Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Behavior Composition as Fully Observable Non-Deterministic Planning

Miquel Ramirez, Nitin Yadav and **Sebastian Sardina***
School of Computer Science and IT
RMIT University
Melbourne, Australia
{*miquel.ramirez,nitin.yadav,sebastian.sardina*}*@rmit.edu.au*

Abstract

The behavior composition problem involves the automatic synthesis of a controller able to "realize" (i.e., implement) a target behavior module by suitably coordinating a collection of partially controllable available behaviors. In this paper, we show that the existence of a composition solution amounts to finding a strong cyclic plan for a special non-deterministic planning problem, thus establishing the formal link between the two synthesis tasks. Importantly, our results support the use of non-deterministic planing systems for solving composition problems in an *off-the-shelf* manner. We then empirically evaluate three state-of-the-art synthesis systems (a domain-independent automated planner and two game solvers based on model checking techniques) on various non-trivial composition instances. Our experiments show that while behavior composition is EXPTIME-complete, the current technology is already able to handle instances of significant complexity. Our work is, as far as we know, the first serious experimental work on behavior composition.

Introduction

The problem of composing (i.e., coordinating) a collection of available behavior modules to implement a desired complex, but non-existent, target behavior module has recently received substantial attention in the literature. The problem, in its various forms, has been studied in many areas of Computer Science, including (web) services (Berardi et al. 2008; Balbiani, Cheikh, and Feuillade 2008), AI reasoning about action (Sardina, Patrizi, and De Giacomo 2008; Stroeder and Pagnucco 2009; De Giacomo, Patrizi, and Sardina 2013), verification (Lustig and Vardi 2009), and even robotics (Bordignon et al. 2007). From a general AI perspective, a behavior refers to the abstract operational model of a device or program, and is generally represented as a non-deterministic transition system. In a smart building setting, for instance, one may look for a so-called *controller* that is able to coordinate the execution of a set of devices installed in a house—e.g., automatic blinds and lights, audio and screen devices, video cameras, etc.—such that it appears as if a complex entertainment system was actually being run. A solution to the problem is called a *composition*.

*We acknowledge the support of the Australian Research Council under a Discovery Project (grant DP120100332).

The literature in the topic has shown how the composition problem can be effectively solved—that is how to check for composition existence—by relying on various techniques, such as PDL satisfiability (De Giacomo and Sardina 2007), direct search (Stroeder and Pagnucco 2009), LTL/ATL synthesis (Lustig and Vardi 2009; De Giacomo and Felli 2010), and computation of special kind of simulation relations (Sardina, Patrizi, and De Giacomo 2008; Berardi et al. 2008). However, all such proposals have remained, so far, at the "proof-of-concept" level and hence no thorough empirical evaluation has been performed to date.

In this paper, we present what is, up to our knowledge, the first in-depth experimental work on behavior composition that provides insights on the practical limits of some of the state-of-the-art synthesis techniques available to date. In particular, we shall focus on two competitive approaches to synthesis, namely, automated planning and model checking based game solvers. The contributions are twofold.

First, to be able to resort to planning technology, we prove *formally* Sardina, Patrizi, and De Giacomo (2008)'s observation that the composition problem—being a synthesis tasks—is closely related to planning under incomplete information (Daniele, Traverso, and Vardi 2000). More concretely, we show that solving a composition problem amounts to finding a strong-cyclic plan for a special fully-observable non-deterministic (FOND) planning problem. In doing so, a major challenge arises: the composition problem is intrinsically a generalized type of planning for safety-goals of the form *"maintain ϕ,"* (De Giacomo and Felli 2010; De Giacomo, Patrizi, and Sardina 2013), whereas most of the work on non-deterministic planning has considered *reachability goals*, that is, goals of the form *"achieve ϕ."* Furthermore, such safety-goals will generally be evaluated over *infinite* runs of the underlying system, as the target module may encode control processes that may repeat infinitely (e.g., elevator controllers, smart house embedded systems, etc). To address this mismatch, we rely on the recent work of Patrizi et al. (2011), who proposed a novel technique that allows a classical planner to *dynamically* define a reachability goal whose achievement guarantees the possibility of looping over a finite plan infinitely many times. By adapting such technique to our non-deterministic setting, our encoding enables the use of existing planners *off-the-shelf*, that is, without requiring *any* modification or extension.

The second contribution of this paper involves an empirical evaluation of three existing state-of-the-art systems that are able to synthesize such type of non-classical plans, namely, one automatic FOND planner and two model checking based game solver systems. In particular, we evaluate our encoding proposal using state-of-the-art FOND planner PRP (Muise, McIlraith, and Beck 2012) and compare it with two competitive game solver verification frameworks, namely, MCMAS (Lomuscio, Qu, and Raimondi 2009) and NUGAT (based on NUSMV (Cimatti et al. 2000)),[1] on various non-trivial classes of composition instances. Interestingly, the results obtained suggest that, despite the high computational complexity of the task at hand, the existing tools can already handle realistically sized composition instances.

The rest of the paper is organized as follows. In the next section, we provide a quick overview on behavior composition and planning under fully-observable non-deterministic domains. After that, we show how to encode a behavior composition problem into a FOND planning problem and prove the soundness and completeness of such encoding. We then report on our empirical evaluation of three state-of-the-art synthesis tools in five different classes of composition instances. We close the paper with a discussion of our findings and future work.

Preliminaries

The Behavior Composition Framework

Informally, the composition task involves the automatic synthesis of a *controller* that is able to suitably *delegate* each action request, compatible with a desired virtual *target* behavior module, to one of the *available* partially controllable behavior modules. All behaviors are modelled using finite, possibly non-deterministic, transition systems. We follow the framework in (De Giacomo and Sardina 2007; Stroeder and Pagnucco 2009; De Giacomo, Patrizi, and Sardina 2013), except that we shall ignore, wlog, the shared environment space where behaviors are meant to execute and interact with.

A *behavior* stands for the operational model of a program or device. For example, in a smart house scenario, behaviors can represent video cameras, automatic blinds, vacuum cleaners, or web browsers. Similarly, in a factory setting, a behavior may stand for a gripper arm, a paint machine, a moving robot, or a laser device. In general, behaviors provide, step by step, the user a set of actions that it can perform (relative to its specification). At each step, the behavior can be instructed to execute one of the legal actions, causing the behavior to transition to a successor state, and thereby providing a new set of applicable actions.

Formally, a *behavior* is a tuple $\mathcal{B} = \langle B, \mathcal{A}, b_0, \varrho \rangle$, where:

- B is the finite set of behavior's states;
- $\mathcal{A}$ is a set of actions;
- $b_0 \in B$ is the initial state;

[1] http://es.fbk.eu/tools/nugat/

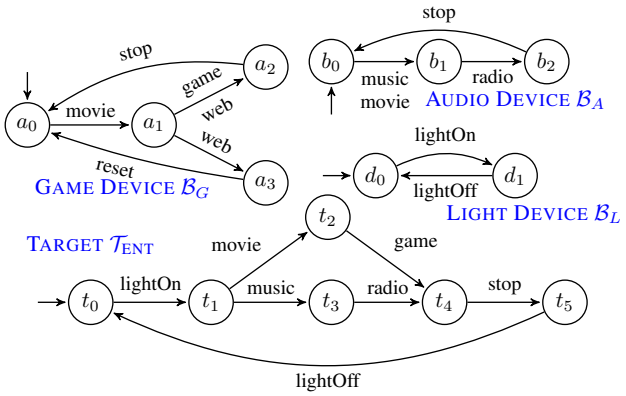

Figure 1: A smart house with three available behaviors.

- $\varrho \subseteq B \times \mathcal{A} \times B$ is the transition relation, where $\langle b, a, b' \rangle \in \varrho$, or $b \xrightarrow{a} b'$ in $\mathcal{B}$, denotes that action a executed in state b may lead the behavior to successor state b'.

Note that we allow behaviors to be non-deterministic: one cannot know beforehand what actions will be available to execute after an action is performed, as the next set of applicable actions would depend on the successor state in which the behavior happens to be in. Hence, we say that non-deterministic behaviors are only *partially controllable*. A *deterministic* behavior is one where there is no state $b \in B$ and action $a \in A$ for which there exist two transitions $b \xrightarrow{a} b'$ and $b \xrightarrow{a} b''$ in $\mathcal{B}$ with $b' \neq b''$. A deterministic behavior is *fully controllable*.

An *system* is a collection of behaviors at disposal (e.g., all devices installed in a smart house, aircraft, or factory) Technically, an *available system* is a tuple $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, where $\mathcal{B}_i = \langle B_i, \mathcal{A}_i, b_{i0}, \varrho_i \rangle$, for $i \in \{1, \ldots, n\}$, is a behavior, called an *available behavior* in the system. Finally, a *target behavior* $\mathcal{T} = \langle T, \mathcal{A}_T, t_0, \varrho_T \rangle$ is a *deterministic* behavior that represents the desired—though not available—functionality, that is, the module that one would like to have but is not readily accessible. For the sake of legibility and easier notation, we shall assume, wlog, that targets are non-blocking that is, they do not have any terminating state with no outgoing transition.[2]

Figure 1 depicts a universal home entertainment system in a smart house scenario. Target $\mathcal{T}_{\text{ENT}}$ encapsulates the desired functionality, which involves first switching on the lights when entering the room, then providing various entertainment options (e.g., listening to music, watching a movie, playing a video game, etc.), and finally stopping active modules and switching off the lights. There are three available devices installed in the house that can be used to bring about such desired behavior, namely, a game device $\mathcal{B}_G$, an audio device $\mathcal{B}_A$, and a light $\mathcal{B}_L$. Note that action web in the device $\mathcal{B}_G$ is non-deterministic: if the device happens to evolve to state a_3, then the device needs to be reset to start functioning again. It can be shown that the desired target module

[2] As customary, e.g., in LTL verification, this can be easily achieved by introducing "fake" loop transitions.

$\mathcal{T}_{\text{ENT}}$ can indeed be fully "realized," that is, implemented, by intelligently coordinating the three available devices.

Informally, the behavior composition task is stated as follows: *Given a system $\mathcal{S}$ and a target behavior $\mathcal{T}$, is it possible to (partially) control 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 the behavior used—so as to "realize" the desired target behavior?* In other words, by adequately controlling the system, it appears as if one was actually executing the target module.

To define a solution to the behavior composition problem, we consider the notion of a *controller* as a component able to activate, stop, and resume any of the available behaviors, and to instruct them to execute an (allowed) action. Note that the controller has *full observability* on the available behaviors. Formally, a *controller* for target $\mathcal{T}$ on system $\mathcal{S}$ is a partial function $C : \mathcal{H}_{\mathcal{S}} \times \mathcal{A} \mapsto \{1, \ldots, n\}$, which, given a history $h \in \mathcal{H}_{\mathcal{S}}$ of the available system (where $\mathcal{H}_{\mathcal{S}}$ is, basically, the set of all finite traces of the asynchronous product of the available behaviors) and a requested (target-compatible) action, returns the index of an available behavior to which the action in question is delegated for execution.

Intuitively, a controller (fully) realizes a target behavior if for every trace (i.e., run) of the target, at every step, the controller returns the index of an available behavior that can perform the requested action. Formally, one first defines when a controller C *realizes a trace* of the target $\mathcal{T}$. Though not required for this paper, the reader is referred to (De Giacomo and Sardina 2007) for details on how to formally characterize trace realization. Then, a controller C realizes the target behavior $\mathcal{T}$ *iff* it realizes all its traces. In that case, C is said to be an *exact composition* for target $\mathcal{T}$ on system $\mathcal{S}$.

Fully Observable Non–Deterministic Planning

Let F be a set of propositions and $L_F = F \cup \{\neg p \mid p \in F\}$ the set of *literals* over F. The *complement* of a literal l is denoted by $\bar{l}$; this notation trivially extends to sets of literals. A *state* s is a consistent subset of L_F such that $|s| = |F|$.

A *fully-observable non-deterministic (FOND) planning problem* is a tuple $\mathcal{P} = \langle F, I, A, G \rangle$, where F is a set of propositional atoms (i.e., fluents), I is a conjunction of fluents $p \in F$ which are true initially, and G is a (consistent) conjunction of literals $l \in L_F$ which are required to be made true by selecting actions $a \in A$ (Daniele, Traverso, and Vardi 2000). An action $a \in A$ is a pair $\langle \mathit{Pre}_a, \mathit{Eff}_a \rangle$ where Pre_a is a DNF formula defined over L_F and Eff_a is a conjunction of *conditional (non-deterministic) effects* (Anderson, Smith, and Weld 1998) of the form $C \rightarrow E$, where C is a conjunction of literals from L_F and $E = e_1 \oplus \cdots \oplus e_n$, where $n \geq 1$ and each e_i is a conjunction over L_F, representing all the possible (exclusive) effects of the action when C holds true—exactly *one-of* the n effects must apply (Bonet and Givan 2005). When $n = 1$, we say that action a is deterministic; otherwise, a is non-deterministic. For legibility, we compactly denote conditional effects of the form $\top \rightarrow E$ by just E.

The semantics of the a FOND problem $\mathcal{P}$ as above is given by a *non-deterministic state model* $\Pi_{\mathcal{P}} = \langle S, s_0, S_G, A, A(s), \delta(a, s) \rangle$, where:

- $S \subseteq 2^F$ is the finite set of *states* of $\Pi_{\mathcal{P}}$;
- $s_0 \in S$ is the (single) *initial state* corresponding exactly to I, such that $s_0 \models I \bigwedge_{p \in I_{\text{CWA}}} \neg p$, where I_{CWA} is the set of propositions not in I, that is, $I_{\text{CWA}} = \{p \in F \mid I \not\models p\}$. Note the closed world semantics in defining s_0;
- $S_G = \{s \in S \mid s \models G\}$ is the set of *goal states*;
- $A(s) = \{a \mid s \models \mathit{Pre}_a\}$ is the *applicability relation* stating what actions are feasible (for execution) in state s; and
- $\delta(a, s) = \{s' \mid \text{for all } C \rightarrow \bigoplus_{\ell=1}^{n} e_\ell \in \mathit{Eff}_a \text{ s.t. } s \models C, \text{ there is } i \leq n \text{ s.t. } s' = (s \setminus \{l \mid \bar{l} \in e_i\}) \cup \{l \mid l \in e_i\}\}$ is the *non-deterministic transition function* denoting all possible states s' resulting from executing action a on state s. Basically, for every conditional effect of the action *when* condition C holds, exactly *one-of* the n possible effects ensue.

Solutions to FOND planning problems are *policies* $\pi : S \mapsto A$, with the intended meaning that action $\pi(s)$ is to be performed in state s, that are guaranteed to transform the initial state s_0 into some goal state $s_g \in S_G$. A policy is *closed* w.r.t. a state s if and only if the set of *reachable* states from s using π, denoted $S_\pi(s)$, is a subset of the domain of π, that is, $S_\pi(s) \subseteq \mathit{Dom}(\pi)$. A policy is deemed *proper* w.r.t. a state s if goal states can be reached using π from *all* π-reachable states $s' \in S_\pi(s)$. A policy π is *acyclic* if for all possible executions $\tau = s_0 s_1 s_2 \cdots$ of π from s_0, it holds that $s_i \neq s_j$, for any $i \neq j$.

Finally, a policy π is a *valid solution* for a FOND planning problem iff π is *both* closed and proper w.r.t. the initial state s_0. In turn, valid policies π can be further classified as either *strong* or *strong–cyclic* plans (Daniele, Traverso, and Vardi 2000), depending on whether π is *acyclic* or not, respectively.

Compiling Behavior Composition into Non–Deterministic Planning

The link between behavior composition and planning is not a novel idea. In fact, it was first suggested by Sardina, Patrizi, and De Giacomo (2008), noting that the task in both problems consists in deriving a *control* function. In planning, this control function maps world states onto actions, while in behavior composition, the control function maps *system histories* and actions onto components that actually execute actions. In this section, we formalize this connection by proposing a scheme to map behavior composition problems into FOND planning tasks.

From now on, let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ be an available system, where $\mathcal{B}_i = \langle B_i, \mathcal{A}_i, b_{i0}, \varrho_i \rangle$, for $i \in \{1, \ldots, n\}$, and $\mathcal{T} = \langle T, \mathcal{A}_T, t_0, \varrho_T \rangle$ be a target module. We define the non-deterministic planning domain $\mathcal{D}_{\langle \mathcal{S}, \mathcal{T} \rangle} = \langle F, I, A \rangle$ as follows. The set of F is built from the following fluents:

- $\mathit{tgt}(t)$, for each $t \in T$, denoting that target $\mathcal{T}$ is in state t;
- $\mathit{beh}(i, b)$, for each $i \in \{1, \ldots, n\}$ and $b \in B_i$, denoting that behavior $\mathcal{B}_i$ is in state b;

- *req*(a) and *srv*(a), for each $a \in \mathcal{A}_T$, denoting that action a has been requested and served, respectively;
- *ready*, denoting that the target $\mathcal{T}$ is ready to issue a new action request.

The initial state I encodes the starting configuration of all available behaviors as well as the target module:

$$I = \mathit{ready} \wedge \mathit{tgt}(t_0) \wedge \bigwedge_{1 \leq i \leq n} \mathit{beh}(i, b_{i0}).$$

The set of actions A is made up of three distinguished action schemas modeling the dynamics of $\mathcal{S}$ and $\mathcal{T}$ and designed to execute in a fixed sequence as follows. First, the target module issues a valid action request (via action generate(t)). Then, a delegation of such action to some behavior is performed (via action delegate(i, a)). Finally, the target evolves and becomes ready to issue a new request (via action progress(t, a)). We formally define these next.

Action generate(t) *non-deterministically* selects what action is requested next when the target is in state t. To issue a request from state t, the target needs to be "ready" and in state t. The effect states that one-of the possible request from state t will ensue and the target is not "ready" anymore:

$$\text{generate}(t) = \langle \mathit{ready} \wedge \mathit{tgt}(t), \bigoplus_{a \in R(\mathcal{T}, t)} (\mathit{req}(a) \wedge \neg \mathit{ready}) \rangle,$$

where $R(\mathcal{B}, s) = \{a \mid s \xrightarrow{a} s' \text{ in } \mathcal{B}\}$ stands for the actions that can be performed by behavior $\mathcal{B}$ in state s. Note the non-determinism on which action is selected.

To delegate an action to a behavior $\mathcal{B}_i$, the action has to be "pending" (that is, currently requested) and the behavior in question able to perform it from its current state. When that happens, the action is considered just "served" (that is, fulfilled) and not requested anymore, and the corresponding behavior evolved as per its transition model. Formally, action delegate$(i, a) = \langle \mathit{Pre}_{\text{del}}, \mathit{Eff}_{\text{del}} \rangle$, where:

$$\mathit{Pre}_{\text{del}} = \bigvee_{(b,a,b') \in \varrho_i} (\mathit{beh}(i, b) \wedge \mathit{req}(a));$$

$$\mathit{Eff}_{\text{del}} = \mathit{srv}(a) \wedge \neg \mathit{req}(a) \wedge \bigoplus_{(b,a,b') \in \varrho_i} \mathit{trans}(b, a, b'),$$

where $\mathit{trans}(b, a, b') \doteq (\mathit{beh}(i, b') \wedge \neg \mathit{beh}(i, b))$ when $b \neq b'$, and $\mathit{trans}(b, a, b') \doteq \mathit{beh}(i, b)$, otherwise.

The final step involves updating the target module via the execution of action progress(t, a), defined for each $t \in T$ and $a \in R(\mathcal{T}, t)$ as follows:

$$\text{progress}(t, a) = \langle \mathit{srv}(a) \wedge \mathit{tgt}(t), \ \mathit{ready} \wedge \neg \mathit{srv}(a) \wedge \bigoplus_{(t,a,t') \in \varrho_T} \mathit{trans}(t, a, t') \rangle,$$

where $\mathit{trans}(t, a, t') \doteq (\mathit{tgt}(t') \wedge \neg \mathit{tgt}(t))$ when $t \neq t'$, and $\mathit{trans}(t, a, t') \doteq \mathit{tgt}(t)$, otherwise.

Looking at the preconditions of the three actions, it is easy to see that execution of the domain will involve sequences of the form generate$(t) \cdot$ delegate$(i, a) \cdot$ progress(t, a), expressing a complete action-request and delegation cycle.

While the above encoding of the dynamics behind a behavior composition problem is relatively straightforward and intuitive, it is still not expressive enough to support the synthesis of composition controllers (i.e., policies) for target behaviors containing loops. The fact is that when it comes to behavior composition, one does not plan for an achievement goal, as standard in FOND planning, but for a safety property: "*always satisfy request*." Nonetheless, computing *finite* plans which satisfy certain safety constraints using classical planners has received considerable attention in the last years (Baier, Bacchus, and McIlraith 2009; Bauer and Haslum 2010). However, none of these approaches deal with the general problem of computing finite plans that are meant to be executed infinitely many times, as required in many control scenarios (e.g., elevator controllers, smart houses, etc). It is precisely those type of plans that are required in order to handle target modules containing loops that may yield infinite execution runs.

To be able to deal with safety goals over infinite runs of the target module, we resort to the recent novel technique proposed in (Patrizi et al. 2011) and adapt it for our non-deterministic safety-goal synthesis scenario. Roughly speaking, the idea is to find a classical finite plan encoding a so-called *lasso*. Namely, a sequence of actions π_1, mapping the initial state of a system into some state s, is followed by a second action sequence π_2 that maps s into itself, which is assumed to be repeated infinitely often. Their technique, then, amounts to non-deterministically selecting the current state of the system as "start looping" configuration, and then trying to reach the exact same configuration a second time. This allows the authors to synthesize finite plans for temporally extended goals over infinite runs using classical planners. In our case, we are dealing with a non-deterministic domain. Nonetheless, the same idea generalizes to the set of *executions* of a strong cyclic plan π, where there may be a (possibly empty) prefix of the execution of π achieving the state s where the loop starts, and subsequent actions in the execution lead to a *terminal* state that can be mapped into s.

So, we further extend $\mathcal{D}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ above with auxiliary fluents and actions and obtain the following non-deterministic planning problem $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle} = \langle F', I', A', G \rangle$. First, let $\hat{F} = \{\mathit{beh}(i, b) \mid i \in \{1, \ldots, n\}, b \in B_i\} \cup \{\mathit{tgt}(t) \mid t \in T\}$ be the subset of F containing all the fluents used to keep track of the state of all behaviors. The set of fluents is defined as:

$$F' = F \cup \{\mathit{rec}(p), \mathit{clsd}(p) \mid p \in \hat{F}\} \cup \{L_{\mathit{init}}, L_{\mathit{end}}, \mathit{canEnd}\},$$

where *rec*(p) and *clsd*(p)—p is recorded and closed—are (auxiliary) fluents that will be used by the planner to "record" the underlying composition configuration to which we aim to return after executing a lasso-type plan/policy. In turn, fluents L_{init} and L_{end} denote that the (search for the) lasso loop has already been initiated (and is yet active) and completed, respectively, whereas fluent *canEnd* is used to rule out empty lasso-loops with no action delegation.

To better understand the new fluents, we next explain the two new actions—loop and end—used to "guess" the lasso in the policy. The action loop $= \langle \mathit{Pre}_{\text{loop}}, \mathit{Eff}_{\text{loop}} \rangle$ is used to *commit* to a particular configuration of $\langle \mathcal{S}, \mathcal{T} \rangle$ as the

achievement goal to be bring about—*the start of the lasso*:

$$\mathit{Pre}_{\text{loop}} = \neg L_{\mathit{init}} \wedge \mathit{ready};$$
$$\mathit{Eff}_{\text{loop}} = L_{\mathit{init}} \wedge \neg \mathit{canEnd} \wedge \bigwedge_{p \in \hat{F}} [p \rightarrow (\mathit{rec}(p) \wedge \neg \mathit{clsd}(p))].$$

That is, a loop can be "started" if it has not yet started and the overall system is in a "ready" state where a new action-request and delegation cycle is about to start. The effects of loop includes "recording" each true fluent p into $\mathit{rec}(p)$ and marking them as not "closed"—they need to be eventually re-instantiated. In addition, it makes L_{init} true to signal the start of the loop synthesis and *canEnd* false to avoid "empty" loops (see end action next).

Second, the action end $= \langle \mathit{Pre}_{\text{end}}, \mathit{Eff}_{\text{end}} \rangle$ is used to mark, or more precisely "guess," the end of the lasso loop. The loop can be ended if it has already started and is legal to end it. The action causes the closure of every proposition that was previously recorded—via a loop action—and has been re-instantiated:

$$\mathit{Pre}_{\text{end}} = L_{\mathit{init}} \wedge \mathit{canEnd};$$
$$\mathit{Eff}_{\text{end}} = \neg \mathit{ready} \wedge \neg L_{\mathit{init}} \wedge L_{\mathit{end}} \wedge \bigwedge_{p \in \hat{F}} p \wedge \mathit{rec}(p) \rightarrow \mathit{clsd}(p).$$

To be able to guess the end of the lasso, at least one action-request and delegation cycle must ensue after the start of the lasso loop, that is, we do not allow end to follow right after loop as this will imply an "empty" loop. To achieve this, we just add proposition *canEnd* to the effects of action progress(t, a), and define $A' = A \cup \{\text{loop}, \text{end}\}$

The initial state of $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ extends that of $\mathcal{D}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ to account for the fact that no commitment has been made on any particular configuration, that is, $I' = I \wedge \bigwedge_{p \in \hat{F}} \mathit{clsd}(p)$. Observe that under I', propositions *canEnd*, L_{init}, L_{end}, and $\mathit{rec}(p)$, for all $p \in F_{\mathcal{S}}$, do not hold initially.

Finally, the goal encodes the requirement of being in a state where the lasso loop has ended and every proposition previously recorded has been re-instantiated (i.e., they are not "open" anymore). Formally:

$$G = L_{\mathit{end}} \wedge \bigwedge_{p \in \hat{F}} \mathit{clsd}(p).$$

Note that due to the encoding, the fact that all opened fluents have been closed is enough to guarantee that the state of the behavior composition domain has come back to that where the loop action was executed before. This is because the propositions encoding the state of each behavior are mutually exclusive. Also, observe that once action end is executed, no more actions are applicable. Therefore, end results in a non-goal terminal state, whenever the state where action end is done cannot be mapped into the state where loop was done.

The main result shows that there is a one-to-one relationship between strong cyclic solution plans π for $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ and controllers C that are an composition solutions for target $\mathcal{T}$ on available system $\mathcal{S}$:

Theorem 1. *Let $\langle \mathcal{S}, \mathcal{T} \rangle$ be a behavior composition problem instance and $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ the resulting FOND planning problem obtained from the above encoding. Then, there exists a composition solution C for $\langle \mathcal{S}, \mathcal{T} \rangle$* if and only if *there exists a* strong cyclic *plan solution π for $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$.*

Proof sketch. We start by recalling that a controller C is a composition solution for $\mathcal{T}$ on system $\mathcal{S}$ if for all the traces of $\mathcal{T}$, C selects, at each step, a behavior $\mathcal{B}_i$ able to execute the corresponding requested action. We then observe that there exists an *exact* correspondence between steps $s \xrightarrow{a,i} s'$ in histories $h \in \mathcal{H}_{\mathcal{S}}$ of the enacted system $\mathcal{E}_{\mathcal{S}}$ (De Giacomo, Patrizi, and Sardina 2013) and the start and end planning states in a sequence of planning actions generate$(t) \cdot$ delegate$(i, a) \cdot$ progress(t, a). Next, if a strong cyclic solution plan π exists for $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$, then the paths in the execution structure K (Cimatti et al. 2003) induced by π correspond to the histories h of $\mathcal{H}_{\mathcal{S}}$, and moreover the valuation of $\hat{F}$ in terminal $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ states of K encodes some previous (composition) state in h, yielding thus the lasso loop. Finally, we demonstrate how all infinite traces of $\mathcal{T}$ can be (finitely) accounted by putting together these "looping" histories h encoded in K. □

We observe that π is encoding a *memory-less* controller C, which are known to suffice for behavior composition problems (De Giacomo, Patrizi, and Sardina 2013).

Strong cyclic plans formalize the notion of *acceptable* "trial and error" strategies. Such strategies are encoded by plans whose partial executions, possibly containing an undetermined number of cycles of unbounded length, can always be extended to reach the goal (Cimatti et al. 2003). Nonetheless, strong cyclic policies *can* yield executions that loop forever when an action always fails to produce some of its outcomes. There are two actions with non-deterministic outcomes in our compilation, namely, generate(t) and delegate(i, a). Built into the compilation, yet not on the definition of behavior composition problems, is the assumption that both of these actions are "fair," that is, all of their outcomes *eventually* occur. This assumption is needed in order to produce controllers that can account for any (legal) target request and available behaviors' evolutions.

Importantly, our compilation scheme accounts for targets with any number of nested loops, as strong cyclic plans allow executions to feature an arbitrary number of cycles of any length. For simplicity in the presentation, it does not however account for targets $\mathcal{T}$ with *finite* traces, for instance, when $\mathcal{T}$ is a chain. Nonetheless, it is straightforward to account for these cases, by adding dummy *no-op* looping requests, as is standard in verification.

We close this section by noting that the encoding developed above is optimal w.r.t. computational complexity. Since FOND planning is EXP-complete and the encoding is polynomial on the size of the composition problem, it follows that checking the existence of a solution for $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ is EXP-complete, thus matching the complexity of the behavior composition (De Giacomo and Sardina 2007).

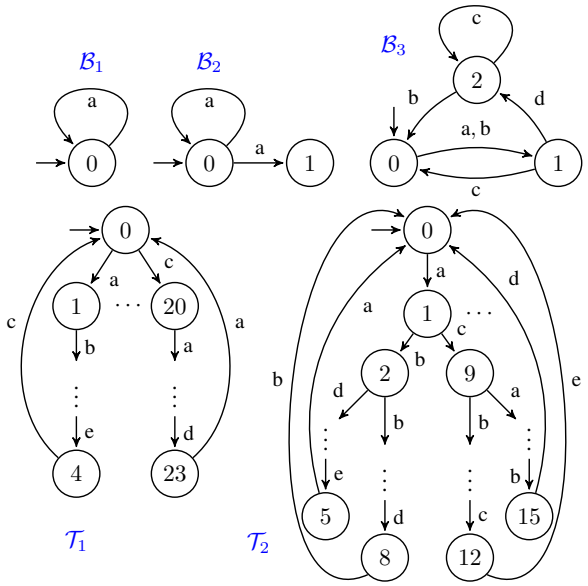

Figure 2: Structure of behaviors used in the benchmarks. Actions requested by targets and implemented by behaviors are denoted as lower case letters.

Experimental Evaluation

In order to assess the performance of the state-of-the-art FOND planners on composition planning tasks $\mathcal{P}_{\langle \mathcal{S},\mathcal{T} \rangle}$ we have designed five different synthetic benchmarks,[3] identified with the letters A, C, A_U, R_1 and R_2. Each benchmark contains behavior composition instances made up of behavior and target modules with specific structural features, as shown in Figure 2:

A **&** C Available behaviors of type $\mathcal{B}_1$ and targets of type $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Both solvable and unsolvable instances are generated.

A_U Like benchmark A, but for every behavior of type $\mathcal{B}_1$, we included several behaviors of type $\mathcal{B}_2$, as "noise." Problems in this benchmark are all solvable and account for scenarios where the composition solver has to deal with a mix of reliable behaviors ($\mathcal{B}_1$) and unreliable behaviors ($\mathcal{B}_2$). Note that type $\mathcal{B}_2$ behaviors may become useless at any point, if they evolve to state 1.

R_1 **&** R_2 Available (deterministic) behaviors are generated randomly with up to five states, resulting in modules like $\mathcal{B}_3$ with intricate structures, and target modules of type $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively.

Benchmarks A, C, R_1 and R_2 feature between 10 and 60 available behaviors (i.e., devices), while we allowed for much larger (up to 120) set of behaviors in benchmark A_U. As such, the number of behaviors is large enough to preclude any manual composition. See that the various examples in the literature, e.g., (De Giacomo and Sardina 2007; Stroeder and Pagnucco 2009; Yadav and Sardina 2012), never contain more than three available behaviors and are still not trivial to solve manually. When it comes to the structure of behaviors, target modules of type $\mathcal{T}_1$ and $\mathcal{T}_2$ capture, intuitively, *sets* and *trees* of (repeating) processes that the target user may request. In turn, available behaviors $\mathcal{B}_1$ and $\mathcal{B}_2$ stand for one action actuators, while behaviors of type $\mathcal{B}_3$ with up to five states are able to model simple though concrete devices like switches, automatic blinds and lights, vacuum cleaners, and microwaves, among others. In addition, in benchmarks R_1 and R_2, the various $\mathcal{B}_3$ type devices may overlap in their functionalities (i.e., actions). One could then argue that the number and structures of behaviors used in our benchmarks are enough to capture some real-world settings, such as smart house rooms equipped with a few dozen (simple) devices.

The structure of behaviors in benchmarks A, C and A_U was deliberately made simple to highlight the power of the domain-independent strategy at the heart of state-of-the-art FOND planners (Yoon, Fern, and Givan 2007). This strategy amounts to first finding a *weak plan* (Cimatti et al. 2003) for $\mathcal{P}_{\langle \mathcal{S},\mathcal{T} \rangle}$, by using a *classical* planner on an automatically derived deterministic relaxation of $\mathcal{P}_{\langle \mathcal{S},\mathcal{T} \rangle}$, and then using such plan to guide the search for strong cyclic plans over the original FOND problem $\mathcal{P}_{\langle \mathcal{S},\mathcal{T} \rangle}$. Concretely, actions that are not part of a weak plan are ruled out, as they will never be part of any strong cyclic plan. In benchmarks R_1 and R_2, the structure of the problems generated is more complex, leading to a greater variety of interaction between behaviors' internal states and target requests.

We have evaluated[4] the performance of state-of-the-art FOND planner PRP over the five benchmarks described above. We chose PRP over other FOND planners such as FIP (Fu et al. 2011), since PRP has been shown to clearly outperform FIP over the International Planning Competition FOND benchmark (Muise, McIlraith, and Beck 2012). Behavior Composition problems can be also mapped into *safety games* (De Giacomo and Felli 2010), which are then solved with game solvers typically using model checking techniques. In order to get a better picture of the feasibility of our mapping of behavior composition problems into FOND problems, we compared PRP performance with that of two state-of-the-art verification frameworks, MCMAS (Lomuscio, Qu, and Raimondi 2009) and NuGaT, a two-player safety game solver implemented on top of NuSMV (Cimatti et al. 2000). We did not modify in any way the source code of any of the software systems used, and relied on the default options suggested by the software authors, with the exception of NuGaT, where we found that allowing it to use dynamic variable ordering during BDD construction had a critical effect on its performance performance (see discussion below).

[3] Unfortunately, as far as we know, there are no readily available benchmarks for behavior composition.

[4] Experiments were run on a dual-processor Intel E8500, 3.2GHz CPUs, 8 GB of RAM, with processes being allowed to run for a max of 30 minutes and use up to 2 GB of physical memory.

		NuGaT			McMAS			PRP			
	#I	#S	#T	AT	#S	#T	AT	#S	#T	#M	AT
A	144	144	0	0.13	90	54	112	144	0	0	0.16
C	252	252	0	0.12	171	81	85	252	0	0	0.6
A_U	192	192	0	11.9	12	180	197.4	192	0	0	0.9
R_1	56	31	25	98.75	4	52	195.5	18	7	31	95
R_2	55	29	26	218	3	52	1104	10	8	37	63.3

Table 1: Performance of NuGaT, McMAS and PRP on the proposed set of benchmarks (see details in the text).

The results of our experimental evaluation are summarize in Table 1. For each of the benchmarks, we report the number of composition instances (column #I), the number of problems for which each solver finished and did not finish its execution within the allotted time (columns #S and #T, resp.), and, lastly, the average run-time in seconds reported for each solver (column AT). For PRP, we also report the number of problems that exhausted the memory limit (column #M); neither NuGaT nor McMAS ever exhausted the available memory resources. The proportion of problems in benchmarks A and C for which there does not exist a composition solution is about 50% (65 out of 144 in benchmark A; 135 out of 252 in benchmark C). In these benchmarks, there was no significant difference between the average times for problems which had a solution and those which did not.

We note the remarkable performance of both NuGaT and PRP on benchmark A: both solve all the problems within 30 minutes and show similar run-times. In comparison, McMAS can only solve 60% of the problems in the benchmark, with run-times several orders of magnitude bigger than both NuGaT and PRP. The performance of McMAS is indeed in line with previously observations on the performance of verification tools on classical planning domains (Patrizi et al. 2011), or the performance of FOND planners with embedded classical planners such as NDP (Kuter and Nau 2008) or FIP (Fu et al. 2011) compared to FOND planners that rely on a verification framework like MBP (Cimatti et al. 2003). Indeed, the size of the OBDD formula reported by McMAS, when it solves a problem, are already in the billions of nodes for $|\mathcal{S}| = 30$.

The outstanding performance of NuGaT is explained by the usage of dynamic variable ordering on the BDD formulas denoting set of states of the fully combined system $\mathcal{S} \times \mathcal{T}$ when performing the strong reachability analysis required to solve a safety-game. While the construction of BDD using dynamic variable ordering may be slower, their size is also reduced, and on the particular class of formulas conveyed by the problems in our benchmarks, the speedup of the fixed-point computation performed by NuGaT fully amortizes the overhead associated with the dynamic variable ordering scheme. When we ran NuGaT over this same benchmark without enabling dynamic variable ordering, NuGaT performance was similar to that of McMAS.

PRP outperforms NuGaT on benchmark A_U. In Figure 3, we can see that as $|\mathcal{S}|$ grows, PRP advantage over NuGaT becomes apparent, finding solutions up to two order of magnitudes faster than NuGaT. In comparison, McMAS is only able to handle the smallest set of problems in the benchmark, which feature 15 behavior modules.

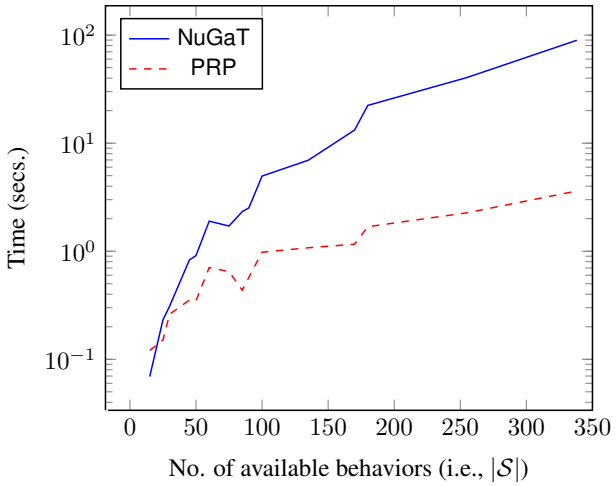

Figure 3: Run-time comparison on benchmark A_U. Each data point is the average of the run-times for all 12 problems with the same number of available behaviors.

On the other hand, PRP performance degrades on benchmarks C, R_1, and R_2. In benchmark C, both NuGaT and PRP are able to solve all the problems within 30 minutes but, as shown in Figure 4, PRP becomes orders of magnitude slower than NuGaT as the level of non-determinism in the planning task $\mathcal{P}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ increases. When the target module $\mathcal{T}$ can request several actions in each target state, as is indeed the case with target type $\mathcal{T}_2$ depicted in Figure 2, the number of possible executions of a policy π becomes bounded by $O(b^d)$, where b is the maximum number of possible requests per target state and d the length of the maximal loop induced by $\mathcal{T}$. The strong cyclic plans for planning tasks in benchmark A and A_U have, in contrast, a linear number of executions. This is made evident in benchmarks R_1 and R_2, where for smaller problems—in terms of number of fluents and actions—than the largest on benchmark A_U, PRP is either running out of time or, interestingly, out of memory well before the 30 minutes time limit. When increasing the time allowed from 30 minutes to 4 hours we observe that for benchmark R_1, 2 more tasks are solved by PRP, running out of memory in 8 additional tasks. On benchmark R_2, the effect of increasing the time limit similarly removes the time outs, PRP solving 5 additional tasks, and running out of memory in 3 more tasks. NuGaT performance profile is not affected by the increase in CPU time on benchmarks R_1, yet it solves 5 more tasks on R_2. Although NuGaT lacks the heuristic machinery that guides PRP policy search so effectively in benchmarks A, C and A_U, it still shows a robust performance profile. At the same time, when the structure of the behavior composition problems is not exploited by PRP domain-independent heuristics, the blind search performed by NuGaT on a more succint search space pays off, and NuGaT performance is clearly better than that of PRP.

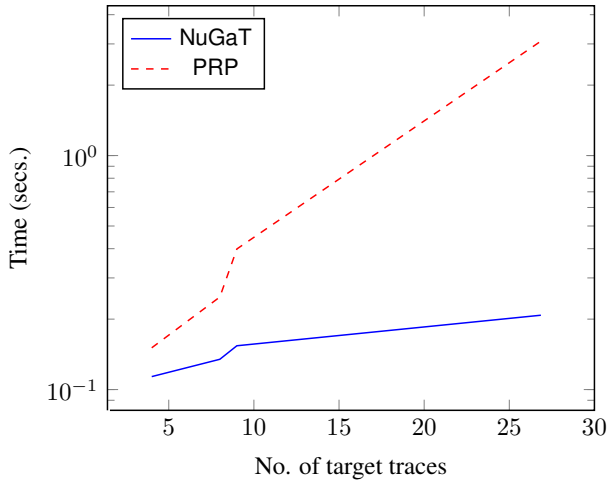

Figure 4: Runtime comparison over benchmark C. Each data point is the average of the runtimes for all problems with the same number of possible target traces.

Conclusions

In this paper we have provided, as far as we are aware, the first empirical analysis on solving behavior composition problems with state-of-the-art synthesis techniques and tools. In particular, we have used automated planning and model checking based techniques to synthesize controllers able to bring about a (virtual) target desired module by suitably coordinating a set of available modules. To that end, we first showed how behavior composition can be formally related to finding strong-cyclic solutions for a nondeterministic planning problem, thus finally providing the link between behavior composition and planning. Then, we performed an empirical evaluation of three synthesis tools—one FOND planner and two game solvers—to solve behavior composition instances over five different type of instances.

Our findings suggest that despite the high computational complexity of the problem at hand, current tools can handle non-trivial instances that are arguably of sizes expected in many real-world settings, such as smart houses, web-services, or manufacturing environments. For example, a couple of dozen behaviors with three to five states would be enough to account for smart spaces with a plethora of simple devices like automatic lights and blinds, video cameras, TVs, vacuum cleaners, among others.

When it comes to reducing behavior composition to a planning task, our encoding relies on two main ingredients. First, we transform a safety (i.e., maintenance) goal of the form *"always delegate pending requests"*, or equivalently *"never fail delegating requests,"* into a standard reachability goal compatible with the planning paradigm. It should be noted that computing *finite* plans which satisfy certain safety constraints using classical planners has in fact received considerable attention in the last years (Mattmüller and Rintanen 2007; Baier, Bacchus, and McIlraith 2009; Bauer and Haslum 2010). However, none of these approaches deal with the general problem of computing finite plans that are meant to be *executed infinitely many times*, as required in many control scenarios (e.g., elevator controllers, web-services, smart houses, etc.). This is indeed the case with behavior composition when the target module contains loops encoding non-terminating processes—sequences of requests are to be served an unbounded, or possibly infinite, number of times. In order to address this, the second ingredient of our encoding involves adapting Patrizi et al. (2011)'s novel technique so as to force policies to generate delegation executions that always include *lasso* loops (over the composition domain state model), thus allowing the infinite repetition of finite delegation strategies. We note that the work in (Pistore and Traverso 2001; Bertoli, Pistore, and Traverso 2010) addresses this as well, by integrating CTL model checking facilities into the pioneering planner MBP. We argue, though, that while CTL does indeed capture well our requirements, it is actually far more expressive than strictly necessary. More importantly, we are interested here in using existing FOND planning systems without modifications or extensions of any kind.

The results reported in this paper demonstrate that using state-of-the-art FOND planners *off-the-shelf* is an effective approach to solve behavior composition instances. However, it is clear that in order to outperform systems such as NuGaT, based on model checking technology, one needs to do significant changes into existing planners. First and simplest, one needs to change the classical planner embedded in PRP in order to strengthen the planning heuristic. Recent work in classical planning (Keyder, Hoffman, and Haslum 2012) shows how to obtain very strong non-admissible heuristics by keeping track a suitable set of critical conjunctions while approximating the delete-relaxation (no other components of the planner, such as the search, are modified). While identifying this set of conjunctions could be difficult in general, it is fairly easy, in our context, to identify good candidates for such conjunctions as part of the process of building composition planning tasks $\mathcal{P}_{\langle \mathcal{S},\mathcal{T} \rangle}$. The empirical results also make us curious about NuGaT's fixed-point computation algorithm and compressed representation of the nondeterministic state model underlying $\langle \mathcal{S}, \mathcal{T} \rangle$, so as to wonder whether state-of-the-art planners can be pushed forward by looking at model checking based synthesis techniques. We plan to pursue these ideas in our future work.

References

Anderson, C.; Smith, D.; and Weld, D. 1998. Conditional effects in graphplan. In *Proc. of the International Conference on AI Planning & Scheduling (AIPS)*, 44–53.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 1(173):593–618.

Balbiani, P.; Cheikh, F.; and Feuillade, G. 2008. Composition of interactive web services based on controller synthesis. In *Proc. of the IEEE Congress on Services (SERVICES)*, 521–528.

Bauer, A., and Haslum, P. 2010. LTL goal specifications

revisited. In *Proc. of the European Conference in Artificial Intelligence (ECAI)*, 881–886.

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

Bertoli, P.; Pistore, M.; and Traverso, P. 2010. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence Journal* 174(3-4):316–361.

Bonet, B., and Givan, R. 2005. 5th international planning competition: Non-deterministic track. call for participation. Technical report.

Bordignon, M.; Rashid, J.; Broxvall, M.; and Saffiotti, A. 2007. Seamless integration of robots and tiny embedded devices in a PEIS-ecology. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3101–3106.

Cimatti, A.; Clarke, E. M.; Giunchiglia, F.; and Roveri, M. 2000. NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)* 2(4):410–425.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1):35–84.

Daniele, M.; Traverso, P.; and Vardi, M. 2000. Strong cyclic planning revisited. *Recent Advances in AI Planning* 35–48.

De Giacomo, G., and Felli, P. 2010. Agent composition synthesis based on ATL. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, 499–506.

De Giacomo, G., and Sardina, S. 2007. Automatic synthesis of new behaviors from a library of available behaviors. In Veloso, M. M., ed., *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1866–1871.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2013. Automatic behavior composition synthesis. *Artificial Intelligence Journal* 196:106–142.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1949–1954.

Keyder, E.; Hoffman, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 128–136.

Kuter, U., and Nau, D. 2008. Using classical planners to solve non-deterministic planning problems. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 513–518.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. of the International Conference on Computer Aided Verification (CAV)*, 682–688.

Lustig, Y., and Vardi, M. Y. 2009. Synthesis from component libraries. In *Proc. of the International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, 395–409.

Mattmüller, R., and Rintanen, J. 2007. Planning for temporally extended goals as propositional satisfiability. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1966–1971.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 172–180.

Patrizi, F.; Lipovetzky, N.; Giacomo, G. D.; and Geffner, H. 2011. Computing infinite plans for LTL goals using a classical planner. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 479–486.

Sardina, S.; Patrizi, F.; and De Giacomo, G. 2008. Behavior composition in the presence of failure. In Brewka, G., and Lang, J., eds., *Proc. of Principles of Knowledge Representation and Reasoning (KR)*, 640–650.

Stroeder, T., and Pagnucco, M. 2009. Realising deterministic behaviour from multiple non-deterministic behaviours. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 936–941.

Yadav, N., and Sardina, S. 2012. Qualitative approximate behavior composition. In *Proc. of the European Conference on Logics in Artificial Intelligence (JELIA)*, volume 7519 of *Lecture Notes in Computer Science (LNCS)*, 450–462. Springer.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 7, 352–359.