Department of Computer Science

# Compositional Controller Synthesis for Stochastic Games

Nicolas Basset, Marta Kwiatkowska, and Clemens Wiltsche

CS-RR-14-05

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

# Compositional Controller Synthesis for Stochastic Games

Nicolas Basset, Marta Kwiatkowska, and Clemens Wiltsche

Department of Computer Science, University of Oxford, United Kingdom

**Abstract.** Design of autonomous systems is facilitated by automatic synthesis of correct-by-construction controllers from formal models and specifications. We focus on stochastic games, which can model the interaction with an adverse environment, as well as probabilistic behaviour arising from uncertainties. We propose a synchronising parallel composition for stochastic games that enables a compositional approach to controller synthesis. We leverage rules for compositional assume-guarantee verification of probabilistic automata to synthesise controllers for games with multi-objective quantitative winning conditions. By composing winning strategies synthesised for the individual components, we can thus obtain a winning strategy for the composed game, achieving better scalability and efficiency at a cost of restricting the class of controllers.

## 1 Introduction

With increasing pervasiveness of technology in civilian and industrial applications, it has become paramount to provide formal guarantees of safety and reliability for autonomous systems. We consider the development of correct-by-construction controllers satisfying high-level specifications, based on formal system models. Automated synthesis of controllers has been advocated, for example, for autonomous driving [2] and distributed control systems [14].

**Stochastic Games.** When designing autonomous systems, often a critical element is the presence of an uncertain and adverse environment, which introduces stochasticity and requires the modelling of the non-cooperative aspect in a game-theoretical setting [6,13]. Hence, we model a system we wish to control as a two-player turn-based stochastic game [17], and consider automated synthesis of strategies that are winning against every environment (Player $\Box$), which we can then interpret as controllers of the system (Player $\Diamond$). In addition to probabilities, one can also annotate the model with rewards to evaluate various quantities, for example, profit or energy usage, by means of expectations.

**Compositionality.** We model systems as a composition of several smaller components. For controller synthesis for games, a compositional approach requires that we can derive a strategy for the composed system by synthesising only for the individual components. Probabilistic automata (PAs) are naturally suited to modelling multi-component probabilistic systems, where synchronising composition is well-studied [16]; see also [19] for a taxonomic discussion. While PAs can be viewed as stochastic games with strategies already applied, it is not immediately clear how to compose games in a natural way.

**Our Composition.** We formulate a composition of stochastic games where component games synchronise on shared actions and interleave otherwise. The composition is inspired by interface automata [8], which are well-suited to compositional assume-guarantee verification of component-based systems, where we preserve the identity of Player $\Diamond$ and Player $\Box$ in the composition by imposing similar compatibility conditions. Our composition is commutative and associative, and reduces to PA composition when only Player $\Box$ is present. The results we prove are independent of specific winning conditions, and thus provide a general framework for the development of compositional synthesis methods.

**Compositional Synthesis.** We show that any rule for compositional verification of PAs carries over as a synthesis rule for games. First, after applying winning strategies to a game, the resulting PAs still satisfy the winning condition for any environment. Then, compositional rules for PAs can be used, such as the assume-guarantee rules in [12] developed for winning conditions involving multi-objective total expected reward and probabilistic LTL queries. These first two steps, described also as "schedule-and-compose" [7], are applicable when strategies can only be implemented locally in practice, for instance, when wanting to control a set of physically separated electrical generators and loads in a microgrid, where no centralised strategy can be implemented. One key property of our game composition is that strategies applied to individual components can be composed to a strategy for the composed game, while preserving the probability measure over the traces. Hence, we obtain a winning strategy for the composed game, which alleviates the difficulty of having to deal with the product state space by trading off the expressiveness of the generated strategies.

**Winning Conditions.** Each player plays according to a *winning condition*, specifying the desirable behaviour of the game, for example "the probability of reaching a failure state is less than 0.01." We are interested in synthesis for zero-sum games for which several kinds of winning conditions are defined in the literature, including $\omega$-regular [3], expected total and average reward [9], and multi-objective versions thereof [6]. Our game composition is independent of such winning conditions, since they are definable on the trace distributions.

**Work we Build Upon.** In this paper we extend the work of [12] by lifting the compositional verification rules for PAs to compositional synthesis rules for games. Typically, such rules involve multi-objective queries, and we extend the synthesis methods for such queries in [5,6] to compositional strategy synthesis.

**Contributions.** Several notions of (non-stochastic) game composition have recently been proposed [10,11], but they do not preserve player identity, i.e. which player controls which actions, and hence are not applicable to synthesising strategies for a specific player. In this paper, we make the following contributions.

– We define a composition for stochastic games, which, to our knowledge, is the first composition for competitive probabilistic systems that preserves the control authority of Player $\Diamond$, enabling compositional strategy synthesis.

– We show how to apply strategies synthesised for the individual components to the composition, such that the trace distribution is preserved.
– We lift compositional rules for PAs to the game framework for synthesis.
– We apply our theory to demonstrate how to compositionally synthesise controllers for games with respect to multi-objective total expected reward queries, and demonstrate the benefits on a prototype implementation.

**Structure.** In Section 2 we introduce stochastic games, their normal form, and their behaviour under strategies. In Section 3 we define our game composition, and show that strategies for the individual components can be applied to the composed game. We demonstrate in Section 4 how to use proof rules for PAs and previously developed synthesis methods to compositionally synthesise strategies.

## 2 Stochastic games, induced PAs and DTMCs

We introduce notation and main definitions for stochastic games and their behaviour under strategies.

**Distributions.** A *discrete probability distribution* (or *distribution*) over a (countable) set $Q$ is a function $\mu : Q \to [0,1]$ such that $\sum_{q\in Q}\mu(q) = 1$; its *support* $\{q \in Q \,|\, \mu(q) > 0\}$ is the set of values where $\mu$ is nonzero. We denote by $\mathcal{D}(Q)$ the set of all distributions over $Q$ with finite support. A distribution $\mu \in \mathcal{D}(Q)$ is *Dirac* if $\mu(q) = 1$ for some $q \in Q$, and if the context is clear we just write $q$ to denote such a distribution $\mu$. We denote by $\boldsymbol{\mu}$ the *product distribution* of $\mu^i \in \mathcal{D}(Q^i)$ for $1 \le i \le n$, defined on $Q^1\times\cdots\times Q^n$ by $\boldsymbol{\mu}(q^1,\ldots,q^n) \overset{\text{def}}{=} \mu^1(q^1)\cdot\cdots\cdot\mu^n(q^n)$.

**Stochastic Games.** We consider turn-based action-labelled stochastic two-player games (henceforth simply called *games*), which distinguish two types of nondeterminism, each controlled by a separate player. Player $\Diamond$ represents the controllable part for which we want to synthesise a strategy, while Player $\Box$ represents the uncontrollable environment. Examples of games are shown in Figure 1.

**Definition 1.** *A* game *is a tuple* $\langle S, (S_\Diamond, S_\Box), \varsigma, \mathcal{A}, \longrightarrow\rangle$*, where $S$ is a countable set of* states *partitioned into* Player $\Diamond$ states $S_\Diamond$ *and* Player $\Box$ states $S_\Box$*;* $\varsigma \in \mathcal{D}(S)$ *is an* initial distribution*;* $\mathcal{A}$ *is a countable set of* actions*; and* $\longrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times \mathcal{D}(S)$ *is a* transition relation*, such that, for all $s$,* $\{(s,a,\mu) \in \longrightarrow\}$ *is finite.*

We adopt the infix notation by writing $s \xrightarrow{a} \mu$ for a *transition* $(s,a,\mu) \in \longrightarrow$, and if $a = \tau$ we speak of a $\tau$-transition. The action labels $\mathcal{A}$ on transitions model observable behaviours, whereas $\tau$ can be seen as internal: it cannot be used in winning conditions and is not synchronised in the composition.

We denote the set of *moves* by $S_\bigcirc \overset{\text{def}}{=} \{(a,\mu) \,|\, \exists s \in S\,.\, s \xrightarrow{a} \mu\}$. A move $(a,\mu)$ is *incoming* to a state $s$ if $\mu(s) > 0$, and is *outgoing* from a state $s$ if $s \xrightarrow{a} \mu$. Note that, as for PAs [16], there could be several moves associated to each action. We define the set of actions *enabled* in a state $s$ by $\mathsf{En}(s) \overset{\text{def}}{=} \{a \in \mathcal{A} \,|\, \exists\mu\,.\, s \xrightarrow{a} \mu\}$.

A finite (infinite) *path* $\lambda = s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2\ldots$ is a finite (infinite) sequence of alternating states and moves, such that $\varsigma(s_0) > 0$, and, for all $i \ge 0$,

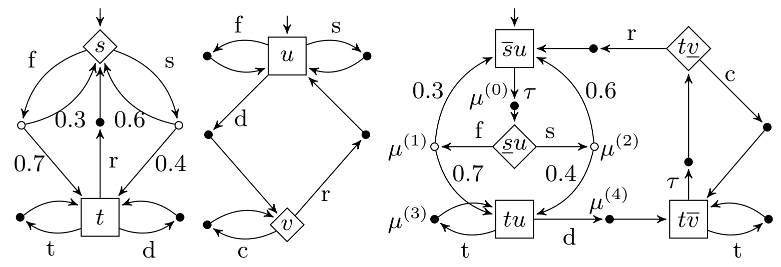


Fig. 1: Three games: the game on the right is the composition of the normal forms of the other two games. Dirac distributions are shown as filled circles.

$s_i \xrightarrow{a_i} \mu_i$ and $\mu_i(s_{i+1}) > 0$. A finite path $\lambda$ ends in a state, and we write $\mathsf{last}(\lambda)$ for the last state of $\lambda$. We denote the set of finite (infinite) paths of a game $G$ by $\Omega^+_G$ ($\Omega_G$), and by $\Omega^+_{G,\#}$ the set of paths ending in a $\mathsf{Player}\ \#$ state, for $\# \in \{\Diamond, \Box\}$. A finite (infinite) *trace* is a finite (infinite) sequence of actions. Given a path, its trace is the sequence of actions along $\lambda$, with $\tau$ projected out. Formally, $\mathsf{trace}(\lambda) \stackrel{\text{def}}{=} \text{PROJ}_{\{\tau\}}(a_0a_1\ldots)$, where, for $\alpha \subseteq \mathcal{A} \cup \{\tau\}$, $\text{PROJ}_\alpha$ is the morphism defined by $\text{PROJ}_\alpha(a) = a$ if $a \notin \alpha$, and $\epsilon$ (the empty trace) otherwise.

**Strategies.** Nondeterminism for each player is resolved by a strategy. A *strategy* for $\mathsf{Player}\ \#$, for $\# \in \{\Diamond, \Box\}$, is a function $\sigma_\# : \Omega^+_{G,\#} \to \mathcal{D}(S_\bigcirc)$ such that $\sigma_\#(\lambda)(a,\mu) > 0$ only if $\mathsf{last}(\lambda) \xrightarrow{a} \mu$. The set of $\mathsf{Player}\ \#$ strategies in game $G$ is denoted by $\Sigma^G_\#$. A strategy is called *memoryless* if, for each path $\lambda$, the choice $\sigma_\#(\lambda)$ is uniquely determined by $\mathsf{last}(\lambda)$.

**Normal Form of a Game.** We can transform every game into its corresponding normal form, which does not affect the winning conditions. Transforming a game to normal form is the first step of our game composition.

**Definition 2.** *A game is in* normal form *if the following hold:*

– *Every $\tau$-transition $s \xrightarrow{\tau} \mu$ is from a $\mathsf{Player}\ \Box$ state $s$ to a $\mathsf{Player}\ \Diamond$ state $s'$ with a Dirac distribution $\mu = s'$.*
– *Every $\mathsf{Player}\ \Diamond$ state $s$ can only be reached by an incoming move $(\tau, s)$. In particular, every distribution $\mu$ of a non-$\tau$-transition, as well as the initial distribution, assigns probability zero to all $\mathsf{Player}\ \Diamond$ states.*

Given a game $G$ without $\tau$-transitions, one can construct its normal form $\mathcal{N}(G)$ by splitting every state $s \in S_\Diamond$ into a $\mathsf{Player}\ \Box$ state $\overline{s}$ and a $\mathsf{Player}\ \Diamond$ state $\underline{s}$, s.t.

– the incoming (resp. outgoing) moves of $\overline{s}$ (resp. $\underline{s}$) are precisely the incoming (resp. outgoing) moves of $s$, with every $\mathsf{Player}\ \Diamond$ state $t \in S_\Diamond$ replaced by $\overline{t}$;
– and the only outgoing (resp. incoming) move of $\overline{s}$ (resp. $\underline{s}$) is $(\tau, \underline{s})$.

Intuitively, at $\overline{s}$ the game is idle until $\mathsf{Player}\ \Box$ allows $\mathsf{Player}\ \Diamond$ to choose a move in $\underline{s}$. Hence, any strategy for $G$ carries over naturally to $\mathcal{N}(G)$, and we can operate w.l.o.g. with normal-form games. Also, $\tau$ can be considered as a scheduling choice. In the transformation to normal form, at most one such scheduling choice is introduced for each Player $\Box$ state, but in the composition more choices can be added, so that Player $\Box$ resolves nondeterminism arising from concurrency.

**Game Unfolding.** Strategy application is defined on the unfolded game. The *unfolding* $\mathcal{U}(G) = \langle \Omega_G^+, (\Omega_{G,\Box}^+, \Omega_{G,\Diamond}^+), \varsigma, \mathcal{A}, \longrightarrow' \rangle$ of the game $G$ is such that $\lambda \xrightarrow{a}' \mu_{\lambda,a}$ if and only if $\mathsf{last}(\lambda) \xrightarrow{a} \mu$ and $\mu_{\lambda,a}(\lambda(a,\mu)s) \overset{\text{def}}{=} \mu(s)$ for all $s \in S$.

An unfolded game is a set of trees (the roots are the support of the initial distribution), with potentially infinite depth, but finite branching. The entire history is stored in the states, so memoryless strategies suffice for unfolded games; formally, each strategy $\sigma_\Diamond \in \Sigma_\Diamond^G$ straightforwardly maps to a memoryless strategy $\mathcal{U}(\sigma_\Diamond) \in \Sigma_\Diamond^{\mathcal{U}(G)}$ by letting $\mathcal{U}(\sigma_\Diamond)(\lambda)(a, \mu_{\lambda,a}) = \sigma_\Diamond(\lambda)(a,\mu)$. We denote by $\mathcal{U}(G)_\bigcirc$ the set of moves of the unfolded form of a game $G$ and by $\mathcal{U}(G)_{\#\bigcirc}$ the set of moves following a Player # state that is of the form $(a, \mu_{\lambda,a})$ with $\lambda \in \Omega_{G,\#}^+$. We remark that the unfolding of a normal form game is also in normal form.

## 2.1 Induced PA

When only one type of nondeterminism is present in a game, it is a *probabilistic automaton (PA)*. PAs are well-suited for compositional modelling [16], and can be used for verification, i.e. checking whether *all* behaviours satisfy a specification (when only Player $\Box$ is present), as well as strategy synthesis (when only Player $\Diamond$ is present) [13]. A PA is a game where $S_\Diamond = \emptyset$ and $S_\Box = S$, which we write here as $\langle S, \varsigma, \mathcal{A}, \longrightarrow \rangle$. This definition corresponds to modelling nondeterminism as an adverse, uncontrollable, environment, and so, by applying a Player $\Diamond$ strategy to a game to resolve the controllable nondeterminism, we are left with a PA where only uncontrollable nondeterminism for Player $\Box$ remains.

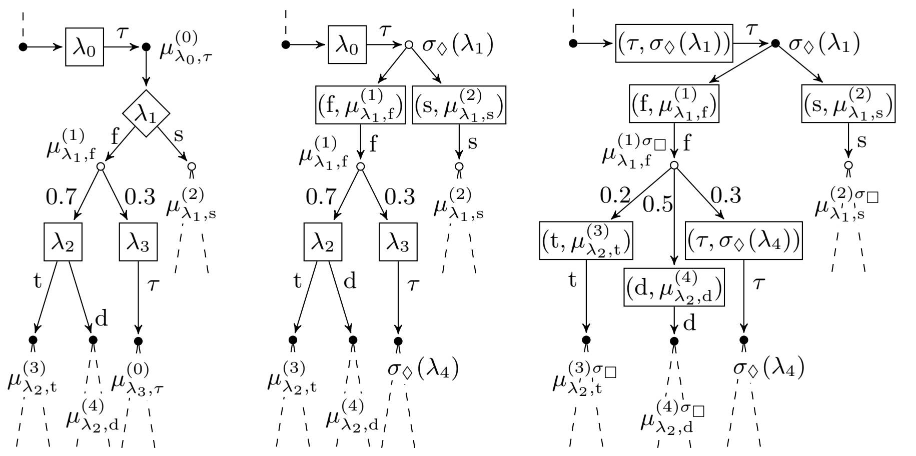
**Definition 3.** *Given an unfolded game* $\mathcal{U}(G) = \langle \Omega_G^+, (\Omega_{G,\Box}^+, \Omega_{G,\Diamond}^+), \varsigma, \mathcal{A}, \longrightarrow \rangle$ *in normal form and a strategy* $\sigma_\Diamond \in \Sigma_\Diamond^G$*, the* induced PA *is* $G^{\sigma_\Diamond} = \langle S', \varsigma, \mathcal{A}, \longrightarrow' \rangle$*, where* $S' \subseteq \Omega_{G,\Box}^+ \cup \mathcal{U}(G)_{\Diamond\bigcirc}$ *is defined inductively as the reachable states, and*

*(I1)* $\lambda \xrightarrow{\tau}' \mathcal{U}(\sigma_\Diamond)(\lambda')$ *iff* $\lambda \xrightarrow{\tau} \lambda'$ *(Player $\Diamond$ strategy chooses a move);*
*(I2)* $(a, \mu_{\lambda,a}) \xrightarrow{a}' \mu_{\lambda,a}$ *for* $(a, \mu_{\lambda,a}) \in \mathcal{U}(G)_{\Diamond\bigcirc}$ *(the chosen move is performed);*
*(I3)* $\lambda \xrightarrow{a}' \mu_{\lambda,a}$ *iff* $\lambda \xrightarrow{a} \mu_{\lambda,a}$ *and* $\lambda \in \Omega_{G,\Box}^+$ *(external transitions from older Player $\Box$ state remain unchanged).*

The unfolded form of the game in Figure 1(right) is shown in Figure 2(a), and strategy application is illustrated in Figure 2(b).

## 2.2 Induced DTMC

A discrete-time Markov chain (DTMC) is a model for systems with probabilistic behaviour only. When applying a Player $\Box$ strategy to an induced PA, all nondeterminism is resolved, and a DTMC is obtained. A (labelled) *DTMC D* is a PA such that, for each $s \in S$, there is at most one transition $s \xrightarrow{a} \mu$.



$\lambda_0$
$\tau$
$\mu^{(0)}_{\lambda_0,\tau}$
$\lambda_1$
f
s
$\mu^{(1)}_{\lambda_1,\mathrm{f}}$
$\mu^{(2)}_{\lambda_1,\mathrm{s}}$
0.7
0.3
$\lambda_2$
$\lambda_3$
t
d
$\mu^{(3)}_{\lambda_2,\mathrm{t}}$
$\mu^{(4)}_{\lambda_2,\mathrm{d}}$
$\mu^{(0)}_{\lambda_3,\tau}$
$\sigma_\Diamond(\lambda_1)$
$(\mathrm{f}, \mu^{(1)}_{\lambda_1,\mathrm{f}})$
$(\mathrm{s}, \mu^{(2)}_{\lambda_1,\mathrm{s}})$
$\sigma_\Diamond(\lambda_4)$
$(\tau, \sigma_\Diamond(\lambda_1))$
$\mu^{(1)\sigma_\Box}_{\lambda_1,\mathrm{f}}$
0.2
0.5
0.3
$(\mathrm{t}, \mu^{(3)}_{\lambda_2,\mathrm{t}})$
$(\mathrm{d}, \mu^{(4)}_{\lambda_2,\mathrm{d}})$
$(\tau, \sigma_\Diamond(\lambda_4))$
$\mu^{(2)\sigma_\Box}_{\lambda_1,\mathrm{s}}$
$\mu^{(3)\sigma_\Box}_{\lambda_2,\mathrm{t}}$
$\mu^{(4)\sigma_\Box}_{\lambda_2,\mathrm{d}}$

(a) Game in Figure 1 (right) unfolded.

(b) Induced PA via the strategy $\sigma_\Diamond$ assigning uniform probabilities.

(c) Induced DTMC via the strategy $\sigma_\Box = \mathcal{U}(\sigma'_\Box)$ assigning $\frac{2}{7}$ to t and $\frac{5}{7}$ to d.

Fig. 2: Game unfoldings. The dashed lines represent recursive application of unfolding and strategy application. Denote $\lambda_0 = \bar{s}u$, $\lambda_1 = \lambda_0(\tau, \mu^{(0)}_{\lambda_0,\tau})\underline{s}u$, $\lambda_2 = \lambda_1(\mathrm{f}, \mu^{(1)}_{\lambda_1,\mathrm{f}})\bar{s}u$, $\lambda_3 = \lambda_1(\mathrm{f}, \mu^{(1)}_{\lambda_1,\mathrm{f}})tu$, $\lambda_4 = \lambda_3(\tau, \mu^{(0)}_{\lambda_3,\tau})\underline{s}u$.

**Definition 4.** *Given an unfolded PA $\mathcal{U}(M) = \langle S, \varsigma, \mathcal{A}, \longrightarrow\rangle$ and a strategy $\sigma_\Box \in \Sigma^M_\Box$, the* induced DTMC *$M^{\sigma_\Box} = \langle S', \varsigma', \mathcal{A}, \longrightarrow'\rangle$ is such that $S' \subseteq \mathcal{U}(M)_\bigcirc$ is defined inductively as the states reachable via $\varsigma'$ and $\longrightarrow'$, where*

- *$(a, \mu_{\lambda,a}) \xrightarrow{a}{}' \mu^{\sigma_\Box}_{\lambda,a}$, such that, for all moves $(b, \nu_{\lambda(a,\mu)t,b})$, we let the distribution $\mu^{\sigma_\Box}_{\lambda,a}(b, \nu_{\lambda(a,\mu)t,b}) \overset{\text{def}}{=} \mu(t)\sigma_\Box(\lambda(a,\mu)t)(b,\nu)$; and*
- *for all moves $(b, \nu_{t,b})$, $\varsigma'(b, \nu_{t,b}) \overset{\text{def}}{=} \varsigma(t)\sigma_\Box(t)(b,\nu)$.*

Note that an induced PA is already unfolded, and does not need to be unfolded again. We illustrate in Figure 2(c) the application of a $\mathsf{Player}$ $\Box$ strategy.

**Probability Measures.** We define the probability measure $\mathrm{Pr}_D$ of a DTMC $D$ in the usual way. The *cylinder set* of a path $\lambda \in \Omega^+_D$ (resp. trace $w \in \mathcal{A}^*$) is the set of infinite paths (resp. traces) with prefix $\lambda$ (resp. $w$). For a finite path $\lambda = s_0(a_0, \mu_0)s_1(a_1, \mu_1)\dots s_n$ we define $\mathrm{Pr}_D(\lambda)$, the measure of its cylinder set, by: $\mathrm{Pr}_D(\lambda) \overset{\text{def}}{=} \varsigma(s_0)\prod_{i=0}^{n-1}\mu_i(s_{i+1})$. We write $\mathrm{Pr}^{\sigma_\Diamond,\sigma_\Box}_G$ (resp. $\mathrm{Pr}^{\sigma_\Box}_M$) for the measure $\mathrm{Pr}_{G^{\sigma_\Diamond,\sigma_\Box}}$ (resp. $\mathrm{Pr}_{M^{\sigma_\Box}}$). The measures uniquely extend to infinite paths due to Carathéodory's extension theorem.

Given a finite trace $w$, $\mathsf{paths}(w)$ denotes the set of minimal finite paths with trace $w$, i.e. $\lambda \in \mathsf{paths}(w)$ if $\mathsf{trace}(\lambda) = w$ and there is no path $\lambda' \neq \lambda$ with $\mathsf{trace}(\lambda') = w$ and $\lambda'$ being a prefix of $\lambda$. The measure of the cylinder set of $w$ is $\mathrm{Pr}_D(w) \overset{\text{def}}{=} \sum_{\lambda \in \mathsf{paths}(w)} \mathrm{Pr}_D(\lambda)$, and we call $Pr_D$ the *trace distribution* of $D$.

**Winning Conditions.** Providing strategies for both players resolves all nondeterminism in a game, resulting in a distribution over paths. A *specification* $\varphi$ is a predicate on trace distributions, and for a DTMC $D$ we write $D \models \varphi$ if $\varphi(P_D)$ holds. A specification $\varphi$ is *defined on traces* if $\varphi(P_D) = \varphi(P_{D'})$ for all DTMCs $D, D'$ such that $P_D(w) = P_{D'}(w)$ for all traces $w$.

## 3 Composing stochastic games

We now introduce composition operators for games and Player $\Diamond$ strategies, leading towards a framework for compositional synthesis in Section 4. Games can be composed of several component games, and our composition is inspired by interface automata [8], which have a natural interpretation as (concurrent) games.

### 3.1 Game Composition

We provide a synchronising composition of games so that controllability is preserved for Player $\Diamond$, that is, actions controlled by Player $\Diamond$ in the components are controlled by Player $\Diamond$ in the composition. We endow each component game with an *alphabet* of actions $\mathcal{A}$, where synchronisation on *shared actions* in $\mathcal{A}^1 \cap \mathcal{A}^2$ is viewed as a (blocking) communication over ports, as in interface automata, though for simplicity we do not distinguish inputs and outputs. Synchronisation is multi-way and we do not impose input-enabledness of IO automata [7].

Given $n$ games $G^i$ in normal form with respective state spaces $S^i$, for $i \in I$ (let $I = \{1, \ldots, n\}$ be the *index set* of the composed game), the state space of the composition is a subset of the Cartesian product $S^1 \times \ldots \times S^n$, whose states contain at most one Player $\Diamond$ component thanks to the normal form. We denote by $s^i$ the $i$th component of $\boldsymbol{s} \in S^1 \times \ldots \times S^n$. Furthermore, every probability distribution in the composition is a product distribution. We say that a transition $\boldsymbol{s} \xrightarrow{a} \boldsymbol{\mu}$ *involves* the $i$th component if $s^i \xrightarrow{a} \mu^i$.

**Definition 5.** *Given $n$ games in normal form $G^i = \langle S^i, (S^i_\Diamond, S^i_\Box), \varsigma^i, \mathcal{A}^i, \longrightarrow^i \rangle$, $i \in I$, their* composition *is the game $\|_{i\in I} G^i \overset{\text{def}}{=} \langle S, (S_\Diamond, S_\Box), \boldsymbol{\varsigma}, \mathcal{A}, \longrightarrow \rangle$, where*

- *$S \subseteq S_\Diamond \cup S_\Box$, with $S_\Box \subseteq S^1_\Box \times \cdots \times S^n_\Box$, and $S_\Diamond \subseteq \{\boldsymbol{s} \in S^1 \times \cdots \times S^n \mid \exists! \iota \,.\, s^\iota \in S^\iota_\Diamond\}$ inductively defined to contain the states reachable from the initial distribution through the transition relation;*
- *$\boldsymbol{\varsigma} = \varsigma^1 \times \cdots \times \varsigma^n$; (note that, due to the normal form, $\boldsymbol{\varsigma}(\boldsymbol{s}) > 0$ only if $\boldsymbol{s} \in S_\Box$)*
- *$\mathcal{A} = \bigcup_{i=1}^n \mathcal{A}^i$;*
- *The transition relation $\longrightarrow$ is defined such that*
  - *$\boldsymbol{s} \xrightarrow{a} \boldsymbol{\mu}$ for $a \neq \tau$ if*

    *(C1) at least one component is involved;*

    *(C2) the components involved in the transition are exactly those having $a$ in their action alphabet;*

    *(C3) for the uninvolved components $j$ (equivalently, that do not have $a$ in their action alphabet), the state remains the same ($\mu^j = s^j$);*

*(C4) if $\boldsymbol{s}$ is a Player $\Diamond$ state then its only Player $\Diamond$ component $G^\iota$ is involved in the transition; and*

- $\boldsymbol{s} \xrightarrow{\tau} \boldsymbol{t}$ *if only one component $G^i$ is involved, $\boldsymbol{s} \in S_\Box$, and s.t. $\mathsf{En}(\boldsymbol{t}) \neq \emptyset$.*

We take the view that identity of the players must be preserved through composition to facilitate synthesis, and thus Player $\Diamond$ actions of the individual components are controlled by a single Player $\Diamond$ in the composition. Player $\Box$ in the composition acts as a scheduler, controlling which component advances and, in Player $\Box$ states, selecting among available actions, whether synchronised or not. Synchronisation in Player $\Diamond$ states means that Player $\Diamond$ in one component may indirectly control some Player $\Box$ actions in another component. In particular, we can impose assume-guarantee contracts at the component level in the following sense. Player $\Diamond$ of different components can cooperate to achieve a common goal: in one component Player $\Diamond$ satisfies the goal $B$ under an assumption $A$ on its environment behaviour (i.e. $A \rightarrow B$), while Player $\Diamond$ in the other component ensures that the assumption is satisfied, against all Player $\Box$ strategies.

Our game composition is both associative and commutative, facilitating a modular model development, and is closely related to PA composition [16], with the condition *(C4)* added. As PAs are just games without Player $\Diamond$ states, the game composition restricted to PAs is the same as classical PA composition. The condition $\mathsf{En}(\boldsymbol{t}) \neq \emptyset$ for $\tau$-transitions ensures that a Player $\Diamond$ state is never entered if it were to result in deadlock introduced by the normal form transformation. Deadlocks that were present before the transformation are still present in the normal form. In the composition of normal form games, $\tau$-transitions are only enabled in Player $\Box$ states, and Player $\Diamond$ states are only reached by such transitions; hence, composing normal form games yields a game in normal form.

In Figure 1, the game on the right is the composition of the normal forms of the two games on the left. The actions "f" and "s" are synchronised and controlled by Player $\Diamond$ in $\underline{s}u$. The "d" action is synchronised and controlled by Player $\Box$ in both components, and so it is controlled by Player $\Box$ in the composition, in $tu$. The action "t" is not synchronised, and thus available in $t\overline{v}$ and $tu$; it is, however, not available in $t\underline{v}$, as it is a Player $\Diamond$ state controlled by $\underline{v}$. The action "c" is also not synchronised, and is available in $t\underline{v}$. The "r" action is synchronised; it is available both in $t$ and in $\underline{v}$, and hence also in $t\underline{v}$.

Constructing the composition of $n$ components of size $|S|$ clearly requires time $\mathcal{O}(|S|^n)$. In strategy synthesis, the limiting factor is that applying the method on a large product game may be computationally infeasible. For example, the synthesis methods for multi-objective queries of [5] that we build upon are exponential in the number of objectives and polynomial in the size of the state space, and the theoretical bounds can be impractical even for small systems (see [6]). To alleviate such difficulties we focus on compositional strategy synthesis.

## 3.2 Strategy Composition

For compositional synthesis, we assume the following compatibility condition on component games: we require that actions controlled by Player $\Diamond$ in one game are enabled and fully controlled by Player $\Diamond$ in the composition.

**Definition 6.** *Games $G^1, \ldots, G^n$ are* compatible *if, for every* Player $\Diamond$ *state* $\boldsymbol{s} \in S_\Diamond$ *in the composition with* $s^\iota \in S^\iota_\Diamond$*, if* $s^\iota \xrightarrow{a}{}^\iota \mu^\iota$ *then there is exactly one distribution* $\boldsymbol{\nu}$*, denoted by* $\langle \mu^\iota \rangle_{\boldsymbol{s},a}$*, such that* $\boldsymbol{s} \xrightarrow{a} \boldsymbol{\nu}$ *and* $\nu^\iota = \mu^\iota$*. (That is, for* $i \neq \iota$ *such that* $a \in \mathcal{A}^i$*, there exists exactly one* $a$*-transition enabled in* $s^i$*.)*

Our compatibility condition is analogous to that for single-threaded interface automata [8]. It remains to be seen if this condition can be relaxed without affecting preservation properties of the winning conditions.

**Composing Strategies.** Given a path $\lambda$ of a composed game $\mathcal{G} = \|_{i \in I} G^i$, for each individual component $G^i$ one can retrieve the corresponding path $[\lambda]^i$ that contains precisely the transitions $G^i$ is involved in. The *projection* $[\cdot]^i : \Omega^+_{\mathcal{G}} \to \Omega^+_{G^i}$ is defined inductively so that, for all states $\boldsymbol{t} \in S$ and paths $\lambda(a, \boldsymbol{\mu})\boldsymbol{t} \in \Omega^+_{\mathcal{G}}$ (with possibly $a = \tau$), we have $[\boldsymbol{s}]^i \stackrel{\text{def}}{=} s^i$; and inductively that $[\lambda(a, \boldsymbol{\mu})\boldsymbol{t}]^i \stackrel{\text{def}}{=} [\lambda]^i(a, \mu^i)t^i$ if $\mathsf{last}(\lambda)^i \xrightarrow{a}{}^i \mu^i$, and $[\lambda]^i$ otherwise.

Recall that a Player $\Diamond$ state $\boldsymbol{s}$ of the composed game has exactly one component $s^\iota$ that is a Player $\Diamond$ state in $G^\iota$; we say that the $\iota$th Player $\Diamond$ *controls* $\boldsymbol{s}$. Given a Player $\Diamond$ strategy for each component, the strategy for the composed game plays the strategy of the Player $\Diamond$ controlling the respective states.

**Definition 7.** *Let* $\sigma^i_\Diamond$*,* $i \in I$*, be* Player $\Diamond$ *strategies for compatible games. Their* composition*,* $\sigma_\Diamond = \|_{i \in I} \sigma^i_\Diamond$*, is defined such that* $\sigma_\Diamond(\lambda)(a, \langle \mu^\iota \rangle_{\boldsymbol{s},a}) \stackrel{\text{def}}{=} \sigma^\iota_\Diamond([\lambda]^\iota)(a, \mu^\iota)$ *for all* $\lambda \in \Omega^+_{\mathcal{G}}$ *with* $\boldsymbol{s} = \mathsf{last}(\lambda) \in S_\Diamond$*.*

From this definition, strategy composition is clearly associative. Note that, for each choice, the composed strategy takes into account the history of only one component, which is less general than using the history of the composed game.

### 3.3 Properties of the Composition

We now show that synthesising strategies for compatible individual components is sufficient to obtain a composed strategy for the composed game.

**Functional Simulations.** We introduce functional simulations, which are a special case of classical PA simulations [16], and show that they preserve winning conditions over traces. Intuitively, a PA $M'$ functionally simulates a PA $M$ if all behaviours of $M$ are present in $M'$, and if strategies translate from $M$ to $M'$.

Given a distribution $\mu$, and a partial function $f : S \to S'$ defined on the support of $\mu$, we write $\overline{f}(\mu)$ for the distribution defined by $\overline{f}(\mu)(s') \stackrel{\text{def}}{=} \sum_{f(s)=s'} \mu(s)$. A *functional simulation* from a PA $M$ to a PA $M'$ is a partial function $f : S \to S'$ such that $\overline{f}(\varsigma) = \varsigma'$, and if $s \xrightarrow{a} \mu$ in $M$ then $f(s) \xrightarrow{a}{}' \overline{f}(\mu)$ in $M'$.

**Lemma 1.** *Given a functional simulation from a PA* $M$ *to a PA* $M'$ *and a specification* $\varphi$ *defined on traces, for every strategy* $\sigma_\Box \in \Sigma^M_\Box$ *there is a strategy* $\sigma'_\Box \in \Sigma^{M'}_\Box$ *such that* $(M')^{\sigma'_\Box} \models \varphi \Leftrightarrow M^{\sigma_\Box} \models \varphi$*.*

**Key Lemma.** The PA $\|_{i \in I} (G^i)^{\sigma^i_\Diamond}$ is constructed by first unfolding each component, applying the Player $\Diamond$ strategies, and then composing the resulting PAs, while the PA $(\|_{i \in I} G^i)^{\|_{i \in I} \sigma^i_\Diamond}$ is constructed by first composing the individual components, then unfolding, and applying the composed Player $\Diamond$ strategy. The following lemma justifies, via the existence of a functional simulation, that composing Player $\Diamond$ strategies preserves the trace distribution between such PAs, and hence yields a fully compositional approach.

**Lemma 2.** *Given compatible games $G^i$, $i \in I$, and respective Player $\Diamond$ strategies $\sigma^i_\Diamond$, there is a functional simulation from $(\|_{i \in I} G^i)^{\|_{i \in I} \sigma^i_\Diamond}$ to $\|_{i \in I} (G^i)^{\sigma^i_\Diamond}$.*

In general, there is no simulation in the other direction, as in the PA composition one can no longer distinguish states in the induced PA that were originally Player $\Diamond$ states, and so condition *(C4)* of the composition is never invoked.

## 4 Compositional Synthesis

Applying strategies synthesised for games to obtain induced PAs allows us to reuse compositional rules for PAs. Using Lemma 2, we can then lift the result back into the game domain. This process is justified in Theorem 1 below.

### 4.1 Composition Rules

We suppose the designer is supplying a game $\mathcal{G} = \|_{i \in I} G^i$ composed of *atomic games $G_i$*, together with specifications defined on traces $\varphi_i$, $i \in I$, and show how, using our framework, strategies $\sigma^i_\Diamond$ synthesised for $G^i$ and $\varphi_i$ can be composed to a strategy $\sigma_\Diamond = \|_{i \in I} \sigma^i_\Diamond$ for $\mathcal{G}$, satisfying a specification $\varphi$ defined on traces.

**Theorem 1.** *Given a rule $\mathfrak{P}$ for PAs $\mathcal{M}_i$ and specifications $\varphi^i_j$ and $\varphi$ defined on traces, then the rule $\mathfrak{G}$ holds for all Player $\Diamond$ strategies $\sigma^i_\Diamond$ of compatible games $G^i$ with the same action alphabets as the corresponding PAs, where*

$$\mathfrak{P} \equiv \frac{\mathcal{M}_i \models \varphi^i_j \quad 1 \leq j \leq m \quad i \in I}{\|_{i \in I} \mathcal{M}_i \models \varphi,} \qquad \textit{and} \qquad \mathfrak{G} \equiv \frac{(G^i)^{\sigma^i_\Diamond} \models \bigwedge_{j=1}^m \varphi^i_j \quad i \in I}{(\|_{i \in I} G^i)^{\|_{i \in I} \sigma^i_\Diamond} \models \varphi.}$$

Theorem 1 enables the compositional synthesis of strategies in an automated way. First, synthesis is performed for atomic components $G^i$, $i \in I$, by obtaining for each $i$ a Player $\Diamond$ strategy $\sigma^i_\Diamond$ for $G^i \models \bigwedge_{j=1}^m \varphi^i_j$. We apply $\mathfrak{P}$ with $\mathcal{M}_i \stackrel{\text{def}}{=} (G^i)^{\sigma^i_\Diamond}$ to deduce that $\varphi$ holds in $\|_{i=1}^n (G^i)^{\sigma^i_\Diamond}$ and, using Lemma 1 and 2, that $\|_{i \in I} \sigma^i_\Diamond$ is a winning strategy for Player $\Diamond$ in $\|_{i=1}^n G^i$. The rules can be applied recursively, making use of associativity of the game and strategy composition.

### 4.2 Multi-Objective Queries

In this section we leverage previous work on compositional verification for PAs in order to compositionally synthesise strategies for games.

**Reward and LTL Objectives.** The *expected value* of a function $\rho : \mathcal{A}^* \to \mathbb{R}_{\pm\infty}$ over traces in a DTMC $D$ is $\mathbb{E}_D[\rho] \stackrel{\text{def}}{=} \lim_{n \to \infty} \sum_{w \in \mathcal{A}^n} \Pr_D(w)\rho(w)$, if the limit exists in $\mathbb{R}_{\pm\infty}$. We denote by $\mathbb{E}_G^{\sigma_\Diamond,\sigma_\Box}$ (resp. $\mathbb{E}_M^{\sigma_\Box}$) the expected value in a game $G$ (resp. PA $M$) under the respective strategies. A *reward structure* of a game with actions $\mathcal{A}$ is a function $r : \mathcal{A}_r \to \mathbb{Q}$, where $\mathcal{A}_r \subseteq \mathcal{A}$. Given a reward structure $r$ such that either $r(a) \leq 0$ or $r(a) \geq 0$ for all actions $a$ occurring infinitely often on a path, the *total reward* for a trace $w = a_0 a_1 \ldots$ is $rew(r)(w) \overset{\text{def}}{=} \lim_{t\to\infty} \sum_{i=0}^{t} r(a_i)$, which is measurable thanks to the restrictions imposed on $r$. Given reward structures $r_j$, $j \in J$, for all $a \in \bigcup_{j\in J} \mathcal{A}_{r_j}$ we let $(\sum_{j\in J} r_j)(a)$ be the sum of the $r_j(a)$ that are defined for $a$.

To express LTL properties over traces, we use the standard LTL operators (cf. [15]); in particular, the operators $\mathsf{F}$ and $\mathsf{G}$ stand for *eventually* and *always*, respectively. For a DTMC $D$, and an LTL formula $\Xi$ over actions $\mathcal{A}_\Xi$, define $\Pr_D(\Xi) \overset{\text{def}}{=} \Pr_D(\{\lambda \in \Omega_D \mid \text{PROJ}_{\mathcal{A}\setminus\mathcal{A}_\Xi}(\mathsf{trace}(\lambda)) \models \Xi\})$, that is, the measure of infinite paths with traces satisfying $\Xi$, where actions not in $\mathcal{A}_\Xi$ are disregarded.

A *reward* (resp. *LTL*) *objective* is of the form $r \blacktriangleright v$ (resp. $\Xi \blacktriangleright v$), where $r$ is a reward structure, $\Xi$ is an LTL formula, $v \in \mathbb{Q}$ is a bound, and $\blacktriangleright \in \{\geq, >\}$. A reward objective $r \blacktriangleright v$ (resp. LTL objective $\Xi \blacktriangleright v$) is true in a game $G$ under a pair of strategies $(\sigma_\Diamond, \sigma_\Box)$ if and only if $\mathbb{E}_G^{\sigma_\Diamond,\sigma_\Box}[rew(r)] \blacktriangleright v$ (resp. $\Pr_G^{\sigma_\Diamond,\sigma_\Box}(\Xi) \blacktriangleright v$), and similarly for PAs and DTMCs. Minimisation of rewards can be expressed by reverting signs.

**Multi-Objective Queries.** A *multi-objective query* (MQ) $\varphi$ is a Boolean combination of reward and LTL objectives, and its truth value is defined inductively on its syntax. An MQ $\varphi$ is a *conjunctive query* (CQ) if it is a conjunction of objectives. Given an MQ with bounds $v_1, v_2, \ldots$, call $\boldsymbol{v} = (v_1, v_2, \ldots)$ the *target*. Denote by $\varphi[\boldsymbol{x}]$ the MQ $\varphi$, where, for all $i$, $r_i \blacktriangleright v_i$ is replaced by $r_i \blacktriangleright x_i$, and $\Xi_i \blacktriangleright v_i$ is replaced by $\Xi_i \blacktriangleright x_i$. Given a game $G$ (resp. PA $M$) we write $G^{\sigma_\Diamond,\sigma_\Box} \models \varphi$ (resp. $M^{\sigma_\Box} \models \varphi$), if the query $\varphi$ evaluates to true under the respective strategies. We write $M \models \varphi$ if $M^{\sigma_\Box} \models \varphi$ for all $\sigma_\Box \in \Sigma_\Box^M$. We say that an MQ $\varphi$ is *achievable* in a game $G$ if there is a $\mathsf{Player}\ \Diamond$ strategy $\sigma_\Diamond$ such that $G^{\sigma_\Diamond} \models \varphi$, that is, $\sigma_\Diamond$ is winning for $\varphi$ against all possible $\mathsf{Player}\ \Box$ strategies. We require that expected total rewards are bounded, that is, we ask for any reward structure $r$ in an MQ $\varphi$ that $G^{\sigma_\Diamond,\sigma_\Box} \models r < \infty \wedge r > -\infty$ for all $\sigma_\Diamond$ and $\sigma_\Box$.

**Fairness.** Since PA rules as used in Theorem 1 often include fairness conditions, we recall here the concept of unconditional process fairness based on [1]. Given a composed PA $\mathcal{M} = \|_{i\in I} M^i$, a strategy $\sigma_\Box$ is *unconditionally fair* if $\mathcal{M}^{\sigma_\Box} \models \mathsf{u}$, where $\mathsf{u} \overset{\text{def}}{=} \bigwedge_{i\in I} \mathsf{GF}\,\mathcal{A}_i \geq 1$, that is, each component makes progress infinitely often with probability 1. We write $\mathcal{M} \models^{\mathsf{u}} \varphi$ if, for all unconditionally fair strategies $\sigma_\Box \in \Sigma_\Box$, $\mathcal{M}^{\sigma_\Box} \models \varphi$; this is equivalent to $\mathcal{M} \models \mathsf{u} \to \varphi$ (the arrow $\to$ stands for the standard logical implication), and so MQs can incorporate fairness.

**Applying Theorem 1.** In particular, for the premises in Theorem 1 we can use the compositional rules for PAs developed in [12], which are stated for MQs. Thus, the specification $\varphi$ for the composed game can, for example, be a CQ, or a summation of rewards, among others. Unconditional fairness corresponds precisely to the fairness conditions used in the PA rules of [12]. When the PA rules include fairness assumptions, note that, for a single component, unconditional fairness is equivalent to only requiring deadlock-freedom.

## 4.3 Compositional Pareto Set Computation

We describe in this section how to pick the targets of the objectives $\varphi_i$ in compositional rules, such as those in Theorem 1, so that $\varphi$ is achievable.

**Pareto Sets.** Given an MQ $\varphi$ with $N$ objectives, vector $\boldsymbol{v} \in \mathbb{R}^N$ is a *Pareto vector* if and only if $\varphi[\boldsymbol{v} - \varepsilon]$ is achievable for all $\varepsilon > 0$, and $\varphi[\boldsymbol{v} + \varepsilon]$ is not achievable for any $\varepsilon > 0$. The downward closure of the set of all such vectors is called a *Pareto set*, where the downward closure of a set $X$ is defined as $\mathsf{dwc}(X) \overset{\text{def}}{=} \{\boldsymbol{y} \in \mathbb{R}^N \mid \exists \boldsymbol{x} \in X \,.\, \boldsymbol{x} \geq \boldsymbol{y}\}$. Given $\varepsilon > 0$, an *$\varepsilon$-approximation of a Pareto set* $P$ is a set of vectors $Q$ satisfying that, for any $\boldsymbol{w} \in Q$, there is a vector $\boldsymbol{v} \in P$ such that $\|\boldsymbol{v} - \boldsymbol{w}\| \leq \varepsilon$, and for every $\boldsymbol{v} \in P$ there is a vector $\boldsymbol{w} \in Q$ such that $\|\boldsymbol{v} - \boldsymbol{w}\| \leq \varepsilon$, where $\|\cdot\|$ is the Manhattan norm.

**Under-approximating Pareto Sets.** We can compositionally compute an under-approximation of the Pareto set for $\varphi$, which we illustrate in Figure 3.

Consider $N$ reward structures, $r_1, \ldots, r_N$, and objectives $\varphi^i$, $i \in I$, over these reward structures for respective games $G^i$, as well as an objective $\varphi$, over the same reward structures, for the composed game $\mathcal{G} = \|_{i \in I} G^i$. Note that, for each $1 \leq j \leq N$, the reward structure $r_j$ may be present in several objectives $\varphi^i$. Let $P^i$ be the Pareto set for $G^i \models \varphi^i$, for $i \in I$, and so each point $\boldsymbol{v}^{(i)} \in P^i$ represents a target vector for the MQ $\varphi^i[\boldsymbol{v}^{(i)}]$ achievable in the game $G^i$.

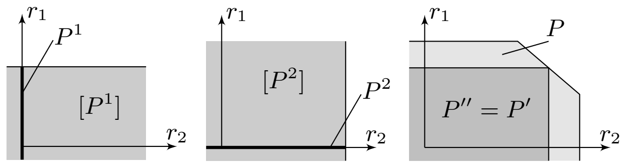
For a Pareto set $P^i$, define the *lifting* $[P^i]$ to all $N$ reward structures by $[P^i] \overset{\text{def}}{=} \{\boldsymbol{v} \in \mathbb{R}^N_{\pm\infty} \mid$ the coordinates of $\boldsymbol{v}$ appearing in $\varphi^i$ are in $P^i\}$. The set $P' \overset{\text{def}}{=} \cap_{i \in I} [P^i]$ is the set of target vectors for all $M$ reward structures, which are consistent with achievability of all objectives $\varphi^i$ in the respective games. The projection[1] $P''$ of $P'$ onto the space of reward structures appearing in $\varphi$ then yields an under-approximation of the Pareto set $P$ for $\varphi$ in the composed game $\mathcal{G}$, that is, $P'' \subseteq P$. A point $\boldsymbol{v} \in P''$ can be achieved by instantiating the objectives $\varphi^i$ with any targets $\boldsymbol{v}^{(i)}$ in $P'$ that match $\boldsymbol{v}$.

## 4.4 Compositional MQ Synthesis

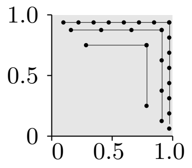
We now describe our compositional strategy synthesis method.

**MQ Synthesis for Component Games.** A game is *stopping* if, under any strategy pair, with probability 1 a part of the game is reached where the properties no longer change, see Appendix E. A strategy is *$\varepsilon$-optimal* for an MQ $\varphi$ with target $\boldsymbol{v}$ if it achieves $\varphi[\boldsymbol{v} - \varepsilon]$ for all $\varepsilon > 0$. From [5] we have that, for atomic stopping games with MQs, it is decidable whether an $\varepsilon$-optimal strategy exists (optimal strategies may not exist), and, $\varepsilon$-optimal strategies can be represented finitely using stochastic memory update (see Appendix A).

[1] More generally, if $\varphi$ contains items such as $r_i + r_j \blacktriangleright v_i + v_j$, as in the (SUM-REWARD) rule of [12], a new dimension is introduced combining the rewards as required.



(a) MQs $\varphi^1 = r_1 \geq v_1$ (left), $\varphi^2 = r_2 \geq v_2$ (center), and $\varphi = r_1 \geq v_1 \wedge r_2 \geq v_2$ (right). Suppose the Pareto sets $P^i$ for $\varphi^i$, $i \in \{1, 2\}$, consist of the thick lines, unbounded towards $-\infty$. Their liftings $[P^i]$ are shown in dark grey. $P''$ under-approximates the Pareto set $P$ for $\varphi$.



(b) Each dot is a weight vector, where the accuracy $a$ is one of $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$.

Fig. 3: Compositional Pareto set computation (a); weight vector selection (b).

We compute a strategy for an MQ in CNF $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} r_{i,j} \geq v_{i,j}$ by implementing value iteration based on [5]. First, set an initial accuracy, $a \leftarrow \frac{1}{2}$. For each $0 \leq i < n$, select a corresponding $0 \leq j_i < m$. Then uniformly iterate over *weights* $\boldsymbol{x}_i \in [0, 1 - a/2]^m$ by gridding with accuracy $a$, keeping the $j_i$th dimension constant at $1 - a/2$. The pattern of selected vectors is shown for $m = 2$ dimensions in Figure 3(b). At each selection of $\boldsymbol{x}_i$, check, using the CQ algorithm of [6], if $\bigwedge_{i=1}^{n} (\sum_{j=1}^{m} \boldsymbol{x}_i^j \cdot r_{i,j} \geq \sum_{j=1}^{m} \boldsymbol{x}_i^j \cdot v_{i,j})$ is realisable, and, if so, return the winning strategy. Otherwise, if all options for selecting $j_i$ are exhausted, refine the accuracy to $a \leftarrow \frac{a}{2}$ and repeat.

Every point $\boldsymbol{y} \in \mathbb{R}^n$ in a CQ Pareto set with weights $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^m_{\geq 0}$ corresponds to intersection of half-spaces $\boldsymbol{x}_i \cdot \boldsymbol{z} \geq y^i$; the union over all choices of weight vectors is the $\varepsilon$-approximation of the corresponding MQ Pareto set.

**MQ Synthesis for Composed Games.** Our method for compositional strategy synthesis, based on synthesis for atomic games, is summarised as follows:

(S1) **User Input:** A composed game $\mathcal{G} = \|_{i \in I} G^i$, MQs $\varphi^i$, $\varphi$, and matching PA rules for use in Theorem 1.
(S2) **First Stage:** Obtain $\varepsilon$-approximate Pareto sets $P^i$ for $G^i \models \varphi^i$, and compute $P''$ as in Section 4.3.
(S3) **User Feedback:** Pick targets $\boldsymbol{v}$ for $\varphi$ from $P''$ and matching targets $\boldsymbol{v}^{(i)}$ for $\varphi^i$ from $P^i$.
(S4) **Second Stage:** Synthesise strategies $\sigma_\Diamond^i$, for $G^i \models \varphi^i[\boldsymbol{v}^{(i)}]$, and compose them using Definition 10 in Appendix A.
(S5) **Output:** A composed strategy $\|_{i \in I} \sigma_\Diamond^i$, winning for $\mathcal{G} \models \varphi[\boldsymbol{v}]$ by Theorem 1.

Steps (S1), (S4) and (S5) are sufficient if the targets are known, while (S2) and (S3) are an additional feature enabled by the Pareto set computation.

## 4.5 Case Study

We illustrate our approach with an example, briefly outlined here; see Appendix F for more detail. We model an Internet-enabled fridge that autonomously

Table 1: Run time comparison between compositional and monolithic strategy synthesis. For the CQ value iteration (cf. [6]) we use 60, 400 and 200 iterations for the fridge, trader and composed model, respectively. Computations were done on a 2.8 GHz Intel® Xeon® CPU, with 32 GB RAM, under Fedora 14.

| Traders ($n$) | State Space Size | | | Running Time [s] | | | | |
|---|---|---|---|---|---|---|---|---|
| | $F$ | $T_i$ | $C$ | Composition | Compositional | | Monolithic | |
| | | | | | Pareto Set | Strategies | Pareto Set | Strategy |
| 1 | 11 | 7 | 17 | 0.006 | 31.0 | 0.2 | 10.9 | 0.26 |
| 2 | 23 | 7 | 119 | 0.1 | 400.0 | 0.37 | 6570.0 | 161.0 |
| 3 | 39 | 7 | 698 | 2.0 | 407.0 | 2.4 | $> 3$h | – |
| 4 | 59 | 7 | 3705 | 75.0 | 4870.0 | 1.4 | $> 5$h | – |

selects between different digital agents selling milk whenever restocking is needed. We compute the Pareto sets and strategies in a prototype implementation as an extension of PRISM-games [4].

The fridge repeatedly invites offers from several traders, and decides whether to accept or decline the offers, based on the quality of each offer. The objective is for the traders to maximise the unit price, and for the fridge to maximise the amount of milk it purchases. For $n$ traders $T_i$, $1 \leq i \leq n$, and a fridge $F$, denote the composition $C \stackrel{\text{def}}{=} (\|_{i=1}^{n} T_i) \parallel F$. We use the following reward objectives $O_i \equiv$ "offers made by $T_i$" $\geq v_{o_i}$, $A_i \equiv$ "offers of $T_i$ accepted" $\geq v_{a_i}$, $Q_i \equiv$ "quality of offers made by $T_i$" $\geq v_{q_i}$, $\$_i \equiv$ "unit price of $T_i$" $\geq v_{\$_i}$, and $\# \equiv$ "amount of milk obtained by $F$" $\geq v_{\#}$, and synthesise strategies as explained in Section 4.1 according to the rule:

$$\frac{F \models \bigwedge_{j=1}^{n}(O_j \to A_j) \wedge (\bigwedge_{j=1}^{n} Q_j \to \#) \qquad T_i \models A_i \to (Q_i \wedge \$_i) \qquad 1 \leq i \leq n}{C \models \bigwedge_{j=1}^{n}(O_j \to \$_j) \wedge (\bigwedge_{j=1}^{n} O_j \to \#).}$$

The main advantages of compositional synthesis are a dramatic improvement in efficiency and the compactness of strategies, as indicated in Table 1. In general, the strategies are randomised and history dependent. For the case of two traders, with the target that we selected, we generate a strategy where the traders make an expensive offer in the first round with probability 0.91, but from then on consistently make less expensive bulk offers.

## 5 Conclusion

We have defined a synchronising composition for stochastic games, and formulated a compositional approach to controller synthesis by leveraging techniques for compositional verification of PAs [12] and multi-objective strategy synthesis of [5,6]. We have extended the implementation of [6] to synthesise $\varepsilon$-optimal strategies for two-player stochastic games for total expected reward objectives in conjunctive normal form. We intend to investigate relaxing the compatibility condition and consider notions of fairness weaker than unconditional fairness to broaden the applicability of our methods.

**Acknowledgements.** The authors thank Dave Parker and Vojtěch Forejt for helpful discussions. The authors are partially supported by ERC Advanced Grant VERIWARE.

## References

1. C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *ATVA'09*, volume 5799 of *LNCS*, pages 135–150. Springer.
2. M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Phil. Trans. R. Soc. A*, 368(1928):4649–4672, 2010.
3. K. Chatterjee, M. Jurdziński, and T. Henzinger. Simple stochastic parity games. In *CSL'03*, volume 2803 of *LNCS*, pages 100–113. Springer.
4. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *TACAS'13*, volume 7795 of *LNCS*, pages 185–191. Springer.
5. T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In *MFCS'13*, volume 8087 of *LNCS*, pages 266–277. Springer.
6. T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *QEST'13*, volume 8054 of *LNCS*, pages 322–337. Springer.
7. L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched PIOA: Parallel composition via distributed scheduling. *TCS*, 365(1–2):83–108, 2006.
8. L. de Alfaro and T.A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001.
9. J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 1996.
10. M. Gelderie. Strategy composition in compositional games. In *ICALP'13*, volume 7966 of *LNCS*, pages 263–274. Springer.
11. S. Ghosh, R. Ramanujam, and S. Simon. Playing extensive form games in parallel. In *CLIMA'10*, volume 6245 of *LNCS*, pages 153–170. Springer.
12. M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Information and Computation*, 232:38–65, 2013.
13. M. Kwiatkowska and D. Parker. Automated verification and strategy synthesis for probabilistic systems. In *ATVA'13*, volume 8172 of *LNCS*, pages 5–22. Springer.
14. N. Ozay, U. Topcu, R.M. Murray, and T. Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *ICCPS'11*, pages 45–54. IEEE.
15. A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE.
16. R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
17. Lloyd S Shapley. Stochastic games. *Proc. Natl. Acad. Sci. USA*, 39(10):1095, 1953.
18. A. Simaitis. *Automatic Verification of Competitive Stochastic Systems*. PhD thesis, Univ. Oxford, 2013.
19. A. Sokolova and E.P. de Vink. Probabilistic automata: system types, parallel composition and comparison. In *VOSS'04*, volume 2925 of *LNCS*, pages 1–43. Springer.

## A SU strategies

We introduce stochastic memory update (SU) strategies, since we use them in the proof of Lemma 1, and because the synthesis methods of [6], which we employ in Section 4, produce such strategies.[2] SU strategies keep internal memory that can be updated stochastically. This formulation is equally powerful as the standard formulation used in the main text, but SU strategies can be exponentially more succinct than the standard formulation above [18]. Note, however, that finite memory SU strategies are strictly more powerful than finite memory strategies in the standard formulation, see [5].

**Definition 8.** *A* stochastic update (SU) *strategy* $\sigma_\#$ *of* Player $\#$ *is a tuple* $\langle \mathfrak{M}, \sigma_\#^n, \sigma_\#^u, \alpha \rangle$*, where*

- $\mathfrak{M}$ *is a countable set of* memory elements*;*
- $\sigma_\#^u : \mathfrak{M} \times S \cup S_\bigcirc \to \mathcal{D}(\mathfrak{M})$ *is a* memory update function*;*
- $\sigma_\#^n : S_\# \times \mathfrak{M} \to \mathcal{D}(S_\bigcirc)$ *is a* next move function *s.t.* $\sigma_\#^n(s,m)(a,\mu) > 0$ *only if* $s \xrightarrow{a} \mu$*; and*
- $\alpha : S \to \mathcal{D}(\mathfrak{M})$ *is an* initial distribution *on the memory.*

As the game proceeds, in Player $\#$ states, the next move $\sigma_\#^n(s,m)$ is picked stochastically based on the current memory $m$ and the current state $s$. Memory is updated both when selecting a move $(a,\mu)$, and when entering a state $s$ (sampled from $\mu$), and the next memory element $\sigma_\#^u(m,\gamma')$ is picked stochastically based on the current memory element $m$ and the *next* state or move $\gamma'$. The memory $\mathfrak{M}$ is considered internal and not visible to the other player.

The semantics of SU strategies are defined by inducing a DTMC for the strategies of both players at the same time, and, equivalently, one can define the semantics as follows, by transforming SU strategies into the standard formulation.[3] Note that the induced PA cannot be defined using SU strategies, since this would reveal the internal memory of Player $\Diamond$ to Player $\Box$.

**Definition 9.** *Given an SU strategy* $\sigma_\#$ *define the strategy* $\widehat{\sigma_\#}$ *for all* $\lambda \in \Omega_{G,\#}^+$ *and transitions* $s \xrightarrow{a} \mu$ *for which* $s = \mathsf{last}(\lambda)$*, by letting* $\widehat{\sigma_\#}(\lambda)(a,\mu) \overset{\text{def}}{=} \sum_{m \in \mathfrak{M}} d(\lambda)(m)\sigma_\#^n(s,m)(a,\mu)$*, where* $d(\lambda)$ *is defined inductively by*

- $d(s) \overset{\text{def}}{=} \alpha(s)$*; and*
- $d(\lambda(a,\mu)s')(m) \overset{\text{def}}{=} \sum_{n,n' \in \mathfrak{M}} d(\lambda)(n)\sigma_\#^u(n,(a,\mu))(n')\sigma_\#^u(n',s')(m)$*.*

Intuitively, the function $d$ maps a path to the distribution of memory elements after seeing the path. For an SU strategy $\sigma_\Diamond$, we hence write $G^{\sigma_\Diamond}$ for $G^{\widehat{\sigma_\Diamond}}$.

[2] T. Brázdil, V. Broček, K. Chatterjee, V. Forejt, and A. Kučera. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In LICS'11, pages 33–42.

[3] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, A. Trivedi, and M. Ummels. Playing Stochastic Games Precisely. Technical Report RR-12-03. University of Oxford, 2013.

**Composing SU Strategies.** For compositional synthesis, after obtaining SU strategies, we compose these strategies directly without applying Definition 9, so as not to sacrifice their compactness.

The memory update function of the composed SU strategy ensures that the memory in the composition is the same as if the SU strategies were applied to the games individually. Technically, only the *next* state or move is available to the memory update function, and so, when moving to a state, it can no longer be recovered which components synchronised, unless the memory elements are unique for each state or move. We assume, therefore, w.l.o.g. that we can recover the action $a$ that is part of the current move from the current memory element $m$, written $\mathsf{act}(m) \overset{\text{def}}{=} a$.

We slightly abuse notation and write $\boldsymbol{\gamma}$ for a state $\boldsymbol{s}$ or move $(a, \boldsymbol{\mu})$ of the composition, and denote by $\gamma^i = (a, \mu^i)$ the $i$th component of a move. Define the set of components that synchronise on a transition labelled by $a \neq \tau$ by $\mathsf{Sync}(a) \overset{\text{def}}{=} \{i \in I \,|\, a \in \mathcal{A}_i\}$. In the following definition, $K$ denotes the set of indices of components that update their memory during a transition of the composed game.

**Definition 10.** *Let $\sigma^i_\Diamond = \langle \mathfrak{M}^i, \sigma^{u,i}_\Diamond, \sigma^{n,i}_\Diamond, \alpha^i \rangle$, $i \in I$, be SU Player $\Diamond$ strategies for compatible games. Their* composition *is $|_{i \in I} \sigma^i_\Diamond \overset{\text{def}}{=} \langle \mathfrak{M}, \sigma^u_\Diamond, \sigma^n_\Diamond, \alpha \rangle$, where*

- $\mathfrak{M} = \prod_{i \in I} \mathfrak{M}^i$;
- $\sigma^n_\Diamond(\boldsymbol{s}, \boldsymbol{m})(a, \langle \mu^\iota \rangle_{\boldsymbol{s},a}) = \sigma^{n,\iota}_\Diamond(s^\iota, m^\iota)(a, \mu^\iota)$ *if $s^\iota \in S^\iota_\Diamond$ (the component $G^\iota$ controlling $\boldsymbol{s}$ chooses);*
- *the memory update function is defined as the product of the memory updates in the components selected by $K$:*

$$\sigma^u_\Diamond(\boldsymbol{m}, \boldsymbol{\gamma})(\boldsymbol{m}_+) \overset{\text{def}}{=} \prod_{i \in K} \sigma^{u,i}_\Diamond(m^i, \gamma^i)(m^i_+) \prod_{i \notin K} \delta_{m^i = m^i_+}$$

  *where $\delta$ is the Kronecker delta, and where*

$$K \overset{\text{def}}{=} \begin{cases} \mathsf{Sync}(a) & \text{if } \boldsymbol{\gamma} = (a, \boldsymbol{\mu}) \in S_\bigcirc \text{ and } a \neq \tau \\ \{\iota\} & \text{if } \boldsymbol{\gamma} = (\tau, \boldsymbol{s}) \in S_\bigcirc, \text{ s.t. } s^\iota \in S^\iota_\Diamond \\ \mathsf{Sync}(\mathsf{act}(\boldsymbol{m})) & \text{if } \boldsymbol{\gamma} \in S_\Box \\ \{\iota\} & \text{if } \gamma^\iota \in S^\iota_\Diamond; \end{cases}$$

- $\alpha(\boldsymbol{s}) = \prod_{i \in I} \alpha^i(s^i)$ *for all $\boldsymbol{s} \in S$.*

Given SU strategies $\sigma^i_\Diamond$, the composition $|_{i \in I} \sigma^i_\Diamond$ selects the same moves as the composition $\|_{i \in I} \widehat{\sigma^i_\Diamond}$. We thus justify the equivalence of the compositions in the following lemma.

**Lemma 3.** *For SU strategies $\sigma^i_\Diamond$, $i \in I$, it holds that*

$$\widehat{|_{i \in I} \sigma^i_\Diamond} = \|_{i \in I} \widehat{\sigma^i_\Diamond}.$$

From Lemma 3, we have $(G^i)^{\sigma^i_\Diamond} = (G^i)^{\widehat{\sigma^i_\Diamond}}$, and we can hence use SU strategies in Theorem 1, to obtain as conclusion that $(\|_{i=1}^n G^i)^{|_{i\in I}\sigma^i_\Diamond} \models \varphi$. This allows us to use the synthesis algorithms from [6], and compose SU strategies directly without unrolling them first, sacrificing compactness.

*Proof (of Lemma 3).* We denote by $\sigma_\Diamond = |_{i\in I}\sigma^i_\Diamond$ the composed SU strategy. As $\sigma_\Diamond$ is applied to a composed game, it associates to each path $\boldsymbol{s}_0(a_1\boldsymbol{\mu}_1)\boldsymbol{s}_2 \ldots \boldsymbol{s}_k$, a corresponding sequence of memory elements $\boldsymbol{m}_0\boldsymbol{m}_1\boldsymbol{m}_2 \ldots \boldsymbol{m}_k$, that are sampled by the memory update function, see Definition 9. For even $j > 0$, let $K_j \overset{\text{def}}{=} \mathsf{Sync}(a_{j+1})$ if $a_{j+1} \neq \tau$, and $K_j = \{\iota\}$ if $a_{j+1} = \tau$ and $\mu_{j+1} = \boldsymbol{s}_{j+1}$ such that $s^\iota_{j+1} \in S^\iota_\Diamond$. For odd $j > 0$, let $K_j \overset{\text{def}}{=} \mathsf{Sync}(\mathsf{act}(\boldsymbol{m}_j))$ if $\boldsymbol{s}_{j+1} \in S_\Box$, and $K_j \overset{\text{def}}{=} \{\iota\}$ if $s^\iota_{j+1} \in S^\iota_\Diamond$, see the corresponding cases in the definition of the SU strategy composition (Definition 10). Define

$$\delta^K_i(m, \gamma, m_+) \overset{\text{def}}{=} \begin{cases} \sigma^{u,i}_\Diamond(m, \gamma)(m_+) & \text{if } i \in K \\ \delta_{m=m_+} & \text{otherwise.} \end{cases}$$

Since the SU strategy composition ensures that the composed strategy updates the memory of the same components, namely, the involved components, both when the move is started, and when the move is resolved to enter the next state, we have $K_j = K_{j+1}$ for all odd $j > 0$. Hence, from the definition of path projection, $d(m^i, [\lambda]^i)$ in Definition 9 for $\sigma^i_\Diamond$ is equivalent to

$$\sum_{m^i_0,\ldots,m^i_{k-1}\in\mathfrak{M}^i} \alpha(\lambda^i_0)(m^i_0) \prod_{j=1}^k \delta^{K_j}_i(m^i_{j-1}, \lambda_j)(m^i_j).$$

From Definition 10, we have that $d(\boldsymbol{m}_k, \lambda)$ in Definition 9 for $\sigma_\Diamond$ is equivalent to

$$\sum_{\boldsymbol{m}_0,\ldots,\boldsymbol{m}_{k-1}\in\mathfrak{M}} \prod_{i\in I} \alpha^i(\lambda^i_0)(m^i_0) \prod_{j=1}^k \prod_{i\in I} \delta^{K_j}_i(m^i_{j-1}, \lambda^i_j, m^i_j).$$

The product over the indices $i \in I$ can be taken out of the summation over memory elements, since each term in the product only depends on one index. Hence, $d(\boldsymbol{m}_k, \lambda) = \prod_{i\in I} d(m^i, [\lambda]^i)$. Then, by unrolling and from Definition 10, we have, for every path $\lambda \in \Omega^+_\mathcal{G}$ with $\mathsf{last}(\lambda) = \boldsymbol{s} \in S_\Diamond$, that

$$\begin{aligned} \widehat{\sigma_\Diamond}(\lambda)(a, \langle\mu^\iota\rangle_{\boldsymbol{s},a}) &= \sum_{\boldsymbol{m}} d(\boldsymbol{m}, \lambda)\sigma^{n,\iota}_\Diamond(s^\iota, m^\iota)(a, \mu^\iota) \\ &= \sum_{m^\iota} d(m^\iota, [\lambda]^i)\sigma^{n,\iota}_\Diamond(s^\iota, m^\iota)(a, \mu^\iota) \\ &= \widehat{\sigma^\iota_\Diamond}([\lambda]^\iota)(a, \mu^\iota) \\ &= (\|_{i\in I}\, \widehat{\sigma^i_\Diamond})(\lambda)(a, \langle\mu^\iota\rangle_{\boldsymbol{s},a}), \end{aligned}$$

concluding the proof. □

## B Proof of Lemma 1

**Lemma 1** *Given a functional simulation from a PA $M$ to a PA $M'$ and a specification $\varphi$ defined on traces, for every strategy $\sigma_\Box \in \Sigma_\Box^M$ there is a strategy $\sigma'_\Box \in \Sigma_\Box^{M'}$ such that $(M')^{\sigma'_\Box} \models \varphi \Leftrightarrow M^{\sigma_\Box} \models \varphi$.*

We first prove the following intermediate result.

**Lemma 4.** *Given a functional simulation $f$ from a PA $M$ to a PA $M'$, for all strategies $\sigma_\Box$ for $M$, there is a strategy $\sigma'_\Box$ for $M'$ such that $P_{M'}^{\sigma'_\Box}(\lambda') = \sum_{f(\lambda)=\lambda'} P_M^{\sigma_\Box}(\lambda)$ for all $\lambda' \in \Omega_{M'}^+$.*

*Proof.* We construct an SU strategy $\sigma'_\Box$ that simulates $\sigma_\Box$ applied to $M$ by keeping the paths $\Omega_M^+$ of $M$ in memory. The functional simulation ensures that every path of $M^{\sigma_\Box}$ corresponds to a path in $M'^{\sigma'_\Box}$. Hence, after seeing memory $\lambda$, $\sigma'_\Box$ picks the next move that $\sigma_\Box$ would pick after seeing $\lambda$.

For an SU strategy $\sigma_\Diamond$ of a PA $M$, the probability of the path $\lambda \in \Omega_M^+$ and the memory element $m \in \mathfrak{M}$ is denoted by $\mathrm{Pr}_M^{\sigma_\Diamond}(m, \lambda) \stackrel{\text{def}}{=} \mathrm{Pr}_M^{\sigma_\Diamond}(\lambda) d(\lambda)(m)$, where $d$ is as in Definition 9. Clearly, $\sum_{m \in M} \mathrm{Pr}_M^{\sigma_\Diamond}(m, \lambda) = \mathrm{Pr}_M^{\sigma_\Diamond}(\lambda)$.

We now formally define $\sigma'_\Box \stackrel{\text{def}}{=} \langle \mathfrak{M}, \sigma_\Box^n, \sigma_\Box^u, \alpha \rangle$ with memory $\mathfrak{M} = \Omega_M^+$, and initial distribution on memory given by $\alpha(s')(s) = \varsigma(s)/\varsigma'(s')$ for all $s \in S$ and $s' \in S'$ such that $f(s) = s'$. The next move function is defined by

$$\sigma_\Box^n(s', \lambda)(a, \mu') = \sum_{\overline{f}(\mu)=\mu'} \sigma_\Box(\lambda)(a, \mu),$$

for all $s' \xrightarrow{a} \mu'$, $\lambda \in \mathfrak{M}$, and $s' = f(\mathsf{last}(\lambda))$. The memory update function is defined for moves by

$$\sigma_\Box^u(\lambda, (a, \mu'))[\lambda(a, \mu)] = \frac{\sigma_\Box(\lambda)(a, \mu)}{\sigma_\Box^n(s', \lambda)(a, \mu')}, \tag{1}$$

for all $s' \xrightarrow{a} \mu'$, $\lambda(a, \mu) \in \mathfrak{M}$, and $s' = f(\mathsf{last}(\lambda))$, and for states by

$$\sigma_\Box^u(\lambda(a, \mu), s')(\lambda(a, \mu)s) = \mu(s)/\mu'(s'), \tag{2}$$

for all $\lambda(a, \mu)s \in \mathfrak{M}$, $s' = f(s)$, and $\mu' = \overline{f}(\mu)$.

We now show by induction that $\mathrm{Pr}_{M'}^{\sigma'_\Box}(\lambda, \lambda') = \mathrm{Pr}_M^{\sigma_\Box}(\lambda)$ if $f(\lambda) = \lambda'$, and $\mathrm{Pr}_{M'}^{\sigma'_\Box}(\lambda, \lambda') = 0$ otherwise.

**Base Case.** For any $s \in S$ and $s' \in S'$ such that $f(s) = s'$, we have that $\mathrm{Pr}_{M'}^{\sigma'_\Box}(s, s') = \varsigma'(s')\alpha(s')(s) = \varsigma(s)$ if $f(s) = s'$ and 0 otherwise.

**Induction Step.** The induction hypothesis is that for any $\lambda \in \Omega_M^+$ and $\lambda' \in \Omega_{M'}^+$, $\mathrm{Pr}_{M'}^{\sigma'_\Box}(\lambda, \lambda') = \mathrm{Pr}_M^{\sigma_\Box}(\lambda)$ if $f(\lambda) = \lambda'$, and 0 otherwise. For any $\lambda(a, \mu)s \in \Omega_M^+$ and $\lambda'(a, \mu')s' \in \Omega_{M'}^+$, we have that $\mathrm{Pr}_{M'}^{\sigma'_\Box}(\lambda(a, \mu)s, \lambda'(a, \mu')s') = \mathrm{Pr}_{M'}^{\sigma'_\Box}(\lambda, \lambda') F_1 F_2 F_3 F_4$, where

$F_1 = \sigma_\Box^n(\mathsf{last}(\lambda'), \lambda)(a, \mu')$,
$F_2 = \sigma_\Box^u(\lambda, (a, \mu'))(\lambda(a, \mu))$,
$F_3 = \mu'(s')$, and
$F_4 = \sigma_\Box^u(\lambda(a, \mu), s')(\lambda(a, \mu)s)$.

Consider first the case where $f(\lambda(a,\mu)s) \neq \lambda'(a,\mu')s'$. If $f(\lambda) \neq \lambda'$, then from the induction hypothesis $Pr_{M'}^{\sigma'_\Box}(\lambda, \lambda') = 0$. If $f((a,\mu)s) \neq (a,\mu')s'$, then from (2), $F_4 = 0$. Hence $\Pr_{M'}^{\sigma'_\Box}(\lambda(a,\mu)s, \lambda'(a,\mu')s') = 0$, as required.

Now suppose that $f(\lambda(a,\mu)s) = \lambda'(a,\mu')s'$. By (1), we have that $F_1F_2 = \sigma_\Box(\lambda)(a,\mu)$, and by (2), we have that $F_3F_4 = \mu(s)$. Thus, applying the induction hypothesis, $\Pr_{M'}^{\sigma'_\Box}(\lambda,\lambda')F_1F_2F_3F_4 = \Pr_M^{\sigma_\Box}(\lambda)\sigma_\Box(\lambda)(a,\mu)\mu(s) = \Pr_M^{\sigma_\Box}(\lambda(a,\mu)s)$.

The above induction immediately yields $\Pr_{M'}^{\sigma'_\Box}(\lambda') = \sum_{\lambda\in\mathfrak{M}} \Pr_{M'}^{\sigma'_\Box}(\lambda,\lambda') = \sum_{f(\lambda)=\lambda'} \Pr_M^{\sigma_\Box}(\lambda)$, concluding the proof. □

*Proof (of Lemma 1).* A functional simulation $f$ extends inductively to a total function on paths of $M$ by defining $f(\lambda(a,\mu)s) \overset{\text{def}}{=} f(\lambda)(a,\overline{f}(\mu))f(s)$. Observe that, for all paths $\lambda$ of $M$, $\mathsf{trace}(f(\lambda)) = \mathsf{trace}(\lambda)$. We have that

$$\begin{aligned} P_{M'}^{\sigma'_\Box}(w) &= \sum_{\lambda'\in\text{paths}(w)} P_{M'}^{\sigma'_\Box}(\lambda') \\ &\overset{*}{=} \sum_{\lambda'\in\text{paths}(w)} \sum_{f(\lambda)=\lambda'} P_M^{\sigma_\Box}(\lambda) \\ &= \sum_{\lambda\in\text{paths}(w)} P_M^{\sigma_\Box}(\lambda) \\ &= P_M^{\sigma_\Box}(w), \end{aligned}$$

where the equation marked with $*$ is a consequence of $\mathsf{trace}(\lambda) = \mathsf{trace}(f(\lambda)$. Thus $\sigma'_\Box$ and $\sigma_\Box$ induce the same trace distribution, and $\varphi$, which is defined on traces, satisfies $(M')^{\sigma'_\Box} \models \varphi \Leftrightarrow M^{\sigma_\Box} \models \varphi$. □

## C Proof of Lemma 2

**Lemma 2** *Given compatible games $G^i$, $i \in I$, and any respective* Player $\Diamond$ *strategies $\sigma_\Diamond^i$, there is a functional simulation from $(\|_{i\in I} G^i)^{\|_{i\in I}\sigma_\Diamond^i}$ to $\|_{i\in I} (G^i)^{\sigma_\Diamond^i}$.*

*Proof.* Denote the composed game $\mathcal{G} = \|_{i\in I} G^i$, the induced PA $M = \mathcal{G}^{\|_{i\in I}\sigma_\Diamond^i}$, and the composition of induced PAs $M' = \|_{i\in I} (G^i)^{\sigma_\Diamond^i}$; denote by $S_G$, $S_M$ and $S_{M'}$ their respective state spaces. The proof is by defining a partial function $f : S_M \to S'_{M'}$, and then showing that $f$ is a functional simulation.

For $\kappa \in \Omega_{\mathcal{G},\Box}^+$ (i.e. states in the induced PA $M$ that come from Player $\Box$ states in the game $\mathcal{G}$), let

$$f(\kappa) \overset{\text{def}}{=} ([\kappa]^1, \ldots, [\kappa]^n),$$

and for $(a, \boldsymbol{\mu}_{\kappa,a}) \in \mathcal{U}(\mathcal{G})_{\Diamond\bigcirc}$ (i.e. states in the induced PA $M$ that come from $\mathsf{Player}\ \Diamond$ states in the game $\mathcal{G}$), let

$$f(a, \boldsymbol{\mu}_{\kappa,a}) \overset{\text{def}}{=} ([\kappa]^1, \dots, [\kappa]^{\iota-1}, (a, \mu^{\iota}_{[\kappa]^{\iota}}), [\kappa]^{\iota+1}, \dots, [\kappa]^n),$$

where $\iota$ is the index of the component controlling the state, cf. Definition 5. Recall first that for distributions $\boldsymbol{\mu}$ with support in the domain of $f$,

$$\overline{f}(\boldsymbol{\mu})(\boldsymbol{\lambda}) = \sum_{f(\kappa')=\boldsymbol{\lambda}} \boldsymbol{\mu}(\kappa').$$

Instantiating this with $\boldsymbol{\mu} = \varsigma$ we obtain,

$$\overline{f}(\varsigma)(\boldsymbol{\lambda}) = \sum_{f(\boldsymbol{s})=\boldsymbol{\lambda}} \varsigma(\boldsymbol{s}).$$

**Proof of $f(\varsigma) = \varsigma'$:** For states $\boldsymbol{s} \in S_G \subseteq \Omega^+_{\mathcal{G},\Box}$, the path projection $[\boldsymbol{s}]^i$ just yields $s^i$ for all $i$, and hence $f(\boldsymbol{s}) = \boldsymbol{s}$. We therefore have that $f(\varsigma) = \varsigma = \varsigma'$.
**Proof that if $f(\boldsymbol{s}) \xrightarrow{a} \boldsymbol{\mu}$ in $M$ then $f(\boldsymbol{s}) \xrightarrow{a}{}' \overline{f}(\boldsymbol{\mu})$ in $M'$:** In order to show this, we consider several cases of the transitions in $M$. According to the definition of the induced PA (Definition 3), $M = \mathcal{G}^{\|_{i\in I}\sigma^i_{\Diamond}}$ has three kind of transitions: $\tilde{\kappa} \xrightarrow{\tau} \sigma_{\Diamond}(\kappa)$ *(I1)*, $(a, \boldsymbol{\mu}_{\kappa,a}) \xrightarrow{a} \boldsymbol{\mu}_{\kappa,a}$ *(I2)*, and $\kappa \xrightarrow{a} \boldsymbol{\mu}_{\kappa,a}$ *(I3)*, where $\boldsymbol{\mu}_{\kappa,a}$ is obtained from the move $(a, \boldsymbol{\mu})$ enabled in $\mathsf{last}(\kappa)$. Note that $\kappa$ is a path in $\mathcal{G}$, but owing to the unfolding, $\kappa$ is a state in $M$ in case *(I3)*. Also, $\mathsf{last}(\kappa) = (\mathsf{last}([\kappa]^1), \dots, \mathsf{last}([\kappa]^n))$.

We first present two intermediate results:
**Fact 1:** We show that $\overline{f}(\sigma_{\Diamond}(\kappa))$ is the product distribution whose $\iota$th coordinate is $\sigma^{\iota}_{\Diamond}([\kappa]^{\iota})$, and the other coordinates $j \neq \iota$ are Dirac distributions $[\kappa]^j$.

**Proof of Fact 1:** Recall from the compatibility condition that $\sigma_{\Diamond}(\kappa)$ is nonzero only for a move of the form $(a, \boldsymbol{\mu}_{\kappa,a})$, with $\boldsymbol{\mu} = \langle \mu^{\iota} \rangle_{\mathsf{last}([\kappa]^{\iota}),a}$. Hence

$$\overline{f}(\sigma_{\Diamond}(\kappa))(f(a, \boldsymbol{\mu}_{\kappa,a})) = \sigma_{\Diamond}(\kappa)(a, \boldsymbol{\mu}_{\kappa,a}) = \sigma^{\iota}_{\Diamond}([\kappa]^{\iota})(a, \mu^{\iota}),$$

where the second equality follows from the definition of strategy composition. Since $\sigma^{\iota}_{\Diamond}([\kappa]^{\iota})(a, \mu^{\iota})$ sums to one when $(a, \mu^{\iota})$ ranges over the moves enabled in $[\kappa]^{\iota}$, the move $f(a, \boldsymbol{\mu}_{\kappa,a})$ is chosen in $M'$ at $\tilde{\kappa}$ with probability $\sigma^{\iota}_{\Diamond}([\kappa]^{\iota})(a, \mu^{\iota})$.
**Fact 2:** We now show that $\overline{f}(\boldsymbol{\mu}_{\kappa,a})$ is the distribution $\boldsymbol{\nu}$, such that $\nu^i = \mu^i_{[\kappa]^i,a}$ for the components synchronizing on $a$ ($i \in \mathsf{Sync}(a)$), and $\nu^j = [\kappa]^j$ otherwise ($j \notin \mathsf{Sync}(a)$).

**Proof of Fact 2:** From the definition of unfolding and the game composition, $\boldsymbol{\mu}_{\kappa,a}(\kappa')$ is nonzero only if $\kappa'$ is of the form $\kappa' = \kappa(a, \boldsymbol{\mu})\boldsymbol{s}$ with $[\kappa']^i = [\kappa]^i(a, \mu^i)s^i$ for the involved components and $[\kappa']^j = [\kappa]^j$ otherwise. For such $\kappa'$, it holds that $\overline{f}(\boldsymbol{\mu}_{\kappa,a})(f(\kappa')) = \boldsymbol{\mu}_{\kappa,a}(\kappa') = \prod_{i\in\mathsf{Sync}(a)} \mu^i(s^i) = \prod_{i\in I} \nu^i([\kappa']^i) = \nu(\kappa')$. Hence, $\overline{f}(\boldsymbol{\mu}_{\kappa,a}) = \boldsymbol{\nu}$.
**Proof of cases *(I1)* and *(I2)*:** The transitions $\tilde{\kappa} \xrightarrow{\tau} \sigma_{\Diamond}(\kappa)$ and $(a, \boldsymbol{\mu}_{\kappa,a}) \xrightarrow{a} \boldsymbol{\mu}_{\kappa,a}$ are induced from transitions of the game composition $\mathcal{G}$:

$$\tilde{\kappa} \xrightarrow{\tau} \kappa \xrightarrow{a} \boldsymbol{\mu}_{\kappa,a}, \qquad (3)$$

themselves coming from transitions of the components, according to the game composition (Definition 5):

- $\mathsf{last}([\tilde{\kappa}]^\iota) \xrightarrow{\tau} \mathsf{last}([\kappa]^\iota) \xrightarrow{a} \mu^\iota$ for the only component $G^\iota$ involved in the $\tau$-transition *(C4)*;
- $\mathsf{last}([\tilde{\kappa}]^i) = \mathsf{last}([\kappa]^i) \xrightarrow{a} \mu^i$ for the other components $G^i$ involved in the $a$-transition ($i \in \mathsf{Sync}(a)$) and also $[\tilde{\kappa}]^i = [\kappa]^i$ *(C1)*;
- the other components ($j \notin \mathsf{Sync}(a)$) stay unchanged $[\tilde{\kappa}]^j = [\kappa]^j$ and $\mu^j = \mathsf{last}([\tilde{\kappa}]^j)$ *(C2)*.

Note that presence of the state $(a, \boldsymbol{\mu}_{\kappa,a})$ in $M$ implies that $\sigma_\Diamond(\kappa)(a, \boldsymbol{\mu}_{\kappa,a}) > 0$, because the state space is defined as the reachable states (Definition 3).

By definition of the compatibility condition (Definition 6) and of the composed strategy (Definition 7), the distribution above is $\boldsymbol{\mu} = \langle \mu^\iota \rangle_{\mathsf{last}([\kappa]^\iota),a}$, and so $\sigma^\iota_\Diamond([\kappa]^\iota)(a, \mu^\iota) = \sigma_\Diamond(\kappa)(a, \boldsymbol{\mu}_{\kappa,a}) > 0$. Thus there is a sequence of transitions $[\tilde{\kappa}]^\iota \xrightarrow{\tau} [\kappa]^\iota \xrightarrow{a} \mu^\iota_{[\kappa]^\iota,a}$ in the induced PA $(G^\iota)^{\sigma^\iota_\Diamond}$, and transition $[\kappa]^i = [\tilde{\kappa}]^i \xrightarrow{a} \mu^i$ for the other components synchronising on $a$ ($i \in \mathsf{Sync}(a)$).

From the two facts above, we have that there are transitions $f(\tilde{\kappa}) \xrightarrow{\tau} \overline{f}(\sigma_\Diamond(\kappa))$ (Fact 1), and $f(a, \boldsymbol{\mu}_{\kappa,a}) \xrightarrow{a} \overline{f}(\boldsymbol{\mu}_{\kappa,a})$ (Fact 2), in the product of PA $M' = \|_{i\in I} (G^i)^{\sigma^i_\Diamond}$.

**Proof of case** *(I3)***:** The transition $\kappa \xrightarrow{a} \boldsymbol{\mu}_{\kappa,a}$ of $M$ was already there in the game $G$. It means that there is a transition $[\kappa]^i \xrightarrow{a} \boldsymbol{\mu^i}_{[\kappa]^i,a}$ in the unfolding $\mathcal{U}(G^i)$ for $i \in \mathsf{Sync}(a)$ and thus in $(G^i)^{\sigma^i_\Diamond}$. We use (Fact 2) again and conclude the proof $f(\kappa) \xrightarrow{a} \overline{f}(\boldsymbol{\mu}_{\kappa,a})$

## D Proof of Theorem 1

**Theorem 1** *Given a rule $\mathfrak{P}$ for PAs $\mathcal{M}_i$ and specifications $\varphi^i_j$ and $\varphi$ defined on traces, then the rule $\mathfrak{G}$ holds for all* Player $\Diamond$ *strategies $\sigma^i_\Diamond$ of compatible games $G^i$ with the same action alphabets as the corresponding PAs, where*

$$\mathfrak{P} \equiv \frac{\mathcal{M}_i \models \varphi^i_j \quad 1 \le j \le m \quad i \in I}{\|_{i\in I} \mathcal{M}_i \models \varphi,} \qquad \textit{and} \qquad \mathfrak{G} \equiv \frac{(G^i)^{\sigma^i_\Diamond} \models \bigwedge_{j=1}^m \varphi^i_j \quad i \in I}{(\|_{i\in I} G^i)^{\|_{i\in I}\sigma^i_\Diamond} \models \varphi.}$$

*Proof.* Take games $G^i$, for all $i$, and take respective Player $\Diamond$ strategies $\sigma^i_\Diamond$, such that $(G^i)^{\sigma^i_\Diamond} \models \bigwedge_{j=1}^m \varphi^i_j$. Since $(G^i)^{\sigma^i_\Diamond}$ are PAs, we apply the corresponding PA rule assumed in the Theorem: Let $\mathcal{M}^i = (G^i)^{\sigma^i_\Diamond}$. We clearly have that $\mathcal{M}^i \models \bigwedge_{j=1}^m \varphi^i_j$ for all $i \in I$ from how the strategies $\sigma^i_\Diamond$ were picked. Then, from the PA rule we have that $\|_{i\in I} \mathcal{M}^i \models \varphi$. From Lemma 2, there is a functional simulation from $(\|_{i\in I} G^i)^{\|_{i\in I}\sigma^i_\Diamond}$ to $\|_{i\in I} (G^i)^{\sigma^i_\Diamond}$. Since $\|_{i\in I} (G^i)^{\sigma^i_\Diamond} \models \varphi$, applying Lemma 1 yields $(\|_{i\in I} G^i)^{\|_{i\in I}\sigma^i_\Diamond} \models \varphi$.

## E Stopping Game Assumption

First, note that every action-labelled game of Definition 1 can be translated into the formulation with stochastic states used in [5] and [6] by considering each move as a stochastic state. Further, by unfolding SU strategies as in Definition 9, our definition of strategy application matches that of [5], where the SU strategies of both players are applied at the same time to directly obtain a DTMC, without the intermediate stage of the induced PA.

Finally, the methods of [5] and [6] require that games, under any strategy pair, reach with probability 1 a part of the game where the properties no longer change. In particular, this is sufficient to guarantee bounded rewards.

In our framework this assumption carries over in the following way. A *bottom strongly connected component* (BSCC) of a DTMC $D$ is a nonempty maximal subset of states $T \subseteq S$ s.t. every state in $T$ is reached with positive probability from any other state in $T$, and no state outside of $T$ is reachable. Recall that some BSCC is reached with probability 1. We say that a game $G$ is a *stopping game* with respect to an MQ $\varphi$ if, for every pair of strategies $(\sigma_\Diamond, \sigma_\Box) \in \Sigma_\Diamond \times \Sigma_\Box$, every BSCC in the DTMC $G^{\sigma_\Diamond, \sigma_\Box}$ has zero reward in all dimensions, and LTL formulae do not distinguish between actions within each BSCC. Hence, once a BSCC is reached, the MQ is no longer affected. We remark that any non-stopping game can be transformed into a stopping game, approximating the Pareto set arbitrarily close, with a polynomial increase in the size of the state-space.[4]

## F Case Study Details

The components are modelled in PRISM-games [4]. For the Pareto set computation and strategy construction we extend the prototype implementation of [6].

**Models.** The models for the traders and the fridge are shown in Figure 4 and Figure 5 respectively. The intuition is that in the composition the scheduler picks a trader $i$ that makes an offer, the trader chooses the offer in state $t_1^i$, say $o_{2i}$, and the fridge chooses in state $s_1^{2i}$ whether to accept (with $a_{2i}$) or decline (with $d_{2i}$). The composition of these components does not have a deadlock state, and the components are compatible. Since the games are stopping and synchronise on the "term" action, any Player $\Box$ strategy is unconditionally fair.

**Properties.** The rule in Section 4.5,

$$\frac{F \models \bigwedge_{j=1}^{n}(O_j \to A_j) \wedge (\bigwedge_{j=1}^{n} Q_j \to \#) \qquad T_i \models A_i \to (Q_i \wedge \$_i) \qquad 1 \le i \le n}{C \models \bigwedge_{j=1}^{n}(O_j \to \$_j) \wedge (\bigwedge_{j=1}^{n} O_j \to \#),}$$

can be derived from the assume-guarantee rules of [12], in particular the (Circ-Quant) rule, which is

$$\frac{M_2 \models \varphi_{A_2} \qquad M_1 \models \varphi_{A_2} \to \varphi_{A_1} \qquad M_2 \models \varphi_{A_1} \to \varphi_G}{M_1 \parallel M_2 \models \varphi_G,}$$

[4] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

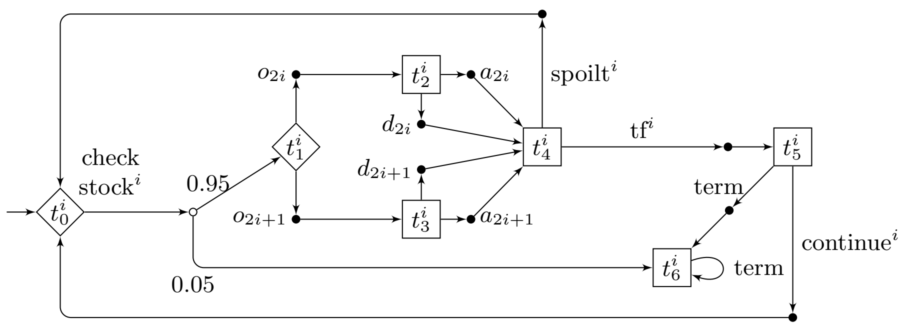


Fig. 4: Model for trader $T_i$ for $1 \leq i \leq n$, for two offers per trader. The action tf$^i$ stands for a finished transaction. Adding another offer, say $o'$, is done by adding another outgoing edge to $t_1^i$ labelled by $o'$, leading to a new Player $\Box$ state $t'$ with actions $a'$ (accept) and $d'$ (decline) leading to $t_4^i$.

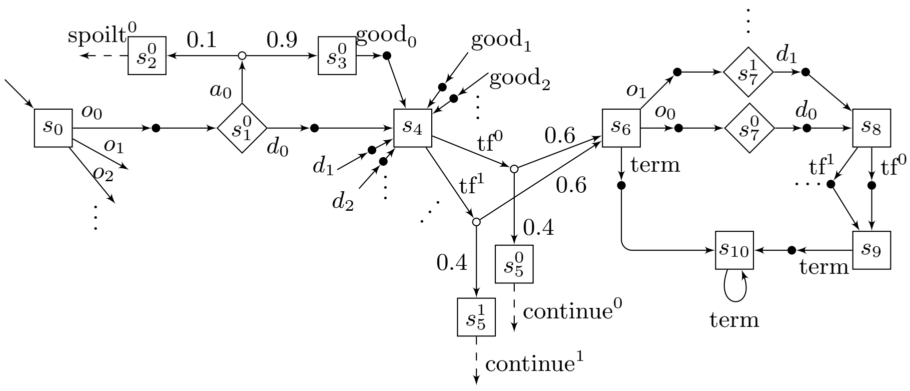
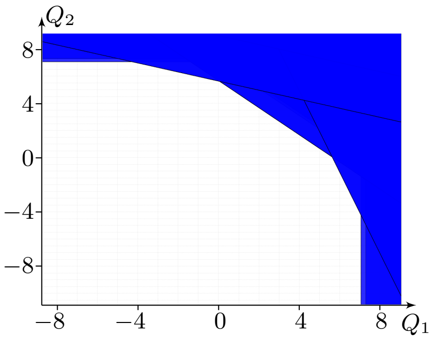
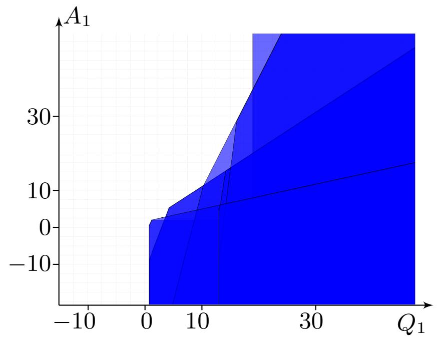


Fig. 5: Model for the fridge $F$ for two traders. Shown is the part for offer $o_0$ of trader $T_0$, and the part for the other offers and trader is indicated by ellipses. Dashed arrows lead with a Dirac distribution to the initial state $s_0$. The action tf$^i$ stands for a finished transaction. Adding more offers is done by duplicating the states $s_2^0$, $s_3^0$, $s_4^0$ and $s_7^0$. Adding another trader is done by first adding a new "tf" action to $s_4$ and to $s_8$, as well as duplicating $s_5^0$ with a new "continue" action. After adding the offers of the new trader (as explained above), additionally the gadget consisting of $s_6$ to $s_8$ is duplicated and attached so that the new $s_6$ is attached where $s_9$ is, and $s_9$ is attached to the new $s_8$ instead.



(a) Pareto set for the fridge $F$, sliced at $v_{o_i} = 6$, $v_{a_i} = 3.8$, and $v_{\#} = 9$.



(b) Pareto set for the trader $T_1$, sliced at $v_{\$_1} = 0.19$.

Fig. 6: Examples of under-approximations of Pareto sets for the model with two traders, sliced to be representable in 2 dimensions. Each separate convex set corresponds to one selection of weights in the algorithm of Section 4.4, and their union is, as expeced, not convex.

for PAs $M_1$ and $M_2$ and MQs $\varphi_{A_1}$, $\varphi_{A_2}$, and $\varphi_G$, such that the actions used in $\varphi_{A_1}$ and $\varphi_{A_2}$ are present in both PAs, and the (Conj-Quant) rule, which is

$$\frac{M_1 \models \varphi_{A_1} \rightarrow \varphi_{G_1} \qquad M_2 \models \varphi_{A_2} \rightarrow \varphi_{G_2}}{M_1 \parallel M_2 \models \varphi_{A_2} \wedge \varphi_{A_1} \rightarrow \varphi_{G_1} \wedge \varphi_{G_2}}$$

for PAs $M_1$ and $M_2$ and MQs $\varphi_{A_1}$, $\varphi_{A_2}$, $\varphi_{G_1}$, $\varphi_{G_2}$.

The reasoning behind it is the following. Assume that sufficiently many offers are made by some trader $T_i$, say $v_{o_i}$. Then the fridge guarantees to also accept, satisfying $A_i$. Therefore, trader $T_i$ can satisfy its primary objective of obtaining a certain unit price, $v_{\$_i}$, and also guarantees that the offers are of a high enough quality, satisfying $Q_i$. If all traders make sufficiently many offers, i.e. $O_i$ is true for all $1 \leq i \leq n$, then the fridge can satisfy its primary objective of obtaining a certain quantity of milk, $v_{\#}$.

We use the following reward objectives $O_i \equiv$ "offers made by $T_i$" $\geq v_{o_i}$, $A_i \equiv$ "offers of $T_i$ accepted" $\geq v_{a_i}$, $Q_i \equiv$ "quality of offers made by $T_i$" $\geq v_{q_i}$, $\$_i \equiv$ "unit price of $T_i$" $\geq v_{\$_i}$, and $\# \equiv$ "amount of milk obtained by $F$" $\geq v_{\#}$, with the reward structures are defined as follows: "offers made by $T_i$" assigns 1 to all offers of $T_i$, i.e. to $o_{2i}$ and $o_{2i+1}$; "offers of $T_i$ accepted" assigns 1 to all accept actions corresponding to the offers of $T_i$, i.e. to $a_{2i}$ and $a_{2i+1}$; "quality of offers made by $T_i$" assigns 0.5 to $o_{2i}$ and 2 to $o_{2i+1}$; "unit price of $T_i$" assigns 1 to $a_{2i}$ and 0.5 to $a_{2i+1}$; and, finally, "amount of milk obtained by $F$" assigns 1.5 to $\text{good}_{2i}$ and 6 to $\text{good}_{2i+1}$. Examples of the Pareto sets computed with our prototype implementation are shown in Figure 6. The targets are $v_{a_i} = 3.8$, $v_{q_i} = 3.3$, $v_{\$_i} = 0.19$, $v_{o_i} = 6.0$ for all $1 \leq i \leq n$, and the quantity of milk is set to $v_{\#} = 8$ for $n = 1$, $v_{\#} = 9$ for $n = 2$, $v_{\#} = 10$ for $n = 3$, and $v_{\#} = 12$ for

$n = 4$. For simplicity, the targets are picked uniformly for the traders, but our approach naturally supports heterogeneous components and specifications.

**Strategies.** The winning strategies for the traders that are synthesised compositionally are history dependent and randomised. In particular, supposing that the strategy starts at the initial state $t_0^i$ with memory $m_0$, the update function assigns $\sigma_\Diamond^u(m_0, t_1^i)(m_1) = 0.09$, and $\sigma_\Diamond^u(m_0, t_1^i)(m_2) = 0.91$, and in state $t_1^i$ the choice of the offer depends on the memory, as follows: $\sigma_\Diamond^n(t_1^i, m_1)(o_{2i}, t_2^i) = 1$, and $\sigma_\Diamond^n(t_1^i, m_2)(o_{2i+1}, t_3^i) = 1$. Moreover, every time the initial state $t_0^i$ is entered again, a different memory element, say $m_4$ is chosen, and the following memory update is Dirac $\sigma_\Diamond^u(m_4, t_1^i)(m_1) = 1$, leading the strategy to choose $o_{2i}$ with probability 1 from then on.