet of this automaton is again Σ (as defined above). Its transition relation Δ_H is defined by

$$\big((q, h), (\gamma_j, \gamma_j'), (q', h')\big) \in \Delta_H \iff h = j \wedge (q, \gamma, q', \gamma', d) \in \Delta_\mathcal{M} \wedge h' = h + d$$

Note that "illegal" transitions are impossible. The players can only *cheat* with respect to the content of the h-th tape cell. Also, no transition labeled with veto_i for any i is possible from a state (q, h). In addition, we now have the following transitions:

$$\begin{aligned}
\big((q_F, h), (\gamma_i, \gamma_i'), C\big) &\in \Delta_H && \text{for all } i, h \in \{1, \dots, k\},\ \gamma, \gamma' \in \Gamma \\
(C, \text{veto}_i, (\bot, 0)) &\in \Delta_H && \text{for all } i \in \{1, \dots, k\} \\
\big(C, (\gamma_i, \gamma_i'), (\top, 0)\big) &\in \Delta_H && \text{for all } i \in \{1, \dots, k\},\ \gamma, \gamma' \in \Gamma
\end{aligned}$$

Suppose $q_0 \in Q_\exists$. The play begins in position $\big((q_0, 1, 0), (\#_1, 0), \dots, (\#_n, 0)\big)$, where $\#_i \in \Gamma_i$ is the blank symbol of $\mathcal{M}$. Player 0 moves (i.e. picks a letter) at all states in which $\mathcal{A}_H$ is in a state from $Q_\exists$, player 1 if it is in a state from $Q_\forall$. This is ensured by the definition of the transition function δ_i, which guarantees that each component changes from a σ-state to a $(1 - \sigma)$-state in every round. The states $(\bot, 0)$ and $(\top, 0)$ in $\mathcal{A}_H$ are 0-states without outgoing transitions. The set player 0 tries to reach is $\mathcal{F}_{\text{loc}}(\{(\top, 0)\}, \{(\bot_1, 0)\}, \dots, \{(\bot_1, 0)\})$.

We now show the correctness of the above construction. If $\mathcal{M}$ accepts w, then player 0 has a winning strategy in the reachability game on the configuration graph of $\mathcal{M}$ on w. If player 1 does not cheat and player 0 plays according to his strategy, the play will finally reach a state $((q_F, h), x_1, \dots, x_n)$ with $x_i \neq (\bot_p, 0)$ for all $p \in \mathbb{B}$ and $i \in \{1, \dots, n\}$. Recall that $q_F \in Q_\exists$. Now player 0 must move to C. Unless player 0 cheats (which is clearly a suboptimal choice at this point), this implies that every component i moves from $x_i = (\gamma_i, 0)$ to $(\gamma_i, 1)$ by the definition of δ_i. Player 1 can play veto_i for some i. However, since the i-th component is in state $(\gamma_i, 1)$ for some $\gamma_i \in \Gamma_i$, we have that this results in the i-th component making a transition to state $(\bot_1, 0)$. Thus player 0 wins. If player 1 plays (γ_i, γ_i') for some i, the play reaches $(\top, 0)$ and thus player 0 wins. If player 1 made an illegal transition at some point in the play, then for some i, the state of $\mathcal{A}_i$ loops between $(\bot_1, 0)$ and $(\bot_1, 1)$ from that point onwards and, again, player 0 wins.

Conversely, if $\mathcal{M}$ rejects w, then player 1 has a winning strategy in the safety game on the configuration graph of $\mathcal{M}$ on w. This implies that, unless player 0 uses an illegal transition, the play never reaches state q_F. On the other hand, if player 0 does make an illegal transition, one component, say i, changes to state $(\bot_0, 1)$ and remains in $\{(\bot_0, \sigma) \mid \sigma \in \mathbb{B}\}$ from this point onwards. If the play ever reaches q_F after that, and thereafter reaches C, player 1 can play veto_i moving $\mathcal{A}_H$ into state $(\bot, 0)$. Component i is in state $(\bot_0, 1)$ when $\mathcal{A}_H$ is in C whereby $\mathcal{A}_i$ never reaches state $(\bot_1, 0)$. Since player 1 never has to make an illegal transition, no component j is in a state $(\bot_1, 0)$. Hence player 0 loses. □

The high complexity of SYNC-REACH$[\mathcal{F}]$ for $\mathcal{F} \in \{\mathcal{F}_{\text{loc}}, \mathcal{F}_{\text{sync}}\}$ prohibits polynomially computable polynomial compositions for reachability games on synchronous arenas (with local or synchronized reachability conditions). Indeed, finding such compositions would amount to showing EXPTIME = PTIME. What can we do differently in order to succeed?

In order to find a winning composition of polynomial size one might suspect that more complex component games are necessary. In this event, finding strategies in the the component games would be more difficult, sparing us the complexity dilemma. Unfortunately, this turns out to be false.

Remark 3. In the reduction above, the constituent arenas are of size $\leq c$ for some constant $c \in \mathbb{N}$ (essentially the size of some APSPACE Turing machine $\mathcal{M}$ deciding an EXPTIME-complete problem). Thus, more complex winning conditions on those arenas will not increase the complexity of finding winning strategies.

Another possibility is to loosen the notion of a polynomial composition. Recall that a winning composition $\mathcal{M}$ of strategies $\mathcal{S}_1, \ldots, \mathcal{S}_k$ is a polynomial winning composition if $\mathcal{M}[\mathcal{S}_1, \ldots, \mathcal{S}_k]$ has polynomial latency and $\mathcal{M}$ has polynomial size. We now loosen this requirement as follows: A winning composition $\mathcal{M}$ is a *poly-space composition* if $\mathcal{M}$ is of polynomial size and the space requirement of $\mathcal{M}[\mathcal{S}_1, \ldots, \mathcal{S}_k]$ is polynomial.

Lemma 3. *Let* $\mathbf{G} = (\mathcal{A}_1 \otimes \cdots \otimes \mathcal{A}_k, \mathcal{F}(F_1, \ldots, F_k))$*, where* $\mathcal{F} \in \{\mathcal{F}_{\text{loc}}, \mathcal{F}_{\text{sync}}\}$*. Given* $\mathbf{G}$*, a strategy machine* $\mathcal{M}$ *with polynomial space requirement implementing a strategy for player 0 in* $\mathbf{G}$ *from position* $p_0 = (\sigma_0, v_0)$*, and sets* $F_1, \ldots, F_k$ *it is decidable in* PSPACE *whether or not* $\mathcal{M}$ *implements a winning strategy from* p_0*.*

The proof uses APTIME = PSPACE to verify that there is an $\mathcal{M}$-consistent loop which does not visit $F = \mathcal{F}(F_1, \ldots, F_k)$ and can be reached without visiting F.

Theorem 4. *The class of reachability games over synchronized products admits poly-space compositions iff* PSPACE = EXPTIME*.*

Proof. Clearly PSPACE = EXPTIME implies that a strategy machine with a polynomial space requirement can compute the next move in some attractor strategy in the course of a single iteration.

Conversely, if the class admits poly-space compositions, there always exists a polynomial sized winning strategy with a polynomial space requirement (assuming the component games are bounded by some constant, cf. Rem. 3) for player 0 (if he wins). Hence, the following NPSPACE procedure is correct: We guess a strategy machine of polynomial size and verify in PSPACE if it implements a winning strategy. By Savitch's theorem, PSPACE = NPSPACE. □

6 Conclusion

We studied the relation between the compositional nature of an arena and the structure of a winning strategy. To this end we introduced two kinds of products on arenas, the parallel and the synchronized product. We defined a notion

of strategy composition which relies on strategy machines. This notion of composition allows to translate winning strategies in component games to winning strategies in the global game. We proved such a composition theorem for the class of reachability games on parallel products. We also showed why a similar result holds on synchronized products iff EXPTIME = PSPACE.

The results of this paper carry through to Büchi games with only minor modifications. We also have results on the case where the reachability condition is given explicitly (instead of as a sequence of k sets). For future research we want to consider more complex winning conditions, such as parity and weak parity. Also, we want to treat different ways of modeling the composite game from constituent arenas, addressing notions of composition from the field of process algebra and formal verification.

Acknowledgments. I would like to thank the anonymous reviewers for many helpful suggestions, both for the presentation and for future research.

References

1. Büchi, J.R., Landweber, L.H.: Solving Sequential Conditions by Finite-State Strategies. Trans. of the AMS 138, 295–311 (1969)
2. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
3. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. 200, 135–183 (1998)
4. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata logics, and infinite games: a guide to current research. Springer, New York (2002)
5. Löding, C.: Infinite games and automata theory. In: Apt, K.R., Grädel, E. (eds.) Lectures in Game Theory for Computer Scientists. Cambridge U. P. (2011)
6. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
7. Fearnley, J., Peled, D., Schewe, S.: Synthesis of succinct systems. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 208–222. Springer, Heidelberg (2012)
8. Harel, D., Kupferman, O., Vardi, M.Y.: On the complexity of verifying concurrent transition systems. Inf. Comput. 173(2), 143–161 (2002)
9. Gelderie, M.: Strategy machines and their complexity. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 431–442. Springer, Heidelberg (2012)
10. Goldin, D.Q., Smolka, S.A., Wegner, P.: Turing machines, transition systems, and interaction. Electr. Notes Theor. Comput. Sci. 52(1), 120–136 (2001)
11. Hunter, P., Dawar, A.: Complexity bounds for regular games (extended abstract). In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 495–506. Springer, Heidelberg (2005)
12. Dawar, A., Horn, F., Hunter, P.: Complexity Bounds for Muller Games. Theoretical Computer Science (2011) (submitted)
13. Horn, F.: Explicit Muller Games are PTIME. In: FSTTCS, pp. 235–243 (2008)