An Experimental Toolchain for Strategy Synthesis with Spatial Properties

Davide Basile ✉, Maurice H. ter Beek, and Vincenzo Ciancia

Formal Methods and Tools Lab
ISTI–CNR, Pisa, Italy
{basile,terbeek,ciancia}@isti.cnr.it

Abstract. We investigate the application of strategy synthesis to enforce spatial properties. The Contract Automata Library (`CATLib`) performs both composition and strategy synthesis of games modelled in a dialect of finite state automata. The Voxel-based Logical Analyser (`VoxLogicA`) is a spatial model checker that allows the verification of properties expressed using the Spatial Logic of Closure Spaces on pixels of digital images. In this paper, we explore the integration of these two tools. We provide a basic example of strategy synthesis on automata encoding motion of agents in spaces represented by images. The strategy is synthesised with `CATLib`, whilst the properties to enforce are defined by means of spatial model checking of the images with `VoxLogicA`.

1 Introduction

Research on strategy synthesis in games is currently a hot topic, with established relations with supervisory control [3, 60], reactive systems synthesis [41], parity games [57] (with recent complexity breakthroughs [27]), automated behaviour composition [44], automated planning [28] and service coordination [13]. Several academic tools have been developed [7, 31–33, 39, 50, 52, 59] and applied to disparate domains, including land transport [12], maritime transport [62], medical systems [53], autonomous agents path planning [46], in which problems are modelled as games and solved using tailored strategy synthesis algorithms.

In an automata-based setting, a strategy is a prescription of the behaviour (transitions) of a particular player for all possible situations (states) that leads that player to a specific goal (final state). Typically, there are other players or an environment with different, often competing goals to account for, and the set of transitions may be partitioned into controllable (by the particular player) and uncontrollable transitions. Strategy synthesis is concerned with the automatic computation of a (safe, optimal) strategy (controller) in such a game-based automata setting.

Another hot topic concerns recent advancements in spatial model checking, which have led to relevant results such as the fully automated segmentation of regions of interest in medical images by brief, unambiguous specifications in spatial logic. The topological approach to spatial model checking of [34] is based on the Spatial Logic of Closure Spaces (SLCS) and provides a fully automated

method to verify properties of points in graphs, digital images, and more recently 3D meshes and geometric structures [25, 56]. Spatial properties of points are related to topological aspects such as being *near* to points satisfying a given property, or being able to *reach* a point satisfying a certain property, passing only through points obeying to specific constraints.

The tool `VoxLogicA` [24,36] (of which the third author is the lead developer) has been designed from scratch for image analysis. Logical operators can be freely mixed with a few imaging operators, related to colour thresholds, texture analysis, or normalisation. The tool is quite fast, due to various factors: most primitives are implemented using the state-of-the-art imaging library SimpleITK[1]; expressions are never recomputed (reduction of the syntax tree to a directed acyclic graph is used as a form of *memoisation*); operations are implicitly parallelised on multi-core CPUs. Ongoing work (cf., e.g., [26]) is devoted to a GPU-based implementation which enables a speedup of 1-2 orders of magnitude.

Returning to the topic of strategy synthesis, the tool `CATLib` [6, 7, 16] is a library (developed by the first author) for performing compositions of contract automata [15] (a dialect of finite-state automata) and synthesising either their supervisory control, their orchestration, or their choreography [13], using novel notions of controllability [9]. Scalability features offered by `CATLib` include a bounded on-the-fly state-space generation optimised with pruning of redundant transitions and parallel streams computations. The software is open source [7], it has been developed using principles of model-based software engineering [6] and it has been extensively validated using various testing and analysis tools to increase the confidence on the reliability of the library.

Contribution. In this paper, we propose a new approach to combine strategy synthesis and spatial model checking. We proceed in a bottom-up fashion by presenting a toolchain based on established off-the-shelf tool-supported theories. We explore the combination of `CATLib` and `VoxLogicA`, to pair the composition and synthesis functionalities of `CATLib` with the spatial model checking functionality of `VoxLogicA`. We provide a proof-of-concept example of strategy synthesis on automata encoding motion of agents in spaces represented by images[2]. The main insight is to encode an image as an automaton, whose states are the pixels of the image. These states are then interpreted as positions of an agent, and transitions to adjacent pixels represent motions of the agent. A composition of automata is thus a multi-agent system, in which each state of the composition is a snapshot of the current position of the agents in the map. A game can thus be played by a set of agents against other opponent agents, where successful states and failure states can be identified using spatial model checking of the images. The strategy is synthesised with `CATLib`, while the properties to enforce are defined by means of spatial model checking of the images with `VoxLogicA`. The

[1] Cf. https://simpleitk.org/.

[2] In the `VoxLogicA` approach, images are seen as a special kind of graphs, where vertices are pixels, and edges represent proximity. Actually, the `VoxLogicA` family of tools can also operate on arbitrary directed graphs. Adapting the present work to the more general setting is left for future work.

developed example is open-source and reproducible at [14]. The benefits include showing the practical applicability of these two tools, providing an original approach to strategy synthesis and spatial model checking, bridging theories and tools developed in different research areas and openings to future research goals.

Related Work. Practical application of spatial logics, including model checking, has been ongoing during the last decade. For instance, the research line originating in [48] merges spatial model checking with signal analysis. In the domain of cyber-physical systems, the approach of [65] demonstrates applications of SLCS in a spatio-temporal domain with linear time, using bigraphical models. An abstract categorical definition of SLCS has been given in [30]. The spatial model checking approach of SLCS and `VoxLogicA` has been demonstrated in case studies ranging from smart transportation [37] and bike sharing [35,38], to brain tumour segmentation [4,24], labelling of white and grey matter [23], and contouring of nevi [22].

Synthesising strategies (or plans/control) for the motion of agents is a widely researched problem [2,42,46,47,55,63]. Spatial logics have been applied to this problem to investigate the synthesis of strategies from properties of spatially distributed systems specified with spatial logics [1,49,54]. Recently, the application domain of smart cities has been explored in [58], and the aforementioned signal-based approach has been enhanced for a hybrid approach to multi-agent control synthesis, by exploiting neural network and spatial-logical specifications in the Spatio-Temporal Reach and Escape Logic (STREL) formalism.

Differently from the above literature, we set out to integrate previously developed off-the-shelf algorithms and tools, with the aim of showing their applicability. Contract automata and their toolkit were introduced to synthesise ochestrations and choreographies of compositions of service contracts exchanging offers and requests [7,9,13,15]. The interpretation of an image as an (agent) contract automaton enables to connect contract automata and `CATLib` with spatial model checking and `VoxLogicA`, showing the flexibility of both approaches.

Structure of the Paper. We start with the background on `CATLib` and `VoxLogicA` in Section 2. The toolchain is described in Section 3, whilst the experiments are reported in Section 4. Conclusions and future work are mentioned in Section 5.

2 Background

We provide some background on the formalisms and tools used in this paper.

2.1 `CATLib`, Automata Composition, and Strategy Synthesis

We first formally introduce contract automata and their synthesis operation. A Contract Automaton (CA) represents either a single service (in which case it is called a *principal*) or a multi-party composition of services performing actions. The number of principals of a CA is called its *rank*. A CA's states are vectors of

states of principals; its transitions are labelled with vectors of *actions* that are either *requests* (prefixed by ?), *offers* (prefixed by !), or *idle* actions (denoted with a distinguished symbol •). Requests and offers belong to the (pairwise disjoint) sets R and O, respectively. Figures 2 and 3 depict example CA. In a vector of actions there is either a single offer, or a single request, or a single pair of request and offer that match, i.e., the ith element of $\vec{a}$, denoted by $\vec{a}_{(i)}$, is $?a$, its jth element $\vec{a}_{(j)} = !a$, and all other elements are •; such vector of action is called *request*, *offer*, or *match*, respectively. Thus, for brevity, we may call action also a vector of actions. A transition is also called a request, offer, or match according to its action label. The goal of each principal is to reach an accepting (*final*) state such that all its requests (and possibly offers) are matched. In [20], CA were equipped with *modalities*, i.e., *necessary* ($\Box$) and *permitted* ($\Diamond$) transitions, respectively. Permitted transitions are controllable, whilst necessary transitions can be uncontrollable or semi-controllable. Here we ignore semi-controllable transitions and consider necessary transitions to be uncontrollable. The resulting formalism is called *Modal Service Contract Automata* (MSCA).

Definition 1 (MSCA). *Given a finite set of states* $\mathcal{Q} = \{q_1, q_2, \ldots\}$*, an MSCA* $\mathcal{A}$ *of rank* n *is a tuple* $\langle Q, \vec{q}_0, A^r, A^o, T, F\rangle$*, with set of states* $Q = Q_1 \times \ldots \times Q_n \subseteq \mathcal{Q}^n$*, initial state* $\vec{q}_0 \in Q$*, set of requests* $A^r \subseteq \mathsf{R}$*, set of offers* $A^o \subseteq \mathsf{O}$*, set of final states* $F \subseteq Q$*, set of transitions* $T \subseteq Q \times A \times Q$*, where* $A \subseteq (A^r \cup A^o \cup \{\bullet\})^n$*, partitioned into* permitted *transitions* $T^{\Diamond}$ *and* necessary *transitions* $T^{\Box}$*, such that: (i) given* $t = (\vec{q}, \vec{a}, \vec{q}\,') \in T$*,* $\vec{a}$ *is either a request, or an offer, or a match; and (ii)* $\forall i \in 1 \ldots n,\ \vec{a}_{(i)} = \bullet$ *implies* $\vec{q}_{(i)} = \vec{q}\,'_{(i)}$.

Composition of services is rendered through the composition of their MSCA models by means of the *composition operator* $\otimes$, which is a variant of a synchronous product. This operator basically interleaves or matches the transitions of the component MSCA, but, whenever two component MSCA are enabled to execute their respective request/offer action, then the match is forced to happen. Moreover, a match involving a necessary transition of an operand is itself necessary. The rank of the composed MSCA is the sum of the ranks of its operands. The vectors of states and actions of the composed MSCA are built from the vectors of states and actions of the component MSCA, respectively.

In a composition of MSCA, typically various properties are analysed. We are especially interested in *agreement*. The property of agreement requires to match all requests, whilst offers can go unmatched.

CA support the synthesis of the most permissive controller from the theory of supervisory control of discrete event systems [29, 60], where a finite state automaton model of a *supervisory controller* (called a strategy in this paper) is synthesised from given (component) finite state automata that are composed. Supervisory control theory has been applied in a variety of domains [13,21,43,45, 61,64], including healthcare. In this paper, we use the synthesis in the framework of games, whose relation with supervisory control is well known [3].

The synthesised automaton, if successfully generated, is such that it is *non-blocking*, *controllable*, and *maximally permissive*. An automaton is said to be

non-blocking if from each state at least one of the *final states* (distinguished stable states representing completed 'tasks' [60]) can be reached without passing through so-called *forbidden states*, meaning that the system always has the possibility to return to an accepted stable state (e.g., a final state). The algorithm assumes that final states and forbidden states are indicated for each component. The synthesised automaton is said to be *controllable* when only controllable actions are disabled. Indeed, the supervisory controller is not permitted to directly block uncontrollable actions from occurring; the controller is only allowed to disable them by preventing controllable actions from occurring. Finally, the fact that the resulting supervisory controller is said to be *maximally permissive* (or least restrictive) means that as much behaviour of the uncontrolled system as possible is still present in the controlled system without violating neither the requirements, nor controllability, nor the non-blocking condition.

Finally, we recall the specification of the abstract synthesis algorithm of CA from [13]. This algorithm will be used to synthesise a strategy for the spatial game in the next sections. The synthesis of a controller, an orchestration, and a choreography of CA are all different special cases of this abstract synthesis algorithm, formalised in [13] and implemented in `CATLib` [6] using map reduce style parallel operations of Java Streams. This algorithm is a fix-point computation where at each iteration the set of transitions of the automaton is refined (pruning predicate ϕ_p) and a set of forbidden states R is computed (forbidden predicate ϕ_f). The synthesis is parametric on these two predicates, which provide information on when a transition has to be pruned from the synthesised automaton or a state has to be deemed forbidden. We refer to MSCA as the set of (MS)CA, where the set of states is denoted by Q and the set of transitions by T (with $T^{\Box}$ denoting the set of necessary transitions). For an automaton $\mathcal{A}$, the predicate $Dangling(\mathcal{A})$ contains those states that are not reachable from the initial state or that cannot reach any final state.

Definition 2 (abstract synthesis [13]). *Let $\mathcal{A}$ be an MSCA, $\mathcal{K}_0 = \mathcal{A}$, and $R_0 = Dangling(\mathcal{K}_0)$. Given two predicates $\phi_p, \phi_f : T \times MSCA \times Q \to \mathbb{B}$, let the* abstract synthesis function $f_{(\phi_p,\phi_f)} : MSCA \times 2^Q \to MSCA \times 2^Q$ *be defined as:*

$$
\begin{aligned}
&f_{(\phi_p,\phi_f)}(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}\\
&T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} - \{\, t \in T_{\mathcal{K}_{i-1}} \mid \phi_p(t, \mathcal{K}_{i-1}, R_{i-1}) = true \,\}\\
&R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to) = t \in T^{\Box}_{\mathcal{A}},\ \phi_f(t, \mathcal{K}_{i-1}, R_{i-1}) = true \,\} \cup Dangling(\mathcal{K}_i)
\end{aligned}
$$

The abstract controller is defined in Equation 1 below as the least fixed point (cf. [13, Theorem 5.2]) where, if the initial state belongs to $R_s^{(\phi_p,\phi_f)}$, then the controller is empty; otherwise, it is the automaton with the set of transitions $T_{\mathcal{K}_s^{(\phi_p,\phi_f)}}$ and without states in $R_s^{(\phi_p,\phi_f)}$.

$$
(\mathcal{K}_s^{(\phi_p,\phi_f)}, R_s^{(\phi_p,\phi_f)}) = sup(\{\, f^n_{(\phi_p,\phi_f)}(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\}) \tag{1}
$$

`CATLib`. CA and their functionalities are implemented in a software artefact, called Contract Automata Library (`CATLib`), which is under continuous devel-

opment [7]. This software artefact is a by-product of scientific research on behavioural contracts and implements results that have previously been formally specified in several publications (cf., e.g., [9–11, 13, 15–20]). `CATLib` has been designed to be easily extendable to support similar automata-based formalisms. Currently, it also supports synchronous communicating machines [40,51]. `CATLib` and the other CA tools [8] allow programmers to use CA for developing more reliable applications. In this paper, we further showcase the flexibility of `CATLib` by using it to synthesise strategies for mobile agents in spatial games. `CATLib` has been implemented using modern established technologies for building, testing, documenting, and delivering high quality source code. `CATLib` is tested up to 100% coverage of all lines, branches, and the strength of the tests is measured with mutation testing with top score.

2.2 `VoxLogicA`, Spatial Model Checking, and Image Analysis

The *Spatial Logic of Closure Spaces* (SLCS) is a modal logical language equipped with a unary 'nearness' modality and two binary operators: 'reaches' and 'is reached'. The language is interpreted on points of a spatial structure, which is, generally speaking, a *Closure Space* (cf. [34] for details). Graphs, digital images, topological spaces, and simplicial complexes are all instances of closure spaces.

Here we concentrate on the interpretation of SLCS on images. In this case, the two reachability modalities collapse and the nearness modality is a derived operator based on the reachability operator, causing a particularly simple syntax.

Definition 3. *Fix a set AP of atomic propositions. The syntax of SLCS is defined by the following grammar (where $p \in AP$):*

$$\phi ::= p \mid \top \mid \neg\,\phi \mid \phi \wedge \phi \mid \rho\,\phi[\phi]$$

Models of SLCS formulae, for the purpose of this paper, are the pixels of digital images; i.e., each SLCS formula induces a truth value for each point of a given digital image. In order to define the interpretation of formulae, a notion of *path* needs to be established, based on a notion of *neighbourhood* or *connectivity* of pixels. Among infinitely many possible choices, `VoxLogicA` normally uses the so-called '8-neighbourhood', i.e., each pixel is adjacent to 8 other pixels, namely those that share an edge or a vertex with it. Connectivity transforms the set of pixels of an image in a (symmetric) graph. Graph-theoretical paths are then well defined, and used below.

The interpretation of formulae depends upon a valuation of atomic propositions, assigning to each atomic proposition the set of points on which it holds, and assigning a direct interpretation to the symbols $p \in AP$. The meaning of the truth value $\top$ (true), negation ($\neg$), and conjunction ($\wedge$) is the usual one. A pixel x satisfies $\rho\,\phi_1[\phi_2]$ if there is a path rooted in x, reaching a pixel satisfying ϕ_1, such that all intermediate points, except eventually the extremes, must satisfy ϕ_2. We make use of the derived operator $\phi_1 \rightsquigarrow \phi_2$ which is similar to $\rho\,\phi_2[\phi_1]$, but the extremes are also required to satisfy ϕ_1. The *near* derived operator $\mathcal{N}\phi \triangleq \rho\,\phi[\neg\top]$ is true at point x if and only if there is a pixel adjacent to x where ϕ holds.

From now on we use the tool's syntax, which uses `tt`, `&`, `|`, `!`, `~>`, and `N` for *true*, conjunction, disjunction, negation, $\rightsquigarrow$, and $\mathcal{N}$, respectively, permits macro abbreviations of the form `let identifier = expression`, function definition of the form `let identifier(argument1,...,argumentN) = expression`, and other constructs not needed for the scope of this paper. On images, atomic propositions can be expressions predicating over the colour components of the pixels. For instance, in our example specification (cf. Figure 5), to characterise the pixels composing a door as the blue pixels (note that 255 is the maximum value since we are using 8-bit images), given that `img` denotes an image, we use:

```
let r = red(img)
let g = green(img)
let b = blue(img)
...
let door = (r =. 0) & (b =. 255) & (g =. 0)
```

Also, the tool permits global formulae that assign a truth value to models, not just pixels in isolation. These can be based on the `volume(phi)` primitive, that computes the number of pixels satisfying the formula `phi`. For instance, existential and universal quantification are defined as follows:

```
let exists(p) = volume(p) .>. 0
let forall(p) = volume(p) .=. volume(tt)
```

The type system of `VoxLogicA` is very simple, and comprises *numbers*, *Boolean values*, *images of numbers* (single-channel images, sometimes called *grayscale*), *images of Boolean values*, very often called *binary images* or *masks*, and ordinary *multi-channel images*. Operators are strongly typed with no type overloading. Therefore, for instance, the pixel-by-pixel *and* of two Boolean-valued images is a different operator with respect to the conjunction of two Boolean values, and it also differs from the conjunction of the Boolean value of each pixel of an image with a Boolean (scalar) constant. With some exceptions, the naming convention of operators reflects their type, having a dot on the side of the 'scalar' value (Boolean or number) and no dot on the side of the image, so for instance `.&.` is Boolean *and*, whereas `&` is pixel-by-pixel *and* of two images. With respect to Figure 5, for instance, we have that `base` and `img` are multi-channel images, with the operators `red`, `green`, `blue`, extracting number-valued images from them. The definition of `mrRed` (a red area) contains the `=.` operator taking a number-valued image on the left, and a number on the right (hence the dot on the right side). In the definition of the property `forbidden1`, one can find an example of the use of the operator `.|.` which takes as arguments two Boolean values.

3 Tool Methodology

In this section, we discuss the tool methodology used to chain `CATLib` and `VoxLogicA` in order to perform strategy synthesis of spatial properties. Later, in Section 5, we will detail scalable techniques that can be adopted to improve the presented methodology. The diagram in Figure 1 depicts the workflow and the various activities in which the whole process is decomposed.

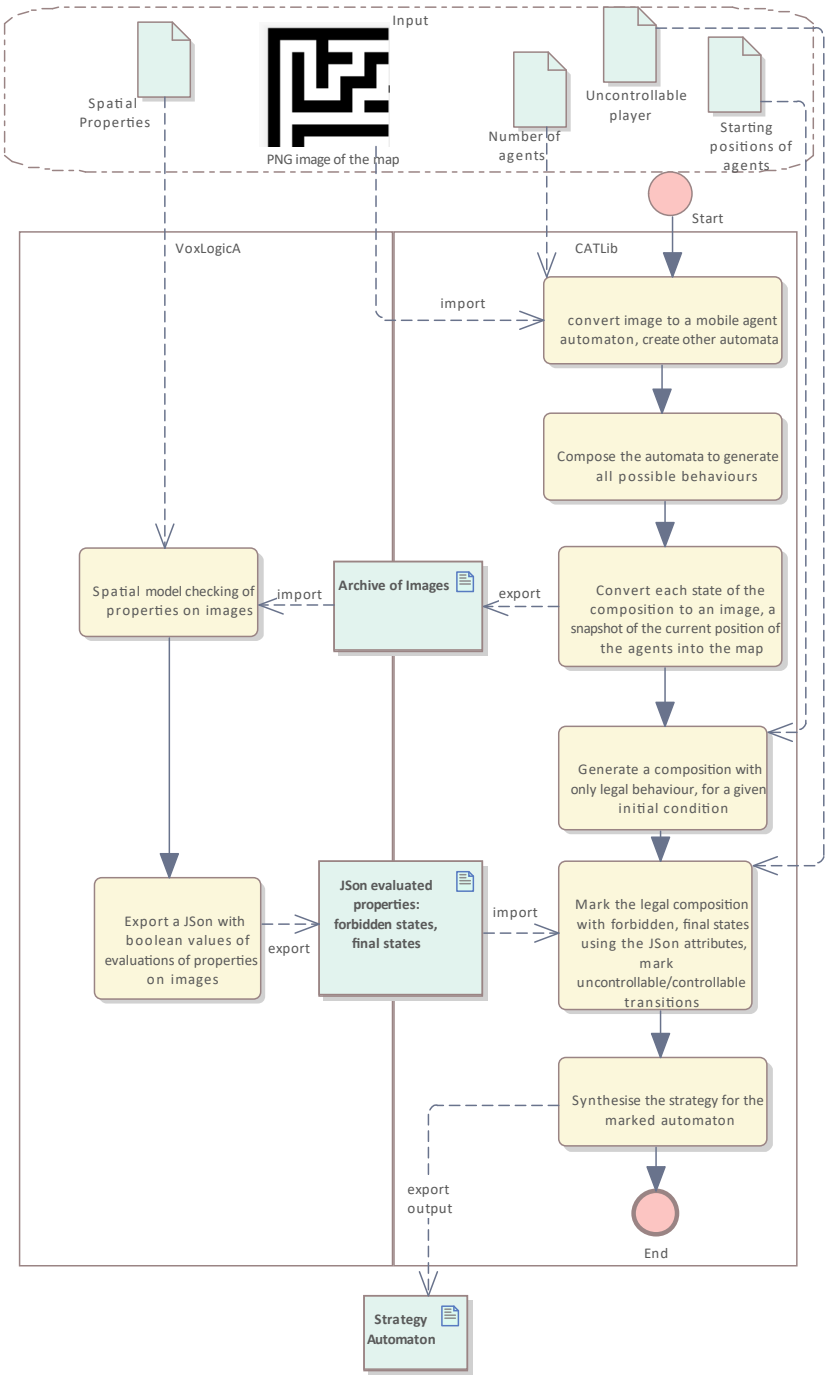

Fig. 1. The workflow showing the integration of the two tools

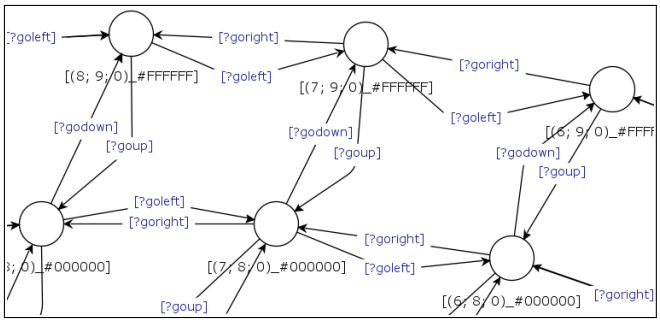

Fig. 2. A zoom-in on a fragment of the agent automaton

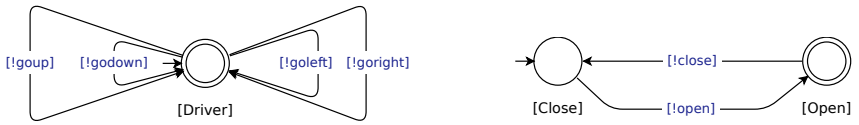

Fig. 3. The driver automaton on the left, the door automaton on the right

The process starts with a `PNG` image, depicting a map or planimetry, for agents to move in. Note that this implies that the state space is discrete, finite, and can be provided by a user with no training on the underlying theories used. Further input concerns the spatial properties that one wants to enforce with the synthesised strategy, modelling the forbidden configurations to avoid and the final configurations to reach, as well as the number of agents in the experiments with their starting position, and an indication of which agents are the controllable players and which are the uncontrollable opponents. The aim of the process is to produce the *maximally permissive* strategy for moving the players against all possible moves of the opponents, such that no forbidden configuration is ever reached and it is always possible to reach a final configuration. In game-theoretical jargon, this is both a safety game and a reachability game [5]. The strategy is maximal, in the sense that it includes all possible behaviour that satisfies the above properties. If the strategy is empty, then there exists no strategy for the players satisfying the given properties. `CATLib` only considers finite traces: infinite looping behaviour where an agent is stalled and is prevented from reaching a reachable final configuration is ruled out.

`CATLib` Activities. In this paper, `CATLib` has been extended to allow the import of `PNG` images, which are internally converted into automata. These automata have pixels as states and transitions connecting adjacent pixels. We interpret these automata as agents, whose position is represented by the current state and transitions are requests to move up, down, left, or right to adjacent pixels/states. If a border is reached, then there will be no request transition in the automaton to move beyond that border. Each state is labelled with both a position, rendered in three coordinates (the third coordinate is currently not used), and the colour of the pixel. Figure 2 depicts a small portion of an agent automaton.

Fig. 4. The state `[(10; 10; 0)_#FFFFFF, (5; 7; 0)_#FFFFFF, Driver, Close]` of the composition of two agents, a driver, and a door. The door is in position `(2; 7)` and is closed. The first agent is depicted red, the second is green, and the door is blue. The attributes of the position of the two agents are both `#FFFFFF`, which is the hexadecimal value for the colour white, i.e., both agents are placed on a white cell of the map.

A driver automaton is used to command an agent to move in a specific direction. It is depicted in Figure 3 (left). The driver can impose some constraints (e.g., never go down). Currently, the driver offers to move in each possible direction. The last automaton that is used models a door, which is initially closed, and which can be opened and closed repeatedly. It is depicted in Figure 3 (right).

The first activity of `CATLib` thus consists of importing and creating the above automata. There can be several instances of agents and doors or different maps according to the parameters of the experiments to perform.

The second activity consists of composing these automata to generate all possible reachable configurations. As stated in Section 2, the composition has unicast synchronisations between offers and requests of agents (called matches), and labels that are only single moves of an agent performing an offer. Agents who perform requests can move only when paired with a corresponding offer. This type of synchronised behaviour is called agreement: all requests must be matched.

In such composition, no restriction is imposed on the agents: they are free to move over walls and doors, and even over other agents. Depending on the initial conditions, some of these configurations could result to be not useful. However, this allows to call the spatial model checker once on all possible configurations. By changing starting conditions in different experiments it is not necessary to invoke again the spatial model checker, since all possible configurations that can be generated have already been analysed offline by the spatial model checker. In the composed automaton, each state is a tuple of states of all agents (included the door and the driver). Each state can be represented as an image, a snapshot of the current configuration. For example, Figure 4 depicts a state rendered as an image. The image is generated by colouring the starting `PNG` image with a red, green, and blue pixel to indicate where, respectively, the first agent, the second agent and the door are located. The door is only coloured when it is closed.

The third activity consists of generating all images for all states of the composition. These images are then passed to `VoxLogicA` (whose activities will be described below) to evaluate for all properties whether or not they are satisfied.

The fourth activity of `CATLib` consists of generating a composition with only legal behaviour. Indeed, to reduce the size of the state space, the composition of `CATLib` allows to avoid generating portions of the state space that are known to violate some property. In case of controllable 'bad' transitions, these will not be generated since they will be pruned by the synthesis. In case of uncontrollable

'bad' transitions, these will be generated (since they cannot be pruned) but their target state will not be visited (the synthesis will try to make these 'bad' states unreachable). Thus, once some agent is rendered as uncontrollable (by changing its transitions to uncontrollable), it cannot be stopped from reaching an illegal configuration. It follows that illegal configurations must be removed before deciding which agents are uncontrollable and which are controllable. In the experiments described in Section 4, the illegal moves are those where an agent is placed on top of a wall (i.e., its state has colour `#000000`), on top of another agent (i.e., in a state of the composition, two agents have the same coordinates), or on top of a closed door (i.e., in a state of the composition, one agent has the same coordinates as the door and the door is closed). Since these are simple invariant properties (it only suffices to check the labels of states), they can be directly checked in `CATLib`. `VoxLogicA` is used to evaluate more complex spatial properties (cf. Figure 5). The aforementioned illegal moves are also specified in `VoxLogicA` under the property `wrong` in Figure 5 below.

In this step it is also decided what are the initial positions of the agents, i.e., the initial state where the state-space generation starts. Depending on the given initial conditions, it is possible that some legal configuration previously generated and passed to `VoxLogicA` will not be generated.

Once the state space for the chosen initial conditions and legal moves is generated, it must be marked with the states that are forbidden and those that are final. This is the fifth activity of `CATLib`. Also, it must be decided which agents are controllable and which are not. This information is provided in part as input parameters of the experiments and in part with a `JSon` file computed with `VoxLogicA`, where each state has as set of Boolean attributes, one for each evaluated spatial property.

After all states and transitions have been marked with the required information, the strategy synthesis is performed as the final, sixth activity of `CATLib`. The algorithm computes the maximal behaviour of the composition (in agreement) such that it is always possible to reach a final configuration and forbidden configurations are never traversed. If the strategy is non-empty, this will provide information on the behaviour to be followed by the controllable agents to ensure that a final configuration is always reached without passing through forbidden configurations, against all possible moves of uncontrollable components.

`VoxLogicA` Activities. The first activity of `VoxLogicA` is the evaluation of the formulae representing final and forbidden states. This is done via an auxiliary python script, that takes as input the logical specification, described by a python function, whose body is constituted by an "f-string", that is a string, where python expressions enclosed in curly braces are evaluated in place, the base image (i.e., the map or planimetry where agents move), and the directory containing all the reachable configurations, encoded as images. The python script then iterates the specification, evaluating expressions where appropriate. The parameters of the python function describing the specification are the base image filename and the currently evaluated configuration, such that the specification can only

Table 1. Summary of the two experiments

	First experiment	**Second experiment**
Controllable	Red and green agents	Door
Uncontrollable	Door	Red and green agents
Initial state	Green agent in front of red agent	Green agent in front of red agent
Final states	Both the red and the green agent reached the exit	The door separates the green agent on the right from the red agent on the left
Forbidden states	The door separates the green agent on the right from the red agent on the left, or the red and green agents are not near each other	Both the red and the green agent reached the exit
Strategy	The red and green agents switch position before traversing the door	Empty

evaluate properties of a single configuration, using the base image to identify relevant regions (like walls).

The second activity of `VoxLogicA` collects all the properties that have been computed in the first activity, locally for each state, and turns them into a single source of information, in the form of a `JSon` file that contains a record for each state, reporting on all the properties that have been described in the specification. In order to do so, a special output mode of `VoxLogicA` is used, where the tool outputs a single `JSon` record of all the user-specified properties that have been printed or saved in the specification.

The presented methodology is a first step towards connecting `CATLib` and `VoxLogicA`. While correct, its efficiency could be improved, especially the input/output overhead. We provide details on future enhancements in Section 5.

4 Experiments

In this section, we describe the experiments that have been performed following the process described in the previous section. We performed two experiments, starting from the same initial conditions but with opposite controllable/uncontrollable agents and forbidden/final states. The setup and outcome of the experiments are reported in Table 1. The repository containing all data, sources, and information on how to reproduce the experiments is publicly available [14].

The `PNG` map image used as planimetry is a 10×10 pixels image that weighs 188 bytes. It is depicted in Figures 1 and 4 (without coloured pixels). Since this is a preliminary exploratory study, we focus on a simple image, leaving more complex scenarios for future work.

The setup for the experiments is of two duplicate mobile agents, one door agent and one driver agent. The door agent is placed in position `(2; 7)` (cf. Figure 4). Initially, the red agent is in the top left corner of the white corridor (position `(1; 1)`), whereas the green agent is just below the red agent (position `(2; 1)`) and the door is closed. The initial state is depicted in Figure 6 (left).

The illegal moves were described in the previous section. We recall that in a legal composition, no agent moves over a wall, a closed door, or another agent.

The invocation of the composition function of `CATLib` is reported below. The composition is instantiated with the list of operands, namely the two agents, the driver, and the door. The second argument is the pruning predicate: if a generated transition satisfies the pruning predicate it will be pruned and not further explored. When applying the composition it is possible to specify a bound on the maximum depth of the generated automaton. In this case, the bound is set to the maximum Integer value. The two agents are instantiated with `maze_tr` and `maze2_tr` being their set of transitions, which only differ in the initial state. The property of agreement is passed as a lambda expression: transitions with a request label will be pruned. Similarly, this condition is put in disjunction with a condition checking whether the target state of the generated transition is 'bad' (i.e., an illegal transition), in which case the transition is pruned.

```
MSCACompositionFunction<String> cf = new MSCACompositionFunction<>
  (List.of(new Automaton<>(maze_tr),new Automaton<>(maze2_tr),driver,door),
    t->t.getLabel().isRequest() || badState.test(t.getTarget()));
Automaton<String, Action, State<String>, ModalTransition<String,Action,State<String>,CALabel>>
  comp = cf.apply(Integer.MAX_VALUE);
```

In the first experiment, the final and forbidden states are set according to the following definitions. Consider the specification given in Figure 5 (cf. Section 2 for an introduction on the operators used therein). The final states are set to be those on the right hand side of the image passing through the corridor where the door is located (property `final`), and are depicted in Figure 6 (middle). Concerning the forbidden states, we experiment with two different spatial properties to identify them. The first property (property `forbidden1`) is a disjunction of two sub-properties. It identifies as forbidden those states that are either illegal (property `wrong`) or in which the two agents are in two areas separated by the closed door, and the green agent is on the right side of the door, i.e., it can reach an escape (a final state), whereas the red agent cannot because it is blocked by the door (property `greenFlees`). In fact, Figure 4 represents one of these forbidden states. The second property (property `forbidden2`) identifies as forbidden those states that are forbidden according to `forbidden1` or in which the two agents are not close to each other, i.e., they are distant more than two pixels (negation of the property `nearBy`).

Finally, in this first experiment we interpret the door as uncontrollable, whereas the red and green agents are controllable. Basically, this is a scenario in which the two players are playing against an uncontrollable door. Below we list the code used to invoke the synthesis operation of `CATLib`. The instantiation of the operation takes as argument the property to enforce, agreement in this case, and the automaton where the synthesis is applied, called `marked` in this case.

```
Automaton<String, Action, State<String>, ModalTransition<String,Action,State<String>,CALabel>>
 strategy = new MpcSynthesisOperator<String>(new Agreement()).apply(marked);
```

The most permissive synthesised strategy consists of 684 states and 2635 transitions (recall that in each transition, only one of the agents is moving). The

```
load base ="{baseimage}"

load img = "{datadir}/{image}"
let r = red(img)
let g = green(img)
let b = blue(img)
let rb = red(base)
let gb = green(base)
let bb = blue(base)

let exists(p) = volume(p) .>. 0
let forall(p) = volume(p) .=. volume(tt)
let forallin(x,p) = forall( (!x) | p)

let door = (r =. 0) & (b =. 255) & (g =. 0)
let floorNoDoor = (rb =. 255) & (bb =. 255) & (gb =. 255)
let floor = floorNoDoor & (!door)
let wall = !floor

let mrRed = (r =. 255) & (b =. 0) & (g =. 0)
let mrGreen = (r =. 0) & (b =. 0) & (g =. 255)

let mrX = mrRed | mrGreen

let initial1 = exists(mrRed & ((x =. 1) & (y =. 1)))
        .&. exists(mrGreen & ((x =. 1) & (y =. 2)))
let initial2 = exists(mrRed & ((x =. 6) & (y =. 3)))
        .&. exists(mrGreen & ((x =. 1) & (y =. 4)))
let wrong = exists(mrX & wall) .|. (!. (exists(mrRed) .&. exists(mrGreen)))
let exit = (x =. 9) & (y >. 2) & (y <. 9)
let pathToExit = (floor ~> exit)
let canExit(mr) = forallin(mr,pathToExit)
let sameRoom = forallin(mrGreen,(mrGreen|floor) ~> mrRed)

let greenFlees = (!.wrong) .&. canExit(mrGreen) .&. (!.(canExit(mrRed)))
let nearby = exists(mrRed & (N N mrGreen))

let forbidden1 = greenFlees .|. wrong
let forbidden2 = forbidden1 .|. (!. nearby)

let final = exists(mrRed & exit) .&. exists(mrGreen & exit)
```

Fig. 5. `VoxLogicA` specification of the properties used in the experiments

length of a shortest path from the initial state to a final state is composed of 33 transitions to be executed. In the initial state (Figure 6 (left)), the green agent is in front of the red agent on the path to the exit. However, in the strategy the green agent cannot traverse the open door before the red agent. Indeed, in this case, since the door is uncontrollable, it is not possible to prevent the door from closing and separating the red agent (blocked by the door) from the green agent (who can reach the exit). This is indeed a forbidden state that the strategy must avoid. In the strategy, to overcome this problem, the two agents switch position before crossing the door. Figure 6 (right) depicts the moment where the red agent is crossing the door right after exchanging position with the green agent who is still in the corridor. Indeed, in the shortest path they switch position near the door. Note that no forbidden state occurs if the door closes after only the red agent has traversed it. Indeed, in this scenario the green agent is prevented from reaching an exit because it is blocked by the door. Hence,

Fig. 6. On the left the initial configuration of both experiments. In the middle the final states of the first experiment (marked in violet). On the right a configuration traversed by one of the shortest paths of the first experiment's strategy, in which the red agent is crossing the door before the green agent does, thus avoiding forbidden configurations.

after the red agent has traversed the door, the strategy guides the green agent to safely cross the door such that they can both reach a final state.

To confirm the first experiment, we performed a second experiment by inverting the setup of the first experiment. In this second experiment, the door is controllable, whereas the green and red agents are both uncontrollable. The final states are those in which the door separates the green agent (on the right side of the door) from the red agent (on the left side of the door). These are basically the forbidden states of the first experiment. Similarly, the forbidden states in the second experiment are those states in which both the green and the red agent have reached the exit, i.e., the final states of the first experiment. The initial configuration is the same as in the first experiment. As expected, in this dual case the returned strategy is empty. Indeed, if this were not the case, then we would have a contradiction because the green and red agents have a strategy to reach the exit without being separated by the door with the red agent blocked, for every possible finite behaviour of the door.

There is no strategy for the door to reach a final configuration mainly because the door cannot ensure that the uncontrollable green agent traverses the door first. Moreover, the door cannot prevent the agents from reaching the exit by always remaining closed since (unless only the green agent has traversed the door) a final state would not be reachable.

Performance of Experiments. We conclude this section by reporting the time needed for computing various phases of the experiments and measures of the computed automata. The experiments were performed on a machine with Intel(R) Core(TM) i9-9900K CPU @ 3.60 GHz equipped with 32 GB of RAM. The time performance is reported in Table 2. We note that the synthesis is more expensive (computationally) than the composition. Indeed, as showed in Section 2, each iteration of the synthesis requires to compute the set of dangling states, which requires a forward and backward visit of the automaton. The marking is the most computationally expensive phase of `CATLib` because each marking of either a final or a forbidden state requires to search whether that state has a final or forbidden attribute in the `JSon` file provide by `VoxLogicA`.

Table 3 reports the number of states, the number of transitions, and the size (in bytes) of the various automata. As expected, the number of states of the agent automaton is exactly the number of pixels of the image. The largest automaton is the one with the unconstrained composition, whose number of

Table 2. Time needed to perform the experiments' phases

Phase	**Both Experiments**	
Computing the unconstrained composition	26643 ms	
Generating images	7910 ms	
Running `VoxLogicA`	6140 s	
Computing the legal composition	2487 ms	
	First experiment	**Second experiment**
Marking the composition with `VoxLogicA` properties and controllability	108058 ms	118291 ms
Synthesis	2942 ms	33472 ms

Table 3. Number of states, transitions and size of the automata used in the experiments

Automaton	**#States**	**#Transitions**	**Size (bytes)**
Agent	100	360	18723
Unconstrained composition	20000	164800	21858146
Legal composition	3200	15176	2004736
Marked composition (first experiment)	3202	17665	2339874
Marked composition (second experiment)	3202	15552	2066368
Strategy (first experiment)	684	2635	347652

states is the product of the states of the two agent automata and the door automaton ($100 \times 100 \times 2$). We note that the agent automaton (encoding an image as automaton) requires more space than the `PNG` image (188 bytes of the image against 18723 bytes of the corresponding automaton). Moreover, the legal states given the initial conditions are only a small fraction (16%) of the total number of states passed to `VoxLogicA`. Finally, the marked compositions for the two experiments have two additional states with respect to the legal composition, which are the added initial and final states. The number of transitions of these two automata differs according to the number of states marked as final, to which a transition to the newly added final state is added, and the number of forbidden states, to which a bad transition is added as a self-loop.

The evaluation of the given `VoxLogicA` specification (also reported in Table 2) takes about 530 milliseconds per image, of which only 45 milliseconds are spent on the actual computation; the rest is spent in file input/output, parsing the specification, and recovering the results from python. Since all the images are processed sequentially, the total analysis time for the 11562 images that are generated is therefore a bit less than two hours, which dominates the total computation time for the experiment. As discussed in Section 3, much of the overhead could be eliminated (cf. the Conclusion for more information).

5 Conclusion

We have discussed an integration of the tools `CATLib` and `VoxLogicA` to perform strategy synthesis on images processed with spatial model checking. Our con-

tribution constitutes the first application of `CATLib` and `VoxLogicA` to build a framework for modelling and solving multi-agents mobile problems. The result clearly demonstrates the feasibility of a full-fledged tool chain built from `CATLib` and `VoxLogicA` and shows an original approach to combine strategy synthesis with spatial model checking. The experiments performed in this paper are still preliminary and not much thought has been given to the efficiency of the encodings, the computations, and the tool integration. Hence, this paper offers a lot of interesting opportunities for future work.

Future Work. The proof-of-concept example in this paper uses a 10×10 pixels map. Efficiency and scalability are two key issues to address in the future. Several possible scalable solutions are viable and some ideas are provided next.

In the current approach, many states are used to move agents up and down the ends of corridors (each agent has a state for each pixel of the image). However, fewer states could actually be sufficient. Relaxing the representation of an image to one where each state is a zone of the image (e.g., a corridor) rather than a pixel would drastically reduce the state space.

Another scalable solution could be to decompose a large image into smaller images. For example, the final states of the first experiment in Section 4 could be entering points to a new portion of the map. Several small maps could be linked together by ports for entering and exiting.

Yet another scalable solution could be to drop the requirement of a strategy to be most permissive in favour of some objective function to optimise. A near-optimal solution could be synthesised as a trace using statistics over runs, in the style of [46].

Currently, each new parameter setup requires to be implemented manually. Similarly, the various `CATLib` and `VoxLogicA` activities depicted in Figure 1 need to be invoked manually. Future research is needed to completely automatise our proposal, providing a tool that takes as input the setup of an experiment, including the map, and outputs the synthesised strategy, if any, in a push-button way. This could result in an optimisation of the methodology presented in Section 3. Indeed, as shown in Section 4 and Table 2, currently a bottleneck is present in the processing of the images and `JSon` logs, mainly due to the offline processing of all images by `VoxLogicA`, for all possible initial conditions of the experiments. For example, the actual time spent on computing the evaluation of properties using `VoxLogicA` is a small fraction of its total evaluation time. The rest is spent in parsing, loading, and saving, which is repeated for each image and could mostly be eliminated. The number of images and total size of the logs could be reduced drastically by making `CATLib` and `VoxLogicA` interact online at each new experiment. In this way, there would be no need for `CATLib` to initially generate all possible states. Only those states that are actually reachable given the setup of the experiment at hand could be generated. This would result in far fewer images to be processed by `VoxLogicA` and a smaller `JSon` log to be parsed by `CATLib` in return. Concerning `VoxLogicA`, the input/output overhead could also be eliminated by loading several files at once in parallel, parsing the specification only once, and exploiting the recent GPU implementation [26].

CRediT author statement

D. Basile: Conceptualization, Software, Formal Analysis, Investigation, Writing - Original Draft, Writing - Review & Editing. **M.H. ter Beek**: Writing - Original Draft, Writing - Review & Editing, Supervision, Funding Acquisition, Project Administration. **V. Ciancia**: Conceptualization, Software, Formal Analysis, Investigation, Writing - Original Draft, Writing - Review & Editing.

Acknowledgments. Research partially funded by the MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems).

References

1. Alsalehi, S., Mehdipour, N., Bartocci, E., Belta, C.: Neural network-based control for multi-agent systems from spatio-temporal specifications. In: Proceedings of the 60th IEEE Conference on Decision and Control (CDC 2021). pp. 5110–5115. IEEE (2021). https://doi.org/10.1109/CDC45484.2021.9682921
2. Alur, R., Moarref, S., Topcu, U.: Compositional synthesis of reactive controllers for multi-agent systems. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 251–269. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_14
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. IFAC Proc. Vol. **31**(18), 447–452 (1998). https://doi.org/10.1016/S1474-6670(17)42032-5
4. Banci Buonamici, F., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial logics and model checking for medical imaging. Int. J. Softw. Tools Technol. Transf. **22**(2), 195–217 (2020). https://doi.org/10.1007/s10009-019-00511-9
5. Basile, D., ter Beek, M.H., Legay, A.: Timed service contract automata. Innovations Syst. Softw. Eng. **16**(2), 199–214 (2020). https://doi.org/10.1007/s11334-019-00353-3
6. Basile, D., ter Beek, M.H.: A clean and efficient implementation of choreography synthesis for behavioural contracts. In: Damiani, F., Dardha, O. (eds.) COORDINATION 2021. LNCS, vol. 12717, pp. 225–238. Springer (2021). https://doi.org/10.1007/978-3-030-78142-2_14
7. Basile, D., ter Beek, M.H.: Contract Automata Library. Sci. Comput. Program. (2022). https://doi.org/10.1016/j.scico.2022.102841, https://github.com/contractautomataproject/ContractAutomataLib
8. Basile, D., ter Beek, M.H.: A runtime environment for contract automata. arXiv:2203.14122 (2022), 10.48550/arXiv.2203.14122
9. Basile, D., ter Beek, M.H., Degano, P., Legay, A., Ferrari, G.L., Gnesi, S., Di Giandomenico, F.: Controller synthesis of service contracts with variability. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102344
10. Basile, D., ter Beek, M.H., Di Giandomenico, F., Gnesi, S.: Orchestration of dynamic service product lines with featured modal contract automata. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC 2017). vol. 2, pp. 117–122. ACM (2017). https://doi.org/10.1145/3109729.3109741

11. Basile, D., ter Beek, M.H., Gnesi, S.: Modelling and analysis with featured modal contract automata. In: Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC 2018). vol. 2, pp. 11–16. ACM (2018). https://doi.org/10.1145/3236405.3236408
12. Basile, D., ter Beek, M.H., Legay, A.: Strategy synthesis for autonomous driving in a moving block railway system with Uppaal Stratego. In: Gotsman, A., Sokolova, A. (eds.) FORTE 2020. LNCS, vol. 12136, pp. 3–21. Springer (2020). https://doi.org/10.1007/978-3-030-50086-3_1
13. Basile, D., ter Beek, M.H., Pugliese, R.: Synthesis of orchestrations and choreographies: Bridging the gap between supervisory control and coordination of services. Log. Methods Comput. Sci. **16**(2) (2020). https://doi.org/10.23638/LMCS-16(2:9)2020
14. Basile, D., Ciancia, V.: Repository for reproducing the experiments, https://github.com/contractautomataproject/CATLib_PngConverter
15. Basile, D., Degano, P., Ferrari, G.L.: Automata for specifying and orchestrating service contracts. Log. Methods Comput. Sci. **12**(4) (2016). https://doi.org/10.2168/LMCS-12(4:6)2016
16. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Playing with our CAT and communication-centric applications. In: Albert, E., Lanese, I. (eds.) FORTE 2016. LNCS, vol. 9688, pp. 62–73. Springer (2016). https://doi.org/10.1007/978-3-319-39570-8_5
17. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Relating two automata-based models of orchestration and choreography. J. Log. Algebr. Methods Program. **85**(3), 425–446 (2016). https://doi.org/10.1016/j.jlamp.2015.09.011
18. Basile, D., Di Giandomenico, F., Gnesi, S.: Enhancing models correctness through formal verification: A case study from the railway domain. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017). pp. 679–686. SciTePress (2017). https://doi.org/10.5220/0006291106790686
19. Basile, D., Di Giandomenico, F., Gnesi, S.: FMCAT: Supporting dynamic service-based product lines. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC 2017). vol. 2, pp. 3–8. ACM (2017). https://doi.org/10.1145/3109729.3109760
20. Basile, D., Di Giandomenico, F., Gnesi, S., Degano, P., Ferrari, G.L.: Specifying variability in service contracts. In: Proceedings of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2017). pp. 20–27. ACM (2017). https://doi.org/10.1145/3023956.3023965
21. ter Beek, M.H., Reniers, M.A., de Vink, E.P.: Supervisory controller synthesis for product lines using CIF 3. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 856–873. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_59
22. Belmonte, G., Broccia, G., Vincenzo, C., Latella, D., Massink, M.: Feasibility of spatial model checking for nevus segmentation. In: Proceedings of the 9th International Conference on Formal Methods in Software Engineering (FormaliSE 2021). pp. 1–12. IEEE (2021). https://doi.org/10.1109/FormaliSE52586.2021.00007
23. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Innovating medical image analysis via spatial logics. In: ter Beek, M.H., Fantechi, A., Semini, L. (eds.) From Software Engineering to Formal Methods and Tools, and Back. LNCS, vol. 11865, pp. 85–109. Springer (2019). https://doi.org/10.1007/978-3-030-30985-5_7

24. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: VoxLogicA: A spatial model checker for declarative image analysis. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 281–298. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_16
25. Bezhanishvili, N., Ciancia, V., Gabelaia, D., Grilletti, G., Latella, D., Massink, M.: Geometric model checking of continuous space. arXiv:2105.06194 (2021), https://arxiv.org/abs/2105.06194
26. Bussi, L., Ciancia, V., Gadducci, F.: Towards a spatial model checker on GPU. In: Peters, K., Willemse, T.A.C. (eds.) FORTE 2021. LNCS, vol. 12719, pp. 188–196. Springer (2021). https://doi.org/10.1007/978-3-030-78089-0_12
27. Calude, C.S., Jain, S., Khoussainov, B., Li, W., Stephan, F.: Deciding parity games in quasipolynomial time. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017). pp. 252–263. ACM (2017). https://doi.org/10.1145/3055399.3055409
28. Camacho, A., Bienvenu, M., McIlraith, S.A.: Towards a unified view of AI planning and reactive synthesis. In: Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2018). pp. 58–67. AAAI (2019), https://ojs.aaai.org/index.php/ICAPS/article/view/3460
29. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer (2006). https://doi.org/10.1007/978-0-387-68612-7
30. Castelnovo, D., Miculan, M.: Closure hyperdoctrines. In: Gadducci, F., Silva, A. (eds.) Proceedings of the 9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021). LIPIcs, vol. 211, pp. 12:1–12:21 (2021). https://doi.org/10.4230/LIPIcs.CALCO.2021.12
31. Cauchi, N., Abate, A.: StocHy: Automated verification and synthesis of stochastic processes. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 247–264. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_14
32. Ceska, M., Pilar, P., Paoletti, N., Brim, L., Kwiatkowska, M.Z.: PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems. In: Chechik, M., Raskin, J. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 367–384. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_21
33. Cheng, C., Lee, E.A., Ruess, H.: autoCode4: Structural controller synthesis. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 398–404. Springer (2017). https://doi.org/10.1007/978-3-662-54577-5_23
34. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Model checking spatial logics for closure spaces. Log. Methods Comput. Sci. **12**(4) (2016). https://doi.org/10.2168/LMCS-12(4:2)2016
35. Ciancia, V., Latella, D., Massink, M., Paškauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 657–673. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_46
36. Ciancia, V., Belmonte, G., Latella, D., Massink, M.: A hands-on introduction to spatial model checking using VoxLogicA – invited contribution. In: Laarman, A., Sokolova, A. (eds.) SPIN 2021. LNCS, vol. 12864, pp. 22–41. Springer (2021). https://doi.org/10.1007/978-3-030-84629-9_2
37. Ciancia, V., Gilmore, S., Grilletti, G., Latella, D., Loreti, M., Massink, M.: Spatio-temporal model checking of vehicular movement in public transport systems. Int. J. Softw. Tools Technol. Transf. **20**(3), 289–311 (2018). https://doi.org/10.1007/s10009-018-0483-8

38. Ciancia, V., Latella, D., Massink, M., Paškauskas, R.: Exploring spatio-temporal properties of bike-sharing systems. In: Proceedings of the Workshops at the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015). pp. 74–79. IEEE (2015). https://doi.org/10.1109/SASOW.2015.17
39. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
40. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) Proceedings of the 21st European Symposium on Programming (ESOP 2012). LNCS, vol. 7211, pp. 194–213. Springer (2012). https://doi.org/10.1007/978-3-642-28869-2_10
41. Ehlers, R., Lafortune, S., Tripakis, S., Vardi, M.Y.: Supervisory control and reactive synthesis: a comparative introduction. Discret. Event Dyn. Syst. **27**(2), 209–260 (2017). https://doi.org/10.1007/s10626-015-0223-0
42. Fan, C., Miller, K., Mitra, S.: Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In: Lahiri, S.K., Wang, C. (eds.) Proceedings of the 32nd International Conference on Computer Aided Verification (CAV'20). LNCS, vol. 12224, pp. 629–652. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_31
43. Farhat, H.: Web service composition via supervisory control theory. IEEE Access **6**, 59779–59789 (2018). https://doi.org/10.1109/ACCESS.2018.2874564
44. Felli, P., Yadav, N., Sardina, S.: Supervisory control for behavior composition. IEEE Trans. Autom. Control **62**(2), 986–991 (2017). https://doi.org/10.1109/TAC.2016.2570748
45. Forschelen, S.T.J., van de Mortel-Fronczak, J.M., Su, R., Rooda, J.E.: Application of supervisory control theory to theme park vehicles. Discrete Event Dyn. Syst. **22**(4), 511–540 (2012). https://doi.org/10.1007/s10626-012-0130-6
46. Gu, R., Jensen, P.G., Poulsen, D.B., Seceleanu, C., Enoiu, E., Lundqvist, K.: Verifiable strategy synthesis for multiple autonomous agents: a scalable approach. Int. J. Softw. Tools Technol. Transf. **24**(3), 395–414 (2022). https://doi.org/10.1007/s10009-022-00657-z
47. Guo, M., Dimarogonas, D.V.: Multi-agent plan reconfiguration under local LTL specifications. Int. J. Robotics Res. **34**(2), 218–235 (2015). https://doi.org/10.1177/0278364914546174
48. Haghighi, I., Jones, A., Kong, Z., Bartocci, E., Grosu, R., Belta, C.: SpaTel: A novel spatial-temporal logic and its applications to networked systems. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC 2015). pp. 189–198. ACM (2015). https://doi.org/10.1145/2728606.2728633
49. Haghighi, I., Sadraddini, S., Belta, C.: Robotic swarm control from spatio-temporal specifications. In: Proceedings of the 55th IEEE Conference on Decision and Control (CDC 2016). pp. 5708–5713. IEEE (2016). https://doi.org/10.1109/CDC.2016.7799146
50. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 475–487. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_25
51. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015). pp. 221–232. ACM (2015). https://doi.org/10.1145/2676726.2676964

52. Lavaei, A., Khaled, M., Soudjani, S., Zamani, M.: AMYTISS: Parallelized automated controller synthesis for large-scale stochastic systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 461–474. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_24
53. Lehmann, S., Rogalla, A., Neidhardt, M., Reinecke, A., Schlaefer, A., Schupp, S.: Modeling $\mathbb{R}^3$ needle steering in Uppaal. In: Dubslaff, C., Luttik, B. (eds.) Proceedings of the 5th Workshop on Models for Formal Analysis of Real Systems (MARS 2022). EPTCS, vol. 355, pp. 40–59 (2022). https://doi.org/10.4204/EPTCS.355.4
54. Liu, Z., Wu, B., Dai, J., Lin, H.: Distributed communication-aware motion planning for networked mobile robots under formal specifications. IEEE Trans. Control. Netw. Syst. **7**(4), 1801–1811 (2020). https://doi.org/10.1109/TCNS.2020.3000742
55. Loizou, S.G., Kyriakopoulos, K.J.: Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: Proceedings of the 43rd IEEE Conference on Decision and Control (CDC 2004). pp. 153–158. IEEE (2004). https://doi.org/10.1109/CDC.2004.1428622
56. Loreti, M., Quadrini, M.: A spatial logic for a simplicial complex model. arXiv:2105.08708 (2021), https://arxiv.org/abs/2105.08708
57. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Inform. **57**(1-2), 3–36 (2020). https://doi.org/10.1007/s00236-019-00349-3
58. Ma, M., Bartocci, E., Lifland, E., Stankovic, J.A., Feng, L.: A novel spatial–temporal specification-based monitoring system for smart cities. IEEE Internet Things J. **8**(15), 11793–11806 (2021). https://doi.org/10.1109/JIOT.2021.3069943
59. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 578–586. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_31
60. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987). https://doi.org/10.1137/0325013
61. van der Sanden, B., Reniers, M.A., Geilen, M., Basten, T., Jacobs, J., Voeten, J., Schiffelers, R.R.H.: Modular model-based supervisory controller design for wafer logistics in lithography machines. In: Proceedings of the 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015). pp. 416–425. IEEE (2015). https://doi.org/10.1109/MODELS.2015.7338273
62. Shokri-Manninen, F., Vain, J., Waldén, M.: Formal verification of COLREG-based navigation of maritime autonomous systems. In: de Boer, F.S., Cerone, A. (eds.) SEFM 2020. LNCS, vol. 12310, pp. 41–59. Springer (2020). https://doi.org/10.1007/978-3-030-58768-0_3
63. Sun, D., Chen, J., Mitra, S., Fan, C.: Multi-agent motion planning from signal temporal logic specifications. IEEE Robotics Autom. Lett. **7**(2), 3451–3458 (2022). https://doi.org/10.1109/LRA.2022.3146951
64. Theunissen, R.J.M., van Beek, D.A., Rooda, J.E.: Improving evolvability of a patient communication control system using state-based supervisory control synthesis. Adv. Eng. Inform. **26**(3), 502–515 (2012). https://doi.org/10.1016/j.aei.2012.02.009
65. Tsigkanos, C., Kehrer, T., Ghezzi, C.: Modeling and verification of evolving cyber-physical spaces. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). pp. 38–48. ACM (2017). https://doi.org/10.1145/3106237.3106299