Synthesizing strategies for homogenous multi-agent systems with incomplete information

Jan Calta[1], Dmitry Shkatov[2], and Holger Schlingloff[1]

[1] Humboldt University, Berlin, Germany
[calta|hs]@informatik.hu-berlin.de
[2] University of the Witwatersrand, Johannesburg, South Africa
dmitry@cs.wits.ac.za

Abstract. We present an algorithm for synthesizing strategies for multi-agent systems composed of homogeneous agents possessing incomplete information about the system as a whole. The algorithm finds all maximal strategies for such agents that enforce a certain property of the system. In contrast to other algorithms known from the literature, our algorithm can be used for automated program synthesis for systems in which agents are required to be homogeneous (i.e., every agent has to follow the same strategy), which is a more restrictive setting.

1 Introduction

Multi-agent systems have, over the last decade, emerged as an active research area on the borderline between game theory, logic, computer science, and artificial intelligence. The two most common computational tasks associated with multi-agent systems are checking if the system conforms to a given specification (technically, this is a model-checking problem for multi-agent systems) and synthesizing strategies for agents that enforce a certain property (technically, this is a synthesis problem). In the present paper, we are concerned with the latter problem.

In the case of systems where agents are heterogeneous—that is, are not required to behave in the same way—every strategy for a group of agents is a set of strategies for individual agents (an agent's strategy is a set of "rules" prescribing the agent how to act given particular circumstances) so that, when agents follow their respective strategies, they bring about a desired outcome. In the case, considered here, of systems where agents are homogeneous, a strategy for a group of agents is a bunch of identical individual strategies since all agents are required to follow the same rules; therefore, a single individual strategy automatically gives us a strategy for the group.

Another distinction commonly made when considering multi-agent systems is the one between those in which agents have complete information about the current state of the whole system and those in which agents have only partial—or, incomplete—information about the current state of the system. In the present paper, we consider the case of systems made up of agents possessing incomplete information.

As alluded to above, in this paper, we present an algorithm for automatically synthesizing strategies for multi-agent systems of homogeneous agents with incomplete information. In practice, such a synthesis allows us to automatically generate an algorithm for a system in which all software components must execute the same program and, moreover, the components have no access to the complete information about the system.

2 Formal model

In this section, we introduce the formal framework for the synthesis of the strategies; namely, structures used to model multi-agent systems and the language for expressing the desired system properties.

2.1 Modular Models

We use modular models (inspired by modular interpreted systems from [5]) to formally model homogeneous multi-agent systems with incomplete information and a fixed topology. More precisely, in such systems,

- all agents have the same set of local states and the same set of available actions;
- each agent has (complete) information about its local state and a limited information about the states of its "neighbors"; thus, an agent may not be able to distinguish between separate system states (a system state is a tuple of local states, one for each agent);
- the neighborhood relation does not change during the runtime (the connections between the agents are fixed throughout the computation).

An example of such system is a multi-core processor, where agents are the cores, agents' actions are processor instructions, the local states of the cores are given by the content of their registers, and the topology of the system is a grid.

The above assumptions allow us to model an agent as a *module*. Since our agents are homogenous, modules for all agents are identical. Thus, the entire system can be represented by a single agent module and the topology of the system. The use of modular models allows us to possibly save a considerable amount of storage space in comparison to the traditional model — a transition system built upon the global statespace of the entire system (such models are referred to in the literature as "concurrent epistemic game structures").

Definition 1. *A modular model* $\mathfrak{M}$ *is a tuple*

$$\mathfrak{M} = \langle Agt, Act, St, \Pi, \pi, neig, k, \Sigma, man, tran \rangle, \text{ where}$$

- $Agt = \{1, \ldots, n\}$ *is a set of agents;*
- *Act is a finite nonempty set of actions; arbitrary actions will be denoted by lower-case Greek letters from the beginning of the alphabet, such as* $\alpha, \beta, \ldots$*;*

- *St is a finite nonempty set of agent's states (the set of system states is then St^n); we denote the members of St using lower-case Latin letters such as q and members of St^n using upper-case Latin letters such as Q;*
- $\Pi = \Pi_1 \cup \cdots \cup \Pi_n$ *is the union of sets of atomic propositions, one for each agent,*
- $\pi : \Pi \to \mathcal{P}(St)$ *is a valuation function,*
- k *is the maximal number of neighbors an agent can have,*
- $neig : Agt \times \{1, \ldots, k\} \to Agt \cup \{\sharp\}$ *is a neighborhood function (thus, if $a \in Agt$ and $1 \leq i \leq k$, then $neig(a, i)$ is the i^{th} neighbor of a; $neig(a, i) = \sharp$ means that a does not have the i^{th} neighbor),*
- Σ *is a finite nonempty set of manifestation symbols,*
- $man : St \to \Sigma$ *is a manifestation function; for technical reasons, $man(Q[\sharp]) = \lambda$,*[3] *where $\lambda \in \Sigma$ is a manifestation of an "absent" neighbor;*
- $tran : St \times \Sigma^k \times Act \to St$ *is a transition function.*

In our example of the multi-core processor, there are four cores with identifiers 1 through 4 ($Agt = \{1, \ldots, 4\}$). Each of them uses two Boolean registers, `l` ("low") and `h` ("high"). The content of each register is represented by one atomic proposition ($\Pi_a = \{\mathtt{l_a}, \mathtt{h_a}\}$ for every $a \in Agt$). Thus, each core has four possible states: $St = \{q_0, q_1, q_2, q_3\}$ and $\pi(\mathtt{l_a}) = \{q_1, q_3\}, \pi(\mathtt{h_a}) = \{q_2, q_3\}$ for every $a \in Agt$ (i is the binary value of the combination of registers `hl` for every $q_i \in St$). The cores are arranged in square, so that $k = 2$ and $neigh(a, 1) = a - 1$ for $a \in \{2, 3, 4\}$, $neigh(a, 2) = a + 1$ for $a \in \{1, 2, 3\}$, $neigh(1, 1) = 4$ and $neigh(4, 2) = 1$. Each core makes his register `l` observable to its neighbors, that is, $\Sigma = \{0, 1\}$, $man(q_i) = 0$ for $i \in \{0, 2\}$ and $man(q_i) = 1$ for $i \in \{1, 3\}$. The agents are capable of two actions ($Act = \{nop, add\}$). Action *nop* has no effect on the state of any agent and action *add* adds to the binary value of the agent's registers the binary value of the observable registers of the agent's neighbors. The transition function defines the effects of the actions as follows:

$$\begin{aligned} &\text{For every } i \in \{0, \ldots, 3\} \text{ and every } \sigma_1, \sigma_2 \in \Sigma \\ &\qquad tran(q_i, \sigma_1, \sigma_2, nop) = q_i \text{ and} \\ &\qquad tran(q_i, \sigma_1, \sigma_2, add) = q_{i+\sigma_1\sigma_2 \bmod 4}\,. \end{aligned} \tag{1}$$

Manifestation function models the incomplete information available to the agents; if $man(q_1) = man(q_2)$ for two different states of agent a then these states can not be distinguished by a's neighbors and no other agent beside a's neighbors has any information about a's state. Notice that all of a's neighbors have the same partial view of a's state (this is part of our assumption that the system being modeled is homogenous).

In what follows, we refer to the combinations of agent's state and manifestations of her neighbors as her *perception*. Then we can think of the transition function as assigning a's state to a combination of a's perception and action (thus, given a's perception and her action, her state changes according to the transition function).

[3] Throughout the paper, we write $t[i]$ for the i^{th} element of a tuple t.

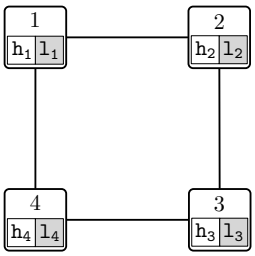


Fig. 1: A four-core processor with cores $1, \ldots, 4$, each of the cores has two boolean registers — `h` and `l` — where the value of `l` is observable by the neighboring cores.

2.2 Logical formalism

Over the recent years, the logic of choice for reasoning about multi-agent systems has been ATL, introduced in [1]. ATL has pretty elaborate syntax, as it allows to reason about strategic ability of coalitions of agents, which requires the use of the so-called coalition modalities. As all the agents in the systems we consider in this paper are under our control, and we can model the presence of environment as a choice of what systems states the system is in at the beginning of the computation, we do not need the full syntactic machinery of ATL. Moreover, as all our computations are meant to be deterministic, it is sufficient for our purposes to use the syntax of the Linear-Time Temporal Logic LTL. (On the other hand, the semantics we are going to propose will differ substantially from the standard LTL semantics, to reflect the particularities of the models we consider.) Our syntax reflects the fact that propositional symbols are evaluated at agents', not system, states:

$$\gamma ::= \top \mid \mathtt{p_a} \mid \neg \mathtt{p_a}, \text{where } a \in Agt \text{ and } \mathtt{p_a} \in \Pi_a \tag{2}$$

$$\varphi ::= \gamma \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{G}\varphi \mid \varphi \mathsf{U} \varphi, \text{ where } \gamma \text{ is defined as in (2).} \tag{3}$$

Such syntax only allows LTL formulas in positive normal form, that is, the ones where nothing but the statements about agent states can be negated. Every LTL formula can be translated into an equivalent formula in positive normal form.

Definition 2. *A* homogenous strategy *is a function* $S : St \times \Sigma^k \to Act$ *assigning actions to perceptions.*

For now, we only consider total functions. Following function S is one possible homogenous strategy for the model of our four-core processor:

$$\begin{aligned} &\text{For every } i \in \{0, \ldots, 3\} \text{ and } \sigma_1, \sigma_2 \in \Sigma \\ &\qquad S(q_i, \sigma_1\sigma_2) = add \text{ if } \sigma_1 = \sigma_2 \text{ and} \\ &\qquad S(q_i, \sigma_1\sigma_2) = nop \text{ if } \sigma_1 \neq \sigma_2. \end{aligned}$$

Definition 3. *The* perception function $per : St^n \times Agt \to St \times \Sigma^k$ *assigns to a system state and an agent her perception of that system state:* $per(Q, a) = \langle Q[a], man(Q[neig(a, 1)]), \ldots, man(Q[neig(a, k)])\rangle$ *where* Q *is a system state,* $a \in Agt$ *and* k *is the neighborhood size.*

For example, $per(\langle q_0, q_2, q_3, q_3\rangle, 3)$, that is, agent's 3 perception of system state $Q = \langle q_0, q_2, q_3, q_3\rangle$, would be $\langle q_3, 0, 1\rangle$, because the state of agent 3 is q_3, the manifestation of the state of its left neighbor (agent 2) is 0 and the manifestation of the state of its right neighbor (agent 4) is 1.

Definition 4. *An* action vector *is an* n*-tuple of actions from* Act*, one for each* $a \in Agt$*.*

Given system states Q and Q', we say that action vector A leads from Q to Q' if $tran(per(Q, a), A[a]) = Q'[a]$ for every $a \in Agt$.

Definition 5. *A* path *is an infinite sequence of system states* $\Lambda = Q_1, Q_2, Q_3 \ldots$ *that can be effected by subsequent action vectors; that is, for every* $j \geq 1$*, there exists an action vector* $A \in Act^n$ *leading from* Q_j *to* Q_{j+1}*. The* j^{th} *component of* Λ *is denoted by* $\Lambda[j]$*.*

Definition 6. *The* outcome *of a homogenous strategy* S *at system state* Q*, denoted by* $out(Q, S)$*, is the path* Λ *such that* $\Lambda[1] = Q$ *and*

$$\Lambda_{j+1} = \prod_{a \in Agt} tran(per(\Lambda_j, a), S(per(\Lambda_j, a))),$$

for every $j \geq 1$*.*

Given a set of system states $\mathbb{Q}$, we use notation $out(\mathbb{Q}, S)$ as a shorthand for $\bigcup_{Q \in \mathbb{Q}} out(Q, S)$.

Informally, given a modular model $\mathfrak{M}$ and a set of system states $\mathbb{Q}$, an LTL formula φ holds at $\mathbb{Q}$ if there exists a homogenous strategy S such that for every system state $Q \in \mathbb{Q}$ the outcome of S at Q satisfies φ. Formally:

$\mathfrak{M}, \mathbb{Q} \models \varphi$ iff there is a strategy S such that $\Lambda \Vdash_{\text{LTL}} \varphi$ for every $\Lambda \in out(\mathbb{Q}, S)$,

where $\Lambda \Vdash_{\text{LTL}} \varphi$ holds iff Λ satisfies φ according to the standard LTL semantics. From the semantics above it follows that,

- $\mathfrak{M}, \mathbb{Q} \models \mathtt{p_a}$ iff $Q[a] \in \pi(\mathtt{p_a})$ for every $Q \in \mathbb{Q}$
- $\mathfrak{M}, \mathbb{Q} \models \neg\mathtt{p_a}$ iff $Q[a] \notin \pi(\mathtt{p_a})$ for every $Q \in \mathbb{Q}$

where $a \in Agt$ and $\mathtt{p_a} \in \Pi_a$.

If the outcome of a strategy at some state satisfies a formula φ then we say that the strategy *enforces* φ. For a given strategy enforcing a formula φ we want to know the set of *all* system states at which the outcome of the strategy satisfies φ; we refer to this set as the *domain* of the strategy. This is the reason why we evaluate formulas at sets of system states rather than at states themselves. We took the idea of evaluating formulas at sets of states from [4].

3 Synthesis of All Maximal Homogenous Strategies

Our aim in this paper is, given a modular model $\mathfrak{M}$ and an LTL formula φ, to synthesize *all maximal homogenous strategies enforcing* φ, i.e., all homogenous strategies enforcing φ whose domains cannot be extended.[4]

3.1 Naive Solution

The naive solution (Alg. 1) is to generate all the possible strategies for a given modular model, and then, for each of those strategies, to check at which system states its outcome satisfies φ. Finally, we have to check the resulting strategies for maximality, excluding all non-maximal strategies from our solution.

We now briefly outline this "naive" approach. The algorithm below uses function *mc*, which for a given LTL formula φ and a transition system T returns the set of all states in T at which φ holds. The transition system T passed to *mc* represents all outcomes of a strategy; therefore, there exists exactly one outgoing transition from each state of T. Therefore, the number of transitions in T equals the number of system states in T, i.e., $|St|^n$. The model checking of φ on the outcomes of a strategy is thus polynomial to the number of system states and length of the formula.

Algorithm 1 Naive Synthesis

Data an LTL formula φ to enforce and a model $\mathfrak{M}$
Result a set of all maximal strategies enforcing φ, together with their domains
1: $Out := \emptyset$
2: *//generate the set of all homogenous strategies Str*
3: **for all** $S \in Str$ **do**
//generate the entire transition system T by following S at every system state
4: $T := \{\langle Q, Q' \rangle \mid Q'[i] = tran(per(Q,i), S(per(Q,i)))$ for every $1 \leq i \leq n\}$
//do global model checking for $T \models \varphi$
5: $D = mc(\varphi, T)$ *//D is the set of all system states from T where* φ *holds*
6: **if** $D \neq \emptyset$ **then**
7: $Out := Out \cup \{\langle S, D \rangle\}$
//remove non-maximal solutions
8: **for all** $\langle S, D \rangle \in Out$ **do**
9: **if** $\exists \langle S', D' \rangle \in Out$ such that $D \subset D'$ **then** $Out := Out \setminus \{\langle S, D \rangle\}$
return Out

The naive solution outlined above has two major drawbacks. First, we have to go through all possible (total) strategies, whose number equals $|Act|^{|St| \cdot |\Sigma|^k}$, which is a pretty large number. Second, we have, prior to the model checking stage, generated the whole transition system corresponding to our modular

[4] From now on, we only consider homogenous strategies and omit the word "homogenous".

model (i.e., the transition system containing every possible system state; the number of such states, as we have seen, is $|St|^n$). In other words, following this "naive" approach, we do not take advantage of the fact that, from the very beginning, we know what formula we want to enforce—we blindly generate all the possible total strategies, and only then check which of them enforce φ.

The more sophisticated approach that we are going to consider in the rest of the paper immediately takes advantage of such knowledge; namely, we only ever generate strategies enforcing φ.

3.2 Incremental solution

In this subsection, we explore the solution that only ever considers strategies that enforce the desired property φ (as opposed to exploring all the strategies).

Since we construct the strategies for a formula φ incrementally, we work with partial functions defined only on a subset of all possible perceptions. We can see a partial strategy S as a set of pairs "perception—assigned action": $S \subseteq \{\langle q, m_1, \ldots, m_k, \alpha\rangle \in St \times \Sigma^k \times Act\}$.

Definition 7. *The outcome of a* partial *strategy S at a system state Q, denoted by $out(Q, S)$, is the (possibly finite) sequence Λ of system states such that $\Lambda[1] = Q$ and $\Lambda_{j+1} = \prod_{a \in Agt} tran(per(\Lambda_j, a), S(per(\Lambda_j, a)))$ for every $1 \leq j \leq |\Lambda|$. If Λ is finite, then it ends with a system state Q' such that S is not defined for $per(Q', a)$ for at least one $a \in Agt$.*

Given a set of system states $\mathbb{Q}$, we use $out(\mathbb{Q}, S)$ as a shorthand for $\bigcup_{Q \in \mathbb{Q}} out(Q, S)$. We use Λ^i for the suffix of Λ starting at $\Lambda[i]$. We use $\overline{\Lambda}$ for the set of all paths with the prefix Λ. If Λ is infinite then $\overline{\Lambda} = \{\Lambda\}$. We say that a sequence of states Λ satisfies an LTL formula φ if $\Lambda' \Vdash_{\text{LTL}} \varphi$ for every $\Lambda' \in \overline{\Lambda}$.

Definition 8 (strategy for a formula). *A partial strategy* for an LTL formula *φ with domain $\mathbb{D}$ is a partial strategy S such that $\Lambda' \Vdash_{LTL} \varphi$ for every $\Lambda' \in \overline{\Lambda}$ and every $\Lambda \in out(\mathbb{D}, S)$.*

Definition 9 (compatibility). *Two partial strategies S and S' are* compatible *if they assign the same actions to the same perceptions, i.e., if both S and S' are defined for $\langle q, m_1, \ldots, m_k\rangle$ then $S(q, m_1, \ldots, m_k) = S'(q, m_1, \ldots, m_k)$.*

If partial strategies S' and S'' are compatible, we define the joint strategy $S = S' \cup S''$ at every state at which either S' or S'' is defined. We can join compatible strategies into a *maximal* strategy:

Definition 10 (maximal strategy). *A partial strategy S for an LTL formula φ with domain $\mathbb{D}$ is* maximal *if:*

1. *there is no other partial strategy $S' \supset S$ for φ with domain $\mathbb{D}' \supset \mathbb{D}$ and*
2. *there is no other partial strategy $S'' \subset S$ for φ with domain $\mathbb{D}$*

A *minimal* strategy for an LTL formula φ and a system state Q is defined exactly for all the agent states that occur in the computation starting at Q until φ is satisfied. Formally:

Definition 11 (minimal strategy). *Let S be a partial strategy for an LTL formula φ with a system state Q in its domain, let Λ be the outcome of S at Q and let Λ_p be the shortest prefix of Λ such that $\Lambda' \Vdash_{LTL} \varphi$ for every $\Lambda' \in \overline{\Lambda_p}$. S is a* minimal *strategy for φ and Q if S is defined exactly for the set $\{\langle q, m_1, \cdots m_k\rangle \in St \times \Sigma^k \mid \langle q, m_1, \cdots m_k\rangle = per(Q, a)$ for some $Q \in \Lambda_p$ and some $a \in Agt\}$*

We construct all maximal strategies for an LTL formula φ in two steps:

1. We iteratively find every system state Q from which φ can be enforced and every minimal strategy for φ and Q.
2. We find every combination of the compatible minimal strategies such that the joint strategy is a maximal strategy for φ.

To accomplish the first step, for every subformula φ' of φ, we construct a directed graph representing the outcomes of minimal strategies enforcing φ'. Every vertex $\langle Q, S\rangle$ of such graph represents a system state Q and a minimal strategy S that enforces φ' at Q. Every edge directed from vertex v to vertex u represents an action vector given by the strategy from v, that leads from the system state from v to the system state from u, that is, given $v = \langle Q, S\rangle$ and $u = \langle Q', S'\rangle$, if an edge is directed from v to u then $\Lambda[2] = Q'$ where $\Lambda = out(Q, S)$ and $S' \subseteq S$. Given a subgraph $F = \langle V_F, E_F\rangle$ of such graph, we refer with $[F]$ to the set $\{Q \in St^n \mid \langle Q, S\rangle \in V_F$ for some $S\}$.

Informally, for each subformula φ' of φ, we obtain a *directed forest* $F_{\varphi'}$ (see Def. 12). A vertex $\langle Q, S\rangle$ is included in $F_{\varphi'}$ iff S is a minimal strategy for φ' and Q.[5] Maximal strategies for φ consist of partial strategies given by the directed forests for the subformulas with incompatible branches cut off.

Definition 12. *A* directed tree *is a connected directed graph $\langle V, E\rangle$ such that $|V| = |E| + 1$ and every vertex has at most one outgoing edge. The* root *of a directed forest $\langle V, E\rangle$ is $v \in V$ such that there is a path from u to v for every $u \in V$. A* directed forest *is a set of directed trees.*

Remark 1. Let F and F' be directed forests, let $\langle Q_1, S_1\rangle$ be a non-root vertex in F with its direct successor being $\langle Q_2, S_2\rangle$. Then $S_2 \subseteq S_1$ and if there is a non-root vertex $\langle Q_1, S_1' \supseteq S_1\rangle$ in F' then its direct successor is $\langle Q_2, S_2'\rangle$ where $S_2 \subseteq S_2' \subseteq S_1'$.

Consequently, given a vertex $\langle Q, S\rangle$ in a tree of directed forest F with root $\langle Q_R, S_R \subseteq S\rangle$, whenever there is a vertex $\langle Q, S' \supseteq S\rangle$ in another directed forest F', then for the branch $\langle Q, S\rangle, \ldots, \langle Q_R, S_R\rangle$ of F there is a sequence of vertices $\langle Q, S'\rangle, \ldots, \langle Q_R, S_R'\rangle$ such that $S_R \subseteq S_R' \subseteq S'$ in F'.

[5] If $S = \emptyset$ then *any* strategy enforces φ' at Q.

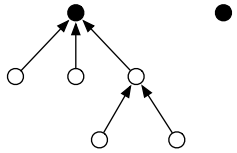


Fig. 2: A directed forest consisting of two trees. The black vertices are the roots.

The basic operation in constructing a directed forest for a formula φ is computing the set of all predecessors of a vertex $v = \langle Q, S\rangle$, using function pre:

$$\begin{aligned} pre(Q,S) = \{\langle Q', S'\rangle \mid & \ S' \text{ is the smallest strategy such that} \\ & \Lambda[2] = Q \text{ where } \Lambda = out(Q', S') \text{ and } S \subseteq S'\} \end{aligned}$$

To compute $pre(Q, S)$, we construct a colored undirected graph $C_{(Q,S)} = \langle V, E\rangle$ where V is partitioned into sets of vertices $V_1, \ldots, V_n$ ($n = |Agt|$), with every vertex from V_a representing a perception and an action of agent a:

$$\begin{aligned} V_a = \{\langle s, m_1, \ldots, m_k, \alpha\rangle \in St \times \Sigma^k \times Act \mid \\ tran(s, m_1, \ldots, m_k, \alpha) = Q[a] \text{ and} \\ S(s, m_1, \ldots, m_k) = \alpha \text{ or is undefined } \} \end{aligned}$$

We say that every vertex in V_a has color a. Two vertices from V are connected iff they represent mutually non-conflicting perceptions of the neighboring agents:

$$\begin{aligned} E = \{&\langle s, m_1, \ldots, m_k, \alpha\rangle \in V_a, \langle s', m'_1, \ldots, m'_k, \alpha'\rangle \in V_b \mid \\ &\exists 1 \leq l \leq k \text{ such that } neig(a, l) = b \text{ and } m_l = man(s') \text{ and} \\ &\exists 1 \leq l' \leq k \text{ such that } neig(b, l') = a \text{ and } m'_{l'} = man(s)\} \end{aligned}$$

Consider a subgraph $\langle V', E'\rangle$ of $C_{(Q,S)}$ such that:

1. V' contains, for every $a \in Agt$, exactly one vertex v_a from V_a (let $v_a = \langle s, m_1, \ldots m_k, \alpha\rangle$),
2. for every two $a, a' \in Agt$, $\langle v_a, v_{a'}\rangle \in E'$ iff a and a' are neighbors and
3. for every $v_a, v_{a'} \in V'$, if v_a and $v_{a'}$ represent the same perceptions then they also represent the same actions.

Every such subgraph represents one system state Q' where $Q'[a] = s$ and one partial strategy $S' = S \cup \bigcup_{a \in Agt} v_a$. Thus, $\langle Q', S'\rangle \in pre(Q, S)$ and the subgraphs of $C_{(Q,S)}$ satisfying the conditions above represent all predecessors of $\langle Q, S\rangle$. An example of such subgraph for $pre(Q = \langle q_0, q_2, q_3, q_3\rangle, S = \emptyset)$ and for our model with transition function $tran$ (1) is depicted in Fig. 3.

Other auxiliary functions used for construction of the directed trees are:

– $roots(F)$, returns all roots of the directed forest F;

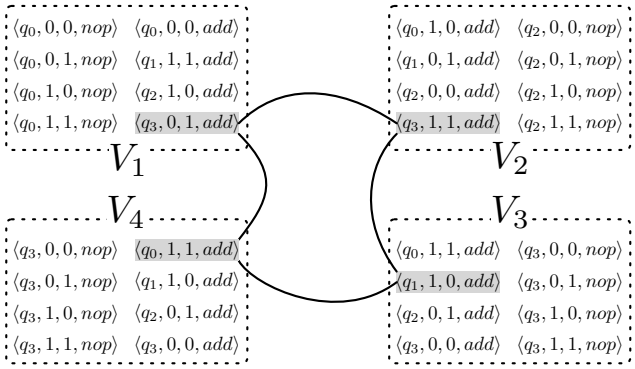


Fig. 3: Graph $C_{Q,S}$ where $Q = \langle q_0, q_2, q_3, q_3 \rangle$ and $S = \emptyset$ (each tuple represents one vertex, edges of $C_{Q,S}$ are not depicted). The dashed squares represent the colored partitions of the vertices for the respective agents. One subgraph representing a predecessor of Q is depicted, with grey vertices and solid edges. The subgraph represents state $Q' = \langle q_3, q_3, q_1, q_0 \rangle$ and strategy S' composed of the vertices of the subgraph. The second state of the outcome of S' at Q' is Q.

- *predecessors*$(\langle Q, S \rangle, F)$, returns all predecessors of the vertex $\langle Q, S \rangle$ in the directed forest F;
- *traverse*$(\langle Q, S \rangle, F)$, returns all vertices of the (sub)tree rooted in $\langle Q, S \rangle$ in the directed forest F; and
- *prune*(T, R, S'), in directed tree T cuts off all subtrees rooted in a vertex representing state R or representing a strategy incompatible with strategy S' and extends the strategies in the remaining vertices with S' (T may be empty after this operation).

Using the functions described above, we construct, for each subformula φ' of φ, the following directed forest $F_{\varphi'} = \langle V_{\varphi'}, E_{\varphi'} \rangle$:

- **case** $\varphi' = \mathtt{p_a}$:
  $V_{\varphi'} = \{\langle Q, \emptyset \rangle \mid \{Q\} \models \mathtt{p_a}\}$; $E_{\varphi'} = \emptyset$;
- **case** $\varphi' = \neg\mathtt{p_a}$:
  $V_{\varphi'} = \{\langle Q, \emptyset \rangle \mid \{Q\} \models \neg\mathtt{p_a}\}$; $E_{\varphi'} = \emptyset$;
- **case** $\varphi' = \psi_1 \vee \psi_2$:
  $F_{\varphi'} = F_{\varphi_1} \cup F_{\varphi_2}$;
- **case** $\varphi' = \psi_1 \wedge \psi_2$:

  $V_{\varphi'} = \{\langle Q, S \rangle \mid$ either $\langle Q, S \rangle \in F_{\psi_1}$ and $\langle Q, S' \rangle \in F_{\psi_2}$ for some $S' \subseteq S$ or
  $\langle Q, S \rangle \in F_{\psi_2}$ and $\langle Q, S' \rangle \in F_{\psi_1}$ for some $S' \subseteq S\}$;
  $E_{\varphi'} = \emptyset$;

- **case** $\varphi' = \mathsf{X}\psi$

We alter a copy of F_ψ as follows: For every vertex $\langle Q, S\rangle$ in F_ψ (obtained by function *traverse*), we find all its predecessors by $pre(Q, S)$ and append to $\langle Q, S\rangle$ those that are not preceding $\langle Q, S\rangle$ in F_ψ. Moreover, we remove every vertex that is a root in F_ψ. Thus, the predecessors of the removed roots become roots themselves. The resulting directed forest is $F_{\varphi'}$.

$$\begin{aligned} V_{\varphi'} =& (V_\psi \cup \{\langle Q, S\rangle \mid \langle Q, S\rangle \in pre(Q', S') \text{ for some } \langle Q', S'\rangle \in F_\psi\}) \setminus \\ & \{\langle Q, S\rangle \mid \langle Q, S\rangle \in roots(F_\psi)\}; \\ E_{\varphi'} =& \{\langle \langle Q, S\rangle, \langle Q', S'\rangle\rangle \mid \langle Q, S\rangle \in pre(Q', S')\}; \end{aligned}$$

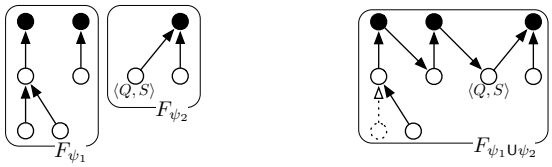
– **case** $\varphi' = \psi_1 \mathsf{U}\, \psi_2$ (Fig. 4):
We use function $reach(\langle Q, S\rangle, R, F_1, F_2)$ to construct $F_{\varphi'}$. The function appends to every vertex $\langle Q, S\rangle$ from the forest F_2, directly or indirectly, every branch of the forest F_1 from which $\langle Q, S\rangle$ is reachable. Where possible, the function extends the strategy S' of a vertex $\langle Q', S'\rangle$ from the directed forest F_1 so that, when following the extended strategy at Q', after reaching the root of $\langle Q', S'\rangle$ in F_1, vertex $\langle Q, S\rangle$ in F_2 is reached solely through the system states represented by vertices of F_1 (the function uses *pre*, *roots* and *prune*, see Alg. 2). In case that Q' is identical with the system state R represented by the root of $\langle Q, S\rangle$, the entire subtree rooted in $\langle Q', S'\rangle$ is excluded from the result. Thus, every system state is represented in any given branch of the resulting tree at most once and unfolding of potential cycles is prevented. At the end, altered forest F_2 is $F_{\varphi'}$.

Algorithm 2 $reach(\langle Q, S\rangle, R, F_1, F_2)$

Data a directed forest F_2, a vertex $\langle Q, S\rangle$ from F_2, the state R represented by the root of the tree containing $\langle Q, S\rangle$ and a directed forest F_1

1: **for all** $\langle Q', S'\rangle \in roots(F_1)$ such that $Q' = Q$ and S' is compatible with S **do**
2: copy the tree rooted in $\langle Q', S'\rangle$ as T;
3: $prune(T, R, S \cup S')$;
4: append every subtree of T rooted in a predecessor of the root of T to $\langle Q, S\rangle$ in F_2;
5: **for all** $\langle Q', S'\rangle \in roots(F_1)$ such that $Q' \neq Q$ and there is $\langle Q', S''\rangle \in pre(Q, S)$ where S' is compatible with S'' **do**
6: copy the tree rooted in $\langle Q', S'\rangle$ as T;
7: $prune(T, R, S'')$;
8: append T to $\langle Q, S\rangle$ in F_2;
9: **for all** $\langle Q', S'\rangle \in predecessors(\langle Q, S, \rangle, F_2)$ **do**
10: $reach(\langle Q', S'\rangle, R, F_1, F_2)$;

– **case** $\varphi' = \mathsf{G}\, \psi$:
We represent with vertices of $F_{\varphi'}$ all strategies that eventually enforce cycles visiting only the states from $[F_\psi]$. We use function $cycle(\langle Q, S\rangle, R, \{\langle Q, S\rangle\}, F_\psi, F_{\varphi'})$ to add to $F_{\varphi'}$ all such strategies that, in addition, include state $\langle Q, S\rangle$ in the cycle, where R is the state represented by the root of $\langle Q, S\rangle$ in F_ψ (see Alg. 3).



(a) Forests for ψ_1 and ψ_2. (b) Forest for $\psi_1 \mathsf{U} \psi_2$.

Fig. 4: An example of $F_{\psi_1 \mathsf{U} \psi_2}$. The dashed vertex is excluded from $F_{\psi_1 \mathsf{U} \psi_2}$ by $\mathit{reach}(\langle Q, S\rangle, R, F_{\psi_1}, F_{\psi_2})$ because its strategy is incompatible with S or because it represents the same state as the root of F_{ψ_2}.

Since we use $\mathit{cycle}(\langle Q, S\rangle, R, \{\langle Q, S\rangle\}, F_\psi, F_{\varphi'})$ for every vertex $\langle Q, S\rangle$ that is a root in F_ψ, the resulting forest is $F_{\varphi'}$.

Algorithm 3 $\mathit{cycle}(\langle Q, S\rangle, R, T, F_1, F_2)$

Data a directed forest F_1, a tree T, a vertex $\langle Q, S\rangle$ from T, the state R represented by the root of the tree in F_1 containing $\langle Q, S\rangle$, a directed forest F_2 to add the results.

1: $\mathbb{P} = \{\langle Q', S'\rangle \in \mathit{pre}(Q, S) \mid \exists \langle Q', S'' \subseteq S'\rangle \in F_1\}$;
2: append every $\langle Q', S'\rangle \in \mathbb{P}$ to $\langle Q, S\rangle$ in T;
3: **for all** $\langle Q', S'\rangle \in \mathbb{P}$ **do**
4: **if** $Q' = R$ **then**
5: cut off $\langle Q', S'\rangle$ from T;
6: copy T to F_2 as T';
7: $\mathit{prune}(T', R, S')$;
8: **else**
9: $\mathit{cycle}(\langle Q', S'\rangle, R, T, F_1, F_2)$;

Now we show, how the directed forest F_φ for a formula φ represents all minimal strategies for φ.

Claim. Let φ be an LTL formula and let F_φ be the directed forest for φ. S is a minimal homogenous strategy for φ and a system state Q iff $\langle Q, S\rangle$ is a vertex of F_φ.

Proof. We only proof the nontrivial cases:

- **case** $\varphi = \mathsf{X}\psi$:
  S_ψ is a minimal strategy for ψ and state Q_ψ iff there is $\langle Q_\psi, S_\psi\rangle \in V_\psi$. Let S be a strategy, let Q be a state and let $\Lambda = \mathit{out}(Q, S)$. Vertex $\langle Q, S\rangle$ is present in F_φ iff $\langle Q, S\rangle \in \mathit{pre}(Q', S')$ for some $\langle Q', S'\rangle \in V_\psi$ iff, for some $\langle Q', S'\rangle \in V_\psi$, $\Lambda[2] = Q'$ and $S \supseteq S'$ is defined for the same agent states as S' and, moreover, for the agent states from Q iff S is a minimal strategy for φ and Q.

– **case** $\varphi' = \psi_1 \mathsf{U}\, \psi_2$:
($\Leftarrow$): Let $\langle Q, S\rangle$ be a vertex from $F_{\varphi'}$. Either (i) $\langle Q, S\rangle \in V_{\psi_2}$ or (ii) $\langle Q, S\rangle$ is added to $F_{\varphi'}$ by function *reach* from F_{ψ_1} and thus some $\langle Q, S_{\psi_1} \subseteq S\rangle \in V_{\psi_1}$. $\langle Q, S\rangle \in V_{\psi_2}$ iff S is a minimal strategy for ψ_2 and Q. Every minimal strategy for ψ_2 and Q is also a minimal strategy for φ' and Q. Thus, if (i) then S is a minimal strategy for φ' and Q. If (ii) then S is the union of compatible strategies S_r and S_{ψ_2} where (1) the last state of $\Lambda = out(Q, S_r)$ is Q' such that $\langle Q', S_{\psi_2}\rangle \in V_{\psi_2}$ and (2) for every other state $Q_\Lambda \in \Lambda$ there is some vertex $\langle Q_\Lambda, S_\Lambda \subseteq S_r\rangle \in F_{\psi_1}$. Thus, the outcome $\Lambda_{\varphi'}$ of $\langle Q, S\rangle$ consists of Λ followed by $\Lambda_2 = out(Q', S_{\psi_2})$. Since Λ_r is composed of the outcomes of minimal strategies for ψ_1 and ends with Q', every path starting before Q' and following $\Lambda_{\varphi'}$ satisfies ψ_1 and every path starting from Q' on and following $\Lambda_{\varphi'}$ satisfies ψ_2. Thus, there is $i \geq 1$ such that $\Lambda' \Vdash_{\mathrm{LTL}} \psi_2$ for every $\Lambda' \in \overline{\Lambda^i_{\varphi'}}$ and $\Lambda' \Vdash_{\mathrm{LTL}} \psi_1$ for every $\Lambda' \in \overline{\Lambda^j_{\varphi'}}$ and every $1 \leq j < i$, namely such i that $\Lambda_{\varphi'}[i] = Q'$. Thus, S is a strategy for φ' and Q.
Strategy S_r is defined for the same agent states as the minimal strategies for ψ_1 from which S_r is composed and, moreover, for the agent states from the last system states of the outcomes of the minimal strategies for ψ_1. Thus, every agent state for which S_r is defined is present at some system state from the outcome of S_r at Q. Since S_{ψ_2} is a minimal strategy for ψ_2 and Q', every for which S_{ψ_2} is defined is present at some system state from the outcome of S_{ψ_2} at Q'. Since $S = S_r \cup S_{\psi_2}$, every agent state for which S is defined is present at some system state from the outcome of S at Q. Let $\Lambda_{\varphi'} = out(S, Q)$ with length l and $Q' = \Lambda_{\varphi'}[i]$ for some i. Since every path satisfying φ' must contain a state from $[F_{\psi_2}]$ and $\Lambda_{\varphi'}[i]$ is the first such state in $\Lambda_{\varphi'}$, there is no $j < i$ such that $\Lambda' \Vdash_{\mathrm{LTL}} \varphi'$ for every $\Lambda' \in \overline{\Lambda^j}$. Since S_{ψ_2} is a minimal strategy for ψ_2 and Q', there is no prefix of the outcome of S_{ψ_2} such that all paths starting with this prefix satisfy ψ_2 and thus, there is no $i \leq j < l$ such that $\Lambda' \Vdash_{\mathrm{LTL}} \varphi'$ for every $\Lambda' \in \overline{\Lambda^j}$. Hence, S is a minimal strategy for φ' and Q.
($\Rightarrow$): Let S be a minimal strategy for ψ and state Q. Then there is $i \geq 1$ such that, for every path Λ' with prefix $\Lambda = out(Q, S)$, $\Lambda'^i \Vdash_{\mathrm{LTL}} \psi_2$ and $\Lambda'^j \Vdash_{\mathrm{LTL}} \psi_1$ for every $1 \leq j < i$. Thus, some $S_{\psi_2} \subseteq S$ is a minimal strategy for ψ_2 and $\Lambda[i]$ and therefore there is a vertex $\langle \Lambda[i], S_{\psi_2} \subseteq S\rangle$ in F_{ψ_2}. If $i = 1$ then S itself is a minimal strategy for ψ_2 and Q and thus, vertex $\langle Q, S\rangle \in F_{\psi_2}$. Since $F_{\psi_2} \subseteq F_{\varphi'}$, $\langle Q, S\rangle$ is a vertex of $F_{\varphi'}$. If $i > 1$ then for every $\Lambda[j]$ there is a vertex representing $\Lambda[j]$ in F_{ψ_1}. Since $\Lambda[i]$ is reachable from every $\Lambda[j]$ by visiting only states from $[F_{\psi_1}]$, $\Lambda[j]$ is found by function *reach* for every $1 \leq j < i$ and vertex $\langle \Lambda[j], S_j\rangle$ is added to $F_{\varphi'}$. Strategy S_j contains S_{ψ_2} and, in addition, is defined for every agent state present in some $\Lambda[l]$ where $j \leq l < i$ so that the outcome of S_j at $\Lambda[j]$ is Λ^j. Thus, $out(\Lambda[1], S_1) = \Lambda^1 = \Lambda$ where $\Lambda = out(Q, S)$. Since S is a minimal strategy for φ' and Q and the outcomes of S_1 and S at Q are identical, $S^1 = S$. Thus, $\langle \Lambda[1], S_1\rangle = \langle Q, S\rangle$ and $\langle Q, S\rangle \in F_{\varphi'}$.
– **case** $\varphi' = \mathsf{G}\, \psi$:

($\Leftarrow$): Let $\langle Q, S\rangle$ be a vertex from $F_{\varphi'}$ contained in a tree T. Tree T with root $\langle R, S_R\rangle$ is added by *cycle* to $F_{\varphi'}$ only if $out(R, S_R)[2] = Q_R$ for some $\langle Q_R, S_R\rangle \in T$. For every vertex $\langle Q', S'\rangle$ from T, the outcome of S' at Q' contains R and thus, also the outcome of S_R at Q_R contains R. Therefore, $\Lambda_R = out(R, S_R)$ contains R infinitely often and thus, Λ_R is infinite. Since $S_R \subseteq S'$ and $R \in out(Q', S')$ for every $\langle Q', S'\rangle \in T$, Λ_R is a suffix of $out(Q', S')$ for every $\langle Q', S'\rangle \in T$. Thus, $out(Q, S)$ is infinite.
As we have shown, $\Lambda = out(Q, S)$ is a path (i.e., an infinite sequence) consisting solely of Q followed by the system states represented by the successors of $\langle Q, S\rangle$ in T. Since T is constructed by function *cycle* from F_ψ, for every $\langle Q', S'\rangle \in T$ there is some $\langle Q', S_\psi \subseteq S'\rangle \in F_\psi$. Thus, for every $i \geq 1$, Λ^i has a prefix $out(\Lambda[i], S_i \subseteq S)$ where $\langle \Lambda[i], S_i\rangle$ is a vertex from F_ψ. Since S_i is a minimal strategy for ψ and $\Lambda[i]$, every path starting with $out(\Lambda[i], S_i)$ satisfies ψ and thus, also Λ^i satisfies ψ. Therefore, S is a strategy for $\varphi' = \mathsf{G}\,\psi$ with Q in its domain.
Strategy S is composed of strategies S_i such that $\langle \Lambda[i], S_i\rangle \in F_\psi$ where $\Lambda = out(Q, S)$ and $i \geq 1$. Since every S_i is a minimal strategy for ψ and $\Lambda[i]$, it is defined solely for the agent states present in all system states in $out(\Lambda[i], S_i)$ except the last one. Additionally, S is defined also for all agent states present in the last system state of $out(\Lambda[i], S_i)$ for every $i \geq 1$ so that Λ is infinite. Thus, S is defined exactly of the agent states present in the system states from $\Lambda = out(Q, S)$ and since Λ satisfies φ', S is a minimal strategy for φ' and Q.
($\Rightarrow$): Let S be a minimal strategy for φ' and state Q. Then $\Lambda = out(Q, S)$ is infinite and $\Lambda^i \Vdash_{\text{LTL}} \psi$ for every $i > 0$. Thus, for every $i > 0$ there is some minimal strategy S_i for ψ and $\Lambda[i]$ such that $out(\Lambda[i], S_i)$ is a prefix of Λ^i and therefore $\langle \Lambda[i], S_i\rangle \in F_\psi$ for every $i > 0$ and some $S_i \subseteq S$.
First, consider the case that $Q = \Lambda[1]$ occurs in Λ infinitely often. Thus, Q is part of a cycle and can be reached from Q through the sequence of predecessors of Q. Since $\langle Q, S_1\rangle \in F_\psi$, function *cycle* looks for vertices $\langle Q', S'\rangle$ such that $\Lambda'[i] = Q$ for some $i > 0$ and $\Lambda' = out(Q', S')$ and $\langle \Lambda'[j], S'_j \subseteq S'\rangle \in F_\psi$ for every $0 < j < i$ until $Q' = Q$. S' is always defined exactly for the agent states occurring at system states from $out(Q', S')$. In case that $Q' = Q$, outcome of S' at Q' contains Q infinitely often. Since S is also defined exactly for the for the agent states occurring at system states from $out(Q, S)$, vertex $\langle Q, S\rangle$ is found be *cycle* and a tree with root $\langle Q, S\rangle$ is added to $F_{\varphi'}$.
Second, consider that $Q = \Lambda[1]$ does not occur in Λ infinitely often (that implies that it occurs in Λ only once). Since Λ is infinite, there is $Q' \neq Q$ that occurs in Λ infinitely often and a vertex $\langle Q', S' \subseteq S\rangle$ was found by *cycle* such that $out(Q', S')$ is a suffix of Λ. Function *cycle* adds to F_ψ every vertex $\langle Q'', S'' \supseteq S'\rangle$ such that $Q' \in out(Q'', S'')$ and S'' is only defined for the agent states occurring at the system states from $out(Q'', S'')$. Thus, $\langle Q, S\rangle$ is found as well and added to $F_{\varphi'}$. □

Definition 13. *A* branch *is a path in a directed tree that starts with a leaf and ends with the root. A* subbranch *is a suffix of a branch.*

Definition 14. *A* maximal compatible subforest M *of a directed forest* F *is a set of subbranches of* F *such that:*

1. *all strategies from the vertices of* M *are mutually compatible and*
2. *no other subbranch of* F *can be added to* M *without violating condition 1.*

In the second step, all maximal compatible subforests of F_φ are generated during depth-first search through F_φ.

Claim. Let F_φ be the directed forest for an LTL formula φ. M is a maximal strategy for φ iff M is a maximal compatible subforest of F_φ.

Proof. Straightforward. □

Hence, after completing the second step, all maximal strategies for φ are found and the task of the strategy synthesis is fulfilled.

4 Complexity and Comparison

The naive approach described in Section 3.1 involves model checking on every transition system generated by a total strategy. Since a total strategy assigns $\alpha \in Act$ to every possible perception from $St \times \Sigma^k$, where k is the maximal size of the neighborhood, there exist $|Act|^{|St|\cdot|\Sigma|^k}$ total strategies. As the number of manifestations ($|\Sigma|$) is in the worst case equal to the number of agent states ($|St|$), the number of all strategies is always less or equal $S = |Act|^{|St|^{k+1}}$. As we have shown in Section 3.1, the model checking of one total strategy for a formula φ of length l involves at most $l \cdot |St|^n$ steps. Thus, the worst case complexity of finding all total strategies for φ is $O(l \cdot |St|^n \cdot |Act|^{|St|^{k+1}})$. To keep only *maximal* strategies for φ, we have to drop strategies with non-maximal domains. Thus, we have to mutually compare the domains of all total strategies for φ and the overall complexity is $O(|Act|^{2\cdot|St|^{k+1}})$.

The worst case complexity of the incremental approach described in Section 3.2 is worse than that of the naive approach. To show this, we estimate the upper bound on the size of the directed forest F_φ from which all maximal strategies for formula φ are computed. We denote the number of vertices of F_φ with $|F_\varphi|$. This number is given by the maximal number of the predecessors of a vertex in the forest (denoted by deg_{in}) to the power of the maximal depth of the tree. Since every branch represents in its vertices each global state at most once, the maximal depth of the tree is the number of global states, which is $|St|^n$. Thus, $O(|F_\varphi|) = O({deg_{in}}^{|St|^n})$. Assume that number of agents is bigger than the size if the neighborhood ($n > k + 1$) and that deg_{in} is at least two (which is usually the case), and therefore, regardless of the complexity of *pre* function used for finding predecessors in F_φ, the construction of F_φ already may take more steps than synthesizing all maximal strategies for φ with the naive approach ($O(2^{|St|^n}) > O(|Act|^{2^{|St|^{k+1}}})$.

The incremental approach may take more time than the naive one for example if $\varphi = \mathsf{G}\top$. While any total strategy is a maximal strategy for $\mathsf{G}\top$, the non-trivial

approach would still look for all predecessors of all global states while iteratively generating strategies that eventually enforce cycles. This computation would be in this case more expensive than simple generation of all total strategies. However, we believe that in case of non-trivial formulas, which usually cannot be enforced by every arbitrary strategy, incremental constructing of only such strategies that enforce the formula takes substantially fewer steps that following the naive approach. This is a phenomenon known in other areas. For example, as shown in [3], a 2-EXPTIME algorithm for deciding satisfiability in LTL vastly outperforms in practice an algorithm whose worst-time theoretical complexity is singly exponential.

5 Conclusion

In this paper, we described a technique for the synthesis of strategies for homogenous multi-agent systems with incomplete information. While there exists a general solution for this problem in the context of heterogenous systems [2], it is not directly applicable in scenarios where the agents are homogenous and each of them must follow the same strategy.

We proposed a modular system model that takes the advantage of the assumption on the homogeneity of the agents and allows for a compact representation of the systems, especially in comparison to the traditional models based on the global system statespace. We used the language of Linear Temporal Logic (LTL) to express the system properties that the synthesized strategies shall enforce. We proposed alternative semantics for LTL that enables us to express these properties intuitively in our context. We described a naive solution to the stated problem and then we proposed a non-trivial algorithm, which constructs the strategies incrementally. Although the latter algorithm has higher worst case complexity than the naive solution, we argue that in practice, the incremental strategy synthesis works much better than the naive solution.

References

1. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
2. Jan Calta, Dmitry Shkatov, and Bernd-Holger Schlingloff. Finding uniform strategies for multi-agent systems. In *Proceedings of 11th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA XI)*, volume 6245 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2010.
3. Valentin Goranko, Angelo Kyrilov, and Dmitry Shkatov. Tableau tool for testing satisfiability in ltl: Implementation and experimental analysis. *Electronic Notes in Theoretical Computer Science*, 262:113–125, 2010.
4. Wojciech Jamroga and Thomas Ågotnes. Constructive knowledge: what agents can achieve under imperfect information. *Journal of Applied Non-classical Logics*, 17(4):423–475, 2007.
5. Wojciech Jamroga and Thomas Ågotnes. Modular interpreted systems. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8. ACM, 2007.