SICS Software-Intensive Cyber-Physical Systems (2019) 34:201–212
https://doi.org/10.1007/s00450-019-00424-7

SPECIAL ISSUE PAPER

# Strategic logics for collaborative embedded systems

## Specification and verification of collaborative embedded systems using strategic logics

**Damian Kurpiewski**[1] · **Diego Marmsoler**[2]

Published online: 5 December 2019


**Abstract**
In embedded systems, there is a clear movement from autonomous systems towards collaborative systems, forming so-called collaborative system groups (CSGs), which collaborate to achieve common goals. Verification of CSGs, however, imposes new challenges, which are difficult to address with traditional verification techniques. In the following, we investigate the use of strategic logics for the analysis of CSGs, by means of a use-case in the domain of smart production systems. Our results show that strategic model checking is useful to investigate certain aspects of CSGs, such as the impact of environmental changes. However, our results also show some limitations of the approach, when it comes to the analysis of implementation-level aspects, such as performance. Thus, we conclude that strategic model checking might complement existing approaches for the analysis of CSGs.

**Keywords** Collaborative embedded system · Collaborative system group · Strategic logic · ATL

## 1 Introduction

More than ever, our daily life is determined by smart systems which are embedded into our environment. We use our smart phones regularly in our daily life activities. Our cars are equipped with 30–100 microprocessors with up to 100 million lines of software [28]. Smart production facilities will raise our production of goods by up to 25 percent with the potential to create as much as 1.8 trillion in new value per year across the worlds factories by 2025 [27]. Due to the impact of embedded systems on society, their verification has become an important task and usually it is done using traditional model checking techniques [11]: A system and its environment is modeled in terms of state machines and then analyzed whether or not the model satisfies certain safety or liveness properties in a given environment.

✉ Damian Kurpiewski
d.kurpiewski@ipipan.waw.pl

Diego Marmsoler
diego.marmsoler@tum.de

[1] Institute of Computer Science, Polish Academy of Sciences, Jana Kazimierza 5, 01-248 Warsaw, Poland

[2] Technische Universität München, Boltzmannstr. 3, 85748 Garching bei München, Germany

While embedded systems were traditionally designed to act autonomously, the trend goes to groups of systems, which collaborate to achieve common goals more efficiently [30]. This development led to the notion of so-called *collaborative embedded systems* (CESs), systems which behavior is driven by certain goals and which may collaborate with other CESs to achieve these goals. To this end, they may be grouped into so-called *collaborative system groups* (CSGs), consisting of several CESs with similar goals [34]. A simple example of a CSG is a fleet of self-driving cars, in which each car represents a CES and collaborates with other cars to optimize road traffic and avoid collisions. Another example, which also serves as a use case for the work presented in this paper, is a smart production system, in which production machines are served by a group of autonomous transport robots with the goal to provide a flexible production chain.

Compared to the verification of traditional embedded systems, verification of CSGs provide new challenges: In addition to check whether certain properties hold in combination with an environment, we are now also interested in whether a group of systems is able to achieve a certain goal, no matter what its environment does. For example, when planning an autonomous car fleet, we would be interested in determining whether the cars are indeed able to reach their goal and optimize road traffic. Or, when planning the intro-

duction of smart production systems, in which autonomous transport robots carry items between machines which produce goods, we want to determine whether or not a given set of robots is able to serve all the requests issued by the machines. In summary, we want to answer two questions regarding CSGs prior to implementing them:

- *Is the goal of a CSG indeed feasible, i.e., is it possible for the participants of the CSG to achieve the goal*?
- *What strategies can be used to achieve a certain goal*?

Such questions, however, are difficult (if not to say impossible) to answer using traditional verification techniques.

Thus, in the following, we propose an approach for the analysis of CSGs, based on strategic logics [7]. To evaluate it, we applied it for the analysis of smart production systems. Therefore, we first specified the CESs and its environment in terms of so-called concurrent game structures (CGS). Then, we formalized goals in terms of ATL formulas and analyzed them over the CGS using strategic model checking.

In this paper we report on our experience in applying strategic model checking to the analysis of CSGs. Its major contributions can be summarized as follows:

- It introduces a methodology for the analysis of CGSs using strategic model checking.
- It presents the outcome of applying the method for the analysis of a smart production factory.
- It reports our experience in applying strategic logic for the analysis of CSGs.

To do so, the paper is structured as follows: First, we provide an overview of strategic logics and describe our approach in more detail. Then, we introduce our use-case from the area of smart production factories. We then describe how we applied the approach for the analysis of smart production factories: we describe how we specified the group and its environment in terms of a concurrent game structure and how the group was analyzed by means of strategic model checking. Finally, we discuss challenges and limitations of using ATL for the analysis of CSGs and point to future work to address them.

## 2 Strategic logics for collaborative embedded systems

In the following, we present our approach. To this end, we first provide some general background on ATL. Then, we describe how it can be applied for the analysis of CSGs.

### 2.1 Alternating-time temporal logic

#### 2.1.1 Models

We interpret *ATL* specifications over a variant of transition systems where transitions are labeled with combinations of actions, one per agent. Moreover, epistemic relations are used to indicate states that look the same to a given agent. Formally, an *imperfect information concurrent game structure* or *iCGS* is given by $M = \langle \mathbb{A}\text{gt}, St, Props, V, Act, d, o, \{\sim_a | a \in \mathbb{A}\text{gt}\}\rangle$ which includes a nonempty finite set of agents $\mathbb{A}\text{gt} = \{1, \ldots, k\}$, a nonempty set of states $St$, a set of atomic propositions $Props$ and their valuation $V : Props \to 2^{St}$, and a nonempty finite set of (atomic) actions $Act$. The protocol function $d : \mathbb{A}\text{gt} \times St \to 2^{Act}$ defines nonempty sets of actions available to agents at each state; we will write $d_a(q)$ instead of $d(a, q)$, and define $d_A(q) = \prod_{a \in A} d_a(q)$ for each $A \subseteq \mathbb{A}\text{gt}, q \in St$. Furthermore, $o$ is a (deterministic) transition function that assigns the outcome state $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to each state $q$ and tuple of actions $\langle \alpha_1, \ldots, \alpha_k \rangle$ such that $\alpha_i \in d(i, q)$ for $i = 1, \ldots, k$. Every $\sim_a \subseteq St \times St$ is an epistemic equivalence relation with the intended meaning that, whenever $q \sim_a q'$, the states $q$ and $q'$ are indistinguishable to agent $a$. The iCGS is assumed to be *uniform*, in the sense that $q \sim_a q'$ implies $d_a(q) = d_a(q')$. Note that perfect information can be modeled by assuming each $\sim_a$ to be the identity relation.

#### 2.1.2 Strategies

A *strategy* of agent $a \in \mathbb{A}\text{gt}$ is a conditional plan that specifies what $a$ is going to do in every possible situation. Formally, a *perfect information memoryless strategy* for $a$ can be represented by a function $s_a : St \to Act$ satisfying $s_a(q) \in d_a(q)$ for each $q \in St$. An *imperfect information memoryless strategy* additionally satisfies $s_a(q) = s_a(q')$ whenever $q \sim_a q'$. We refer to the former as Ir -*strategies*, and to the latter as ir -*strategies*. Capital letter I stands for *Perfect Information*, while lowercase i stands for *Imperfect information*. Similarly lowercase r stands for *Imperfect Recall*, hence memoryless strategies.

A *collective x-strategy* $s_A$, for coalition $A \subseteq \mathbb{A}\text{gt}$ and strategy type $x \in \{\text{Ir}, \text{ir}\}$, is a tuple of individual $x$-strategies, one per agent from $A$. The set of all such strategies is denoted by $\Sigma^x_A$. By $s_A|_a$ we denote the strategy of agent $a \in A$ selected from $s_A$.

#### 2.1.3 Outcome paths

A *path* $\lambda = q_0 q_1 q_2 \ldots$ is an infinite sequence of states such that there is a transition between each $q_i, q_{i+1}$. We use $\lambda[i]$ to denote the $i$th position on path $\lambda$ (starting from $i = 0$) and $\lambda[i, j]$ to denote the part of $\lambda$ between positions $i$ and $j$.

Function $out(q, s_A)$ returns the set of all paths that can result from the execution of a (complete) strategy $s_A$, beginning at state $q$. Formally:

$$out(q, s_A) = \{\lambda = q_0, q_1, q_2 \ldots \mid q_0 = q \text{ and for each } i = 0, 1, \ldots \text{ there exists } \langle \alpha^i_{a_1}, \ldots, \alpha^i_{a_k} \rangle \text{ such that } \alpha^i_a \in d_a(q_i) \text{ for every } a \in \mathbb{A}\text{gt, and } \alpha^i_a = s_A|_a(q_i) \text{ for every } a \in A, \text{ and } q_{i+1} = o(q_i, \alpha^i_{a_1}, \ldots, \alpha^i_{a_k})\}.$$

We will sometimes write $out^{\mathrm{Ir}}(q, s_A)$ instead of $out(q, s_A)$. Moreover, the function $out^{\mathrm{ir}}(q, s_A) = \bigcup_{a \in A} \bigcup_{q \sim_a q'} out(q', s_A)$ collects all the outcome paths that start from states that are indistinguishable from $q$ to at least one agent in $A$.

#### 2.1.4 Syntax

We use a variant of **ATL**hat explicitly distinguishes between perfect and imperfect information abilities. Formally, the syntax is defined by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle A \rangle\!\rangle_x \mathbf{X}\phi \mid \langle\!\langle A \rangle\!\rangle_x \mathbf{G}\phi \mid \langle\!\langle A \rangle\!\rangle_x \phi \,\mathbf{U}\, \phi,$$

where $x \in \{\mathrm{Ir}, \mathrm{ir}\}$, $p \in Props$ and $A \subseteq \mathbb{A}\mathrm{gt}$. We read $\langle\!\langle A \rangle\!\rangle_{\mathrm{ir}} \gamma$ as "*A can identify and execute a strategy that enforces* $\gamma$," $\mathbf{X}$ as *"in the next state,"* $\mathbf{G}$ as *"now and always in the future,"* and $\mathbf{U}$ as *"until."* The perfect information modality $\langle\!\langle A \rangle\!\rangle_{\mathrm{Ir}} \gamma$ can be read as *"A might be able to bring about* $\gamma$ *if allowed to make lucky guesses whenever uncertain."* We focus on the kind of ability expressed by $\langle\!\langle A \rangle\!\rangle_{\mathrm{ir}}$. The other strategic modality (i.e., $\langle\!\langle A \rangle\!\rangle_{\mathrm{Ir}}$) will prove useful when approximating $\langle\!\langle A \rangle\!\rangle_{\mathrm{ir}}$.

#### 2.1.5 Semantics

With given iCGS model $M$ and state $q$, the semantics of **ATL**an be defined as follows:

- $M, q \models p$ iff $q \in V(p)$,
- $M, q \models \neg\phi$ iff $M, q \not\models \phi$,
- $M, q \models \phi \wedge \psi$ iff $M, q \models \phi$ and $M, q \models \psi$,
- $M, q \models \langle\!\langle A \rangle\!\rangle_x \mathbf{X}\phi$ iff there exists strategy $s_A \in \Sigma^x_A$ such that for all $\lambda \in out^x(q, s_A)$ we have $M, \lambda[1] \models \phi$,
- $M, q \models \langle\!\langle A \rangle\!\rangle_x \mathbf{G}\phi$ iff there exists $s_A \in \Sigma^x_A$ such that for all $\lambda \in out^x(q, s_A)$ and $i \in \mathbb{N}$ we have $M, \lambda[i] \models \phi$,
- $M, q \models \langle\!\langle A \rangle\!\rangle_x \psi \,\mathbf{U}\, \phi$ iff there exists $s_A \in \Sigma^x_A$ such that for all $\lambda \in out^x(q, s_A)$ there is $i \in \mathbb{N}$ for which $M, \lambda[i] \models \phi$ and $M, \lambda[j] \models \psi$ for all $0 \leq j < i$.

The standard boolean operators (logical constants $\top$ and $\bot$, disjunction $\vee$, and implication $\rightarrow$) are defined as usual. We will often write $\langle A \rangle \varphi$ instead of $\langle\!\langle A \rangle\!\rangle_{\mathrm{ir}} \mathbf{X}\varphi$ to express one-step abilities under imperfect information. Additionally, we define *"now or sometime in the future"* as $\mathbf{F}\varphi \equiv \top \,\mathbf{U}\, \varphi$. It is easy to see that $M, q \models \langle\!\langle A \rangle\!\rangle_x \mathbf{F}\phi$ iff there exists a collective strategy $s_A \in \Sigma^x_A$ such that, on each path $\lambda \in out^x(q, s_A)$, there is a state satisfying $\phi$. In that case, we can also say that $\phi$ is *x-reachable from* $q$.

## 3 Smart production factories

In the following, we describe our use-case in the domain of Industry 4.0, in general, and specifically smart production factories (SPFs) [32]. In such a factory, different machines are placed in different locations to produce certain items. Thereby, production of certain items may require other items and machines actually form a distributed production chain. Since machines are distributed amongst the factors, items somehow need to be transported from one machine to another one.

Usually, this is done by means of so-called automated guided vehicles with pre-defined, fixed routes. In SPF, however, items are carried by a fleet of *autonomous transport robots* (Fig. 1) which can navigate freely in their environment, without being bound to a fixed track. A typical fleet consists of 4–20 robots which can carry 50–200 kg load each. Each robot has a pre-recorded map of the factory (Fig. 3) depicting no-go areas and the position of charging stations, machines, and storage areas. Moreover, robots have a laser scanner which delivers an accurate image of its environment and which is combined with the map in order to determine its current position. In order to move around, robots need electricity which is stored in lithium batteries. In order to keep the robots moving, their batteries need to be recharged from time to time. Machines communicate the production of a new item as well as their need for certain items from other machines by broadcasting it to all the robots via wireless communication. Robots may also communicate to each other via a wireless network.

Compared to traditional automated guided vehicles, autonomous transport robots provide the following benefits:

- A fleet of transport robots can be extended even during operation of the factory; no intervention in the running operation is necessary.
- Changing the location of a machine just requires to adapt the corresponding target point in the internal map of the robots; it is not even necessary to designate new routes.
- Broken transport robots do not interrupt production since another robot may take over the responsibilities of the broken one.
- Similarly, obstacles or people blocking a route do not interrupt production since they may be easily circumvented by a robot.

**Fig. 1** Loading of a transport robot

– Finally, workload may be optimized since robots responsible for serving certain machines which are currently idle may autonomously move to support the transportation of items from other machines which are more productive.

## 4 Analyzing smart production factories using strategic logics

Figure 2 depicts our approach for the use-case presented above. It is basically an adapted version of the general approach presented in Sect. 2 with one addition: in order to support the creation of CGSs for different factory settings, we developed a language for the specification of factory setups and a corresponding tool[1] to map a specification to a corresponding CGS. We then specified different variants of the use-case using our language and generated corresponding CGSs using our tool. In the next step, we specified goals for the use-case in terms of ATL formulæ over the CGS. Finally, we analyzed them using a strategic model checker, based on the algorithm explained in [21].

### 4.1 Specifying smart production factories

As mentioned above, we developed a language to support the specification of different factory settings. Our language can be used to specify factory layouts and settings for the machines.

[1] The tool is available online and can be downloaded at https://github.com/blackbat13/ATLFormulaChecker.

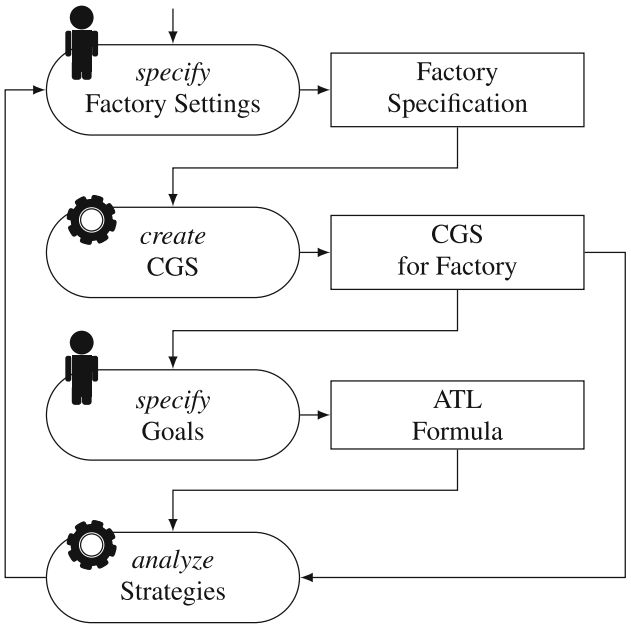


**Fig. 2** Verification approach showing manual (stick-figure) and automated (gear-wheel) activities (rounded rectangles) and corresponding artifacts (rectangles)

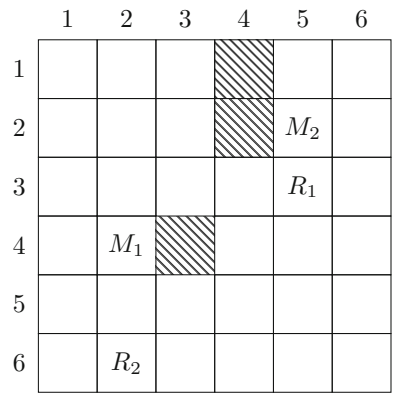


**Fig. 3** Conceptual representation of factory map with three obstacles (dashed parcels), two machines $M_1$ and $M_2$, and two robots $R_1$ and $R_2$

#### 4.1.1 Factory layout

A factory layout describes the *size* of a factory in terms of a matrix of floor-parcels. Moreover, it describes the number and position of static *obstacles* and *machines*. Finally, it also provides the number and initial position of available *robots*.

***Example 1*** (Simple Factory Layout) Figure 3 depicts an example of a factory map consisting of $6 * 6 = 36$ parcels. Obstacles are present at parcel $(3, 4)$, $(4, 1)$, and $(4, 2)$. Two machines are placed at parcels $(2, 4)$ (machine $M_1$) and $(5, 2)$ (machine $M_2$), respectively. In addition, two robots are placed at parcels $(2, 6)$ (robot $R_2$) and $(5, 3)$ (robot $R_1$).

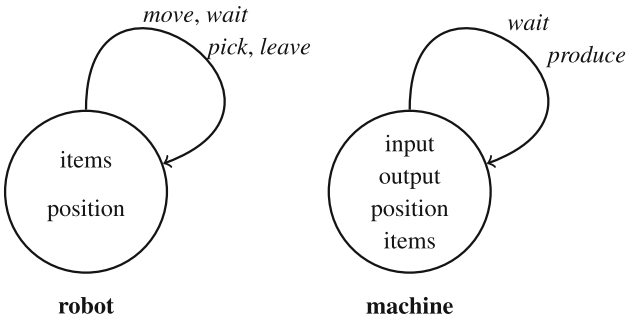


**Fig. 4** CGS for robots and machines

#### 4.1.2 Machine requirements

In addition to the layout of the factory, production requirements need to be specified for machines. To this end, our language allows to specify how many items of a certain type are required by a machine to produce a new item. Thereby, the type of an item is modeled by the identifier of a machine able to produce that item.

***Example 2*** (Production requirements for machine $M2$) We can add production requirements for machine $M2$ of the layout created in Example 1. With

$$[M1 \mapsto 1 * M2],$$

for example, we specify that machine $M1$ requires one item from machine $M2$, in order to produce an item itself. □

In addition to required items, one needs to provide a maximal number of produced items per machine. This is required in order to limit the size of the produced model, since unbounded production would lead to an infinite CGS.

### 4.2 From the specification to the model

A factory setup can be used to generate a corresponding CGS which, in turn, can be used for analyses. A CGS produced from a factory specification has two types of agents: *robots* and *machines*. Figure 4 depicts the state and possible actions for both types of agents.

#### 4.2.1 Robots

A robot's overall state is determined by the following variables:

- A pair of integers, representing the coordinate of the robot's current *position* within the factory.
- An integer representing the *type of item* currently carried by the robot. The item type is represented by the identifier of the machine able to produce this type of item. For example: a situation in which the robot currently carries an item produced by machine 1 is modeled with a value "1". A situation in which a robot does not carry any item is modeled by "−1".

At any point in time, a robot may perform one of the following actions:

- It may *wait* or *move* one parcel to the south, north, west, or east. Moving will change a robot's position and it is restricted by the factory layout: A robot can only move inside a factory and has to avoid obstacles. Moreover, collision of robots is possible and occurs whenever two robots enter the same field. After collision, the corresponding robots cannot move again.
- If the robot is located at a parcel containing a machine, it may *pick* an item from, or *leave* it to the corresponding machine. Picking or leaving items changes the carrying state of the robot.

#### 4.2.2 Machines

A machines state, on the other hand, is determined by four different variables:

- Similar to a robot, also a machine is located at a *position* within the factory which is modeled by a pair of integers.
- In addition, a machine's state is also determined by the number of *input* items of each type, available to the machine. This is modeled by a list of integers, with one entry for each type of item (again, the type is determined by the identifier of a machine able to produce this item). The first entry, for example, determines the number of items required from machine 1. The second entry determines the number of items required from machine 2 etc.
- Also the *output* produced by a machine influences its state. It is modeled by an integer, representing the amount of items waiting for pick-up by a robot.
- Finally, variable *items* determines the total number of items produced by the machine, so far. In contrast to the output variable, the *items* variable does not represent the number of items waiting for pickup but rather the total amount of items currently produced by the machine. The variable is introduced to limit the size of the produced model.

A machine may perform one of two actions:

- It may *wait* for items it requires for the production of a new item.
- If all the required items are available, it may *produce* a new item. Production of an item reduces a machine's input state and increases its output state and item count.

***Example 3*** (CGS for simple factory) The CGS produced for the example factory setting specified in Examples 1 and 2 consists of 3581 different states in total. Figure 5 depicts a small fragment of the overall CGS. It shows how the different robots and machines are encoded into the model:

| | |
|---|---|
| r_pos | models the position of each robot. |
| m_pos | models the position of each machine. |
| r_items | models the carried items for each robot. |
| m_in | models the required items from each machine for each robot. |
| m_out | models the produced items by each machine. |
| it_count | models the number of produced items by each machine. |
| pr_times | models the number of steps required by each machine to produce a new item. |
| m_clocks | models the elapsed time since a machine started producing a new item. |
| prop_stuck | marks that a machine was stuck at some point in time, i.e., that it could not produce any item due to a full output. |

The state at the bottom of Fig. 5, for example, is a state in which one machine is located at parcel (1, 3) and another one at (4, 1). Both machines have an empty input and the second machine has one item in its output waiting for pickup. The robots, on the other hand, are located at parcels (1, 3) and (4, 1), respectively. None of them currently carries an item. In total, one of the machines produced one item, so far. From this state, the first robot may move one parcel to the north, causing the state to change to the upper right one. □

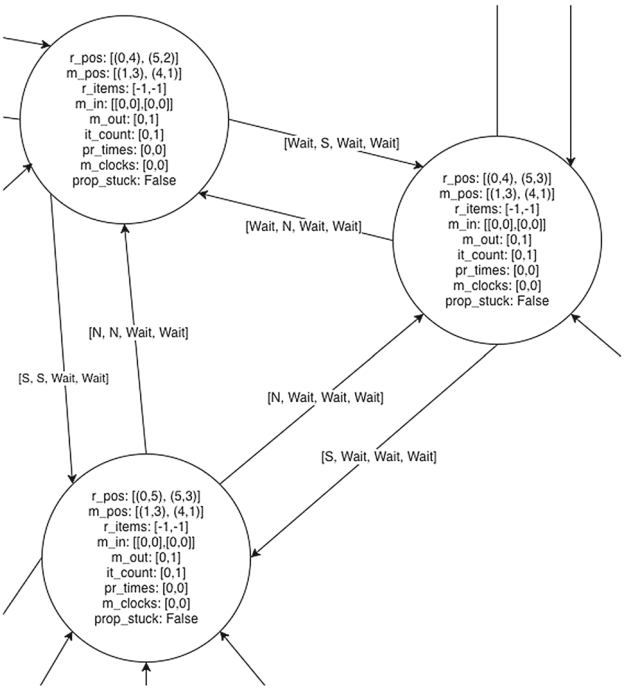


**Fig. 5** Fragment of the CGS produced for the example factory setting specified in Examples 1 and 2

## 4.3 Verification

Defining the model and its specification is only part of the work. Once the CGS is created and desired properties are known, verification can be done. As previously mentioned, we specify our goals in ATL, both with perfect and imperfect information and imperfect recall. Then we run model checking algorithms to check whether or not a given formula holds over the model. We show how it works on a simple example. First we begin with specifying the configuration we would like to verify. We will use the factory setting as specified in Examples 1 and 2. In the next step we generate the model in terms of a corresponding CGS. Once the model is generated, we need to provide a formula for the verification, such as $\langle\langle R\rangle\rangle F\ produce_1$. The latter formula can be interpreted as follows: a coalition of robots $R$ has a strategy to enforce that eventually each machine will produce at least one item. Given the formula and the model we can run model checking algorithms to check whether the formula holds over the model. The algorithm itself is a fixpoint computation that works as follows: first, we begin with all the states satisfying the desired property (winning states), i.e., states where each machine has produced at least one item. Then we look for states from which we can reach our current states in one step. In other words, we look for states where the coalition $R$ has a strategy to enforce moving from this state to one of the winning states. Then we add these states to our set (mark them as the winning states) and repeat the process, until a fixpoint is reached. If the final set of states contains the beginning state of the model, then the result of the formula is true, otherwise it is false. An overview of the verification results obtained for our use-case is presented in Sect. 5.

## 4.4 Variants

To investigate different variants of the use-case, we extended the basic model which resulted in three different extensions of the model presented so far.

### 4.4.1 Charging for robots

In the basic version of our model we do not consider energy for robots. Rather, we assume infinite energy for each robot. In order to investigate scenarios in which a robot may indeed run out of battery we extended our model with a notion of energy consumption for robots. Thereby, the specification of factory layouts may contain number and position of charging stations as well as the maximal amount of energy for

each robot. The generated model is in a way that it requires one energy consumption for each movement of a robot. Moreover, robots are charged immediately after arriving at a charging station and performing action "charge".

#### 4.4.2 Storage areas

In some settings, a factory may provide storage areas for items. To cover such cases, we provide an additional variant of the basic model. Here, a user may specify the number and position of storage areas. In the generated model, storage areas can be used to store items and thus avoid machines to be stuck due to full output.

#### 4.4.3 Production time

In the basic model, production of an item occurs instantaneous. However, it could be interesting to investigate also the impact of production time on service strategies. To this end, we provide an additional extension of the basic model in which we can provide production times for each machine. The production of an item then requires the provided amount of time.

## 5 Results

In previous sections we have defined the use case, its configuration and possible variants thereof. We have also introduced the language to model it and shown how one can verify a given property of such a model. In this section, we present the outcome of the analysis of different factory settings. To this end, we first introduce different factory configurations and some properties we would like to analyze. Then, we describe the outcome of the analysis for each of the properties.

### 5.1 Factory configurations

Different variants of the model may result in different number of states. For example adding storage areas to the factory layout changes the actual size of the model and may have an impact on the verification times. In order to have a good understanding of how such changes may affect the experiments, we present some details regarding the generation process of these different models in Table 1. Headers should be interpreted as follows:

- En.—initial energy of the robots; inf means that robots don't use energy
- # Ch.—number of charging stations in the factory
- # Stor.—number of storage areas in the factory
- Pr. t.—production time for the machines
- It. l.—item limit for the machines, i.e. how many items can any machine produce
- # St.—number of states in the generated model
- Gen.—generation time of the model in seconds

**Table 1** Model generation results

| En. | # Ch. | # Stor. | Pr. t. | It. l. | $\#St.$ | $Gen.$ |
|---|---|---|---|---|---|---|
| $inf$ | 0 | 0 | 0 | 1 | 3581 | 0.76 |
| $inf$ | 0 | 0 | 1 | 1 | 4161 | 0.83 |
| $inf$ | 0 | 0 | 0 | 2 | 13,912 | 3.07 |
| 10 | 0 | 0 | 0 | 1 | 28,667 | 6.23 |
| 10 | 1 | 0 | 0 | 1 | 48,426 | 10.65 |
| $inf$ | 0 | 1 | 0 | 1 | 4670 | 1.32 |

As the results show, the modification that drastically affects the size of the model is adding energy to the robots. For example lets take a look on the first, simplest configuration: no energy limit, no charging stations and storage areas, no time requirements for productions and production limited to one item per machine. Such a simple model consist of 3581 different states. If we change this configuration by only specifying initial energy for robots of 10 units, the number of states in the generated model goes up to 28, 667 (this corresponds to an increase in size by a factor of 8). If we also add one charging station to the model, thus allowing robots to recharge their energy, we end up with 48,426 states, which is 13 times more than in the initial, simplest configuration.

### 5.2 Properties

We considered four different properties that are important for the safe functioning of the presented factory scenario. Formulas representing these properties are as follows:

- $\phi_1 : \langle\langle R \rangle\rangle F(produce_n)$
- $\phi_2 : \langle\langle R \rangle\rangle F(\neg stuck \wedge produce_n)$
- $\phi_3 : \langle\langle R \rangle\rangle F(energy > 0 \wedge produce_n)$
- $\phi_4 : \neg\langle\langle R' \rangle\rangle G(\neg produce_n)$

where:

- $produce_n$ denotes that each machine has produced at least n items, and
- $stuck$ denotes that a machine is stuck, i.e., that its input requirements are met, but its output isn't empty.

The first three of the presented formulas describe the properly functioning of the factory. $\phi_1$ means that when robots work together they can make sure that each machine will produce the required number of items. As one can see this

is a rather basic property, but it's a necessary condition for the properly functioning of the factory. Still this may be not enough to describe an efficiently operating factory. That we try to achieve using formula $\phi_2$, where robots not only need to make each machine produce the required number of items, but they also need to make sure that each machine will produce a new item when all required items are delivered. This way we know that a factory setting is efficient, i.e., there is no redundant waiting time for machines. The third formula $\phi_3$ describes configurations in which no robot will be left with zero energy after executing its commands.

The last formula $\phi_4$, unlike the others, describes a security property of the model. Imagine a possible scenario: someone wants to disturb the work in the factory by hijacking, or maybe just disabling some of the robots. The question is, how secure is the factory against such an attack? That is what is the minimal subset of the robots that the adversary must take control of, in order to disturb the factory production line?

### 5.3 Basic production

In this section we consider formula $\phi_1 : \langle\langle R\rangle\rangle F(produce_n)$. The goal for the coalition of robots is to reach a state, in which each machine has produced at least $n$ items. We will test this property in different configurations of the model. First, we begin with a classic model based on a factory layout as presented in Fig. 3. The model consist of two robots and two machines. Robots can move freely, apart from places marked as an obstacle. Robots don't consume energy and can carry at most one item—there is no storage in the factory. Requirements for machines are simple: the first machine doesn't need anything to produce an item and the second machine needs one item from the first machine. By manipulating the parameters of the model we can change: items limit, machines requirements and production times for each machine. We can also modify parameter $n$ of the formula. As the experiments show, the formula always holds, both under perfect and imperfect information, except when there is a conflict in the parameters. A conflict occurs when, for example, machine requirements are constructed in such a way, that at least one of the machines will be always blocked. Results are shown in Table 2. For simplicity, machines are named A and B. The configuration should be interpreted as follows: the first parameter defines the limit for the production, and the second one defines production times for machines. For simplicity, if production times for both machines are 0, we will write only 0 as the second parameter. If a machine requires some time to produce an item, we will write *machine* : *number*, where machine is the letter representing the machine and number is production time for this machine. If some machines are ommitted we assume that production time for them is by default 0.

The results should be interpreted as follows:

**Table 2** Results for model checking $\phi_1$

| Conf. | #*states* | IR t. | IR | iR t. | iR (appr.) |
|---|---|---|---|---|---|
| (1, 0) | 3581 | 1.1 | True | 0.3 | True |
| (3, 0) | 26,039 | 2.4 | True | 1.1 | True |
| (3, A: 5) | 72,573 | 8.3 | True | 5.7 | True |

- conf.—configuration, as explained above
- # states—number of states in the generated model
- IR t.—perfect information verification time in seconds
- IR—result under perfect information
- iR t.—imperfect information verification time in seconds
- iR (appr.)—approximated result for the imperfect information

Of course, in such a simple environment, results of the experiments are as expected. Perhaps one can get more interesting scenarios by specifying finite charges for the robots. In this version of the model, apart from the previous parameters, we can also modify the number of charging stations, their positions and initial charge of the robots. Again, we conducted some experiments, using previous configurations with additional parameters. First we begun with a simpler scenario, i.e., a factory without charging stations. In such an environment one can already suspect the result of the formula. The strategy for robots seems to be simple: find shortest path to fulfill machine requirements. Of course, there are some obstacles along the way, such as avoiding collisions.

While conducting experiments it is possible to find the minimal initial charge for the robots in a given configuration, under which the formula is satisfied. There can be some differences based on the considered type of information. Under imperfect information, robots may require more energy than under perfect information. We can also manipulate positions of the charging station to find optimal positioning.

### 5.4 No stuck time

As said before, proper functioning of the factory is sometimes not enough. Sometimes we need to ensure that a considered configuration is not only stable, but also efficient. After the previous section, we know under which configurations the system is stable, i.e. robots can enforce production of n items from each machine. Now, we want to know when the system is efficient, i.e. each machine can produce a new item immediately after its requirements are fulfilled (when it receive all items needed for production from the robots). In other words, we will call a configuration efficient, if a coalition of robots has a strategy to ensure that each machine will produce the required number of items and while doing so, no machine will be stuck. Our stuck property in the model is sticky—once a machine is stuck it is stuck forever. We took the configura-

**Table 3** Results for model checking $\phi_2$

| Conf. | $\#states$ | $IRt.$ | IR | $iRt.$ | iR (appr.) |
|---|---|---|---|---|---|
| (1, 0) | 3581 | 0.92 | True | 0.18 | True |
| (3, 0) | 26,039 | 0.002 | False | 0.002 | False |
| (3, A: 5) | 72,573 | 0.005 | False | 0.005 | False |
| (3, A: 5, st: 1) | 198,762 | 10.5 | True | 8.7 | True |

**Table 4** Results for model checking $\phi_3$

| Conf. | $\#states$ | $IRt.$ | IR | $iRt.$ | iR (appr.) |
|---|---|---|---|---|---|
| (1, 0, en: 5, ch: 0) | 2681 | 0.0002 | False | 0.0001 | False |
| (1, 0, en: 9, ch: 0) | 20,439 | 0.6 | True | 0.05 | True |
| (1, 0, en: 5, ch: 1) | 5660 | 0.0004 | False | 0.0003 | False |
| (1, 0, en: 6, ch: 1) | 16,489 | 1.3 | True | 0.3 | True |

tions from the previous section and run our model-checking algorithm again, this time with formula $\phi_2$. Results are shown in Table 3.

### 5.5 No robot left behind

In order to talk about energy level we need a model with energy. When checking formula $\phi_3$ we will only consider models with energy. Based on its initial level, robots may or may not be able to fulfill their duties. When thinking about the property described by formula $\phi_3$, some questions arise: What is the minimal level of the initial energy required for each robot? What is the optimal positioning of the charging stations and how many are required? Does storage affect this property? How does imperfect information affect this property? How do production times of the machines affect this property? We try to answer these questions by conducting several experiments on different configurations. First, lets describe basic requirements for our scenario. As previously, we consider a $6 \times 6$ factory layout with two machines and two robots. We will consider two configurations of the production times for the machines: [A:0, B:0] and [A:2, B:3]. This way, we can see how production times will affect our results. As for the number of items we want to produce, we choose 1 and 2. As shown before, the number of states grows fast, when we add energy to the robots, so it is only sensible to limit the size of the model by using low limits for production. When combined together, our requirements give us four different configurations and we try to find optimal configurations for the rest of the parameters: initial energy, charging stations and storage areas. The results are summarized in Table 4.

### 5.6 Security property

There are various ways to describe security properties. In our factory scenario, the level of security can be described by the minimal number of robots that an adversary needs to take control of, in order to be able to disturb the work of the factory. For example, lets say that we have a factory with 10 robots. Rendering 3 of them malfunctioning won't affect the system, but if the adversary takes control of the fourth robot, then the rest of them won't be able to ensure proper functioning of the factory anymore. In this situation the minimal number of robots needed to disturb the production is 4 out of 10. Of course, the result may vary depending on the factory configuration, layout, and even on the concrete robots that are hacked. Work of some of the robots may be more important to ensure proper production, than the work of other robots. Table 5 show results of the experiments conducted on a smaller, $4 \times 4$ factory. First column (#robots) contains number of robots in the tested configuration. Next columns shows minimum number of robots that need to be hacked in order to disrupt the work of the factory in the different variants of the model: without energy and storage, with energy and charging stations, with storage areas.

**Table 5** Results for model checking $\phi_4$

| #robots | Normal | Charging | Storage |
|---|---|---|---|
| 2 | 1 | 1 | 1 |
| 3 | 2 | 1 | 2 |

## 6 Discussion

In the following, we reflect on our experience in using strategic logics for the analysis of CSGs. To this end, we discuss promising observations as well as possible limitations of the approach. Based on our observation, we then provide some suggestions for the use of our approach.

### 6.1 Advantages and limitations

In general, strategic logics are well-suited for the analysis of situations in which agents collaborate to achieve common goals. Since such situations are common in the context of CSGs, strategic logic seems well-suited to support the analysis of such system groups. This suspicion is indeed also confirmed by our results, in which we were able to verify a set of interesting properties for a special type of CSG.

As with every automatic verification approach, model checking reaches its limit when the models become too large. In our experiments we were unable to test models with more than 4 robots and 2 machines, due to the memout caused by the space explosion.

### 6.2 Suggestions

Based on our lessons learned, we provide some suggestions for using strategic model checking for the analysis of CSGs.

First, strategic model checking seems really promising to investigate the impact of different setups on a group's ability to achieve its goals. To this end, one can specify a goal of a CSG, adapt the settings and investigate whether the goal is still feasible. In our use case, for example, we applied strategic model checking to find the minimal level of initial energy of robots which is necessary for them to achieve their goal. In another example, we used the approach to investigate the minimal number of robots required to achieve a certain goal.

An interesting supplement in using strategic logic is its ability to deal with imperfect information. This is interesting, from the point of view of CSGs, since it allows to analyze the effect of communication problems within the group. In our use case, for example, there were situations in which a group of robots could indeed serve all requests of machines, as long as communication was working. However, for the case in which communication was not properly working, the robots where not able to achieve their goal anymore.

## 7 Related work

Collaborative embedded systems can be seen as a special case of a broader metaphor, namely that of agents in a multi-agent system [36,39,40]. A *multi-agent system* (*MAS*) is a system that involves several autonomous entities that act in the same environment. Agents in a MAS are usually software entities, as opposed to CES where the actors are often physical artifacts supervised by an intelligent controller. On the other hand, most models of multi-agent systems are general enough to cover systems involving both virtual and physical components. The most prominent applications of the MAS framework in the multi-robot context focus on market mechanisms for multi-robot coordination [12,31] and in particular on auction-based job allocation and scheduling [24,25]. A preliminary analysis of logistic robots from the MAS point of view was presented recently in [33].

Many relevant properties of multi-agent systems refer to *strategic abilities* of agents and their groups. Properties of this kind can be conveniently specified in *modal logics of strategic ability*, of which alternating-time temporal logic (ATL) [4,5] is probably the most popular. Typical contributions include results concerning the conceptual soundness of a given semantics [1,2,13,16,20,35], meta-logical properties [8,17], and the complexity of model checking [6, 14,17,35,37]. From the practical point of view, a number of model checking tools have been created or extended to accept models of multi-agent systems as input – most notably *Mocha* [3], *MCK* [15,38], and *MCMAS* [26]. A comprehensive overview can be found in [19]. Moreover, the last couple of years have witnessed a growing number of serious attempts at overcoming the complexity of verification for ATL in more sophisticated scenarios [9,10,18,23,29], and even actual attempts at model checking of a concrete voting procedure [22]. All of this suggests that the time is ripe to use both multi-agent models and multi-agent logics to enhance the design, specification, and verification of collaborative embedded systems.

## 8 Conclusion

In this paper, we presented an approach for the verification of collaborative system groups (CSGs) using strategic logics. We present a case study in the area of smart production factories and evaluated the approach by using it to analyze such systems. To this end, we developed a simple language for the specification of smart production factories and a corresponding algorithm to map such specifications to a corresponding concurrent game structure (CGS). Then, we investigated different properties for such systems and analyzed them by means of model checking over the generated CGS.

Our results suggest that strategic logics are well suited to analyze important aspects of CSGs, such as their ability to achieve certain collaboration goals under different circumstances. On the other hand, the approach seems less suited for the analysis of implementation aspects such as performance.

Our approach may be used to complement existing verification approaches for CSGs which are based on traditional model checking techniques. To this end, future work should investigate further case studies in different CSG domains, such as fleets of self-driving cars.

**Acknowledgements** The authors acknowledge the support of the National Centre for Research and Development (NCBR), Poland, under the PolLux Project VoteVerif (POLLUX-IV/1/2016). We also thank Wojtek Jamroga for assistance with the related work section.

## References

1. Ågotnes T (2004) A note on syntactic characterization of incomplete information in ATEL. In: Proceedings of workshop on knowledge and games, pp 34–42
2. Ågotnes T, Goranko V, Jamroga W, Wooldridge M (2015) Knowledge and ability. In: van Ditmarsch H, Halpern J, van der Hoek W, Kooi B (eds) Handbook of epistemic logic. College Publications, New York
3. Alur R, de Alfaro L, Grossu R, Henzinger T, Kang M, Kirsch C, Majumdar R, Mang F, Wang BY (2001) jMocha: a model-checking tool that exploits design structure. In: Proceedings of international conference on software engineering (ICSE), pp 835–836. IEEE Computer Society Press
4. Alur R, Henzinger TA, Kupferman O (1997) Alternating-time temporal logic. In: Proceedings of the 38th annual symposium on

foundations of computer science (FOCS), pp 100–109. IEEE Computer Society Press
5. Alur R, Henzinger TA, Kupferman O (2002) Alternating-time temporal logic. J ACM 49:672–713. https://doi.org/10.1145/585265.585270
6. Bulling N, Dix J, Jamroga W (2010) Model checking logics of strategic ability: complexity. In: Dastani M, Hindriks K, Meyer JJ (eds) Specification and verification of multi-agent systems. Springer, Berlin
7. Bulling N, Goranko V, Jamroga W (2015) Logics for reasoning about strategic abilities in multi-player games. In: van Benthem J, Ghosh S, Verbrugge R (eds) Models of strategic reasoning. Springer, Berlin, pp 93–136
8. Bulling N, Jamroga W (2014) Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. J Auton Agents Multi Agent Syst 28(3):474–518
9. Busard S, Pecheur C, Qu H, Raimondi F (2014) Improving the model checking of strategies under partial observability and fairness constraints. In: Formal methods and software engineering, lecture notes in computer science, vol 8829, pp 27–42. Springer. https://doi.org/10.1007/978-3-319-11737-9_3
10. Busard S, Pecheur C, Qu H, Raimondi F (2015) Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. Inf Comput 242:128–156. https://doi.org/10.1016/j.ic.2015.03.014
11. Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8(2):244–263. https://doi.org/10.1145/5397.5399
12. Dias M, Zlot R, Kalra N, Stentz A (2006) Market-based multirobot coordination: a survey and analysis. Proc IEEE 94(7):1257–1270. https://doi.org/10.1109/JPROC.2006.876939
13. Dima C, Enea C, Guelev D (2010) Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In: Proceedings of games, automata, logics and formal verification (GandALF), pp 103–117
14. Dima C, Tiplea F (2011) Model-checking ATL under imperfect information and perfect recall semantics is undecidable. CoRR abs/1102.4225
15. Gammie P, Meyden R (2004) MCK model checking the logic of knowledge. In: Proceedings of the 16th international conference on computer aided verification (CAV'04), LNCS, vol 3114, pp 479–483. Springer
16. Guelev D, Dima C (2012) Epistemic ATL with perfect recall, past and strategy contexts. In: Proceedings of computational logic in multi-agent systems (CLIMA), lecture notes in computer science, vol 7486, pp 77–93. Springer. https://doi.org/10.1007/978-3-642-32897-8_7
17. Guelev D, Dima C, Enea C (2011) An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking. J Appl Non Class Logics 21(1):93–131
18. Huang X, van der Meyden R (2014) Symbolic model checking epistemic strategy logic. In: Proceedings of AAAI conference on artificial intelligence, pp 1426–1432
19. Jamroga W (2015) Logical methods for specification and verification of multi-agent systems. ICS PAS Publishing House, Manila
20. Jamroga W, van der Hoek W (2004) Agents that know how to play. Fundam Inform 63(2–3):185–219
21. Jamroga W, Knapik M, Kurpiewski D(2017) Fixpoint approximation of strategic abilities under imperfect information. In: Proceedings of the 16th international conference on autonomous agents and multiagent systems (AAMAS), pp 1241–1249. IFAAMAS
22. Jamroga W, Knapik M, Kurpiewski D (2018) Model checking the selene e-voting protocol in multi-agent logics. In: Proceedings of the 3rd international joint conference on electronic voting (E-VOTE-ID), lecture notes in computer science, Springer. To appear
23. Jamroga W, Knapik M, Kurpiewski D, Mikulski Ł (2018) Approximate verification of strategic abilities under imperfect information. Artificial intelligence, To appear
24. Koenig S, Keskinocak P, Tovey CA (2010) Progress on agent coordination with cooperative auctions. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010
25. Lagoudakis M, Markakis V, Kempe D, Keskinocak P, Koenig S, Kleywegt A, Tovey C, Meyerson A, Jain S (2005) Auction-based multi-robot routing. In: Proceedings of the international conference on robotics: science and systems, pp 343–350
26. Lomuscio A, Qu H, Raimondi F (2015) MCMAS: an open-source model checker for the verification of multi-agent systems. Int J Softw Tools Technol Transf. https://doi.org/10.1007/s10009-015-0378-x Availabe online
27. Manyika J, Chui M, Bisson P, Woetzel J, Dobbs R, Bughin J, Aharon D (2015) The internet of things: mapping the value beyond the hype. https://web.archive.org/web/20180726101247/https://www.mckinsey.com/~/media/McKinsey/Business
28. Motavalli J (2010) The dozens of computers that make modern cars go (and stop). https://web.archive.org/web/20180726104858/https://www.nytimes.com/2010/02/05/technology/05electronics.html
29. Pilecki J, Bednarczyk M, Jamroga W (2014) Synthesis and verification of uniform strategies for multi-agent systems. In: Proceedings of CLIMA XV, lecture notes in computer science, vol 8624, pp 166–182. Springer
30. Rajkumar R, Lee I, Sha L, Stankovic J (2010) Cyber–physical systems: the next computing revolution. In: Design automation conference (DAC), 2010 47th ACM/IEEE, pp 731–736. IEEE
31. Sandholm TW (1999) Distributed rational decision making. In: Weiss G (ed) Multiagent systems: a modern approach to distributed artificial intelligence. The MIT Press, Cambridge, pp 201–258
32. Schlingloff B (2018) Specification and verification of collaborative transport robots. In: 4th international workshop on emerging ideas and trends in the engineering of cyber–physical systems, EITEC@CPSWeek 2018, 10 April 2018, Porto, Portugal, pp 3–8. IEEE Computer Society. https://doi.org/10.1109/EITEC.2018.00006
33. Schlingloff B, Stubert H, Jamroga W (2016) Collaborative embedded systems—a case study. In: 3rd international workshop on emerging ideas and trends in engineering of cyber–physical systems, EITEC@CPSWeek, pp 17–22. https://doi.org/10.1109/EITEC.2016.7503691
34. Schlingloff BH, Stubert H, Jamroga W (2016) Collaborative embedded systems—a case study. In: 2016 3rd international workshop on Emerging ideas and trends in engineering of cyber–physical systems (EITEC), pp 17–22. IEEE
35. Schobbens PY (2004) Alternating-time logic with imperfect recall. Electron Notes Theor Computer Sci 85(2):82–93
36. Shoham Y, Leyton-Brown K (2009) Multiagent systems—algorithmic, game-theoretic, and logical foundations. Cambridge University Press, Cambridge
37. van der Hoek W, Lomuscio A, Wooldridge M (2006) On the complexity of practical ATL model checking. In: Proceedings of international joint conference on autonomous agents and multiagent systems (AAMAS), pp 201–208. ACM
38. van der Meyden R (2017) Optimizing epistemic model checking using conditional independence. In: Proceedings of theoretical aspects of rationality and knowledge, pp 398–414
39. Weiss G (ed) (1999) Multiagent systems. A modern approach to distributed artificial intelligence. MIT Press, Cambridge
40. Wooldridge M (2002) An introduction to multi agent systems. Wiley, Amsterdam

**Damian Kurpiewski** is a researcher at the Polish Academy of Sciences. His research interests include modelling, specification and verification of strategic interaction between agents in various systems. Currently he is in the middle of his PhD under the supervision of Prof. Jamroga, with the focus on logic-based verification of voting protocols. Mr. Kurpiewski has played a vital role in recent algorithmic advances for model checking of alternating-time logic with imperfect information. He also leads the development of the STV model checker.

**Diego Marmsoler** is a postdoctoral researcher at the Software and Systems group of Prof. Manfred Broy at the Technical University of Munich. He obtained a B.Sc. from the Free University of Bozen-Bolzano and an M.Sc. from the Technical University of Munich, Ludwig Maximilian University of Munich, and Augsburg University. He received a Ph.D. in Computer Science from the Technical University of Munich in 2019. His research focuses on the formal specification and verification of distributed, component-based systems. In particular, he works on the integration of various formal methods for the verification of such systems.

# Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com