Decision Theoretic Behavior Composition

Nitin Yadav
School of Computer Science and IT
RMIT University
Melbourne, Australia
nitin.yadav@rmit.edu.au

Sebastian Sardina
School of Computer Science and IT
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

ABSTRACT

The behavior composition problem involves realizing a virtual target behavior (i.e., the desired module) by suitably coordinating the execution of a set of partially controllable available components (e.g., agents, devices, processes, etc.) running in a shared partially predictable environment. All existing approaches to such problem have been framed within *strict* uncertainty settings. In this work, we propose a framework for automatic behavior composition which allows the seamless integration of classical behavior composition with decision-theoretic reasoning. Specifically, we consider the problem of *maximizing the "expected realizability"* of the target behavior in settings where the uncertainty can be quantified. Unlike previous proposals, the approach developed here is able to (better) deal with instances that do not accept "exact" solutions, thus yielding a more practical account for real domains. Moreover, it is provably strictly more general than the classical composition framework. Besides formally defining the problem and what counts as a solution, we show how a decision-theoretic composition problem can be solved by reducing it to the problem of finding an optimal policy in a Markov decision process.

Categories and Subject Descriptors

I.12.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods.

General Terms

Theory, Verification, Algorithms.

Keywords

Behavior composition, decision theory, synthesis.

1. INTRODUCTION

In this work, we develop a decision theoretic account for behavior composition, that is, the problem of synthesizing a smart controller that is able to realize a virtual (i.e., non-available) target behavior module by suitably coordinating a set of available behaviors acting in a shared environment. Such problem has been extensively studied in the web-service composition literature (e.g., [1, 2]), where behaviors are deterministic and represent services, and more recently within the AI literature in more general settings (e.g., [5, 10, 11]), where nondeterministic behaviors may stand for the logic of various artifacts, such as agents or agents' high-level plans, physical devices, business processes, or software modules. Nonetheless, all approaches to behavior composition have assumed a setting of *strict uncertainty* [6], in that the incomplete information, for example, on the dynamics of the environment or on that of the available behaviors, cannot be quantified in any way. Hence, classical behavior composition problems may only accommodate *exact* solutions, that is, controllers that will guarantee the realization of the given target module no matter what. In many settings, however, while exact solutions may not exist, the ability to obtain a controller realizing the target module to the highest degree—the "optimal" controller—is desirable.

In order to better deal with non-solvable behavior composition instances, a framework in which different non-exact controllers can be compared is required. To that end, we propose an extension of the classical composition problem that goes beyond strict uncertainty, by accommodating ways of quantifying the different uncertainties in the model. Following the literature in behavior composition, we abstract the actual behaviors and environment as finite state *transition systems*. More precisely, each available module is represented as a nondeterministic transition system (to model partial controllability); the target behavior is represented as a deterministic transition system (to model full controllability); and the environment, which is fully accessible by all behaviors, is represented as a nondeterministic transition system (to model partial predictability).

As one can observe, in the above model, there are three sources of uncertainty stemming from the potential nondeterminism in both the environment and the available behaviors as well as in the potential different transitions in the target behavior. In the extended behavior composition framework to be developed here, all three uncertainties can be quantified. Note this is a reasonable assumption in many realistic settings, in which such information is readily available to the modeller. Consider a domain in which different bots are meant to maintain a garden, by performing various gardening activities such as cleaning, watering, and plucking flowers. Some bots may be equipped with buckets that may, nondeterministically, get filled after using it, and such nondeterminism can be quantified depending on various aspects of the domain (e.g., size of the bucket, average amount of dirt collected in a single action, etc.) Similarly, execution of actions in the garden environment—where the bots are meant to operate—can also be represented stochastically: a single clean operation may not always successfully clean the whole garden; the probability of a successful clean depends, for example, on the size of the garden and the season. More interestingly, given that the desired target behavior for maintaining the garden may involve more than one action from a given state, probabilities can be assigned to these depending on their expected

Cite as: Decision Theoretic Behavior Composition, Yadav, N. and Sardina, S., *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 575-582.

frequency. For instance, in some state, the gardening target system is expected to request the plucking action 30% of the time only, most times it will just request watering the garden.

The contributions of this paper are threefold. First, building on [10, 5, 11], a decision theoretic framework for behavior composition is developed. In doing so, we define the notion of *optimal composition controllers* using the "expected realizability" of the target, as well as the notion of *exact* compositions, that is, controllers that will solve the composition problem robustly. Unlike previous frameworks for behavior composition, the proposed one is able to deal with problems that do not accept exact solutions. Second, we provide a translation of a decision theoretic behavior composition problem into a Markov decision process (MDP) [8, 6], and show that finding an optimal policy for such MDP amounts to finding an optimal composition. This problem reduction provides a readily available technique for solving the new composition framework using the established MDP paradigm. Third, we show that the decision theoretic framework developed here is a *strict* extension of the classical behavior composition frameworks in the literature.

2. THE PROBABILISTIC FRAMEWORK

Classical behavior composition problems are stated on an abstract framework based on a sort of finite state transition systems (see, e.g., [1, 5, 10, 11]). Specifically, the so-called (available) *system* includes a set of available behaviors representing those artifacts or devices at disposal that are meant to run within a shared environment. A *target* behavior then stands for such module that is desired but not directly available and is therefore meant to be "realized" by suitably composing the available behaviors in the system.

In a classical composition problem, incomplete information on any component is modeled by means of nondeterminism in the transition systems (in the available behaviors or in the environment) or different action transitions per state (in the target). However, all the work so far on the problem of behavior composition has assumed a setting of *strict uncertainty* [6] in that the space of possibilities—possible effects of actions, evolution of behaviors, and future action requests—is known, but the probabilities of these potential alternatives is not quantified.

In this section, we extend the framework used in [11, 10] to accommodate stochastic measures in the different components, thus yielding a framework for behavior composition under (non-strict) uncertainty. In particular, we use probabilities to model the uncertainty of the dynamics of the environment and of the available behaviors, as well as of the preferences on actions in the target module. Such probabilities are provided by a domain expert who is able to state how often a device happens to fail, an action brings about its expected effects, or certain requests arrive to the system.

Environment.

As standard in behavior composition, we assume to have a *shared fully observable environment*, which provides an abstract account of actions' preconditions and effects, and a mean of communication among modules. Since, in general, we have incomplete information about the actual preconditions and effects of actions, we shall use a stochastic model of the environment. Thus, given a state and an action to be executed in such state, different successor states may ensue with different probabilities. Formally, an *environment* is a tuple $\mathcal{E} = \langle \mathcal{A}, E, e_0, \mathcal{P}^{\mathcal{E}}_{next} \rangle$, where:

- $\mathcal{A}$ is a finite set of shared actions;
- E is the finite set of environment's states;
- $e_0 \in E$ is the initial state of the environment;
- $\mathcal{P}^{\mathcal{E}}_{next} : E \times \mathcal{A} \times E \mapsto [0, 1]$ is the probabilistic transition function among states: $\mathcal{P}^{\mathcal{E}}_{next}(e, a, e') = p$, or just $e \xrightarrow{a:p} e'$ in $\mathcal{E}$, states that action a when performed in state e leads the environment to a successor state e' with probability p. Furthermore, we require that for every $e \in E$ and $a \in \mathcal{A}$, $\sum_{e' \in E} \mathcal{P}^{\mathcal{E}}_{next}(e, a, e') \in \{0, 1\}$, that is, the action is not executable (the sum is 0) or all possible evolutions of the environment are accounted (the sum is 1).

EXAMPLE 1. A scenario wherein a garden is maintained by several bots is depicted in Figure 1. To keep the garden healthy one needs to regularly water the plants, pluck the ripe fruits and flowers, clean the garden by picking fallen leaves and removing dirt, and emptying the various waste bins. Whereas cleaning and emptying the bins is a regular activity, plucking and watering are done as required. The environment $\mathcal{E}$ models the states the garden can be in. The environment allows plucking and cleaning activities to be done in any order, and plants can be watered in any state. The pluck action results in the flowers and fruits been fully plucked 75% of the time (i.e., 25% of the time the garden still remains to be plucked), whereas the clean action results in the garden being totally cleaned 20% of the time (i.e., dirt still remains 80% of the time). A pluck action from the initial state (e_0) results in the garden been plucked but dirty (e_2) with a probability of .75, a subsequent clean action results in the garden being both plucked and clean (e_3), with a probability of .2. Similarly, a clean action from the initial state results in the garden being fully clean but not plucked (e_1) 20% of the time, and a subsequent pluck action causes the garden being cleaned and plucked (e_3) 75% of the time. For simplicity, we assume that emptying the bins always results in the environment evolving to its initial state. ■

Behaviors.

A behavior stands, essentially, for the logic of some available component (e.g., device, agent, plan, workflow), which provides, step by step, its user with a set of actions that can be performed. At each step, the user selects one action among those provided and executes it. Then, a new set of actions is provided, and so on. As behaviors are intended to interact with the environment (cf. above), their dynamics may depend on conditions in the environment. Formally, a *behavior* over an environment $\mathcal{E} = \langle \mathcal{A}, E, e_0, \mathcal{P}^{\mathcal{E}}_{next} \rangle$ is a tuple $\mathcal{B} = \langle B, b_0, \mathcal{P}^{\mathcal{B}}_{next} \rangle$, where:

- B is the finite set of behavior's states;
- $b_0 \in B$ is the initial state of the behavior;
- $\mathcal{P}^{\mathcal{B}}_{next} : B \times E \times \mathcal{A} \times B \mapsto [0, 1]$ is the probabilistic transition function of the behavior: $\mathcal{P}^{\mathcal{B}}_{next}(b, a, e, b') = p$, or $e \xrightarrow{a,e:p} e'$ in $\mathcal{B}$, denotes that action a executed in behavior state b when the environment is in state e will result in the behavior evolving to state b' with probability p. Since *all* potential transitions are accounted for in the model, we require that for every $b \in B$, $a \in \mathcal{A}$, and $e \in E$, $\sum_{b' \in B} \mathcal{P}^{\mathcal{B}}_{next}(b, a, e, b') \in \{0, 1\}$.

Behaviors are, in general, *nondeterministic*, that is, given a state and an action, there may be several transitions enabled by the environment. Hence, when choosing the action to execute next, one cannot be certain of the resulting state and of which actions will be available later on, since this depends on what particular transition happens to take place. In other words, nondeterministic behaviors are only *partially controllable*.

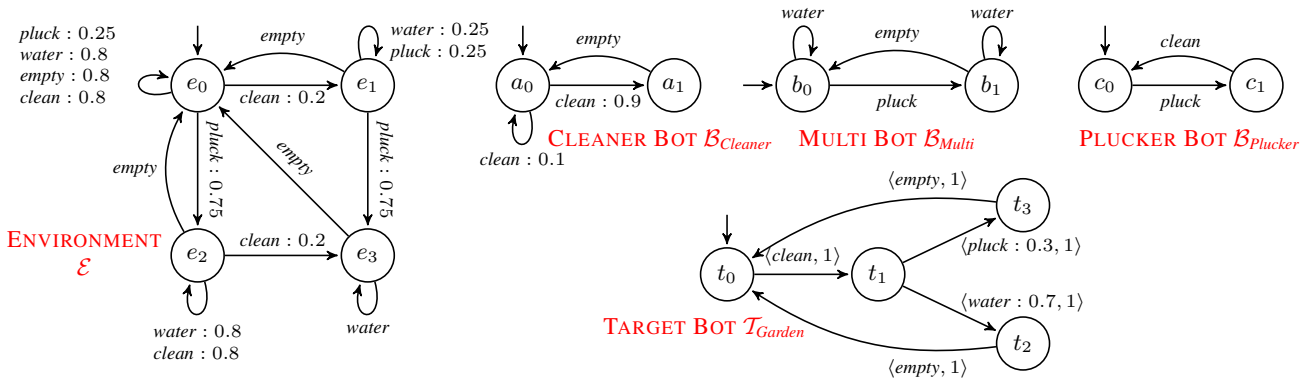

Figure 1: The garden bots system $\mathcal{S}_{Garden} = \langle \mathcal{B}_{Cleaner}, \mathcal{B}_{Multi}, \mathcal{B}_{Plucker}, \mathcal{E} \rangle$ and the target behavior $\mathcal{T}_{Garden}$.

EXAMPLE 2. In the gardening scenario, we assume there are three available garden bots; see Figure 1. The cleaner bot $\mathcal{B}_{Cleaner}$ cleans the garden by collecting the fallen leaves, dirt, waste, etc., into its own bucket. Most generally—90% of the time—its bucket gets filled up with a cleaning session, and the bot has to empty it to be able to start cleaning again. We assume the *empty* action involves emptying all garden bins as well as the bots' buckets. The plucker bot $\mathcal{B}_{Plucker}$ can pluck and clean the garden; since it is not equipped with a bucket, it plucks and collects from the ground directly. Finally, the multi-bot $\mathcal{B}_{Multi}$ has the capability to water the plants and pluck. It has a small bucket, and so it needs to empty it after every plucking session. ■

A behavior is deterministic if given a state and a legal action in that state, we always know exactly *the* next behavior state—the behavior is *fully controllable* through the selection of the next action to perform. Formally, a behavior $\mathcal{B} = \langle B, b_0, \mathcal{P}^{\mathcal{B}}_{next} \rangle$ over an environment $\mathcal{E} = \langle \mathcal{A}, E, e_0, \mathcal{P}^{\mathcal{E}}_{next} \rangle$ is *deterministic* iff for every $b, b' \in B$, $e \in E$, and $a \in \mathcal{A}$, it is the case that $\mathcal{P}^{\mathcal{B}}_{next}(b, e, a, b') \in \{0, 1\}$. In such case, the dynamics of the behavior can be represented using a transition relation $\delta_{\mathcal{B}} \subseteq B \times E \times A \times B$, where $\delta_{\mathcal{B}}(b, e, a, b')$ holds iff $\mathcal{P}^{\mathcal{B}}_{next}(b, e, a, b') = 1$.

Target behavior.

A target behavior is basically a *deterministic* behavior over $\mathcal{E}$ that represents the fully controllable desired behavior. A target behavior is virtual, in the sense that it does not exist in reality and, hence, is meant to be "realized" through the available behaviors.

Formally, a *target behavior* over an environment $\mathcal{E} = \langle \mathcal{A}, E, e_0, \mathcal{P}^{\mathcal{E}}_{next} \rangle$ is a tuple $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{req} \rangle$, where:

- T is the finite set of target's states;
- $t_0 \in T$ is the initial state of the target;
- $\delta \subseteq T \times E \times \mathcal{A} \times T$ is the target's deterministic transition relation: $\langle t, e, a, t' \rangle \in \delta$, or $t \xrightarrow{e:a} t'$ in $\mathcal{T}$, states that action a executed in the target state t, when the environment is in a state e, results in the target evolving to (unique) state t';
- $R : T \times A \mapsto \mathbb{R}^+$ is the reward function of the target: $R(t, a)$ denotes the reward obtained when the action a is successfully executed in target state t;
- $\mathcal{P}_{req} : T \times E \times \mathcal{A} \mapsto [0, 1]$ is the probabilistic action request function: $\mathcal{P}_{req}(t, e, a)$ denotes the probability of the target requesting the execution of action a when it is in state t and the environment is in state e. For consistency, we require that $\sum_{a \in \mathcal{A}} \mathcal{P}_{req}(t, e, a) \in \{0, 1\}$, for every $t \in T$, $e \in E$ (i.e., all possible requests are accounted for), and moreover, for all $a \in \mathcal{A}$, we have $\mathcal{P}_{req}(t, e, a) = 0$ whenever there is no state $t' \in T$ such that $\langle t, e, a, t' \rangle \in \delta$.

A *uniform-reward* target behavior is one where all actions have the same reward, that is, there exists $\alpha \in \mathbb{R}^+$ such that for all $a \in \mathcal{A}$ and $t \in T$, we have $R(t, a) = \alpha$.

This concludes the definition of the basic components for a decision-theoretic behavior composition problem. As the reader can easily note, this framework is essentially that of [5, 10, 11], except that *stochastic probabilistic transitions* are used instead of transition relations, a probability distribution over the potential action requests is used in the specification of the target, and a reward function is used in the target to state how "important" a particular request is. Note also that the probability function $\mathcal{P}_{req}$ in the target is very different to the ones used in the available behaviors and the environment. In the former, it denotes the probability of the target executing (i.e., requesting) an action from a given state, whereas in the latter the corresponding function simply denotes the stochastic evolutions of the entity.

EXAMPLE 3. The desired behavior required to maintain the garden in a particular season is not directly represented by any of the existing bots in the garden, and is modeled by the deterministic uniform-reward target bot $\mathcal{T}_{Garden}$ shown in Figure 1. Intuitively, the garden should always be cleaned first to remove any fallen leaves and dirt, followed by either plucking or watering the garden. Since flowers and fruits do not grow everyday, the plucking is required only 30% of the time; 70% of the time a request for watering the garden will be issued. Finally, the bins are to be emptied, and the whole process can repeat again. All requests are of equal value, namely, 1 unit (second component in each transition label). ■

Enacted system.

A *system* $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ is built form n, possibly nondeterministic, *available behaviors* $\mathcal{B}_i$, with $i \in \{1, \ldots n\}$, acting in a shared *environment* $\mathcal{E}$. Since, in the simplest case, one action can be executed at a given time, available behaviors in a system are meant to act concurrently in an interleaved fashion.

To refer to the behavior that emerges from the behaviors' joint (interleaved) executions, we use the notion of enacted system behavior, the synchronous product of the environment with the asynchronous product of all available be-

haviors. Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system, where $\mathcal{E} = \langle \mathcal{A}, E, e_0, \mathcal{P}^{\mathcal{E}}_{next} \rangle$ and $\mathcal{B}_i = \langle B_i, b_{i0}, \mathcal{P}^{\mathcal{B}_i}_{next} \rangle$, for $i \in \{1, \ldots, n\}$. The *enacted system behavior* of $\mathcal{S}$ is the tuple $\mathcal{T}_{\mathcal{S}} = \langle S, \mathcal{A}, \{1, \ldots, n\}, s_0, \mathcal{P}_{\mathcal{S}} \rangle$, where:

- $S = B_1 \times \cdots \times B_n \times E$ is the finite set of $\mathcal{T}_{\mathcal{S}}$'s states; when $s = \langle b_1, \ldots, b_n, e \rangle$, we denote b_i by $beh_i(s)$, for $i \in \{1, \ldots, n\}$, and e by $env(s)$;
- $s_0 \in S$, with $env(s_0) = e_0$ and $beh_i(s_0) = b_{i0}$, for each $i \in \{1, \ldots, n\}$, is $\mathcal{T}_{\mathcal{S}}$'s initial state;
- $\mathcal{P}_{\mathcal{S}} : S \times \mathcal{A} \times \{1, \ldots, n\} \times S \mapsto [0, 1]$ is $\mathcal{T}_{\mathcal{S}}$'s probabilistic transition function, defined as follows:

$$\mathcal{P}_{\mathcal{S}}(s, a, k, s') = \mathcal{P}^{\mathcal{E}}_{next}(env(s), a, env(s')) \times \mathcal{P}^{\mathcal{B}_k}_{next}(beh_k(s), a, env(s), beh_k(s')),$$

  if $beh_i(s) = beh_i(s')$, for each $i \in \{1, \ldots, n\} \setminus \{k\}$; and $\mathcal{P}_{\mathcal{S}}(s, a, k, s') = 0$, otherwise.

To distinguish which behavior acts in each enacted transition, we label each stochastic transition in $\mathcal{T}_{\mathcal{S}}$ with the corresponding behavior index—all other behaviors remain still. We observe that the sources of nondeterminism in enacted behaviors stem from two sources, namely, the nondeterminism in the environment and the nondeterminism in the available behaviors.

So, informally, the decision-theoretic (DT) behavior composition task is stated as follows: Given a system $\mathcal{S}$ and a target behavior $\mathcal{T}$, find the "optimal" way of (partially) controlling the available behaviors in $\mathcal{S}$ in a step-by-step manner—by instructing them on which action to execute next and observing, afterwards, the outcome in both the behavior used as well as in the environment—so as to "best realize" a specific deterministic target behavior. In the next section, we make this problem definition precise.

3. DT-COMPOSITION

In order to bring about the desired virtual target behavior in an available system, we assume the existence of a (central) *controller* module that is able to control the available behaviors, in the sense that, at each step, it can observe all behaviors, instruct them to execute an action (within their capabilities), stop, and resume them. In classical behavior composition, one then looks for a controller that *guarantees* that the target will be implemented in the system *always*, that is, no matter how the target happens to requests actions within its logic or how the available behaviors and the environment happen to evolve with actions. Such controller is then deemed an (exact) solution to the problem. From a (generalized) planning perspective, the composition task can be seen as that of planning for a "maintenance" goal, namely, always maintain target realization.

When it comes to realizing a target module in a composition framework as the one described above, though, one should not just look for *exact* solutions, as in general there may be none. Instead, one shall look for *optimal* ways of maximizing the "expected realizability" of the target in the available system.

Controller.

Before formally defining the central module in charge of coordinating the available behaviors, we first need to define the technical notions of traces and histories of a system. A *trace* for a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ is a, possibly infinite, sequence of states from the enacted system behavior of the form $s^0 \xrightarrow{a^1,k^1} s^1 \xrightarrow{a^2,k^2} \cdots$ such that *(i)* $s^0 = s_0$; and *(ii)* $\mathcal{P}_{\mathcal{S}}(s^j, a^{j+1}, k^{j+1}, s^{j+1}) > 0$, for all $j \geq 0$. Intuitively, a trace represents a possible (legal) evolution of the (enacted) system, where k^j is the index of the behavior which has executed action a^j. A *history* is a just a *finite prefix* $h = s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$ of a trace. We denote s^ℓ by $last(h)$, and the length ℓ of the history by $|h|$. The set of all histories for a given system will be denoted by $\mathcal{H}$.

So, formally, a *controller* for an available system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ is a total function $C : \mathcal{H} \times \mathcal{A} \mapsto \{1, \ldots, n, u\}$ such that, given a system history $h \in \mathcal{H}$ and an action $a \in \mathcal{A}$ that ought to be performed, returns the index of the behavior to which the action a is to be delegated for execution. For technical convenience, a special value u ("undefined") may be returned, thus making C a total function which returns a value even for actions that no behavior can perform.[1]

Now, informally, a "dead-end" is reached in a history if the controller in use selects a behavior which is not capable of executing the delegated action. Then, given two controllers, one should prefer the one that reaches a dead-end with lower probability, or put it differently, the one that has the highest probability of honoring the target's requests. In particular, a controller that is guaranteed not to ever reach a dead-end will be an exact, and thus optimal, solution.

We say that a history is reachable by a controller, if starting from the initial state of the enacted system, the behavior executing the action at each state of the history is indeed the one selected by the controller. More formally, a history $h = s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$ is *reachable* by a controller C (in a system $\mathcal{S}$) iff $k^i = C(s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^{i-1},k^{i-1}} s^{i-1}, a^i)$, for each $i \in \{1, \ldots, \ell\}$. We denote with $\mathcal{H}^\ell_C$ the set of all reachable histories of length ℓ and $\mathcal{H}_C = \bigcup_{i \geq 0} \mathcal{H}^i_C$ the set of all histories reachable by C.

Value of a controller and compositions.

In order to evaluate and compare controllers, we define the value of a controller for a given target and system. Roughly speaking, a controller is "rewarded" for every action request from the target that it fulfills by a successful delegation to an available behavior. More specifically, at every point, a controller gets a reward that depends both on the frequency of such request and the value of (fulfilling) it.

From now on, let $\mathcal{T} = \langle T, t_0, \delta, \mathcal{P}_{req} \rangle$ be a target behavior to be realized in a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$. Let C be a controller for system $\mathcal{S}$, and $\mathcal{T}_{\mathcal{S}}$ be the enacted system behavior as defined in the previous section. First, consider the case of evaluating the performance of a controller over a finite number of requests. The *value of C* for $k \geq 1$ requests at system history $h \in \mathcal{H}$ when the target is in state $t \in T$, denoted $\mathcal{Y}^C_k(h, t)$, is defined as follows:

$$\mathcal{Y}^C_k(h, t) = \sum_{a \in \mathcal{A}} [\mathcal{P}_{req}(t, env(last(h)), a) \times IR^C(h, t, a) + \sum_{\substack{s' \in S \\ \langle t, e, a, t' \rangle \in \delta}} \mathcal{P}_{\mathcal{S}}(last(h), a, C(h, a), s') \times \mathcal{Y}^C_{k-1}(h \xrightarrow{a, C(h,a)} s', t')],$$

where $\mathcal{Y}^C_0(h, t) = 0$, for all $h \in \mathcal{H}$ and $t \in T$, and $IR^C(h, t, a)$ stands for the *immediate* reward collected by the controller C when

[1] Although, as we shall see later, under the full observability assumption, it is enough for a controller to depend only on the final state of the enacted system—rather than the whole history–we shall work with the most general definition that could also be used in settings with partial observability.

requested to delegate action a at history h:

$$IR^C(h,t,a) = \begin{cases} R(t,a) & \text{if } \exists s'.\mathcal{P}_{\mathcal{S}}(last(h),a,C(h,a),s') > 0; \\ 0 & \text{if } C(h,a) = u; \\ -R(t,a) & \text{otherwise.} \end{cases}$$

The value of a controller for k steps of target $\mathcal{T}$ in a system $\mathcal{S}$ is defined as $\mathcal{Y}^C_k = \mathcal{Y}^C_k(s_0,t_0)$. We say that a controller C^* is a *k-composition* if for all other controllers C, $\mathcal{Y}^{C^*}_k \geq \mathcal{Y}^C_k$ holds.

Since the target may include infinite traces, we are in general interested in controllers that are optimal for any number of potential requests, that is, for infinite executions of the target behavior. To cope with unbounded executions of the target, we appeal to the use of a *discount factor*, as customary in sequential decision making over infinite episodes [6, 3]. The idea is that the satisfaction of later target-compatible requests are less important than those issued earlier. Formally, the value of a controller C, denoted by $\mathcal{Y}^C_\gamma(h,t)$, relative to a discount factor $0 \leq \gamma < 1$, is defined as follows:

$$\mathcal{Y}^C_\gamma(h,t) = \sum_{a\in\mathcal{A}} [\mathcal{P}_{req}(t, env(last(h)), a) \times IR^C(h,t,a) + \gamma \sum_{\substack{s'\in S \\ \langle t,e,a,t'\rangle\in\delta}} \mathcal{P}_{\mathcal{S}}(last(h),a,C(h,a),s') \times \mathcal{Y}^C_\gamma(h \xrightarrow{a,C(h,a)} s', t')].$$

The use of a discount factor plays the same role as in infinite horizon Markov decision processes, namely, it allows convergence of the value of a controller [3, 8]. Note that the assumption that temporally closer rewards are more important than distant ones is particularly suitable in the context of composition problems, where behaviors may fail, the target and available system may be reset, or the problem may not be fully solvable.

As with the finite case, the value of a controller for a given target $\mathcal{T}$ and system $\mathcal{S}$ is defined as $\mathcal{Y}^C_\gamma = \mathcal{Y}^C_\gamma(s_0,t_0)$. Finally, we say that a controller C^* is a *γ-composition* (of target $\mathcal{T}$ in system $\mathcal{S}$) if for all other controllers C, it is the case that $\mathcal{Y}^{C^*}_\gamma \geq \mathcal{Y}^C_\gamma$.

Put it all together, the decision theoretic behavior composition problem, or simply *DT-composition problem*, amounts to synthesize a γ-composition for a given system $\mathcal{S}$, target behavior $\mathcal{T}$, and discount factor γ.

Exact compositions.

A behavior composition problem has an exact solution when there exists a controller that can fully realize the target, that is, a controller that can *always* honor the target's requests, no matter what. There have recently been various approaches in the literature to synthesize such a controller, called a *composition*, if any exists (see, e.g., [5, 10, 11, 4]). Within our decision theoretic setting, it is important to clearly define what an exact solution is and its relationship with "optimal" controllers.

Since the target behavior is deterministic, its specification can be seen as the set of all possible sequences of actions that can be requested, starting from the initial state. Thus, given any finite run of the target, the most one could expect is that every single action has been successfully realized in the system. This would imply that all possible rewards in the run have indeed been collected. Since one does not know a priori which actual run will ensue, we consider the maximum expected reward when running the target. To make this precise, we define $\mathcal{R}^{\max}_k(t,e)$ as the *maximum* expected reward when running the target from its state t at environment state e for $k \geq 0$ steps as follows (here, $\mathcal{R}^{max}_0(t,e) = 0$):

$$\mathcal{R}^{\max}_{k\geq 1}(t,e) = \sum_{a\in\mathcal{A}} [\mathcal{P}_{req}(t,e,a) \times R(t,a)] + \sum_{\substack{e'\in E \\ \langle t,e,a,t'\rangle\in\delta}} [\mathcal{P}_{\mathcal{E}}(e,a,e') \times \mathcal{R}^{\max}_{k-1}(t',e')].$$

As above, we take $\mathcal{R}^{\max}_k = \mathcal{R}^{\max}_k(t_0,e_0)$, for any $k \geq 0$. Note that this definition is well defined for both cyclic and acyclic targets. Of course, for an acyclic target with a longest path of length ℓ, it is easy to show that $\mathcal{R}^{\max}_k = \mathcal{R}^{\max}_\ell$, for every $k \geq \ell$.

Thus, a controller C is an *exact composition* if $\mathcal{Y}^C_k = \mathcal{R}^{\max}_k$, for all $k \geq 1$, that is, C can *fully* and *always* realize a target behavior in the available system. Note that controllers are meant to have full observability of the current history. A *Markovian* (i.e., memoryless) controller C is one that only looks at the current state of the system to decide the delegation; formally, for all histories $h, h' \in \mathcal{H}$ such that $last(h) = last(h')$ and action $a \in \mathcal{A}$, $C(h,a) = C(h',a)$ applies. When it comes to exact solutions, Markovian controllers are enough under full observability.

THEOREM 1. *Let $\mathcal{S}$ be a system and $\mathcal{T}$ be a target behavior. Then, if there exists an exact solution for realizing $\mathcal{T}$ in $\mathcal{S}$, then there exists a Markovian controller which is also an exact solution.*

PROOF. Let C^* be an exact solution for realizing $\mathcal{T}$ in $\mathcal{S}$. For any $h \in \mathcal{H}$ and $a \in \mathcal{A}$, we define a new controller $\widehat{C}(h,a) = C^*(h',a)$ if $h' \in \mathcal{H}_{C^*}$ is such that $last(h) = last(h')$ and for all $h'' \in \mathcal{H}_{C^*}$ such that $last(h'') = last(h)$, it is the case that $C(h',a) \leq C(h'',a)$ (we assume $u > i$, for any $i \in \{1,\ldots,n\}$). Otherwise, if such history h' does not exist, we take $\widehat{C}(h,a) = u$.

It is easy to check that $\widehat{C}$ is well-defined. In addition, $\widehat{C}$ is Markovian. In fact, consider two histories $h_1, h_2 \in \mathcal{H}$ such that $last(h_1) = last(h_2)$, and suppose that $\widehat{C}(h_1,a) = k_1$. Then, $C^*(h'_1,a) = k_1$, for some $h'_1 \in \mathcal{H}_{C^*}$ and it is easy to show that $\widehat{C}(h,a) = C^*(h'_1,a) = k_1$ as well, the same witness history h'_1 can be used for h_2 too. Furthermore, because h'_1 is reachable by C^*, together with the fact that C^* is indeed an exact solution, implies that $k_1 \in \{1,\ldots,n\}$ is a correct delegation, in the sense that behavior $\mathcal{B}_{k_1}$ is able to perform a legal step on action a when the environment is in state $env(last(h))$, and since $last(h) = last(h'_1)$, such delegation is also legal at history h and $\widehat{C}$ is also exact. □

More importantly, exact solutions are guaranteed to be always optimal controllers under unbounded runs, independently of the discount factor chosen.

THEOREM 2. *If a controller is an exact composition for a decision-theoretic behavior composition problem, then such controller is a γ-composition, for any $0 \leq \gamma < 1$.*

PROOF (SKETCH). Let C^* be an exact solution to a DT-composition problem, and assume, wlog, a target with a uniform reward α. Then, at each step, C^* collects the maximum possible reward of α. If a discount factor γ is used, then C^* will collect a reward of $\alpha \times \sum_{n=1}^{\ell} \gamma^{n-1}$ over ℓ steps, which is indeed the maximum possible reward for a γ-composition after ℓ steps. Hence, C^* is also a γ-composition for the given composition problem. □

4. SOLVING DT-COMPOSITIONS

Various techniques have been used to actually solve classical behavior composition problems, including PDL satisfiability [5],

search-based approaches [11], LTL/ATL synthesis [9, 4], and computation of special kind of simulation relations [10, 2]. Unfortunately, in the context of the decision theoretic framework from Section 2, none of these techniques can be applied. In this section, we show how to solve a decision theoretic composition problem, by reducing it to a Markov decision problem in a natural manner. We also demonstrate the reduction with a proof of concept implementation using an off-the-shelf existing MDP solver.

Markov decision processes.

A Markov decision process (MDP) is a discrete time stochastic control process [8, 3]. At each step, the process is in a state q, the decision maker chooses an action a, the process evolves to a successor state q' with some probability, and the decision maker receives certain reward r. The "best" decision maker is one that collects maximum (potential) rewards over time.

Formally, a *Markov decision process* (MDP) is a tuple $\mathcal{M} = \langle Q, A, p, r\rangle$, where:

- Q is a (finite) set of states;
- A is a (finite) set of actions;
- $p : Q \times A \times Q \mapsto [0, 1]$ is the probabilistic state transition function: $p(q, a, q')$ denotes probability of the process evolving to state q' when action a is executed in state q;
- $r : Q \times A \mapsto \mathbb{R}$ is the reward function: $r(q, a)$ denotes the immediate reward obtained when action a in executed state q.

A policy—the decision maker—is a collection of state-action mappings stating what action to take in each state of the process. Formally, a (stationary) *policy* is a function $\pi : Q \mapsto A$; $\pi(q)$ denotes the action to be taken in state q. Solving an MDP involves then computing a policy that accumulates maximum reward over time. In doing so, one can be interested in finite horizon problems, where the decision maker is meant to perform a fixed number of sequential decisions, or infinite horizon problems, where rewards over infinite runs of the MDP are considered.

So, the value of an optimal policy in a state q for a *finite* horizon k is given by the following Bellman's *principle of optimality* [8, 3]:

$$V_k^*(q) = \max_{a\in A}\{r(q, a) + \sum_{q'\in Q} p(q, a, q') \times V_{k-1}^*(q')\}.$$

Similarly, the value of an optimal policy in a state q for *infinite* horizon relative to a discount factor of $0 \leq \gamma < 1$ is as follows [7]:

$$V^*(q) = \max_{a\in A}\{r(q, a) + \gamma \sum_{q'\in Q} p(q, a, q') \times V^*(q')\}.$$

Howard [7] showed that there always exists an optimal *stationary* policy for infinite horizon problems, that is, one that does not depend on which stage a decision is taken.

From behavior composition to MDPs.

With the notion of MDPs at hand, we show next how to reduce a DT-composition problem, as described in Sections 2 and 3, to the problem of solving an MDP.

Informally, in our setting, the decision maker is the controller, and thus, the possible actions that can be taken are those of behavior delegation. Consider then a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$, with $\mathcal{T}_\mathcal{S}$ denoting the corresponding enacted system behavior, and a target $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{req}\rangle$. We define the corresponding MDP encoding $\mathcal{M}_{\mathcal{S},\mathcal{T}} = \langle Q, A, p, r\rangle$ as follows:

- $Q = \mathcal{S} \times T \times \mathcal{A} \cup \{q_{init}\}$, where for all $\langle s, t, a\rangle \in Q$, $\mathcal{P}_{req}(t, env(s), a) > 0$. Given an MDP state $q = \langle s, t, a\rangle \in Q$, we define $sys(q) = s$, $tgt(q) = t$, and $req(q) = a$. A special, domain independent, state q_{init} is used as a "dummy" initial state of the process.
- $A = Index = \{1, \ldots, n, u\}$, that is, an action in the encoded MDP stands for a behavior selection (or no selection at all).
- The state transition function is defined as follows:

$$p(q, i, q') = \begin{cases} \mathcal{P}_{req}(tgt(q'), env(sys(q')), req(q')), \text{ if} \\ \quad q = q_{init}, sys(q') = s_0, tgt(q') = t_0; \\ \\ \mathcal{P}_\mathcal{S}(sys(q), req(q), i, sys(q')) \times \\ \quad \mathcal{P}_{req}(tgt(q'), env(sys(q')), req(q')), \text{ if} \\ \quad q \neq q_{init}; \\ \\ 0, \text{ otherwise.} \end{cases}$$

- The reward function is defined as ($\alpha = R(tgt(q), req(q))$):

$$r(q, i) = \begin{cases} \alpha & \text{if } \mathcal{P}_\mathcal{S}(sys(q), req(q), i, sys(q')) > 0 \\ & \quad \text{for some } q' \in Q \text{ and } q \neq q_{init}; \\ \\ 0 & \text{if } i = u \text{ or } q = q_{init}; \\ \\ -\alpha & \text{otherwise.} \end{cases}$$

In the resulting MDP, a state is built from the state of the enacted system behavior (which includes the states of the environment and those of all available behaviors), the state of target behavior, and an action being requested; in other words, a "snapshot" of the whole composition problem. Each transition in the MDP represents the behavior—through its index—to which the current request is delegated for execution. The dynamics of the MDP encodes both the dynamics of the enacted system behavior and the target behavior, as well as that of the stochastic process (i.e., the user of the target) that is requesting actions. Finally, the reward function in the MDP merely mimics that of the encoded behavior composition problem; no reward is given from the initial dummy state, and an unfeasible delegation (i.e., one where the chosen behavior may not perform the action) receives a penalty (i.e., it is better to prescribe "u").

Given a policy $\pi : Q \mapsto Index$ for the MDP $\mathcal{M}_{\mathcal{S},\mathcal{T}}$, we define the *induced controller* $C_\pi(h, a)$, where $h = s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$, with $\ell \geq 0$ and $a \in \mathcal{A}$, as follows:

$$C_\pi(h, a) = \begin{cases} \pi(q) & \text{if } sys(q) = last(h), a = req(q), \text{ and} \\ & \quad t_0 \xrightarrow{env(s^0):a^1} \cdots \xrightarrow{env(s^{\ell-1}):a^\ell} tgt(q) \text{ in } \mathcal{T}; \\ u & \text{otherwise.} \end{cases}$$

Note that the output for histories that do not yield any legal evolution of the target is irrelevant, and hence, we just output value u.

We now state the main result of the section, namely, a solution to the encoded MDP yields an optimal controller for the corresponding DT-composition problem.

THEOREM 3. *Let $\mathcal{S}$ be an available system and $\mathcal{T}$ a target behavior. Let $\mathcal{M}_{\mathcal{S},\mathcal{T}}$ be the corresponding MDP encoding as described above. If π^* is an γ-optimal policy for $\mathcal{M}_{\mathcal{S},\mathcal{T}}$, then its induced controller C_{π^*} is an γ-composition for realizing $\mathcal{T}$ in $\mathcal{S}$.*

PROOF (SKETCH). This is proved by showing that a solution to the optimality equation of the encoded MDP conforms to the solution of the equation for a composition. Specifically, we show, by induction on ℓ, that $\gamma \mathcal{Y}_{\ell}^{C_\pi}(s_0, t_0) = V_{\ell+1}^{\pi}(q_{init})$, where γ is a discount factor, $\ell \geq 0$, and π is a policy for $\mathcal{M}_{\mathcal{S},\mathcal{T}}$. To that end, we rely on the fact that the value of a controller and that of a policy in the MDP can be re-written as follows. First, the value of the controller for $k+1$ steps can be re-written as:

$$\mathcal{Y}_{k+1}^{C_\pi}(s_0, t_0) = \mathcal{Y}_{k}^{C_\pi}(s_0, t_0) + \gamma^k \sum_{h \in \mathcal{H}_k^{C_\pi}} [Pr_{C_\pi}(h) \sum_{a \in \mathcal{A}} \mathcal{P}_{req}(t_h, env(last(h)), a) \times IR^{C_\pi}(h, t_h, a)],$$

where $\mathcal{H}_k^{C_\pi}$ is the set of all histories of length k that may be reached by following controller C_π, $Pr_{C_\pi}(h)$ is probability of such history arising, and t_h stands for the resulting (unique) state of the target after having performed all the actions in h.

Second, the cumulative reward gained by policy π after $k+2$ steps can be re-written as follows:

$$V_{k+2}^{\pi}(q_{init}) = V_{k+1}^{\pi}(q_{init}) + \gamma^{k+1} \sum_{\lambda \in \Lambda_{k+1}^{\pi}} [Pr_\pi(\lambda) \times r(last(\lambda), \pi(last(\lambda)))],$$

where Λ_k^π is the set of all MDP sequence of states of length k that may be traversed when following policy π, and $Pr_\pi(\lambda)$ is the probability of sequence λ arising.

The fact that $\gamma \mathcal{Y}_{\ell}^{C_\pi}(s_0, t_0) = V_{\ell+1}^{\pi}(q_{init})$, together with the fact that every controller is always related to some policy in the MDP (even non-Markovian controllers), is enough to prove the thesis. □

This result proves the correctness of the encoding, and provides us with a technique for solving DT-composition problems, by using, for instance, policy-iteration implementations [7].

EXAMPLE 4. We generated the optimal policy for the garden scenario from Figure 1 by using a simple existing MDP solver.[2] The problem does *not* actually have an exact solution. To see that, consider the sequence of action requests *clean · water · empty* compatible with the target $\mathcal{T}_{Garden}$. It is not hard to verify that the first and last actions need to be delegated to bot $\mathcal{B}_{Cleaner}$, whereas the second action *water* ought to be delegated to bot $\mathcal{B}_{Multi}$. However, bot $\mathcal{B}_{Cleaner}$ will be able to perform the last action *empty* only if it has evolved to state a_1 after *clean*'s execution. Otherwise, if $\mathcal{B}_{Cleaner}$ happens to stay in state a_0 instead, action *empty* cannot be realized in the system $\mathcal{S}_{Garden}$ and a dead-end is reached.

Note, though, that the chances of $\mathcal{B}_{Cleaner}$ evolving to state a_0 are indeed low. Hence, an optimal controller—a composition—should still choose $\mathcal{B}_{Cleaner}$ to execute the first *clean* action. This is indeed the controller induced by the optimal policy found when solving the corresponding MDP, which is partially listed below as output by the MDP solver (`BEH0`, `BEH1`, and `BEH2` stand for behaviors $\mathcal{B}_{Cleaner}$, $\mathcal{B}_{Multi}$, and $\mathcal{B}_{Plucker}$, respectively):

```
Beh:0 0 0 | Tgt:0| Env:0|Act:CLEAN ------> BEH0
Beh:0 0 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
Beh:1 0 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
Beh:1 0 0 | Tgt:2| Env:0|Act:EMPTY ------> BEH0
Beh:0 0 0 | Tgt:2| Env:0|Act:EMPTY ------> U
...
```

[2] `http://copa.uniandes.edu.co/software/jmarkov/`

Observe that if after doing a *clean* action, behavior $\mathcal{B}_{Cleaner}$ (`BEH0`) stays in its state a_0, the policy prescribes `U`, thus signaling a dead-end in the composition.

In turn, the following rules in the policy will successfully realize the request sequence *clean · pluck · empty*:

```
Beh:0 0 0 | Tgt:0| Env:0|Act:CLEAN ------> BEH0
Beh:0 0 0 | Tgt:1| Env:0|Act:PLUCK ------> BEH1
Beh:0 1 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
Beh:0 1 0 | Tgt:3| Env:0|Act:EMPTY ------> BEH1
```

Finally, bot $\mathcal{B}_{Plucker}$ (`BEH2`) will be used by the induced controller in cases as the following ones:

```
Beh:0 1 0 | Tgt:1| Env:0|Act:PLUCK ------> BEH2
Beh:0 1 1 | Tgt:0| Env:0|Act:CLEAN ------> BEH2
```

Observe that in the configuration of the second rule, behavior $\mathcal{B}_{Cleaner}$ is also able to perform the cleaning action; however, it is best to use the plucker bot as it will bring it to state c_0, from where it is able to pluck again if needed (see that bot $\mathcal{B}_{Multi}$ is in state b_1 from where it cannot pluck).

All the above rules are only for the cases in which the environment remains in its state e_0, other (similar) rules exist in the policy/controller for other environment states. ■

Exact compositions.

As discussed, in a decision theoretic composition problem, one looks, in general, for the "best" possible controller, since exact compositions may not exist. Nonetheless, the following result states that if one does exist, it is enough to restrict to the finite horizon case in the corresponding MDP (without losing optimality).

THEOREM 4. *If there exists an exact composition for realizing a given target $\mathcal{T}$ in a system $\mathcal{S}$, then the controller induced by any $(|Q|+1)$-optimal policy for MDP $\mathcal{M}_{\mathcal{S},\mathcal{T}}$ is an exact composition.*

PROOF (SKETCH). This follows from the fact that there exists an optimal policy for $\mathcal{M}_{\mathcal{S},\mathcal{T}}$ that is stationary (which can be proven by relying on the fact that there exists a Markovian exact composition due to Theorem 1), and the fact that by optimizing the MDP up to $Q+1$ steps, it is guaranteed that *all* possible configurations of the whole composition framework—which includes both available system and target—are taken into account. □

This result is important in that it provides a way of verifying whether a DT-composition problem accepts an exact solution, namely, find an optimal policy π for horizon $|Q|+1$ and check whether $\mathcal{Y}_{|Q|}^{C_\pi} = \mathcal{R}_{|Q|}^{\max}$ (recall the first step in the MDP involves no action request and attracts no reward). Of course, it is possible to restate the above theorem in terms of an infinite horizon problem:

COROLLARY 1. *If there exists an exact composition for realizing a given target $\mathcal{T}$ in a system $\mathcal{S}$, then there exists a discount factor $\hat{\gamma}$ such that for any γ-optimal policy π for MDP $\mathcal{M}_{\mathcal{S},\mathcal{T}}$, with $\gamma \geq \hat{\gamma}$, the induced controller C_π is an exact composition of $\mathcal{T}$ in $\mathcal{S}$.*

When no exact composition exists, though, all one can do is to settle for the (best) controller induced by an optimal policy in the encoded MDP. Since non-exact compositions will include dead-ends, that is, possible histories where some target-compatible request may not be fulfilled, other mechanisms will be required to bring the overall system to a "healthy" configuration, such as resetting the whole system or even some parts of it.

We close this section by relating our approach to behavior composition with the "classical" approaches to the problem in the literature (e.g., [5, 10, 11, 4]). In such approaches, the task amounts to decide whether an *exact* composition controller exists (and to synthesize one if any) in settings under strict uncertainty. The dynamics of behaviors and that of the environment are represented by means of *transition relations*, rather than probabilistic transition functions. As a result, the designer can only model whether a transition is possible or not. In addition, the target behavior does not include a probabilistic request function $\mathcal{P}_{req}$, but simply a transition relation stating what actions can be legally requested.

As expected, the following result states that our DT-composition framework is at least as expressive as the classical one.

THEOREM 5. *For any instance of a classical behavior composition (as in [10, 11]), there is a decision-theoretic behavior composition instance such that there exists a composition solution for the former iff there exists an exact composition for the latter.*

PROOF (SKETCH). This is shown by building a DT-composition problem instance as follows:

- The environment probabilistic transition function is defined such that $\mathcal{P}^{\mathcal{E}}_{next}(e, a, e') = 1/|\Delta(e, a)|$, whenever $\langle e, a, e' \rangle \in \rho$, where ρ is the transition relation of the original classical environment and $\Delta(e, a) = \{e' \mid \langle e, a, e' \rangle \in \rho\}$.
- The probabilistic transition function for each available behavior $\mathcal{B}_i$ is defined as $\mathcal{P}^{\mathcal{B}_i}_{next}(b, e, a, b') = 1/|\Delta(b, e, a)|$, whenever $\langle b, e, a, b' \rangle \in \delta_i$, where δ_i is the transition relation of the original classical available behavior $\mathcal{B}_i$ and $\Delta(b, e, a) = \{b' \mid \langle b, e, a, b' \rangle \in \delta_i\}$.
- The probabilistic action request function of the target behavior is defined $\mathcal{P}_{req}(t, e, a) = 1/|\Delta(t, e)|$, whenever $\langle t, e, a, t' \rangle \in \delta_T$, where δ_T is the transition relation of the original target and $\Delta(t, e) = \{a \mid \langle t, e, a, t' \rangle \in \delta_T\}$.
- The target reward function is defined as $R(t, a) = 1$ for all $a \in \mathcal{A}$ and $t \in T$ such that $\mathcal{P}_{req}(t, e, a) > 0$ for some $e \in E$.

(In all other cases, the probabilities are assumed to be zero.) It is not hard to show that the resulting DT-composition instance has an exact composition iff the original classical one has a solution. □

Clearly, not every DT-composition problem can be mapped to the classical setting, as it is the case with our gardening scenario. It follows then that the framework developed here is, not surprisingly, a strict extension of the classical ones for behavior composition.

We observe that all previous approaches provide an EXPTIME upper bound to the computational complexity. Fully observable MDPs can be solved in time polynomial in the size of state space and actions [8]. Since the size of $\mathcal{M}_{\mathcal{S},\mathcal{T}}$'s state space is indeed exponential in the number of behaviors, such bound still applies here.

5. CONCLUSIONS

In this paper, we have generalized the classical behavior composition problem (e.g., [11, 5, 10]) to one that is able to account for *quantified uncertainties* in the domain, both in the dynamics of the behaviors and environment, as well as in the preferences over requests from the target user. The task then is to find the "best" controller—a composition—that maximizes the *expected realizability* of the target. Unlike previous approaches, the extended decision theoretic composition framework is able to deal with unsolvable problem instances, that is, those that do not accept exact solutions. In addition, it is provably more expressive than the classical version under strict uncertainty. In order to solve a DT-composition problem, we showed how to reduce it to the problem of finding an optimal policy in a Markov decision process, an established framework for sequential stochastic decision making.

There are many open lines of research in this framework. A natural extension is to accommodate preferences over available behaviors. In many applications, using one component may be more costly than using another one, e.g., it is preferred to transport goods by car than to do by truck. Though catering for this may appear straightforward to achieve by simply encoding, for instance, a ranking over available behaviors in the reward function of the MDP, it is not clear that all the results presented here would generalize. In fact, under such setting, exact solutions may no longer be optimal controllers. Another interesting issue is to combine our framework with that of classical behavior composition in the literature. The idea is that some actions in the target may not be compromised and *must* be met in any composition. For example, once the garden has been plucked, it is mandatory that the collected fruit be adequately stored. Yet another possibility is to generalize the framework to one under partial observability, and possibly using partially-observable MDPs (POMDPs) to tackle those cases. Lastly, if rewards and transitions are not fully known, a reinforcement learning framework could be used to find compositions while learning the domain.

Acknowledgments

We thank the anonymous reviewers and Lawrence Cavedon for their helpful comments. We also acknowledge the support of the Australian Research Council (under grant DP1094627).

6. REFERENCES

[1] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *International Journal of Cooperative Information Systems*, 14(4):333–376, 2005.

[2] D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–452, 2008.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and compuatational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, July 1999.

[4] G. De Giacomo and P. Felli. Agent composition synthesis based on ATL. In *Proc. of AAMAS*, pages 499–506, 2010.

[5] G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI*, pages 1866–1871, 2007.

[6] S. French. *Decision Theory: An Introduction to the Mathematics of Rationality*. Ellis Horwood, 1986.

[7] R. Howard. *Dynamic Programming and Markov Process*. The MIT Press, 1960.

[8] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

[9] S. Sardina and G. De Giacomo. Realizing multiple autonomous agents through scheduling of shared devices. In *Proc. of ICAPS*, pages 304–312, 2008.

[10] S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc. of KR*, pages 640–650, 2008.

[11] T. Stroeder and M. Pagnucco. Realising deterministic behaviour from multiple non-deterministic behaviours. In *Proc. of IJCAI*, pages 936–941, 2009.