Hierarchical Models of Multi-Agent Systems: Strategic Ability and Model Checking

Rustam Galimullin[1], **Wojtek Jamroga**[2,3], **Damian Kurpiewski**[2,3], **Vadim Malvone**[4], **Aniello Murano**[5]

[1]University of Bergen, Bergen, Norway
[2]Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
[3]Institute of Advanced Studies, Nicolaus Copernicus University in Torun, Poland
[4]Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France
[5]University of Naples Federico II, Naples, Italy
rustam.galimullin@uib.no, {w.jamroga, d.kurpiewski}@ipipan.waw.pl,
vadim.malvone@telecom-paris.fr, aniello.murano@unina.it

Abstract

Multi-agent systems often involve multi-level interactions that make strategic reasoning hard to scale. To capture such systems, we introduce *hierarchical concurrent game models* (HCGMs) that allow embedding of (sub)systems into other systems. We define the semantics of ATL on HCGMs by an unfolding of an HCGM into a standard (flat) concurrent game model. Building on this semantics, we provide a model checking algorithm for ATL interpreted over HCGMs and discuss its complexity.

1 Introduction

Multi-agent systems involve complex interactions among autonomous agents, whose behaviour often depends on both cooperation and competition. As the number of agents and possible interactions grows, reasoning about their strategic abilities becomes increasingly challenging. To manage this complexity, hierarchical models provide a natural way to describe systems at multiple levels of abstraction: a top-level transition system captures coarse-grained behavior, while selected nodes are refined into lower-level models representing more detailed interactions. One can think of an authorisation module that can be used as a part of different e-voting protocols, or autonomous robots that share the same motor or vision modules. Such hierarchical specifications allow a modeller to represent complex systems more succinctly, focusing on a higher-level behaviour of a system.

Models of hierarchical, and more generally compositional representations have been extensively studied in the context of reactive single-agent systems (Alur, Kannan, and Yannakakis 1999; Alur and Yannakakis 2001; de Alfaro and Henzinger 2001; Murano, Napoli, and Parente 2008; Aminof, Kupferman, and Murano 2012; Faran and Kupferman 2019; Bozzelli et al. 2020; Stefansson and Johansson 2023; Yamane 2025), where one can employ temporal logics for specification and verification of such systems (Pnueli 1977; Queille and Sifakis 1982; Clarke and Emerson 1981). However, extending these ideas to multi-agent game-like scenarios (Alur, Henzinger, and Kupferman 2002; Kupferman, Vardi, and Wolper 2001) is not straightforward: agents act proactively, and their objectives are expressed in terms of strategic abilities rather than mere temporal properties. To address this, we introduce *hierarchical concurrent game models* (HCGMs), which generalise classic concurrent game models (CGMs) from (Alur, Henzinger, and Kupferman 2002) to a hierarchical setting. Their semantics is formally defined via a flattening procedure that unfolds a hierarchical model into an equivalent flat CGM.

Our contribution. We introduce HCGMs, a hierarchical extension of CGMs for proactive multi-agent interaction. We provide a flattening semantics and show how to interpret the *Alternating-time Temporal Logic* (ATL) formulas over hierarchical structures. Finally, we propose a direct ATL model-checking procedure that operates on the hierarchy, avoiding the explicit construction of the (potentially exponentially larger) flat model.

2 Alternating-Time Temporal Logic

We start by recalling the standard concurrent game models and ATL (Alur, Henzinger, and Kupferman 2002). Throughout the paper, let Ap be a countable set of atoms.

Definition 1 (Concurrent game model (CGM))**.** *A* CGM *is a tuple* $M = (\mathbb{A}\mathrm{gt}, St, Act, act, o, V)$ *where* $\mathbb{A}\mathrm{gt} = \{a_1, \ldots, a_n\}$ *is a finite set of agents,* St *is a nonempty set of states,* Act *is a nonempty set of actions,* $act : St \times \mathbb{A}\mathrm{gt} \to 2^{Act} \setminus \emptyset$ *is the enabling function specifying for each state and each agent a non-empty set of actions available to the agent,* $o : St \times Act^{\mathbb{A}\mathrm{gt}} \to St$ *is a total transition function such that* $o(s, \alpha_1, \ldots, \alpha_n)$ *is defined only when* $\alpha_i \in act(s, a_i)$ *for all* $i \in \mathbb{A}\mathrm{gt}$*, and* $V : St \to 2^{\mathsf{Ap}}$ *is a valuation function.*

We will sometimes call such models "flat", to distinguish them from hierarchical structures, defined below.

A *path* π is an infinite sequence of states $s_0, s_1, \ldots$ such that for all $k \geqslant 0$ there is a *joint action* $\alpha \in Act^{\mathbb{A}\mathrm{gt}}$ such that $\alpha_i \in act(s_k, a_i)$ for every $i \in \mathbb{A}\mathrm{gt}$ and $s_{k+1} = o(s_k, \alpha)$.

Definition 2 (Alternating-time Temporal Logic (ATL))**.** *The language of* ATL *is defined recursively as follows:*

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle C \rangle\rangle \mathbf{X}\varphi \mid \langle\langle C \rangle\rangle \varphi \mathbf{U} \varphi \mid \langle\langle C \rangle\rangle \varphi \mathbf{R} \varphi$$

where $p \in \mathsf{Ap}$ *and* $C \subseteq \mathbb{A}\mathrm{gt}$.

Formula $\langle\langle C\rangle\rangle \mathbf{X}\varphi$ reads as "coalition C has a strategy to achieve φ in the *ne***X***t* step". Formula $\langle\langle C\rangle\rangle \psi \mathbf{U}\varphi$ means "coalition C can maintain ψ **U***ntil* φ becomes true. Formula $\langle\langle C\rangle\rangle \psi \mathbf{R}\varphi$ reads as "coalition C can maintain φ until ψ **R***eleases* the requirement for the truth of φ".

Definition 3. *A* (memoryless) strategy *for agent* $i \in \mathbb{A}\mathrm{gt}$ *is a function* $\sigma_i : St \to Act$ *that assigns an action to agent* a_i *in all states. We denote by* $\vec{\sigma}_C$ *a joint strategy for coalition* C*. Joint strategies contain one strategy for each agent in coalition* C*. A joint strategy* $\vec{\sigma}_C$ *is* compatible *with a path* π *iff for every* $j \geqslant 0$*,* $\pi_{j+1} = o(\pi_j, \alpha_C)$ *for some joint action* $\alpha_C \in Act^C$ *s.t. for every* $a_i \in C$*,* $\alpha_i = \sigma_i(\pi_j)$*, and for every* $a_i \in \overline{C}$*,* $\alpha_i \in act(\pi_j, a_i)$*. By* $\mathsf{out}(s, \vec{\sigma}_C)$ *we denote the set of all* $\vec{\sigma}_C$*-compatible paths from* s*.*

Definition 4 (Semantics)**.** *The satisfaction relation* $\models$ *for a CGM* M *and a* $\varphi \in \mathsf{ATL}$ *is defined as follows (clauses for Boolean connectives are immediate and thus omitted):*

$(M, s) \models p$ *iff* $p \in V(s)$
$(M, s) \models \langle\langle C\rangle\rangle \mathbf{X}\psi$ *iff* $\exists \vec{\sigma}_C, \forall \pi \in \mathsf{out}(s, \vec{\sigma}_C)\ (M, \pi_1) \models \psi$
$(M, s) \models \langle\langle C\rangle\rangle \psi \mathbf{U} \psi'$ *iff* $\exists \vec{\sigma}_C, \forall \pi \in \mathsf{out}(s, \vec{\sigma}_C), \exists k \geqslant 0$ $(M, \pi_k) \models \psi'$*, and for all* $1 \leqslant j < k$ $(M, \pi_j) \models \psi$
$(M, s) \models \langle\langle C\rangle\rangle \psi \mathbf{R} \psi'$ *iff* $\exists \vec{\sigma}_C, \forall \pi \in \mathsf{out}(s, \vec{\sigma}_C), \forall k \geqslant 0 :$ $(M, \pi_k) \models \psi'$*, or for some* $0 \leqslant j < k$*,* $(M, \pi_j) \models \psi$

In the paper, we are primarily interested in the model checking problem.

Definition 5. *Given a CGM* M*, a state* $s \in St$*, and a formula* φ*, the* model checking problem *concerns determining whether* $M, s \models \varphi$*.*

3 Hierarchical Models of MAS

Now, taking inspiration from (Alur and Yannakakis 2001), we define Hierarchical CGMs.

Definition 6 (Hierarchical concurrent game model (HCGM))**.** *An* HCGM *(or Hierarchical CGM) is a tuple of* modules $\mathcal{M} = (\mathsf{M}_1, ..., \mathsf{M}_k)$*, where each* M_i *is a tuple* $(\mathbb{A}\mathrm{gt}_i, St_i, \mathsf{In}_i, \mathsf{Out}_i, \mathsf{B}_i, \mathsf{Map}_i, \mathsf{Act}_i, \mathsf{act}_i, \mathsf{o}_i, \mathsf{V}_i)$ *with its elements defined as follows.*

- $\mathbb{A}\mathrm{gt}_i = \{a_1^i, a_2^i, \dots\}$ *is a finite nonempty set of* active agents*. We will use* $n_i = |\mathbb{A}\mathrm{gt}_i|$ *to denote the number of active agents in the module.*
- $St_i = \{s_1^i, s_2^i, \dots\}$ *is a non-empty set of* states*.*
- In_i *is a non-empty set of* inputs*, i.e., the entry points to the module.*
- Out_i *is a set of* outputs*, i.e., exits from* M_i*. We require that sets* St_i*,* In_i*, and* Out_i *are mutually disjoint. Moreover,* $\mathsf{Out}_i = \emptyset$ *if* $i = 1$*, and* $\mathsf{Out}_i \neq \emptyset$ *otherwise.*
- B_i *is a finite set of* boxes*.*
- $\mathsf{Map}_i : \mathsf{B}_i \mapsto \{i+1, ..., k\}$ *maps each box of the current module to a module of level at least one level deeper than the current one. We will denote by* $\mathsf{In}(\mathsf{B}_i)$ *the disjoint union* $\bigsqcup_{b \in \mathsf{B}_i} \mathsf{In}_{\mathsf{Map}_i(b)}$ *of inputs of all boxes* $b \in \mathsf{B}_i$*, and we will denote by* $\mathsf{Out}(\mathsf{B}_i)$ *the disjoint union* $\bigsqcup_{b \in \mathsf{B}_i} \mathsf{Out}_{\mathsf{Map}_i(b)}$ *of outputs of all boxes* $b \in \mathsf{B}_i$*.*
- Act_i *is a non-empty set of* actions*. We assume* $\epsilon \notin \mathsf{Act}_i$*.*
- $\mathsf{act}_i : (St_i \cup \mathsf{In}_i \cup \mathsf{Out}(\mathsf{B}_i)) \times \mathbb{A}\mathrm{gt}_i \to 2^{\mathsf{Act}_i} \setminus \emptyset$ *is an* action enabling function *that specifies which actions are available to agents in each decision point.*
- $\mathsf{o}_i : \{(s, \alpha_1, ..., \alpha_{n_i}) \mid s \in St_i \cup \mathsf{In}_i \cup \mathsf{Out}(\mathsf{B}_i), \alpha_j \in \mathsf{act}_i(s, a_j)\} \to St_i \cup \mathsf{In}(\mathsf{B}_i) \cup \mathsf{Out}_i$ *is a (total)* transition function*.*
- $\mathsf{V}_i : St_i \cup \mathsf{In}_i \cup \mathsf{Out}_i \to 2^{\mathsf{Ap}}$ *is a* valuation function*.*

Intuitively, an HCGM consists of several modules that can embed one another. Each module has its own (possibly overlapping with other modules) sets of agents and actions, as well as disjoint sets of inputs, states, and outputs. We assume that the top-level module M_1 has no outputs, i.e., we reason about self-contained multi-agent systems. Boxes represent embeddings of other modules into the current one. Embedding is well-defined because a module may only embed modules at strictly deeper levels, avoiding recursion.

The transition function o_i ensures that, for all states of the current module, inputs of the current module, and outputs of the boxes of the current module, and for all full action profiles, the outcome of a transition is either a state of the current module, output of the current module, or one of the inputs of the boxes of the current module. Observe that such a definition of o treats outputs of the boxes (or rather of modules assigned to the boxes) of the current module as its own states, and it allows, for example, connecting an output of a module to its own input.

Example 1. *Consider a simple voting procedure with two agents: Voter* (v) *and Election Authority* (ea)*. The procedure consists of three main steps: registration, ballot filling, and vote casting, allowing two registration methods (online and onsite). To manage this complexity, we employ a hierarchical structure: the registration and ballot filling steps are encapsulated as separate, reusable models, while a top-level module coordinates the overall process. This hierarchical decomposition is fundamental, as it allows us to abstract away internal details, verify components independently, and reuse modules across different contexts.*

The HCGM for the voting example $\mathcal{M}_{vote} = (\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3)$ *is depicted in Figure 1. For simplicity, we specify only the topmost module* M_1*. It has two agents* $\mathbb{A}\mathrm{gt}_1 = \{v, ea\}$*, four states* $St_1 = \{s_1^1, s_2^1, s_3^1, s_4^1\}$*, one input* $\mathsf{In}_1 = \{in_1\}$*, and no outputs, i.e.* $\mathsf{Out}_1 = \emptyset$*. Within* M_1*, three boxes* $\mathsf{B}_1 = \{b_1, b_2, b_3\}$ *are mapped to submodules:* b_1 *and* b_2 *to the registration module* M_2*, and* b_3 *to the ballot filling module* M_3*. Crucially, the hierarchy permits* M_2 *and* M_3 *to be defined over different sets of agents, reflecting the natural assumption that only the voter participates in ballot filling.*

In the figure, for readability, we simplify the transition function, labeling arrows with intuitive agent actions. Thus, from the input in_1*, the voter can apply to register online, entering box* b_1*. If registration fails, she may attempt onsite registration via* b_2*. Repeated failure leads to sink state* s_1^1*, while success (ok) advances to* s_2^1*. From there, the voter can fill the ballot inside* b_3*, and upon exit, vote to reach* s_3^1*. Here, she may revote (returning to* s_2^1*), or finish the voting procedure. This hierarchical representation of the voting procedure reuses modules in order to elegantly separate different*

concerns and subsystems, allowing us to focus on the high-level execution logic in the top module.

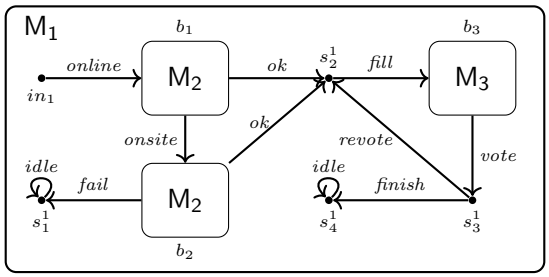

Figure 1: HCGM $\mathcal{M}_{vote}$. The top-level module M_1 contains three boxes: b_1 and b_2 for registration, and b_3 for ballot filling. The internal structures of M_2 and M_3 are omitted for simplicity.

To determine the truth-value of ATL formulas on HCGMs, we define a flattening of a given HCGM into an equivalent CGM, on which formulas are then evaluated.

Definition 7 (Execution semantics for HCGM)**.** *Given an HCGM $\mathcal{M} = \langle \mathsf{M}_1, ..., \mathsf{M}_k \rangle$, we define the corresponding* flat *CGM as $flat(\mathcal{M}) = (\mathbb{A}\mathrm{gt}, St, Act, act, o, V)$ with:*

- $\mathbb{A}\mathrm{gt} = \bigcup_i \mathbb{A}\mathrm{gt}_i$ *collects the agents from all the modules, with $n = |\mathbb{A}\mathrm{gt}|$ denoting the overall number of agents.*
- $St = S_1$, *where $S_i = St_i \cup \mathsf{In}_i \cup \mathsf{Out}_i \cup \bigsqcup_{b \in \mathsf{B}_i} S_{\mathsf{Map}_i(b)}$ recursively defines the set of states corresponding to module M_i and all boxes of M_i. Note that the recursion is well defined, because the nesting of modules in a HCGM is acyclic, and hence (in at most k steps) the recursive rule must encounter a module for which $\mathsf{B}_i = \emptyset$.*
- $Act = \bigcup_i \mathsf{Act}_i \cup \{\epsilon\}$ *collects the actions from all modules, plus the "idle" action ϵ that will be assigned to agents absent in the present module.*
- $act : St \times \mathbb{A}\mathrm{gt} \to 2^{Act} \setminus \emptyset$ *is such that $act(s, a_j) = \mathsf{act}_i(s, a_j)$ if $s \in St_i \cup \mathsf{In}_i \cup \mathsf{Out}(\mathsf{B}_i)$ for some $i \in [1, .., k]$ and $a_j \in \mathbb{A}\mathrm{gt}_i$, and $act(s, a_j) = \{\epsilon\}$ otherwise.*
- *Given an action profile $\alpha = (\alpha_1, ..., \alpha_n)$, let $active(\alpha)$ be the same as α, but with all the ϵ actions removed from it. Then, $o : \{(s, \alpha_1, ..., \alpha_n) \mid s \in St, \alpha_j \in act(s, a_j)\} \to St$ is such that $o(s, \alpha_1, ..., \alpha_n) = \mathsf{o}_i(s, active(\alpha_1, ..., \alpha_m))$ for $s \in S_i, i \in [1, .., k]$. Note that the function is well-defined, as every decision point in $flat(\mathcal{M})$ is unique.*

To flatten an HCGM, we first collect all agents appearing in its modules. The set of states of the flat CGM is defined recursively as a disjoint union of states, inputs, and outputs of modules at all levels. Observe that the same module may appear multiple times as a box, and, therefore, we use disjoint union to distinguish different occurrences. The overall set of actions is the union of actions across all modules.

Since different modules may involve different sets of agents, we introduce the special action ϵ to handle agents that are inactive in a given module. Such agents are assigned only the action ϵ, ensuring that action profiles remain well-defined in the flattened model. This also allows us to have a total transition function, where we can focus only on agents that are active in a given module, and the action ϵ performed by the agents inactive in the given module does not influence the outcome of the function.

Finally, observe that due to the fact that multiple boxes can be assigned to the same module, the size of $flat(\mathcal{M})$ for a given $\mathcal{M}$, even with single inputs, can, in the worst case, be exponential w.r.t. $|\mathcal{M}|$ and the nesting depth $nd(\mathcal{M})$ of $\mathcal{M}$, i.e., the number of consecutive embeddings of modules into one another. Hence, similarly to (Alur and Yannakakis 2001), the size of $flat(\mathcal{M})$ is bounded by $O(|\mathcal{M}|^{nd(\mathcal{M})})$.

4 Model Checking

Given a formula $\varphi \in \mathsf{ATL}$, an HCGM $\mathcal{M}$, and an input in of the topmost module, one can first construct $flat(\mathcal{M})$ and apply the standard ATL model checking algorithm (Alur, Henzinger, and Kupferman 2002), which runs in polynomial time in the size of the model. However, since $flat(\mathcal{M})$ may be exponential in the nesting depth, this yields an overall complexity of $O(|\varphi| \cdot |\mathcal{M}|^{nd(\mathcal{M})})$. However, in practice, models are typically orders of magnitude larger than formulas (see, e.g., (Jamroga, Knapik, and Kurpiewski 2018; Ezekiel et al. 2011)), making this approach undesirable. We therefore propose a direct model checking procedure that operates on the hierarchical structure.

The algorithm is a multi-agent strategic extension of the CTL (Clarke and Emerson 1981) model checking algorithm from (Alur and Yannakakis 2001), and the reader is referred to the paper for more intuitions and examples.

Algorithm 1 Model checking ATL on HCGMs

procedure MC($\mathcal{M} = (\mathsf{M}_1, ..., \mathsf{M}_k), \varphi$)
 for $\psi \in sub(\varphi)$ **do**
  case $\psi = \langle\langle C \rangle\rangle \mathbf{X} \chi$
   for $i \in [k, ..., 1]$ **do**
    $M_i \leftarrow$ HCGM2CGM(M_i)
    ATLMC($M_i, \langle\langle C \cap \mathbb{A}\mathrm{gt}_i \rangle\rangle \mathbf{X} \chi$)
   $\mathcal{M} \leftarrow$ SPLIT($\mathcal{M}, \psi$)
  case $\psi = \langle\langle C \rangle\rangle \psi_1 \mathbf{U} \psi_2$
   $\mathcal{M} \leftarrow$ SUMMARY($\mathcal{M}, \psi$)
   $\mathcal{M} \leftarrow$ SPLIT($\mathcal{M}, \psi$)
 $X \leftarrow \emptyset$
 for $in \in \mathsf{In}_1$ **do**
  if in is labelled with φ **then**
   $X \leftarrow X \cup \{in\}$
 return X
end procedure
procedure SPLIT($\mathcal{M}, \psi$)
 $\mathcal{M}' \leftarrow ()$
 for $\mathsf{M}_i \in \mathcal{M}$ **do**
  for $j \in [0, ..., 2^{\mathsf{Out}_i} - 1]$ **do**
   $\mathsf{M}_i^{\mathrm{BIN}(j)} = \mathsf{M}_i[\mathsf{Map}_i \mapsto \mathsf{Map}_i^{\mathrm{BIN}(j)}]$
   for $bit \in |\mathrm{BIN}(j)|$ **do**
    if $\mathrm{BIN}(j)[bit] = 1$ **then**
     label $out_{bit+1} \in \mathsf{Out_i}$ with ψ
   for $b \in \mathsf{B}_i$ **do**
    $v \leftarrow 0^{|\mathsf{Out}_{\mathsf{Map}_i(b)}|}$
    for $l \in |\mathsf{Out}_{\mathsf{Map}_i(b)}|$ **do**
     if $\psi = \langle\langle C \rangle\rangle \mathbf{X} \chi$ **then**
      if out_l is labelled with ψ **then**
       $v[l] \leftarrow 1$
     else if $\psi = \langle\langle C \rangle\rangle \psi_1 \mathbf{U} \psi_2$ **then**
      $M_i \leftarrow$ HCGM2CGM(M_i)
      $\theta_1 = \langle\langle C \cap \mathbb{A}\mathrm{gt}_i \rangle\rangle \mathbf{X} \langle\langle C \cap \mathbb{A}\mathrm{gt}_i \rangle\rangle (\psi_1 \vee \mathrm{MB}) \mathbf{U} (\psi_2 \vee \mathrm{Y})$

34: $\theta_2^{out} = \langle\langle C \cap \mathbb{A}\mathrm{gt}_i\rangle\rangle \mathbf{X}\langle\langle C \cap \mathbb{A}\mathrm{gt}_i\rangle\rangle(\psi_1 \vee \mathrm{MB})\mathbf{U}(\psi_2 \vee \mathrm{Y} \vee (out \wedge \psi_1))$
35: **if** (out_l is not labelled with ψ and $out_l \in \mathrm{ATLMC}(M_i, \theta_1)$) or ($out_l$ is labelled with ψ and $out_l \in \mathrm{ATLMC}(M_i, \theta_2^{out})$ for some $out \in \mathsf{Out}_i$) **then**
36: $v[l] \leftarrow 1$
37: $\mathsf{Map}_i^{\mathrm{BIN}(j)}(b) = \mathsf{Map}_i(b) + \mathrm{INT}(v)$
38: $\mathcal{M}' \leftarrow \mathrm{APPEND}(\mathcal{M}', \mathsf{M}_i^{\mathrm{BIN}(j)})$
39: **return** $\mathcal{M}'$
40: **end procedure**
41: **procedure** SUMMARY($\mathcal{M}, \psi$)
42: **for** $i \in [k, ..., 1]$ **do**
43: $M_i \leftarrow$ HCGM2CGM(M_i)
44: **if** $\psi = \langle\langle C\rangle\rangle \psi_1 \mathbf{U} \psi_2$ **then**
45: **for** $in_i \in \mathsf{In}_i$ **do**
46: $\theta_1 = \langle\langle C \cap \mathbb{A}\mathrm{gt}_i\rangle\rangle(\psi_1 \vee \mathrm{MB})\mathbf{U}(\psi_2 \vee \mathrm{Y})$
47: $\theta_2^{out} = \langle\langle C \cap \mathbb{A}\mathrm{gt}_i\rangle\rangle(\psi_1 \vee \mathrm{MB})\mathbf{U}(\psi_1 \wedge out)$
48: **if** $in_i \in \mathrm{ATLMC}(M_i, \theta_1)$ **then**
49: label in_i with Y
50: **else if** $in_i \in \mathrm{ATLMC}(M_i, \theta_2^{out})$ for $out \in \mathsf{Out}_i$ **then**
51: label in_i with (MB, out)
52: **else**
53: label in_i with N
54: **return** $\mathcal{M}$
55: **end procedure**
56: **procedure** HCGM2CGM(M_i)
57: $S_i = St_i \cup \mathsf{In}_i \cup \mathsf{Out}_i \cup \mathsf{Out}(\mathsf{B}_i) \cup \mathsf{In}(\mathsf{B}_i)$
58: $o^\star \leftarrow \emptyset$
59: **for** $s \in S_i$ **do**
60: **if** $s \in St_i \cup \mathsf{In}_i \cup \mathsf{Out}(\mathsf{B}_i)$ **then**
61: $o^\star(s) \leftarrow \mathsf{o}_i(s)$
62: **else if** $s \in \mathsf{In}(\mathsf{B}_i)$ and s is labelled with (MB, out) **then**
63: $o^\star(s) \leftarrow \{((\alpha, ..., \alpha, out)\}$ with $|\alpha, ..., \alpha| = |\mathbb{A}\mathrm{gt}_i|$
64: $M_i \leftarrow (\mathbb{A}\mathrm{gt}_i, S_i, \mathsf{Act}_i, \mathsf{act}_i, o^\star, \mathsf{V}_i \cup \mathsf{V}(\mathsf{Out}(\mathsf{B}_i)) \cup \mathsf{V}(\mathsf{In}(\mathsf{B}_i)))$
65: **return** M_i
66: **end procedure**

Given an HCGM $\mathcal{M}$ and a formula $\varphi \in \mathsf{ATL}$, Algorithm 1 returns the set of inputs of the outermost module that satisfy φ. The main difference of model checking HCGMs compared to the standard CGMs is that we need to deal with boxes. In particular, whether a strategic formula is satisfied by an output of a box is dependent on the context, i.e., the module the given box is a part of as well as where the arrows from outputs of the box go. So, following (Alur and Yannakakis 2001), for the cases of strategic modalities the model checking procedure not only returns a labelling but a new HCGM as well. This new HCGM will contain copies of modules that are mapped to boxes and such that there is a copy for each possible assignment of a given formula to a module's outputs. In Algorithm 1, we omit Boolean cases for brevity and skip the **R**elease case, which is similar to **U**ntil. W.l.o.g., we assume that each input in_j and output out_j is labelled with the corresponding unique proposition.

Case $\psi = \langle\langle C\rangle\rangle \mathbf{X}\chi$. For each module M_i (bottom-up), we build a (partial) CGM M_i via HCGM2CGM, where transitions from Out_i and $\mathsf{In}(\mathsf{B}_i)$ are left undefined since they are handled by the parent and child modules correspondingly. We then run standard ATL model checking on M_i with the *local* coalition $C \cap \mathbb{A}\mathrm{gt}_i$, i.e., we compute $\mathrm{ATLMC}(M_i, \langle\langle C \cap \mathbb{A}\mathrm{gt}_i\rangle\rangle \mathbf{X}\chi)$ and propagate the resulting labels. Agents in $C \setminus \mathbb{A}\mathrm{gt}_i$ are inactive in M_i. To account for the dependence of box-outputs on context, we apply SPLIT, which replaces each module M_i by $2^{|\mathsf{Out}_i|}$ copies, one for each possible truth assignment of ψ over its outputs. We use functions BIN and INT to switch between the distribution of outputs that satisfy ψ and the module that the current box should be mapped to. E.g., $str = 00101$ means that out_1 and out_3 of the current box are labelled with ψ, and thus this box would be mapped to module $\mathsf{Map}_i(b) + \mathrm{INT}(str)$, which is the fifth copy of module $\mathsf{Map}_i(b)$.

Case $\psi := \langle\langle C\rangle\rangle(\psi_1 \mathbf{U} \psi_2)$**.** We first compute, bottom-up, an interface summary for each module by classifying its inputs into three sets: *YES* (the coalition can reach a ψ_2-state within the module while maintaining ψ_1), *MAYBE* (the coalition can maintain ψ_1 until exiting the module, so ψ_2 may be reached above), and *NO* otherwise. This is implemented in SUMMARY by model checking suitable ATL formulas on the CGM M_i with coalition $C \cap \mathbb{A}\mathrm{gt}_i$ (we additionally label inputs with Y/MB/N remembering the outputs that satisfied MB and treating boxes as states with these labels). We then apply SPLIT, using the computed labels to update box mappings. To determine the mapping for the copies, we use formulas θ_1 and θ_2^{out} that check whether from the given output the active agents in the coalition can satisfy the **U**ntil formula within the current module, or whether the truth of ψ_1 can be maintained till one of the exits out of the module (and ψ_2 can potentially be satisfied in one of the parent modules).

Case $\psi := \langle\langle C\rangle\rangle(\psi_1 \mathbf{R} \psi_2)$**.** Analogous to **U**ntil, with the following minor modifications. First, in SUMMARY, we label an input with Y, if ψ_2 can be maintained in the current module until we reach a ψ_1-state. Otherwise, if ψ_2 can be maintained throughout the module and we successfully reach one of its exits, we label the corresponding input with MB. Otherwise, we label the input with N.

Complexity. Each strategic modality may split a given HCGM into at most 2^d copies, where d is the maximal number of outputs of a module. Such a splitting occurs for at most $|\varphi|$ strategic subformulas. Hence, the overall running time of the algorithm is bounded by $O(|\mathcal{M}| \cdot 2^{|\varphi| d})$. Moreover, it uses only polynomial space.

Theorem 1. *Model checking* ATL *for HCGMs is PSPACE-complete w.r.t. the sizes of an HCGM and a formula.*

Proof. The argument here follows the argument for CTL given in (Alur and Yannakakis 2001, Section 4.1.5). Indeed, for a given HCGM $\mathcal{M}$ and formula φ of ATL we can run a model checking procedure on $flat(\mathcal{M})$. As noted we noted above, the size of such a structure is in $O(|\mathcal{M}|^{nd(\mathcal{M})})$, i.e. exponential. However, we do not need to keep the full structure in memory. For each possible execution path of φ, we only need space which is bounded by $O(nd(\mathcal{M}) \cdot |\mathcal{M}|)$, i.e. we flatten only the modules that the execution path enters. To check the exponential number of executions paths of φ, we reuse this space, checking paths one by one. The lower bound follows from the facts that hierarchical CTL model checking (Alur and Yannakakis 2001) is PSPACE-complete and that CTL can be seen as a fragment of ATL. □

No less importantly, the practical complexity of our algorithm, while exponential, is much more preferable than the complexity of $O(|\varphi| \cdot |\mathcal{M}|^{nd(\mathcal{M})})$ obtained by verification via flattening. This is because (i) properties for model checking are usually given by *short* ATL formulas, and (ii)

it is a good design practice to use only boxes with few exits. Indeed, regarding (i), as we noticed at the beginning of this section, models are typically orders of magnitude larger than formulas. Regarding (ii), notice that a box conceptually corresponds to a function specification in a programming language. Entries define the possible inputs, and exits the possible outputs of a function. Boxes with few exits model functions with a small output type, e.g., Boolean. This is natural in many practical situations, for instance for all verification/authentication tasks that return only true/false or grant/deny or protocols that return ack/nack/timeout.

Theorem 2. *Model checking* ATL *for formulas of bounded length over HCGMs that include only modules with a bounded amount of exits is P-complete, and can be done in linear time w.r.t. the size of the HCGMs.*

Observe that even under the conditions of Theorem 2, flattening of an HCGM can lead to an exponential blow-up. Consider an HCGM with n modules, such that every module M_i is defined by a sequence of $n - i$ boxes (each with only 1 entry and 1 exit), assigned to modules $\mathsf{M}_{i+1}, \mathsf{M}_{i+2}, ..., \mathsf{M}_n$ respectively. Then, the flattening results in a structure with exponentially many states, while our hierarchical model checking algorithm still runs in P.

5 Conclusion

Many real-world multi-agent systems are intrinsically hierarchical, with recurring components specified once and reused across contexts. A naive verification workflow flattens the hierarchy into a single concurrent game model, but this can cause an exponential blow-up in the nesting depth, quickly making strategic analysis impractical.

In this work, we introduce *hierarchical concurrent game models* (HCGMs), define an unfolding-based semantics, and develop an ATL model-checking procedure that operates *directly on the hierarchy*. We thus extend the results of (Alur and Yannakakis 2001) to the setting of strategically interacting agents, and show that we have the same complexity for the more expressive framework.

As immediate future work, we plan to implement and evaluate our procedure on realistic benchmarks, following the tool-driven line for hierarchical single-agent systems (Alur, McDougall, and Yang 2002; Alur, Grosu, and McDougall 2000; Ferrando et al. 2023; Wasowski 2004). One way to go about this is by extending the existing ATL model checkers, like MCMAS (Lomuscio, Qu, and Raimondi 2017), STV (Kurpiewski, Jamroga, and Knapik 2019), and VITAMIN (Ferrando and Malvone 2025). Along similar lines, we plan to investigate abstractions (Belardinelli et al. 2023) for HCGMs to further explore the practical use of hierarchical models of MAS.

On the conceptual level, it is intriguing to consider more general settings by, e.g., extending our work to more expressive strategic logics like ATL* (Alur, Henzinger, and Kupferman 2002) and Strategy Logic (SL) (Mogavero, Murano, and Vardi 2010), as well as incorporating imperfect information, perfect recall, communication between modules, asynchronous executions, and so on.

Acknowledgments

The contribution of W. Jamroga and D. Kurpiewski has been co-financed by the Polish Minister of Science and Higher Education under the programme "Excellence Initiative – Research University."

AI Declaration

The authors have employed Generative AI tools to assist in improving and polishing the English language in the descriptive sections of the text.

References

Alur, R., and Yannakakis, M. 2001. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems* 23(3):273–303.

Alur, R.; Grosu, R.; and McDougall, M. 2000. Efficient reachability analysis of hierarchical reactive machines. In Emerson, E. A., and Sistla, A. P., eds., *Proceedings of the 12th CAV*, volume 1855 of *LNCS*, 280–295. Springer.

Alur, R.; Henzinger, T.; and Kupferman, O. 2002. Alternating-time temporal logic. *Journal of the ACM* 49(5):672–713.

Alur, R.; Kannan, S.; and Yannakakis, M. 1999. Communicating hierarchical state machines. In Wiedermann, J.; van Emde Boas, P.; and Nielsen, M., eds., *Proceedings of the 26th ICALP*, volume 1644 of *LNCS*, 169–178. Springer.

Alur, R.; McDougall, M.; and Yang, Z. 2002. Exploiting behavioral hierarchy for efficient model checking. In Brinksma, E., and Larsen, K. G., eds., *Proceedings of the 14th CAV*, volume 2404 of *LNCS*, 338–342. Springer.

Aminof, B.; Kupferman, O.; and Murano, A. 2012. Improved model checking of hierarchical systems. *Information and computation* 210:68–86.

Belardinelli, F.; Ferrando, A.; Jamroga, W.; Malvone, V.; and Murano, A. 2023. Scalable verification of strategy logic through three-valued abstraction. In *Proceedings of the 32nd IJCAI*, 46–54. ijcai.org.

Bozzelli, L.; Murano, A.; Perelli, G.; and Sorrentino, L. 2020. Hierarchical cost-parity games. *Theoretical Computer Science* 847:147–174.

Clarke, E. M., and Emerson, E. A. 1981. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Kozen, D., ed., *Logics of Programs*, volume 131 of *LNCS*, 52–71. Springer.

de Alfaro, L., and Henzinger, T. A. 2001. Interface automata. In *Proceedings of SIGSOFT*, 109–120. ACM.

Ezekiel, J.; Lomuscio, A.; Molnar, L.; and Veres, S. M. 2011. Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In Walsh, T., ed., *Proceedings of the 22nd IJCAI*, 1659–1664. IJCAI/AAAI.

Faran, R., and Kupferman, O. 2019. A parametrized analysis of algorithms on hierarchical graphs. *International Journal of Foundations of Computer Science* 30(06n07):979–1003.

Ferrando, A., and Malvone, V. 2025. VITAMIN: A compositional framework for model checking of multi-agent systems. In Rocha, A. P.; Steels, L.; and van den Herik,

H. J., eds., *Proceedings of the 17th ICAART*, 648–655. SCITEPRESS.

Ferrando, A.; Malvone, V.; Murano, A.; and Stranieri, S. 2023. HYASM: A tool to verify hierarchical systems. In *Proceedings of the 31st WETICE*, 1–6. IEEE.

Jamroga, W.; Knapik, M.; and Kurpiewski, D. 2018. Model checking the SELENE e-voting protocol in multi-agent logics. In *Proceedings of the 3rd E-Vote-ID*, volume 11143 of *LNCS*, 100–116. Springer.

Kupferman, O.; Vardi, M.; and Wolper, P. 2001. Module checking. *Information and Computation* 164:322–344.

Kurpiewski, D.; Jamroga, W.; and Knapik, M. 2019. STV: model checking for strategies under imperfect information. In Elkind, E.; Veloso, M.; Agmon, N.; and Taylor, M. E., eds., *Proceedings of the 18th AAMAS*, 2372–2374. IFAAMAS.

Lomuscio, A.; Qu, H.; and Raimondi, F. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer* 19(1):9–30.

Mogavero, F.; Murano, A.; and Vardi, M. Y. 2010. Reasoning about strategies. In Lodaya, K., and Mahajan, M., eds., *Proceedings of the 30th FSTTCS*, LIPIcs, 133–144. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Murano, A.; Napoli, M.; and Parente, M. 2008. Program complexity in hierarchical module checking. In Cervesato, I.; Veith, H.; and Voronkov, A., eds., *Proceedings of the 15th LPAR*, volume 5330 of *LNCS*, 318–332. Springer.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th FOCS*, 46–57. IEEE Computer Society.

Queille, J., and Sifakis, J. 1982. Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M., and Montanari, U., eds., *Proceedings of the 5th Symposium on Programming*, volume 137 of *LNCS*, 337–351. Springer.

Stefansson, E., and Johansson, K. H. 2023. Hierarchical finite state machines for efficient optimal planning in large-scale systems. In *Proceedings of the 21st ECC*, 1–8. IEEE.

Wasowski, A. 2004. Flattening statecharts without explosions. In Whalley, D. B., and Cytron, R., eds., *Proceedings of LCTES*, 257–266. ACM.

Yamane, S. 2025. Specification and verification method of parallel hierarchical timed automata by predicate abstraction and refinement. *IEEE Access* 13:90017–90033.