Two-Player Game Structures for Generalized Planning and Agent Composition

Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi
Dipartimento di Informatica e Sistemistica
SAPIENZA Università di Roma
Roma, ITALY
lastname@dis.uniroma1.it

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

Abstract

In this paper, we review a series of agent behavior synthesis problems under full observability and nondeterminism (partial controllability), ranging from conditional planning, to recently introduced agent planning programs, and to sophisticated forms of agent behavior compositions, and show that all of them can be solved by model checking two-player game structures. These structures are akin to transition systems/Kripke structures, usually adopted in model checking, except that they distinguish (and hence allow to separately quantify) between the actions/moves of two antagonistic players. We show that using them we can implement solvers for several agent behavior synthesis problems.

Introduction

AI has been long concerned with agent behavior synthesis problems: the best known such problem being Automated Planning in its various forms (Ghallab, Nau, and Traverso 2004). Here we consider a variety of agent behavior synthesis problems characterized by *full observability* and *nondeterminism* (i.e., partial controllability). Specifically, we focus on three problems of increasing sophistication. The simplest problem we consider is standard *conditional planning* in nondeterministic fully observable domains (Rintanen 2004), which is well understood by the AI community. Then, we move to a sophisticated form of planning recently introduced in (De Giacomo, Patrizi, and Sardina 2010), in which so-called *agent planning programs*—programs built only from achievement and maintenance goals—are meant to merge two traditions in AI research, namely, Automated Planning and Agent-Oriented Programming (Wooldridge 2009). Solving, that is, realizing, such planning programs requires temporally extended plans that loop and possibly do not even terminate, analogously to (Kerjean et al. 2006). Finally, we turn to *agent behavior composition*. Composition of nondeterministic, fully observable available behaviors for realizing a *single* target behavior was studied, e.g., in (Sardina, De Giacomo, and Patrizi 2008; Stroeder and Pagnucco 2009), and it is linked to composition of stateful, or "conversational," web services (Su 2008). Here, we shall consider an advanced form of that composition, in which several devices are composed in order to realize *multiple virtual agents* simultaneously (Sardina and De Giacomo 2008). Such a problem is relevant for robot ecology, ubiquitous robots, or intelligent spaces (Lundh, Karlsson, and Saffiotti 2008).

The techniques originally proposed for the above synthesis problems are quite diverse, ranging from specific forms of planning (for conditional planning), to simulation (for composition), to LTL-based synthesis (for agent planning programs and advanced forms of composition).

The main contribution of this paper is to show that diverse agent behavior synthesis problems, including all the ones mentioned above, can be treated *uniformly* by relying on two foundational ingredients:

- making explicit the assumption of two different roles in reasoning/synthesis: a role that works *against* the solution, the so-called "environment;" and a role that works *towards* the solution, the so-called "controller;"
- exploiting full observability even in the presence of nondeterminism (i.e., partial controllability) to do reasoning and synthesis based on model checking (Clarke, Grumberg, and Peled 1999).

On the basis of these two points, we introduce *two-player game structures*, which are akin to the widely adopted transition systems/Kripke structures in model checking, except that they distinguish between the actions/moves of two antagonistic players: the *environment* and the *controller*. Such a distinction has its roots in discrete control theory (Ramadge and Wonham 1989), and has lately been adopted in Verification for dealing with synthesis from temporal specifications (Piterman, Pnueli, and Sa'ar 2006). Also, this distinction has been explicitly made in some AI work on reasoning about actions and on agents, e.g., (Lespérance, De Giacomo, and Ozgovde 2008; Genesereth and Nilsson 1987; Wooldridge 2009). Formally, such a distinction allows for separately quantifying over both environment's and controller's moves. To fully exploit this possibility, we introduce a variant of (modal) μ-calculus (Emerson 1996)—possibly the most powerful formalism for temporal specification (Clarke, Grumberg, and Peled 1999)—that takes into account such a distinction.

We demonstrate then that the resulting framework is indeed a very powerful one. To that end, we show that one can reformulate each of the above synthesis problems, as well as many others, as the task of model checking a (typically simple) μ-calculus formula over suitable two-player game structures. By exploiting the result on μ-calculus model checking, we are able to solve, optimally wrt computational complexity and effectively in practice, through model checking tools, several forms of agent behavior synthesis.

Two-player Game Structures

We start by introducing the notion of *two-player game structure* (2GS, for short), largely inspired by those game structures used in synthesis by model checking in Verification (Piterman, Pnueli, and Sa'ar 2006; de Alfaro, Henzinger, and Majumdar 2001), which in turn are at the base of ATL interpretation structures (Alur, Henzinger, and Kupferman 2002), often used in modeling multi-agent systems (Wooldridge 2009). 2GS's are akin to transition systems used to describe the systems to be checked in Verification (Clarke, Grumberg, and Peled 1999), with a substantial difference, though: while a transition system describes the evolution of a system, a 2GS describes the *joint evolution* of two autonomous systems—the *environment* and the *controller*—running together and interacting at each step, as if engaged in a sort of game.

Formally, a two-player game structure (2GS) is a tuple $G = \langle \mathcal{X}, \mathcal{Y}, start, \rho_e, \rho_c \rangle$, where:

- $\mathcal{X} = \{x_1, \ldots, x_m\}$ and $\mathcal{Y} = \{y_1, \ldots, y_n\}$ are two disjoint finite sets representing the environment and controller variables, respectively. Each variable x_i (resp. y_i) ranges over finite domain X_i (resp. Y_i). Set $\mathcal{X} \cup \mathcal{Y}$ is the set of *game state variables*. A valuation of variables in $\mathcal{X} \cup \mathcal{Y}$ is a total function val assigning to each $x_i \in \mathcal{X}$ (resp. $y_i \in \mathcal{Y}$) a value $val(x_i) \in X_i$ (resp. $val(y_i) \in Y_i$). For convenience, we represent valuations as vectors $\langle \vec{x}, \vec{y} \rangle \in \vec{X} \times \vec{Y}$, where $\vec{X} = X_1 \times \cdots \times X_m$ and $\vec{Y} = Y_1 \times \cdots \times Y_n$. Notice that $\vec{X}$ (resp. $\vec{Y}$) corresponds to the subvaluation of variables in $\mathcal{X}$ (resp. $\mathcal{Y}$). A valuation $\langle \vec{x}, \vec{y} \rangle$ is a game state, where $\vec{x}$ and $\vec{y}$ are the corresponding environment and controller states, respectively.
- $start = \langle \vec{x}_o, \vec{y}_o \rangle$ is the *initial state* of the game.
- $\rho_e \subseteq \vec{X} \times \vec{Y} \times \vec{X}$ is the *environment transition relation*, which relates each game state to its possible successor environment states (or *moves*).
- $\rho_c \subseteq \vec{X} \times \vec{Y} \times \vec{X} \times \vec{Y}$ is the *controller transition relation*, which relates each game state and environment move to the possible controller replies. Notice that formally the projection of ρ_c on $\vec{X} \times \vec{Y} \times \vec{X}$, which does not include the controller response, is trivially ρ_e.

The idea of 2GS is that from a current game state $\langle \vec{x}, \vec{y} \rangle$, the environment *moves* by choosing an $\vec{x}'$ such that $\rho_e(\vec{x}, \vec{y}, \vec{x}')$ holds, and, after this, the controller *replies back* by choosing a $\vec{y}'$ such that $\rho_c(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$ holds. Intuitively, a game structure represents the rules of a game played by these two adversaries, the *environment* and the *controller*. More precisely, the game structure defines the *constraints* each player is subject to when moving (but not the goal of the game).

Given a 2GS G as above, one can express the *winning conditions for the controller* (i.e., its goal) through a goal formula over G. To express such goal formulas, we use a variant of the μ-calculus (Emerson 1996) interpreted over game structures. The key building block is the operator $\odot\Psi$ interpreted as follows

$$\begin{array}{l}\langle \vec{x}, \vec{y} \rangle \models \odot\Psi \text{ iff} \\ \quad \exists \vec{x}'.\rho_e(\vec{x}, \vec{y}, \vec{x}') \land \\ \quad \forall \vec{x}'.\rho_e(\vec{x}, \vec{y}, \vec{x}') \to \exists \vec{y}'.\rho_c(\vec{x}, \vec{y}, \vec{x}', \vec{y}') \text{ s.t. } \langle \vec{x}', \vec{y}' \rangle \models \Psi.\end{array}$$

In English, this operator expresses the following: *for every move $\vec{x}$ of the environment from the game state $\langle \vec{x}, \vec{y} \rangle$, there is a move $\vec{y}'$ of controller such that in the resulting state of the game $\langle \vec{x}', \vec{y}' \rangle$ the property Ψ holds.* With this operator at hand, we develop the whole μ-calculus as follows:

$$\Psi \leftarrow \varphi \mid Z \mid \Psi_1 \land \Psi_2 \mid \Psi_1 \lor \Psi_2 \mid \odot\Psi \mid \mu Z.\Psi \mid \nu Z.\Psi,$$

where φ is an arbitrary boolean expression built from propositions of the form $(x_i = \bar{x}_i)$ and $(y_i = \bar{y}_i)$; Z is a predicate variable; $\odot\Psi$ is as defined above; and μ (resp. ν) is the least (resp. greatest) fixpoint operator from the μ-calculus. We say that a 2GS G satisfies goal formula Ψ, written $G \models \Psi$, if and only if $start \models \Psi$.

We recall that one can express arbitrary temporal/dynamic properties using least and greatest fixpoints constructions (Emerson 1996). For instance, to express that the controller wins the game if a state satisfying a formula φ is reached from the initial state, one can write $G \models \diamond\varphi$, where:

$$\diamond\varphi \doteq \mu Z.\ \varphi \lor \odot Z.$$

Similarly, a greatest fixpoint construction can be used to express the ability of the controller to maintain a property φ, namely, we write $G \models \boxdot\varphi$, where:

$$\boxdot\varphi \doteq \nu Z.\varphi \land \odot Z.$$

Fixpoints can be also nested into each other, for example:

$$\boxdot\diamond\varphi \doteq \nu Z_1.(\mu Z_2.((\varphi \land \odot Z_1) \lor \odot Z_2))$$

expresses that the controller has a strategy to force the game so that it is *always* the case that *eventually* a state where φ holds is reached.[1]

In general, we shall be interested in checking whether the goal formula is satisfied in a game structure, which amounts to *model checking* the game structure. In fact, such a form of model checking is essentially identical to the standard model checking of transition systems (Clarke, Grumberg, and Peled 1999), except for the computation of the preimages, which, in the case of game structures are based on the operator $\odot\Psi$. Hence, one can apply classical results in model checking for μ-calculus (Emerson 1996), thus obtaining a computational characterization of the complexity of checking goal formulas in 2GSs.

Theorem 1. *Checking a goal formula Ψ over a game structure $G = \langle \mathcal{X}, \mathcal{Y}, start, \rho_e, \rho_c \rangle$ can be done in time*

$$O((|G| \cdot |\Psi|)^k),$$

where $|G|$ denotes the number of game states of G plus $|\rho_e| + |\rho_c|$, $|\Psi|$ is the size of formula Ψ (considering propositional formulas as atomic), and k is the number of nested fixpoints sharing the same free variables in Ψ.

[1] See (Emerson 1996) for details about fixpoint nesting. Note that the LTL-style abbreviations used cannot be composed trivially.

Proof. The thesis follows from the results in (Emerson 1996) on fixpoints computations and from the definition of $\odot\Psi$, which, though more sophisticated than in standard μ-calculus, only involves local checks, that is, checks on transitions and states directly connected to the current state. □

We are not merely interested in verifying goal formulas, but, also and more importantly, in *synthesizing* strategies to actually fulfill them. A (controller) <u>*strategy*</u> is a partial function $f : (\vec{X} \times \vec{Y})^+ \times \vec{X} \mapsto \vec{Y}$ such that for every sequence $\lambda = \langle \vec{x}_0, \vec{y}_0\rangle \cdots \langle \vec{x}_n, \vec{y}_n\rangle$ and every $\vec{x}' \in \vec{X}$ such that $\rho_e(\vec{x}_n, \vec{y}_n, \vec{x}')$ holds, it is the case that $\rho_c(\vec{x}_n, \vec{y}_n, \vec{x}', f(\lambda, \vec{x}'))$ applies. We say that a strategy f is <u>*winning*</u> if by resolving the controller existential choice in evaluating the formulas of the form $\odot\Psi$ according to f, the goal formula is satisfied. Notably, model checking algorithms provide a *witness* of the checked property (Clarke, Grumberg, and Peled 1999; Piterman, Pnueli, and Sa'ar 2006), which, in our case, consists of a *labeling* of the game structure produced during the model checking process. From *labelled* game states, one can read how the controller is meant to react to the environment in order to fulfill the formulas that *label* the state itself, and from this, define a strategy to fulfill the goal formula.

Conditional Planning

To better understand how game structures work, we first use them to capture conditional planning with full observability (Rintanen 2004; Ghallab, Nau, and Traverso 2004).

Let $\mathcal{D} = \langle P, A, S_0, \rho\rangle$ be a (nondeterministic) dynamic domain, where: *(i)* $P = \{p_1, \ldots, p_n\}$ is a finite set of *domain propositions*, and a *state* is a subset of 2^P; *(ii)* $A = \{a_1, \ldots, a_r\}$ is the finite set of *domain actions*; *(iii)* $S_0 \in 2^P$ is the *initial state*; *(iv)* $\rho \subseteq 2^P \times A \times 2^P$ is the *domain transition relation*. We freely interchange notations $\langle S, a, S'\rangle \in \rho$ and $S \xrightarrow{a} S'$.

Suppose next that φ is the propositional formula over P expressing the (reachability) goal for $\mathcal{D}$. We then define the game structure $G_\mathcal{D} = \langle \mathcal{X}, \mathcal{Y}, start, \rho_e, \rho_c\rangle$ as follows:

- $\mathcal{X} = P$ and $\mathcal{Y} = \{act\}$, with act ranging over $A \cup \{a_{init}\}$;
- $start = \langle S_0, a_{init}\rangle$;
- $\rho_e = \rho \cup \{\langle S_0, a_{init}, S_0\rangle\}$;
- $\rho_c(S, a, S', a')$ iff for some $\mathcal{S}'' \in 2^P$, $\rho(S', a', S'')$ holds (i.e., the action a' is executable next).

In words, the environment plays the role of the the nondeterministic domain $\mathcal{D}$, while the controller is meant to capture a plan. At each step, the controller chooses an action act, which must be executable in the current state of the domain (fourth point above). Once the controller has selected a certain action, the environment plays its turn by choosing the next state to evolve to (third point above). This move basically involves resolving the nondeterminism of the action act selected by the controller. A special action a_{init} is used as the initial (dummy) controller move, which keeps the environment in $\mathcal{D}$'s initial state S_0.

Finally, we represent the planning goal φ as the goal formula $\Diamond\varphi$ over $G_\mathcal{D}$. Such a formula requires that, no matter how the environment moves (i.e., how domain $\mathcal{D}$ happens to evolve), the controller guarantees reachability of a game state where φ holds (i.e., a domain state satisfying φ).

Theorem 2. *There exists a conditional plan for reaching goal φ in the dynamic domain $\mathcal{D}$ iff $G_\mathcal{D} \models \Diamond\varphi$.*

As discussed above, we can check such a property by standard model checking. What is more, from a witness, we can directly compute a winning strategy f, which corresponds to a conditional plan for goal φ in domain $\mathcal{D}$.

As for computational complexity, by applying Theorem 1 and considering that the goal formula $\Diamond\varphi \doteq \mu Z. \varphi \vee \odot Z$ has no nested fixpoints, we get that such a technique computes conditional plans in $O(|G|) = O(|2^P| \cdot |A| + |\rho|)$. That is, its complexity is polynomial in the size of the domain and exponential in its representation, matching the problem complexity, which is EXPTIME-complete (Rintanen 2004).

It is worth noting that, although we have focused on standard conditional planning, similar reductions can be done for other forms of planning with full observability. In particular, all planning accounts tackled via model checking of CTL, including strong cyclic planning (Cimatti et al. 2003), can be directly recast as finding a winning strategy for a goal formula without nesting of fixpoints in a 2GS as above. Also the propositional variant of frameworks where both the agent and the domain behaviors are modeled as a Golog-like program (Lespérance, De Giacomo, and Ozgovde 2008) can be easily captured, see (Fritz, Baier, and McIlraith 2008).

Agent Planning Programs

Next, we consider an advanced form of planning that requires loops and possibly non-terminating plans (De Giacomo, Patrizi, and Sardina 2010). Given a dynamic domain $\mathcal{D} = \langle P, A, S_0, \rho\rangle$ as above, an *agent planning program* for $\mathcal{D}$ is a tuple $\mathcal{T} = \langle T, \mathcal{G}, t_0, \delta\rangle$, where:

- $T = \{t_0, \ldots, t_q\}$ is the finite set of *program states*;
- $\mathcal{G}$ is a finite set of goals of the form *"achieve ϕ while maintaining ψ,"* denoted by pairs $g = \langle \psi, \phi\rangle$, where ψ and ϕ are propositional formulae over P;
- $t_0 \in T$ is the *program initial state*;
- $\delta \subseteq T \times \mathcal{G} \times T$ is the *program transition relation*. We freely interchange notations $\langle t, g, t'\rangle \in \delta$ and $t \xrightarrow{g} t'$ in $\mathcal{T}$.

Intuitively, an agent planning program provides a structured representation of the different goals that an agent may need to satisfy—it encodes the agent's space of deliberation.

Agent planning programs are *realized* as follows (for the formal definition see (De Giacomo, Patrizi, and Sardina 2010)): at any point in time, the planning program is in a state t and the dynamic domain in a state S; the agent requests a transition $t \xrightarrow{\langle \psi, \phi\rangle} t'$ in $\mathcal{T}$ (e.g., $t \xrightarrow{\langle \neg Driving, At(pub)\rangle} t'$, i.e., be at the pub and never be driving); then, a plan π from S that leads the dynamic domain to a state satisfying ϕ, while only traversing states where ψ holds, is synthesized—notice that such a plan *must also guarantee the continuation of the program*; upon plan completion, the agent planning program moves to t' and requests a new transition, and so on. Notice also that, at any point in time, all possible choices available in the agent planning program must be guaranteed by the system, since the actual request that will be made is not known in advance—the whole agent's space of deliberation ought to be accounted for.

For example, imagine a planning program for specifying the routine habits of a young researcher. Initially, the researcher is at home, from where she may choose to go to work or to a friend's house. After work, she may want to go back home or to a pub, and so on. So, in order to fulfill the initial possible request to go to work, plans involving driving or taking a bus to the lab are calculated and one of them is chosen. Further plans are then calculated to fulfill the request of going to the pub, and so on. Now suppose that the researcher would like to always leave the car home when going to the pub. Then, plans involving driving to work are not appropriate, not because they fail to satisfy the goal of being at work, but because they would prevent the fulfillment of further goals (namely, going to the pub with the car left at home). Thus, plans must not only fulfill their goals, but must also make fulfilling later requests encoded in the structure of the program, possibly within loops, possible.

Let us now show how the problem of finding a realization of a planning program $\mathcal{T}$ can be reduced to building a winning strategy for a goal over a 2GS. Precisely, from $\mathcal{D}$ and $\mathcal{T}$, we shall build a 2GS G and a goal formula φ, such that $G \models \varphi$ iff $\mathcal{T}$ is realizable in $\mathcal{D}$. Also, from a witness of the check $G \models \varphi$, we shall extract a winning strategy f that corresponds to a realization of $\mathcal{T}$.

The construction of $G = \langle \mathcal{X}, \mathcal{Y}, start, \rho_e, \rho_c \rangle$ is as follows. The set of *environment variables* is $\mathcal{X} = \{P, tr_{\mathcal{T}}\}$, where $tr_{\mathcal{T}}$ ranges over $\delta \cup \{tr_{init}\}$, that is, the set of $\mathcal{T}$'s transitions (plus tr_{init}, for technical convenience only). The set of *controller variables* is $\mathcal{Y} = \{act, last\}$, where variable act ranges over $A \cup \{a_{init}\}$ (again, a_{init} is introduced for convenience), and $last$ is a propositional variable. Variable act stands for the action to be executed next, while $last$ marks the end of current plan's execution.

As for the transitions of the game structure, we have:

- $start = \langle S_0, tr_{init}, a_{init}, \bot \rangle$;
- the *environment transition relation* ρ_e is such that $\rho_e(\langle S, tr \rangle, \langle a, l \rangle, \langle S', tr' \rangle)$ iff:
  - tr is a transition $t \xrightarrow{\langle \psi, \phi \rangle} t' \in \delta$;
  - $S \models \psi$ and $S \xrightarrow{a} S' \in \rho$, i.e., S both fulfills the maintenance goal required by tr and enables a's execution;
  - if $l = \bot$, then $tr'_{\mathcal{T}} = tr_{\mathcal{T}}$, i.e., if a is not the last action and thus the transition realization is not completed, then the planning program keeps requesting the same (current) transition $tr_{\mathcal{T}}$;
  - if $l = \top$, then $S' \models \phi$ and $tr'_{\mathcal{T}}$ is any $t' \xrightarrow{\langle \psi', \phi' \rangle} t'' \in \delta$, i.e., if a is indeed the last action for tr realization, then goal ϕ is indeed achieved and a new $\mathcal{T}$ transition is chosen according to δ.

  Furthermore, for each initial transition $t_0 \xrightarrow{\psi, \phi} t' \in \delta$ we have that $\rho_e(\langle S_0, tr_{init} \rangle, \langle a_{init}, \bot \rangle, \langle S_0, t_0 \xrightarrow{\psi, \phi} t' \rangle)$, capturing all possible initial moves for the environment;
- the *controller transition relation* ρ_s is such that $\rho_s(\langle S, tr \rangle, \langle a, l \rangle, \langle S', tr' \rangle, \langle a', l' \rangle)$ iff there exists a transition $S' \xrightarrow{a'} S''$ in $\mathcal{D}$ for some state S'' (i.e., action a' is executable in current domain state S'). Observe no constraint is required on controller variable l'.

Intuitively, G represents the synchronous evolution of domain $\mathcal{D}$ and program $\mathcal{T}$, which together form the environment, operated by the controller. The environment includes all $\mathcal{D}$'s and all $\mathcal{T}$'s transitions, both chosen nondeterministically from the controller's viewpoint. The controller, on the other hand, represents the possible decisions made at each step, namely, the action to be performed next and the notification for plan completion, in order to fulfill the requests issued by the environment. Observe that $\mathcal{T}$ can issue a new transition request *only after* its current request is deemed fulfilled by the controller (i.e., $last$ holds).

As for the goal formula, we have $\varphi = \boxdot\Diamond last$, which requires that it is *always* the case that *eventually* the controller does reach the end of the (current) plan, thus fulfilling the (current) request issued by program $\mathcal{T}$.

Theorem 3. *Let $\mathcal{T}$ be a planning program over a dynamic domain $\mathcal{D}$, and G be the corresponding 2GS built as above. Then, there exists a realization of $\mathcal{T}$ in $\mathcal{D}$ iff $G \models \boxdot\Diamond last$.*

Since $\varphi = \boxdot\Diamond last$ has two nested fixpoints, we get that such a technique computes a realization of $\mathcal{T}$ in $\mathcal{D}$ in $O(|G|) = O((|2^P| \cdot (|\delta| + |\rho|))^2)$, i.e., in polynomial time in the size of $\mathcal{D}$ and $\mathcal{T}$ and in exponential time in their representations. This upperbound is indeed *tight*, as conditional planning, which is a special case of agent planning program realization, is already EXPTIME-complete.

Multitarget Composition

We now turn to composition in the style of (Sardina, De Giacomo, and Patrizi 2008; Su 2008). In particular, we focus on a sophisticated form of composition originally proposed in (Sardina and De Giacomo 2008). Namely, we want to realize a *collection* of independent (target) *virtual agents* that are meant to act autonomously and asynchronously on a shared environment. For example, a surveillance agent and a cleaning agent, among others, may all operate in the same smart house environment. Such agents have no fixed embodiment, but must be concretely realized by a set of available *devices* (e.g., a vacuum cleaner, microwave, or video camera) that are allowed to "join" and "leave" the various agents, dynamically depending on agent's requests. Each agent's embodiment is dynamically transformed while in execution.

Both agents and devices (i.e., their logics) are described by *transition systems* of the form $TS = \langle A, S, s_0, \delta \rangle$, where: *(i)* A is a finite set of actions; *(ii)* S is the finite set of possible states; *(iii)* $s_0 \in S$ is the initial state of TS; and *(iv)* $\delta \subseteq S \times A \times S$ is the transition relation, with $\langle s, a, s' \rangle \in \delta$ or $s \xrightarrow{a} s'$ denoting that TS may evolve to state s' when action a is executed in state s. We assume, wlog, that each state may evolve to at least one next state. Intuitively, the transition systems for the agents encode the space of deliberation of these agents, whereas the transition systems for the devices encode the *capabilities* of such artifacts.

So, we consider a tuple of *available devices* $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, where $\mathcal{B}_i = \langle A^{\mathcal{B}}, B_i, b_{0i}, \delta_i \rangle$, and a tuple of *virtual agents* $\langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle$, where $\mathcal{T}_i = \langle A_{\mathcal{T}}, T_i, t_{0i}, \tau_i \rangle$. All $\mathcal{T}_i$'s are *deterministic*, i.e., fully controllable—in the sense that there is no uncertainty on the resulting state obtained by executing an action—whereas each $\mathcal{B}_i$ may be *nondeterministic*, i.e., only partially controllable (though their current state is fully observable). The composition problem we are concerned with involves guaranteeing the concurrent execution of the virtual agents *as if each of them were acting in isolation*, though, in reality, they are all collectively realized by the same set of (actual) available devices. A solution to this problem amounts to synthesizing a *controller* that intelligently delegates the actions requested by virtual agents to concrete devices. The original problem, which was shown to be EXPTIME-complete, allowed the controller to assign agents' requested actions to devices *without* simultaneously progressing those agents whose actions have been done. In other words, the controller may instruct the execution of an action in a certain device without stating to which particular agent it corresponds. Obviously, after m steps (i.e., the number of agents), no more requests are pending so some agent is allowed to issue a new request (Sardina and De Giacomo 2008). 2GSs can be used to encode and solve this problem within the same complexity bound as the original solution.

Here we detail an interesting variant of this problem, in which the served agent is progressed, simultaneously, when an action is executed by a device. Differently from the original problem, such a variant requires to actually identify which devices are "embodying" each agent *at every point in time*. Also, it shows how easy it is to tackle composition variants using 2GSs.

Specifically, from tuples $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ and $\langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle$, we build a 2GS $G = \langle \mathcal{X}, \mathcal{Y}, start, \rho_e, \rho_c \rangle$ as follows. The *environment variables* are $\mathcal{X} = \{s_1^{\mathcal{B}}, \ldots, s_n^{\mathcal{B}}, s_1^{\mathcal{T}}, \ldots, s_m^{\mathcal{T}}, r_1^{\mathcal{T}}, \ldots, r_m^{\mathcal{T}}\}$, where variables $s_i^{\mathcal{B}}$, $s_i^{\mathcal{T}}$ and $r_i^{\mathcal{T}}$ range over B_i, T_i, and $A_{\mathcal{T}}$, respectively. Each $s_i^{\mathcal{B}}$ corresponds to the current state of $\mathcal{B}_i$, $s_i^{\mathcal{T}}$ to the current state of $\mathcal{T}_i$, and $r_i^{\mathcal{T}}$ to the last request issued by $\mathcal{T}_i$. The *controller variables*, on the other hand, are $\mathcal{Y} = \{dev, full\}$, where dev ranges over $\{0, \ldots, n\}$ and $full$ over $\{1, \ldots, m\}$. Variable dev stores the index of the available device selected to execute the action (0 being a dummy value denoting the game's initial state). Variable $full$ stores the index of the virtual agent whose request is to be fulfilled. As before, we denote assignments to $\mathcal{X}$-variables as $\vec{x} \in \vec{X} = B_1 \times \cdots \times B_n \times T_1 \times \cdots \times T_m \times (A_{\mathcal{T}})^m$, and assignments to $\mathcal{Y}$-variables as $\vec{y} \in \vec{Y} = \{0, \ldots, n\} \times \{1, \ldots, m\}$.

The *initial state* is $start = \langle \vec{x}_0, \vec{y}_0 \rangle$, with $\vec{x}_0 = \langle b_{01}, \ldots, b_{0n}, t_{01}, \ldots, t_{0m}, a, \ldots, a \rangle$, for a arbitrarily chosen in $A_{\mathcal{T}}$, and $\vec{y}_0 = \langle 0, 1 \rangle$, i.e., the only state where $dev = 0$.

The *environment transition relation* $\rho_e \subseteq \vec{X} \times \vec{Y} \times \vec{X}$ is such that, for $\langle \vec{x}, \vec{y} \rangle \neq start$, $\langle \vec{x}, \vec{y}, \vec{x}' \rangle \in \rho_e$ iff for $\vec{x} = \langle b_1, \ldots, b_n, t_1, \ldots, t_m, r_1, \ldots, r_m \rangle$, $\vec{y} = \langle k, f \rangle$, and $\vec{x}' = \langle b_1', \ldots, b_n', t_1', \ldots, t_m', r_1', \ldots, r_m' \rangle$ we have that:

- there exists a transition $\langle b_k, r_f, b_k' \rangle$ in δ_k, i.e., the selected device actually executes the action requested by the agent to be fulfilled (i.e., agent $\mathcal{T}_f$); while for each $i \neq k$, $b_i' = b_i$ applies, that is, non-selected devices remain still;
- the f-th agent moves (deterministically) according to transition $\langle t_f, r_f, t_f' \rangle \in \tau_f$, and from t_f' there exists a further transition $\langle t_f', r_f', t_f'' \rangle \in \tau_f$, for some t_f'', i.e., the virtual agent whose request is fulfilled moves according to its transition function and issues a new (legal) request; while for each $i \neq f$, $t_i' = t_i$ and $r_i' = r_i$ apply, i.e., all virtual agents whose request are not yet fulfilled remain still.

In addition, from $start = \langle \vec{x}_0, \vec{y}_0 \rangle$, we have that $\langle \vec{x}_0, \vec{y}_0, \vec{x} \rangle \in \rho_e$ with $\vec{x} = \langle b_{01}, \ldots, b_{0n}, t_{01}, \ldots, t_{0m}, r_1, \ldots, r_m \rangle$ for r_i such that $\langle t_{0i}, r_i, t \rangle \in \tau_i$, for some $t \in T_i$, i.e., from the initial state each virtual agent issues an initial legal request.

Finally, the *controller transition relation* $\rho_c \subseteq \vec{X} \times \vec{Y} \times \vec{X} \times \vec{Y}$ is such that $\langle \vec{x}, \vec{y}, \vec{x}', \vec{y}' \rangle \in \rho_c$ iff $\langle \vec{x}, \vec{y}, \vec{x}' \rangle \in \rho_e$ and for $\vec{x}' = \langle b_1', \ldots, b_n', t_1', \ldots, t_m', r_1', \ldots, r_m' \rangle$ and $\vec{y}' = \langle k', f' \rangle$, there exists a transition $\langle b_{k'}', r_{f'}, b_{k'}'' \rangle \in \delta_{k'}$, for some $b_{k'}''$, i.e., the action requested by agent $\mathcal{T}_{f'}$ is actually executable by device $\mathcal{B}_{k'}$. That completes the definition of 2GS G.

Now, on such 2GS, we define the *goal formula* φ simply as $\varphi = \bigwedge_{f=1}^{m} \Box\Diamond(full = f)$, thus requiring that each time a virtual agent request is issued, it is eventually fulfilled.

Theorem 4. *There exists a composition for the virtual agents $\langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle$ by the available devices $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ (according to the assumptions considered here) iff $G \models \varphi$, for G and φ constructed as above.*

From a witness of $G \models \varphi$, one can extract an actual controller able to do the composition. As for complexity, since the goal formula contains two nested fixpoints, the time needed to check the existence of a composition (and to actually compute one) is $O(|G|^2) = O((|\vec{X}| \cdot |\vec{Y}| + |\rho_e| + |\rho_c|)^2)$, where $|\vec{X}| = O(|B_{max}|^n \cdot |T_{max}|^m \cdot |A_{\mathcal{T}}|^m)$; $|\vec{Y}| = n \cdot m + 1$; $|\rho_e| = O(|\vec{X}| \cdot |\vec{Y}| \cdot |B_{max}| \cdot |A_{\mathcal{T}}|)$; and $|\rho_c| = O(|\rho_e| \cdot m \cdot n)$, with B_{max} and T_{max} being the maximum number of states among devices and virtual agents, respectively. Such a bound, under the natural assumption that the number of actions is at most polynomial in the number of states of virtual agents, reduces to $O(u^{m+n})$, where $u = \max\{B_{max}, T_{max}\}$, which is essentially the complexity of the problem (Sardina and De Giacomo 2008).

Implementation

The approach presented here is ready implementable with model checking tools. However, the need of quantifying separately on environment and controller moves, requires the use of μ-calculus, and not simply CTL or LTL (Clarke, Grumberg, and Peled 1999) for which model checking tools are much more mature. As an alternative, if the goal formula does not require nested fixpoints, we can adopt ATL model checkers (Lomuscio, Qu, and Raimondi 2009). ATL interpretation structures are indeed quite related to 2GSs, and we can split the set of ATL agents into two coalitions representing (possibly in a modular way) the environment and the controller—e.g., (De Giacomo and Felli 2010) encodes single target composition in ATL.

Another alternative is to use (procedure *synth* in) the TLV system (Piterman, Pnueli, and Sa'ar 2006), which is a framework for the development of BDD-based procedures for formal synthesis/verification. TLV takes a generic 2GS G expressed in the concrete specification language SMV (Clarke, Grumberg, and Peled 1999), and uses special SMV *modules* `Env` and `Sys` to distinguish the environment and the controller. Also, it takes an LTL formula φ of "GR1" form: $\varphi = \bigwedge_{i=1}^{n} \Box\Diamond p_i \longrightarrow \bigwedge_{j=1}^{m} \Box\Diamond q_j$, with p_i and q_j propositions over G. Then, it synthesizes, if any, a controller strategy to satisfy φ, by transforming φ into a μ-calculus formula

φ_μ over G, which quantifies separately on the environment and the controller. Because of the fixed structure of φ, the resulting φ_μ has 3 nested fixpoints.

It turns out that TLV can be used to solve both agent planning programs and multi target composition problems; indeed, the goal formulas φ_g of such problems, are special cases of the φ_μ that TLV uses. Specifically, for the multi target problem, we can proceed as follows. To encode a 2GS in SMV, one essentially needs to describe both controller's and environment's behaviors. They are actually encoded as SMV modules—each representing a transition system, with its initial state, state variables and transition rules, specified in SMV sections `VAR`, `INIT` and `TRANS`, respectively—possibly communicating via input/output parameters. We do so by defining: *(i)* one SMV module for each available device $\mathcal{B}_i$ and one for each virtual agent $\mathcal{T}_i$, wrapping them in an `Env` module to form the 2GS environment; and *(ii)* a further `Sys` module, representing the controller, which communicates with all available devices, to delegate them action executions, and with all virtual agents, to inform them of request fulfillment. The controller is modeled so as to guarantee that action delegations are always *safe*, i.e, the device instructed to execute the action can actually do it in its current state. As for goal formula, this corresponds to the μ-calculus formula obtained in TLV by processing the GR1 LTL formula $\Box\Diamond\texttt{true} \longrightarrow \bigwedge_{i=1}^{n} \Box\Diamond\texttt{fulfilled}_i$ (n number of virtual agents), which technically is encoded as a list $\texttt{fulfilled}_1 \ldots \texttt{fulfilled}_n$ in the `JUSTICE` section of controller's module. We used TLV to actually solve several variants of the example in (Sardina and De Giacomo 2008), using the version of the multitarget composition introduced above. The tool was able to find the solution for those simple problems in around 10 seconds (on a Core2Duo 2.4Ghz laptop with 2GB of RAM).

Conclusions

We have shown that model checking 2GS's is a very powerful, yet fully automatically manageable, technique for synthesis in a variety of AI contexts under full observability. By observing that most current model checking algorithms are based on the so-called "global" model checking, we conclude that these can be seen as a generalization of planning by backward reasoning when applied to 2GSs. Model checking algorithms based on forward reasoning are also available, the so-called "local" model checking techniques (Stirling and Walker 1991), though they are currently considered less effective in Verification (Clarke, Grumberg, and Peled 1999). On the contrary, within AI, planning by forward reasoning is currently deemed the most effective, mainly due to use of heuristics that seriously reduce the search space. An interesting direction for future research, thus, would be to apply local model checking to 2GSs, while taking advantage of such heuristics developed by the automated planning community.

Acknowledgments We dedicate this paper to the memory of Amir Pnueli, who first showed us the power of game structures for tackling synthesis problems.

We acknowledge the support of EU Project FP7-ICT ACSI (257593), EU Project FP7-ICT SM4All (224332), and of Agent Oriented Software and the Australian Research Council (under grant DP1094627).

References

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *Journal of the ACM* (49):672–713.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence Journal* 1–2(147).

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model checking*. Cambridge, MA, USA: The MIT Press.

de Alfaro, L.; Henzinger, T. A.; and Majumdar, R. 2001. From verification to control: Dynamic programs for omega-regular objectives. In *Proc. of LICS'01*, 279–290.

De Giacomo, G., and Felli, P. 2010. Agent composition synthesis based on ATL. In *Proc. of AAMAS'10*.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2010. Agent programming via planning programs. In *Proc. of AAMAS'10*.

Emerson, E. A. 1996. Model checking and the mu-calculus. In *Descriptive Complexity and Finite Models*, 185–214.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond. In *Proc. of KR'08*, 600–610.

Genesereth, M. R., and Nilsson, N. J. 1987. *Logical foundations of artificial intelligence*. Morgan Kaufmann.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Kerjean, S.; Kabanza, F.; St.-Denis, R.; and Thiébaux, S. 2006. Analyzing LTL model checking techniques for plan synthesis and controller synthesis (work in progress). *Electronic Notes in Theoretical Computer Science (ENTCS)* 149(2):91–104.

Lespérance, Y.; De Giacomo, G.; and Ozgovde, A. N. 2008. A model of contingent planning for agent programming languages. In *Proc. of AAMAS'08*, 477–484.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. of CAV'09*, 682–688.

Lundh, R.; Karlsson, L.; and Saffiotti, A. 2008. Automatic configuration of multi-robot systems: Planning for multiple steps. In *Proc. of ECAI'08*, 616–620.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of reactive(1) designs. In *Proc. of VMCAI'06*, 364–380. Springer.

Ramadge, P. J., and Wonham, W. M. 1989. The control of discrete event systems. *IEEE Trans. on Control Theory* 77(1):81–98.

Rintanen, J. 2004. Complexity of Planning with Partial Observability. In *Proc. of ICAPS'04*, 345–354.

Sardina, S., and De Giacomo, G. 2008. Realizing multiple autonomous agents through scheduling of shared devices. In *Proc. of ICAPS'08*, 304–312.

Sardina, S.; De Giacomo, G.; and Patrizi, F. 2008. Behavior Composition in the Presence of Failure. In *Proc. of KR'08*, 640–650.

Stirling, C., and Walker, D. 1991. Local model checking in the modal mu-calculus. *Theor. Comput. Sci.* 89(1):161–177.

Stroeder, T., and Pagnucco, M. 2009. Realising deterministic behaviour from multiple non-deterministic behaviours. In *Proc. of IJCAI'09*, 936–941.

Su, J., ed. 2008. *Semantic Web Services: Composition and Analysis. IEEE Data Eng. Bull.*, volume 31. IEEE Comp. Society.

Wooldridge, M. 2009. *Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition.