Emerging Synchrony in Applauding Audiences: Formal Analysis and Specification*

Luca Di Stefano[1] (✉) [0000−0003−1922−3151] and Omar Inverso[2] [0000−0002−9348−1979]

[1] TU Wien, Institute of Computer Engineering, Treitlstraße 3, 1040 Vienna, Austria
[2] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
luca.di.stefano@tuwien.ac.at

Abstract. Applause is an ancient, widespread collective behaviour by which an audience expresses appreciation at the conclusion of a collective event such as an artistic performance or a public ceremony. In some cultures, it is possible to observe a spontaneous transition from an initially incoherent to a surprisingly synchronised form of applause. Such kind of emergent behaviour has long since fascinated researchers from different disciplines. This paper shows a possible application of formal methods to study similar phenomena. The key idea is to model the audience as a concurrent system, where each person is a separate process that follows the same, simple behaviour. The model can then be automatically analysed to study the possible evolutions of the system as a whole, and in particular to assess the likelihood of emerging synchronisation.

1 Introduction

The formal representation of systems of interacting components, as a first step towards their rigorous and possibly automated analysis, has long since been a primary concern in research on concurrency theory. Arising from generalisations of the classic theory of languages and automata for sequential systems, process algebras [11] have been an invaluable asset in this respect.

Owing to the fact that, arguably, no single formalism can capture every form of concurrent computation to a suitable degree of clarity and elegance [31], a number of process algebras have been proposed over the years [5, 23, 30]. More sophisticated formalisms focus on specific aspects of particular classes of systems, such as *mobility*, which allows to define dynamic communication structures [32], and *locality* of interaction [17].

Novel formalisms and mechanisable workflows have recently been proposed to support the study of active subjects of research in different scientific disciplines.

* This work is dedicated to Rocco De Nicola on the occasion of his 70th birthday. Luca is very thankful for having been supervised by Rocco during his doctoral studies and for the enduring collaboration. Omar would like to express his gratitude to Rocco for the opportunities of joint work (including Luca's supervision) and the precious advice. Rocco's work and vision are a continuing source of inspiration for both authors.

Process algebras were already used over two decades ago to model some aspects of the social behaviour of insects [36, 38], but with limited possibilities of carrying out any automated analysis. Researchers in biology, complexity science, and other fields are now slowly becoming increasingly aware and appreciative of the potential benefits of applying formal models of concurrency to their scenarios of interest [4, 6]. More recent contributions following up on this line of research aim at supporting the study of *emergent behaviour* in complex natural or artificial systems. Modern formal languages to naturally express aspects such as *attribute-based communication* [1, 3, 18], which allows for dynamic communication groups as occurring in e.g. social networks [15], or *stigmergic interaction* [12], typical of some biological systems, have facilitated the formal specification and analysis of different classes of systems, such as matching protocols like stable marriage [15, 16], flocking birds [14], and the collective foraging behaviour of ants [13].

Following up on this trend, in this paper we consider the well-known collective behaviour that manifests itself in an applauding audience at the end of a collective event, such as an artistic performance or a public ceremony. We focus on the fascinating phenomenon whereby, from an initial phase of incoherent clapping, the people in the audience end up clapping in unison [33, 34, 37]. Our main goal is to demonstrate once again how the methodologies originally developed for the study of concurrent systems may be profitably applied to undertake mechanised reasonings on phenomena of this kind. To do so, we gradually introduce a model of a clapping audience, then turn the model into formal specifications amenable to mechanised analysis. We rely on formal analysis to understand whether such a model would be able to reproduce anything similar to the mentioned emergent synchrony observed in real audiences.

The paper is structured as follows. Section 2 introduces our model. Section 3 describes the analysis workflow to study our model, and presents our experimental evaluation. Section 4 contains our conclusions and ideas for future research, and discusses some literature related to the phenomenon of synchronised applause and its modelling.

2 Modelling the audience

Let us consider an audience of N people. Initially, everyone claps at his own pace. At the same time, people listen to each other, in the attempt to collectively reach a certain rhythm. The idea is that multiple claps occurring simultaneously will produce a louder sound, which every person can use as a reference to adjust his own pace. We would like to assess whether the whole audience would be able to achieve a synchronised clapping, if every person were to follow this kind of behaviour.

We model the audience as a collection of agents. These agents evolve in steps, which intuitively correspond to the actual moments of time when their actions take place. Notice that the steps do not correspond to actual timestamps, but are merely functional to providing a logical ordering of the actions.

All the agents behave in the same way. At each step, an agent may or may not clap, depending on his current clapping pace; at the same time, the agent pays attention to the rest of the audience. When enough agents are clapping together, so that a loud enough sound is produced, the agent may adjust his way of clapping in an attempt to get closer to that of the others.

Let us use a pseudo-formal notation to represent this behaviour:

$$\textsc{Behavior} \triangleq \{\textsc{Listen};\ \textsc{MaybeClap};\ \textsc{Adjust}\};\ \textsc{Behavior}$$

Semicolons denote a sequence of actions. Listen represents the agent's ability to pay attention to the rest of the audience. MaybeClap is when the agent decides whether or not to clap. Adjust represents the agent's potential attempt to make his pace more in sync with that of the audience. The tree operations are enclosed in braces to indicate that they take place in the same time step. Lastly, the agent repeats these operations indefinitely.

We will soon explain in further detail how we implement each of the actions above. But first, let us introduce the parameters in which our model will be specified and the variables that make up the state of our agents.

State variables and parameters. Let us assume that each agent has an identifier $i \in \{0, \dots, N-1\}$. Each agent i has its own clapping *period* stored in a variable T_i. Intuitively, this means that the agent would clap his hands exactly every T time steps, if it were not influenced by others. In order for an agent to "know" when it is the right moment to clap, we equip it with a clapping *counter* c_i that is regularly decremented and incremented. If c_i reaches a value of 0, it means that the agent i is clapping during that step. The increment or decrement of c_i is governed by a third variable, named $sign_i$. Later on, we will omit indices whenever they are clear from the context.

As stated above, our agents listen to the audience and can detect louder sounds (resulting from many agents clapping at the same time) which they use as a reference to synchronise their clapping. For that reason, each agent uses a variable *loud*, which contains a reference value to distinguish a loud event from a quieter one. Lastly, we equip agents with a counter *tSinceLoud* tracking the number of steps passed since a loud sound was last detected.

Some of the variables above are initialised to fixed values: namely, $sign = 1$ and $tSinceLoud = -\infty$. The others are parameterised, which allows us to tune their initial value and explore how different values affect the system's dynamics. Specifically, the initial clapping period of each agent will be nondeterministically chosen from the interval between two parameters *Tmin* and *Tmax*, and may vary between agents. Setting a lower value of *Tmin* allows for agents that clap faster; higher values of *Tmax*, in turn, allow agents that clap more slowly. The range of allowed values ($Tmax - Tmin$) also affects how chaotic the initial state may be. Trivially, when $Tmin = Tmax$, agents all start with the same clapping period. A broader range allows for a less uniform audience.

Variable *loud* is also initialised according to a parameter $loud^{(0)}$, and will be the same for the entire audience. When $loud^{(0)}$ is low, agents are more inclined

Table 1. Variables and parameters in our model.

Variables		
Name	Meaning	Initial values
T	clapping period: the agent claps every T time steps	$\mathit{Tmin}, \dots, \mathit{Tmax}$
c	clapping counter: when 0, the agent is clapping	$1, \dots, T/2$
sign	determines whether c is increasing or decreasing	1
loud	determines whether the audience is clapping loudly	$\mathit{loud}^{(0)}$
tSinceLoud	number of steps since the last loud clap	$-\infty$

Parameters		
Name	Meaning	Constraints
N	Number of agents	Positive
$\mathit{Tmin}, \mathit{Tmax}$	Range of initial values for T	Positive, $\mathit{Tmin} < \mathit{Tmax}$
$\mathit{loud}^{(0)}$	Initial value of *loud*	Positive

to be influenced by the audience. For instance, when $\mathit{loud}^{(0)} = 2$, any two agents clapping at the same time will be interpreted as a loud event. Lastly, the number of agents itself N is also a parameter. All parameters are assumed to be positive, with $\mathit{Tmin} < \mathit{Tmax}$. We summarise the parameters in Table 1.

Sensing loud claps. During a Listen operation, an agent detects whether the audience is clapping "loudly" during the current time step. To do so, the agent simply counts how many agents have just clapped (i.e. their counter c is currently set to 0) and compares the count to *loud*, which acts as a threshold. The agent then resets *tSinceLoud* if the threshold is reached or exceeded, or increments it otherwise.

$$\begin{aligned}\textsc{Listen} \triangleq\ & \mathit{audienceClap} \leftarrow \#\{i \in \{0, \dots, N-1\} \mid c_i = 0\}\,; \\ & \mathit{tSinceLoud} \leftarrow 0 \text{ if } \mathit{audienceClap} \geq \mathit{loud} \text{ else } \mathit{tSinceLoud} + 1\end{aligned}$$

Clapping. The clapping logic is rather simple, as it just concerns the update of c in a regular pattern. Specifically, the agent just updates c to the result of $c + \mathit{sign}$. As stated earlier, *sign* is initially set to 1, so c is initially increasing. When c reaches $T \div 2$, *sign* is flipped to -1. Thus, from the next time step, c will decrease. When c hits 0 (and the agent claps), is flipped back to 1 and the counter starts increasing again, and so on. Thus, plotting the value of c over time results in a triangle wave of period T.

$$\begin{aligned}\textsc{MaybeClap} \triangleq\ & \mathit{sign} \leftarrow 1 \text{ if } c = 0 \text{ else } -1 \text{ if } c = T \div 2 \text{ else } \mathit{sign}\,; \\ & c \leftarrow c + \mathit{sign}\end{aligned}$$

Adjusting the rhythm. In the last part of their behaviour, agents may perform several adjustments to try and synchronise with the rest of the audience; we will describe them one at a time.

First, we let agents adjust their own clapping *period* by using the time interval between loud claps ($\mathit{tSinceLoud}$) as an approximation of the synchronous applause period. Let us assume that agents have always access to their *previous* value of this variable, and let us call this value $\mathit{tSinceLoud}^{prev}$.

In practice, when an agent hears a loud clap, he computes the average between $\mathit{tSinceLoud}^{prev}$ and his current period T, bounds the result between Tmin and Tmax, and sets his new period to the final result. The first loud clap is not used to adjust the period because, at that point in time, $\mathit{tSinceLoud}^{prev}$ still has a value of $-\infty$; intuitively, this models the fact that at least *two* loud claps have to happen before agents can adjust their pace.

$$\begin{aligned}\textsc{Adjust} \triangleq\ & T \leftarrow T \text{ if } \mathit{tSinceLoud} = 0 \lor \mathit{tSinceLoud}^{prev} = -\infty \\ & \qquad\qquad \text{else } (T + \mathit{tSinceLoud}^{prev}) \div 2\,; \\ & T \leftarrow \max(\mathit{Tmin}, \min(T, \mathit{Tmax}))\,; \\ & \ldots\end{aligned}$$

Agents may also adjust the loudness *threshold*, i.e., their value of loud. If the threshold is crossed by a large margin, agents increase it. In this way, smaller groups are prevented from influencing the agents in the future. Conversely, if no loud clap is heard for a long enough timespan, agents will start listening for weaker claps. However, we constrain this decrement to happen only once every $\mathit{Tmax} \div 2$ steps, so that the threshold does not fall too quickly towards 0. By this mechanism we aim at modelling a cooperative audience; even if they are not actively communicating, the agents still *want* to find synchrony (e.g., because it is a culturally-ingrained way to demonstrate their appreciation of the show), and therefore are willing to synchronise in smaller groups if a large one has not emerged yet.

$$\begin{aligned}\textsc{Adjust} \triangleq\ \ & T \leftarrow \ldots\,; \\ & \mathit{loud} \leftarrow \begin{cases} \mathit{loud} + 1 & \text{if } \mathit{tSinceLoud} = 0 \\ \mathit{loud} - 1 & \text{if } \exists k > 0.\ \mathit{tSinceLoud} = k(T_{max} \div 2) \\ \mathit{loud} & \text{otherwise;} \end{cases} \\ & \ldots\end{aligned}$$

Finally, we may need to slightly adjust the evolution of c. The reason is that different agents may clap with different *phase* even when they converge to the same clapping period, and thus never synchronise fully. Therefore, we need a phase-adjustment mechanism. Our solution is as follows: when an agent has heard a loud clap, but did not take part in it, then it holds his value of c for the current time step. In practice, we implement this by subtracting sign from c, reversing the previous update within $\textsc{MaybeClap}$.

$$\begin{aligned}\textsc{Adjust} \triangleq\ & T \leftarrow \ldots\,;\ \mathit{loud} \leftarrow \ldots\,; \\ & c \leftarrow c - \mathit{sign} \text{ if } \mathit{tSinceLoud} = 0 \land c \neq 1 \text{ else } c\end{aligned}$$

Limitations. Let us conclude this overview of the model by listing some of its current limitations. First, we basically treat agents as very precise oscillators. In isolation, their clap completely lack any *jitter*, i.e., they never deviate from a perfectly periodic behaviour. Additionally, all agents are assumed to clap at the same intensity, and every clap may be heard by the entire audience. So, the perceived loudness of a clap at a given time step is the same for everybody and merely corresponds to how many agents are clapping during that step. Lastly, we assume that the agents can always perfectly measure the number of agents clapping at any given time: they can never misclassify a loud clap as weak, or vice versa. These limitations are not due to the modelling tools, which fully support writing more detailed models that keep the complications above into account. For instance, we might add nondeterministic error values to every measurement the agent performs. We left these complications out because we do not believe that they would affect significantly the emergence of synchrony in this (admittedly idealised) setting.

3 Analysis

To assess whether the rules laid out in Section 2 lead to the emergence of synchronous applause, we formalised them as LAbS specifications [12] and then generated 1000 simulations using our tool SLiVER [19].

LAbS and SLiVER. LAbS is a high-level language for agent-based models. It lets one describe agents' behaviour through intuitive, compact primitives by which agents can observe and react to each other's exposed features, or *attributes*. Thus, it shares some of the motivations and principles originating in the context of attribute-based communication [2]; however, its design favours indirect interaction over explicit message passing. SLiVER, in turn, is a publicly available[3] prototype for automated analysis of LAbS specifications. It sports a modular, extensible architecture allowing reuse of different state-of-the-art verification tools for general-purpose programming languages [19, 21]. For this work, we will rely on SAT-based bounded model checking (BMC) of sequential C programs [9].

LAbS uses a fairly similar syntax to the pseudo-code snippets that we presented in Section 2, and thus implementing our model has been straightforward. We only had to extend the language by adding a **count** primitive that allows to count the number of agents satisfying a given predicate, an operation that we need to encode the Listen operation. This is a straightforward generalisation of the primitives for existential and universal quantification over agents (**exists**, **forall**), which are already part of LAbS. We are aware that this operation requires full knowledge of the system, and our motivations for adding it to the language follows the same argument by which we justified upon introducing **exists**, **forall**, and other similar constructs [14]. In short, these constructs allow us to compactly represent observations that we assume an agent can perform in a real-world context. In our example, the ability to count simply models the fact

[3] https://github.com/labs-lang/sliver

that agents are able to listen to their surroundings. We are aware that these operations would be hard to realise in a traditional distributed setting, but at the moment we prioritise ease of modelling over these concerns.

LAbS specifications. Fig. 1 shows the LAbS specifications that we derived from the model of Section 2. We only resort to minimal embellishments (such as using mathematical notation for comparison and assignment operators, and a different font for parameters) over the actual, machine-readable specification, mostly for sake of compactness. In brief, a LAbS specification always starts with a **system** section that declares external parameters and defines the composition of the system (lines 1–4). This is followed by the definition of agents, with their internal state (called *interface*) and their behaviour (lines 6–59). Notice that each variable in the interface must be initialised, but the initial value may be nondeterministically picked from a range (or a list of alternatives, not shown here). Then, two more sections follow. In the **assume** section, we can provide predicates to constrain the initial state of the system, while in the **check** section we may specify properties of interest.

The specifications closely follow our model, with a few minor changes:

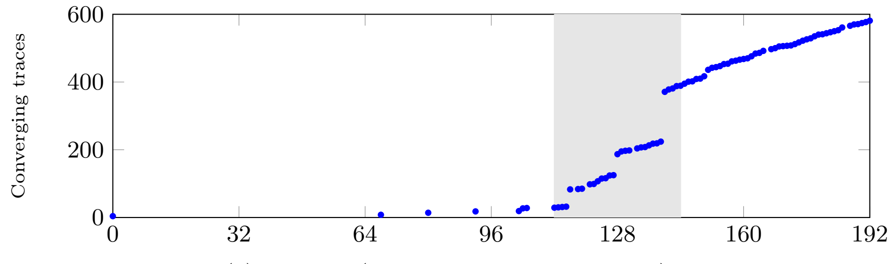
- We shorten some variable names: namely, we use tsl for *tSinceLoud* and audience for *audienceClap*.
- To implement the loudness threshold adjustment, we make use of a helper variable loud_decr that tells us when is the time to decrement the threshold.
- Agents store the clapping *half-period* tau=$T/2$ instead of the period T. This makes parts of the specifications shorter, but is otherwise equivalent to the already discussed model. Accordingly, the specifications are parameterised in $tau_{min}{=}Tmin/2$ and $tau_{max}{=}Tmax/2$ rather than in $Tmin$, $Tmax$.
- We split the Adjust behaviour in two parts and put the MaybeClap operations between them: this is equivalent but takes fewer operations and support variables.
- We add a “dampening” mechanism to the Listen operation: every loud clap after the first one is only recognised as such if the previous one happened at least $2tau_{min}$ steps ago (line 19).

As for the **assume** and **check** sections, we use the former to restrict the initial values of c to the interval $1, \dots, T/2$, as seen in Table 1 (lines 61–65). The latter, in turn, specifies the property of being in synchrony as being in a state where all agents assign the same values to variables c, tau, and sign (lines 67–73). Here, we use the modality after B to indicate that the property should be checked B steps after the initial state.

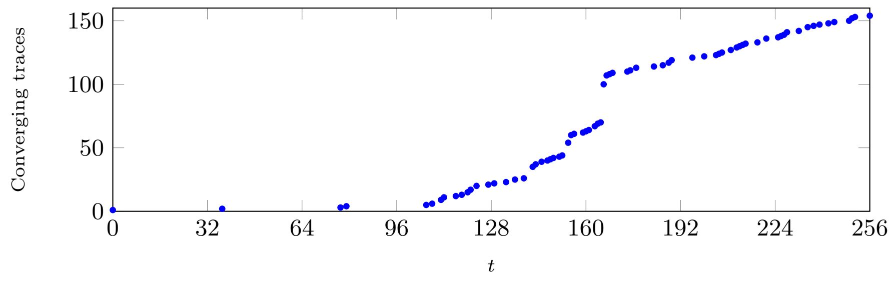
Experimental setup. All the simulations were performed on a dedicated 64-bit GNU/Linux workstation with kernel 5.10.27, equipped with 128 GB of physical memory and a dual 3.10 GHz Xeon E5-2687W 8-core processor. All simulations used the parameters $N = 16$, $loud^{(0)} = 4$, $Tmin = 8$, $Tmax = 20$, and ran for 192 time steps. (Notice that each time step requires $2N$ transitions by individual agents). On average, SLiVER takes around 21 minutes to generate a trace with these parameters on our machine; this is due to the fact that we use the same,

Fig. 1. LAbS listing for our clapping audience model.

```
system {
  extern = N, loud^(0), tau_min, tau_min
  spawn = Agent: N
}

agent Agent {
  interface =
    tau: tau_min..tau_max;
    c: 1..tau_max;
    cPrev: 1; tsl: −1; sign: 1;
    loud: loud^(0);
    loud_decr: tau_max

  Behavior = cPrev ← c; {
    # Listen
    audience := count Agent a,
                   (cPrev of a) = 0;
    loudClap := if audience ≥ loud
                   and (tsl = −1 or tsl ≥ 2∗tau_min)
                   then 1 else 0;

    # Adjust (I)
    tau_avg := (2∗tau + tsl) ÷ 4;
    tauNext := if loudClap = 1
               and tsl ≠ −1
               then tau_avg else tau;
    tauNext := max(tau_min, tauNext);
    tauNext := min(tau_max, tauNext);

    sign ← if c = 0 then 1
           else if c ≥ tauNext then −1
           else sign;

    # MaybeClap
    cNext := if loudClap = 1
             and tsl ≠ −1
             and cPrev ≠ 0
             then c
             else min(c + sign, tauNext);

    # Adjust (II)
    tau ← tauNext;
    c ← cNext;
    loud ← if audience > loud
              then loud + 1
              else if loud_decr = 0
                     then max(loud − 1, 2)
                     else loud;

    loud_decr ← if loudClap = 1
                   or loud_decr = 0
                   then tau_max
                   else loud_decr − 1;

    tsl ← if loudClap then 0
          else if tsl = −1 then −1
          else min(2∗tau_max, tsl + 1)

  }; Behavior
}

assume {
  Initc = forall Agent a,
    1 < (c of a) and
    (c of a) ≤ (tau of a) − 1
}

check {
  Sync = after B
    forall Agent a, forall Agent b,
    (c of a) = (c of b) and
    (tau of a) = (tau of b) and
    (sign of a) = (sign of b)
}
```



(a) Run #1 (1000 simulations, 192 steps).



(b) Run #2 (200 simulations, 256 steps).

Fig. 2. Traces that have reached synchrony by a given time.

verification-oriented encoding for simulation. We obtain traces by performing BMC on our program with a randomised heuristics (to ensure that simulations vary) and a dummy property that fails after the desired trace length has been reached. We do apply some optimizations, e.g., by concretising the initial state and some other sources of nondeterminism, but this approach is bound to be less performant than implementing a bespoke interpreter. At the same time, it lets us reuse the same encoding and workflow we developed for verification, which greatly reduces development and maintenance efforts. A separate interpreter, furthermore, would require a proof that its traces are equivalent to those allowed by the verification encoding. Since this task is essentially single-core and requires a modest amount of memory (about 4.15 GB), we were able to run 10 instances of SLiVER in parallel, bringing our throughput close to 2 minutes per simulation.

Results and discussion. In Fig. 2a we plot the number of simulation that have achieved synchrony against time. By synchrony, we mean that at some point before the end of the simulation every agent has the same period T, the same value of the clapping counter c, and the same value of *sign*, meaning that they are all following the same clapping pattern. SLiVER randomly picked 4 traces that were already synchronous in the initial state. Apart from these, simulations that exhibit synchrony in the first 100 time steps are visibly rare. However, a significant number of traces undergoes a phase transition in the surroundings of $t = 128$ (the shaded region in our plot). In fact, by $t = 112$, the audience has synchronised in only 29 traces out of 1000, while by $t = 144$ this number

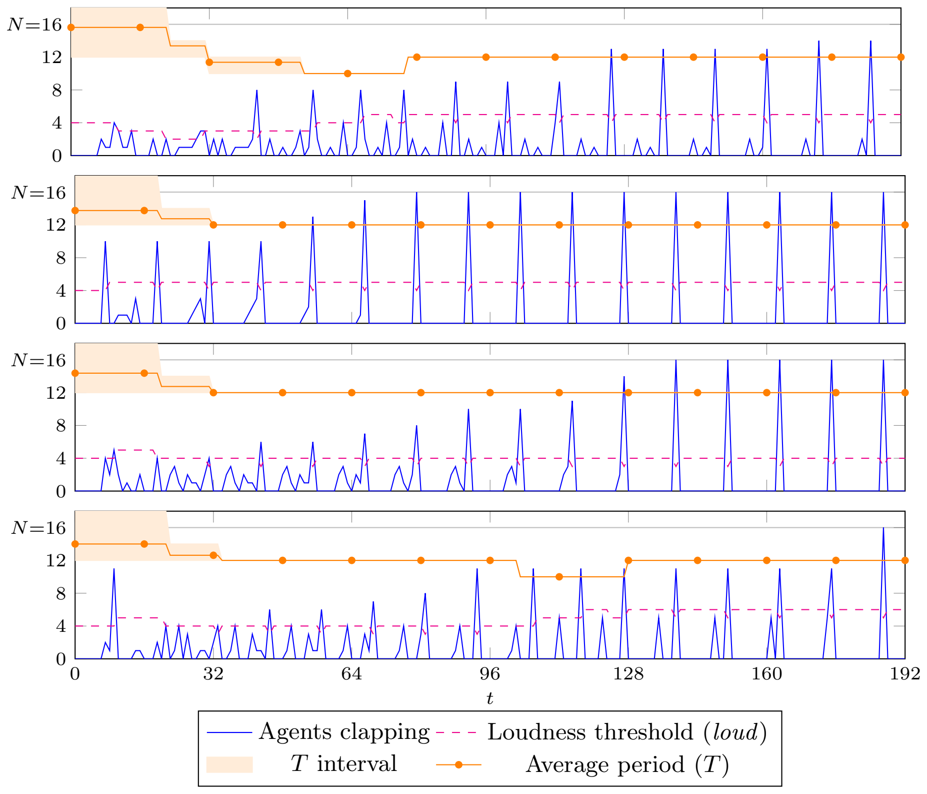


Fig. 3. Visualization of four simulations from our experimental run.

has risen to 389. After that, traces converge to synchrony in a seemingly linear fashion. Overall, 581 of our simulations (i.e., 58%) achieve synchrony. As further evidence of our audience's ability to converge, we then performed an additional 200 simulations with a bound of 256 time steps. Our results (shown in Fig. 2b) largely replicate those of the first run, although the phase transition now appears to be slightly delayed. Once again, we observe that the number of converging simulation grows slowly, but linearly with time after the phase transition, and by step 256 we observe synchrony in 154 simulations out of 200 (77%).

As an additional source of insight into our audience's dynamics, in Fig. 3 we graphically represent a few illustrative simulations. While the topmost simulation cannot achieve synchrony within 192 steps, the other simulations all converge, albeit at different times (from top to bottom: 68, 128, and 176).

For each trace, we plot the number of agents clapping at every time step, along with the average value of the clapping period T and the value of loudness threshold *loud*. We also show the interval between the minimum and maximum values of T within the audience as a shaded area. We are also able to translate these simulations into animations, which gives valuable and intuitive insight into

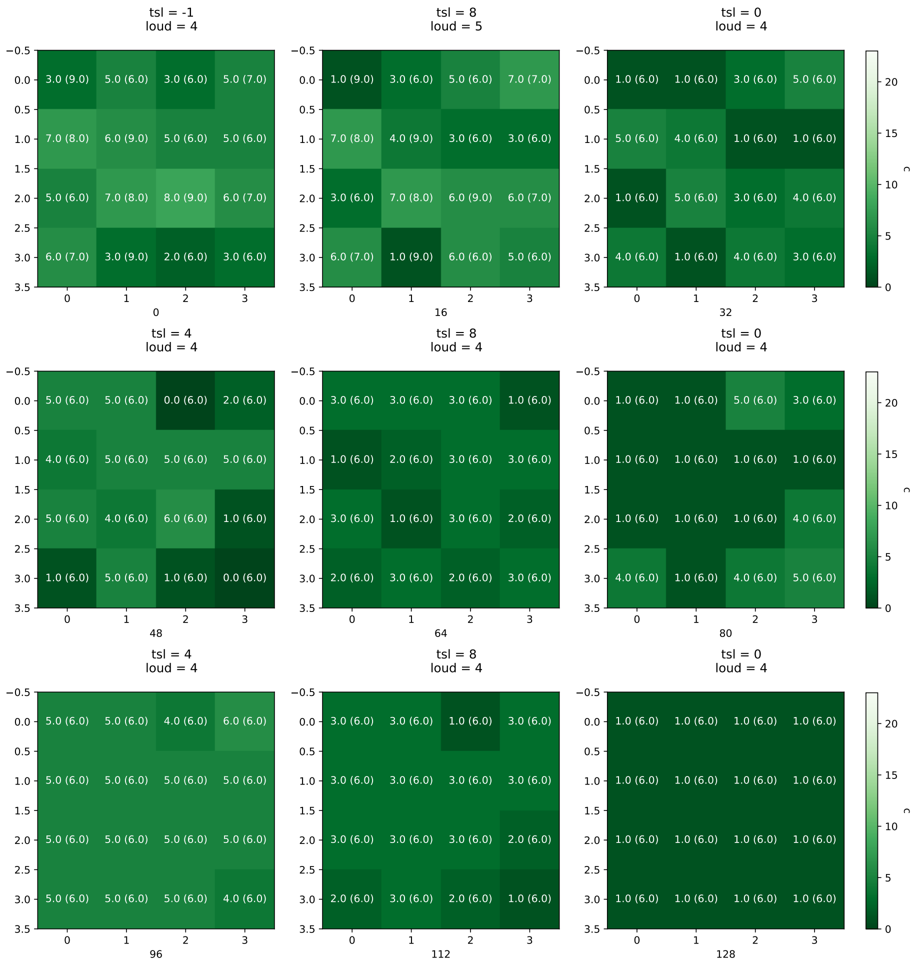


Fig. 4. Visualization of the third simulation from Fig. 3.

the dynamics of our system. In Fig. 4 we showcase a few frames from the third simulation: the full video is available online.[4] Each agent is depicted as a square, marked by its value of the clapping counter c and (in parentheses) the half-period tau. The colour of each square is also determined by the value of c. We also report values for tsl and loud on top of each frame, as well as the step number on the bottom. Here, we chose to focus on the initial part of the simulation, where the system spends some time in a seemingly chaotic behaviour until agents gradually start synchronising, until step 128 when they begin to clap in unison.

On one hand, agents' periods converge rather easily after loud claps start happening: by looking at the peaks of the clapping agents plot, in all convergent traces we observe the quick emergence of a leading group followed by one or two smaller assemblies. In contrast, the topmost trace shows a more erratic behaviour in the early steps, i.e., until agents settle for a lower loudness threshold: at that point, they start recognizing loud claps and gradually gather into synchronised groups (though not quickly enough to converge within the time limit). Sometimes, a very loud clap early in the simulation may speed up convergence, as it raises the loudness threshold which in turn prevents interference from smaller groups (see trace number 2 from the top). However, these early loud claps may be merely caused by chance, as many agents with different periods and phases somehow end up clapping in unison at some point. In this case (shown, for instance, in the bottommost trace), this is counterproductive, as the higher threshold makes agents ignore potentially useful cues from the audience (see the bottommost trace).

Synchronising phase takes a much longer time than synchronising periods: this is evident by looking at how the smaller groups and delay their claps in such a way to get closer and closer to the leading one, until full synchronisation is achieved. Quite obviously this is a result of our implementation of Adjust, by which an agent can only adjust his phase by one unit at a time. While perhaps inefficient, we argue that this design choice is both *general* (i.e., it works across a wide range of situations) and *natural* (i.e., it still lets agents clap with a certain regularity). Manipulating c more abruptly may lead to faster convergence in some cases, but in others it may cause oscillations where the "chasers" never manage to get the phase right, and/or may make them stop clapping for long spans of time (because their value of c stops hitting 0 regularly). Having said that, we cannot argue with certainty that our policy to adjust phase is the absolute best, and leave further investigation about this aspect for future work.

Verifying that synchrony is persistent. In the previous sections, when discussing audiences that start clapping in unison, we have used the word *convergence* because we believe that our model does not allow breaking the synchrony: once reached, there is no way for the agents to drift back into incoherent applause.

While this fact is perhaps easy to deduce by looking at the specifications, we now want to showcase our ability to mechanically prove this kind of statements with SLiVER. Therefore, we slightly revise our specification and add an assumption that all agents are already in synchrony in the initial state. We then

[4] https://doi.org/10.5281/zenodo.11374365

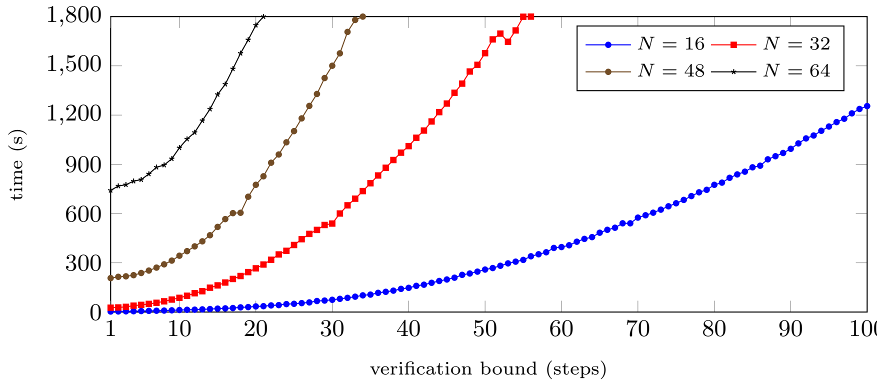


Fig. 5. Verification times for our synchrony preservation experiments.

run SLiVER to prove that the synchrony is not broken (i.e., property Sync at lines 67–67 of Fig. 1 is preserved) after a given number of steps.

In principle, it would be sufficient to prove this for a single step: if the audience starts from a synchronous state and the property is preserved, different should happen in the next round. However, we also want to use this task as a way to benchmark how our chosen technique (BMC) scales in both the verification bound and the size of the system. Therefore, we run tasks for systems of 16, 32, 48, and 64 agents. For each system, we start by verifying the property for a single step. If the task succeeds, we increment the verification bound by one and run another task. We stop when we find a verification bound for which SLiVER is not able to reach a verdict within 30 minutes, or when the bound exceeds 100.

The results of this experiment are summarised in Fig. 5. All experiments that terminated gave a positive verdict, reinforcing our conviction that the agents are not able to break synchrony once it is achieved. For all systems, verification times grow in a slightly super-linear fashion and are mainly affected by the size of the system itself. For N=16, SLiVER never times out and can verify a system up to a bound of 100 within 1255 seconds (about 20 minutes). However, when we double the number of agents (N=32) SLiVER times out with a verification bound of 56, which further decreases to 34 and then 21 for N=48 and N=64, respectively. Notice that this loss of performance is at least partially justified by the fact that, as the system grows in size, computing its evolution after each step requires a higher number of individual state updates. Additionally, more agents require more memory to hold their state variables. These factors are known to negatively affect the underlying satisfiability procedure.

4 Conclusion and related work

In this work, we have once again stressed the growing importance that the study of complexity and emerging behaviour is gaining among multiple research areas.

We have traced a correspondence between this ongoing trend and the overarching concerns of concurrency theory, and argued that process-algebraic techniques and tools may provide a solid foundation for the upcoming challenges that these new applications confront us with. In particular, we have focused on attribute-based formalisms as an effective way to capture the dynamic nature of these systems and of the interactions happening among their components. To illustrate this, we have provided an intuitive specification of an audience of clapping agents, and used the emergence of synchronised applause as a case study for our analysis. To the best of our knowledge, this is the first attempt to integrate formal modelling and analysis of such a scenario.

We have shown that our informal description of individual behaviour can be easily captured by a high-level, formal agent-based language, and that we may use the resulting specification to simulate our system. By doing so, we empirically checked that a group of agents following this behaviour may in fact achieve synchrony. The same specifications also allowed us to formally verify that synchrony cannot be broken: once the agents are clapping in unison, they will maintain the collective pace.

We believe that this ability to combine empirical, trace-based analysis with formal, exhaustive state space exploration is the decisive advantage that these languages and tools offer as a modelling solution. As interest in complex systems cannot but grow in the future, it is essential that the rich toolbox of concurrency theory and formal verification is made easily available to the broader scientific community, through mature and highly usable languages and analysis platforms.

Studying this example gave us a number of ideas for future work. On the semantics side, enforcing synchrony should be made easier, since it is commonplace in important contexts ranging from cellular automata [8] and Boolean networks [25]. Ideally, one should be able to choose this or other forms of agent interleaving at the time of analysis, without having to change the model as we are currently forced to do. On the language side, we should allow users to define and invoke functions, which should be treated as reusable sequences of atomic operations. This would improve readability in the case of complex behaviours such as the one we introduced in this work.

On the analysis side, our simulation workflow could benefit from an increased throughput. We might obtain this by either leveraging state-of-the-art solvers [22], or by pivoting to other symbolic tools with native support for simulation [7]. We should also allow to express a wider range of properties, such as arbitrary LTL formulae [35]. Our alternative, explicit-state backend [20] does support full model checking of safety and liveness properties, but symbolic techniques would likely scale better as the number and complexity of the analysed agents grows.

Data-driven approaches are worth pursuing too. In principle, we could inject empirical measurements as assumptions at specific points in time, and check if there are executions that fit the data. This could be used to validate the model, or as a form of parameter search: we could parameterise parts of the model in one or more symbolic variables and analyse which values fit the empirical data.

Related work. A number of biological phenomena related to rhythmic patterns and their synchronisation may be modelled as a population of coupled oscillators [39]. Analytical solutions for these populations are however hard to find, as the resulting dynamics are highly non-linear. This approach is frustrated by the high non-linearity of the collective dynamics, hence these models typically restrict themselves to *weak* coupling (i.e., the external perturbations have a small instantaneous effect on each single oscillator). However, defining the (small) interaction between any two oscillators as a periodic function of their phase offset results in the Kuramoto model [27], which does allow to analytically derive whether a group of oscillators will synchronise fully, partially, or not at all, depending on the intensity of these interactions. Interestingly, the Kuramoto model appears to be a good fit for collective applause dynamics [33, 34].

At first glance, our agents might also look like a set of oscillators; however, by specifying their evolution as a computational process, we can capture with ease dynamics that are highly non-linear and thus do not lend themselves to a simple mathematical description. Most notably, while the Kuramoto model assumes a constant interaction (i.e., every agent continuously reacts to the phase difference with respect to the others), our model allows agents to detect and react to the general, aggregate loudness of the audience, in a discontinuous fashion.

We are not aware of many other works in this vein. In [28], each agent adjusts his rhythm based on his neighbours' last clapping time and their clapping period, but we believe that relying on the latter makes the model unrealistic since agents do not explicitly communicate that information. However, differently from both our model and the Kuramoto model, agents in [28] influence each other according to an inverse-square law (i.e., the intensity of the interaction decreases with the square power of the distance), and interactions are overall limited to a local neighbourhood. These added constraints still allow for emergent synchrony. In ApplauSim [24], agents react to the perceived collective rhythm, as in our model, but this rhythm is computed as an average over a short observation window. Crucially, both these models seem to somewhat rely on "synchronisers", or "backbone" agents, which receive little influence from the rest of the audience and thus facilitate the emergence of synchrony. Our model seems to perform well even in the absence of synchronisers, but we might study their impact in future work.

Clapping and applause are complex, nuanced social behaviours [10], featuring many aspects not covered in our model. For instance, choosing when to begin and stop clapping is in itself an intricate choice informed by both individual preferences (e.g., eagerness to show appreciation) and social cues (e.g., the perceived mood of the audience). Models that originated in epidemiology [26] appear to be well-suited to capture this and other forms of social contagion [29]. In the future we could use this work to extend our model with a "startup" phase, where agents have to decide when to start clapping. An advantage of our language-based approach is precisely this ability to combine concepts and behaviours coming from different models, and study their interplay.

References

1. Abd Alrahman, Y., Azzopardi, S., Di Stefano, L., Piterman, N.: Language support for verifying reconfigurable interacting systems. International Journal on Software Tools for Technology Transfer **25**(5), 765–784 (2023). https://doi.org/10/gs4rg6
2. Abd Alrahman, Y., De Nicola, R., Loreti, M.: Programming interactions in collective adaptive systems by relying on attribute-based communication. Science of Computer Programming **192** (2020). https://doi.org/10.1016/j.scico.2020.102428
3. Abd Alrahman, Y., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: SAC. ACM (2015). https://doi.org/gf4vn6
4. Bartocci, E., Liò, P.: Computational modeling, formal analysis, and tools for systems biology. PLoS Computational Biology **12** (2016). https://doi.org/10/f8knqc
5. Bergstra, J.A., Klop, J.W., Tucker, J.V.: Algebraic tools for system construction. In: Workshop on Logics of Programs. LNCS, vol. 164. Springer (1983). https://doi.org/10.1007/3-540-12896-4_353
6. Boemo, M.A., Cardelli, L., Nieduszynski, C.A.: The Beacon Calculus: A formal method for the flexible and concise modelling of biological systems. PLOS Computational Biology **16** (2020). https://doi.org/10.1371/journal.pcbi.1007651
7. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: CAV. LNCS, vol. 8559. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
8. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press (1998). https://doi.org/10.1017/CBO9780511549755
9. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: TACAS. LNCS, vol. 2988. Springer (2004). https://doi.org/10/cfm9ks
10. Crawley, A.: Clap, clap, clap - Unsystematic review essay on clapping and applause. Integrative Psychological and Behavioral Science **57** (2023). https://doi.org/10/gt5mb5
11. De Nicola, R.: A gentle introduction to Process Algebras. IMT School for Advanced Studies, Lucca, Italy (2011). https://doi.org/10.5281/zenodo.11065174
12. De Nicola, R., Di Stefano, L., Inverso, O.: Multi-agent systems with virtual stigmergy. Science of Computer Programming **187** (2020). https://doi.org/10/h3kv
13. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Intuitive Modelling and Formal Analysis of Collective Behaviour in Foraging Ants. In: CMSB. LNCS, vol. 14137. Springer (2023). https://doi.org/10.1007/978-3-031-42697-1_4
14. De Nicola, R., Di Stefano, L., Inverso, O., Valiani, S.: Modelling flocks of birds and colonies of ants from the bottom up. International Journal on Software Tools for Technology Transfer **25** (2023). https://doi.org/10.1007/s10009-023-00731-0
15. De Nicola, R., Duong, T., Inverso, O., Trubiani, C.: AErlang: Empowering Erlang with attribute-based communication. Science of Computer Programming **168** (2018). https://doi.org/10.1016/j.scico.2018.08.006
16. De Nicola, R., Duong, T., Loreti, M.: ABEL - A domain specific framework for programming with attribute-based communication. In: COORDINATION. LNCS, vol. 11533. Springer (2019). https://doi.org/10.1007/978-3-030-22397-7_7
17. De Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Transactions on Software Engineering **24** (1998). https://doi.org/10.1109/32.685256
18. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. ACM Transactions on Autonomous and Adaptive Systems **9** (2014). https://doi.org/10.1145/2619998

19. Di Stefano, L., De Nicola, R., Inverso, O.: Verification of Distributed Systems via Sequential Emulation. ACM Transactions on Software Engineering and Methodology **31** (2022). https://doi.org/10.1145/3490387
20. Di Stefano, L., Lang, F.: Verifying temporal properties of stigmergic collective systems using CADP. In: ISoLA. LNCS, vol. 13036. Springer (2021). https://doi.org/10.1007/978-3-030-89159-6_29
21. Di Stefano, L., Lang, F.: Compositional verification of stigmergic collective systems. In: VMCAI. LNCS, vol. 13881. Springer (2023). https://doi.org/10/jxwd
22. Fleury, M., Biere, A.: Mining definitions in Kissat with Kittens. Formal Methods in System Design **60** (2022). https://doi.org/10.1007/s10703-023-00421-2
23. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985), http://www.usingcsp.com/cspbook.pdf
24. Horni, A., Montini, L.: A glimpse at emergence in agent-based simulations. In: STRC. ETH Zürich (2013). https://doi.org/10/m94g
25. Kauffman, S.: Homeostasis and differentiation in random genetic control networks. Nature **224**(5215) (1969). https://doi.org/10.1038/224177a0
26. Kermack, W.O., McKendrick, A.G.: Contributions to the mathematical theory of epidemics—I. Bulletin of Mathematical Biology **53** (1991). https://doi.org/10.1007/BF02464423, reprinted from Proc. R. Soc. A **115** (1927).
27. Kuramoto, Y., Nishikawa, I.: Statistical macrodynamics of large dynamical systems. Case of a phase transition in oscillator communities. Journal of Statistical Physics **49** (1987). https://doi.org/10.1007/BF01009349
28. Li, D., Liu, K., Sun, Y., Han, M.: Emerging Clapping Synchronization From a Complex Multiagent Network With Local Information via Local Control. IEEE Transactions on Circuits and Systems II Express Briefs **56-II** (2009). https://doi.org/10.1109/TCSII.2009.2020931
29. Mann, R.P., Faria, J., Sumpter, D.J.T., Krause, J.: The dynamics of audience applause. Journal of The Royal Society Interface **10** (2013). https://doi.org/10/f2zmr6
30. Milner, R.: A Calculus of Communicating Systems, LNCS, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
31. Milner, R.: Elements of Interaction - Turing Award lecture. Communications of the ACM **36** (1993). https://doi.org/10.1145/151233.151240
32. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Information and Computation **100** (1992). https://doi.org/10.1016/0890-5401(92)90008-4
33. Néda, Z., Ravasz, E., Brechet, Y., Vicsek, T., Barabási, A.L.: The sound of many hands clapping. Nature **403** (2000). https://doi.org/10.1038/35002660
34. Néda, Z., Ravasz, E., Vicsek, T., Brechet, Y., Barabási, A.L.: Physics of the rhythmic applause. Phys. Rev. E **61** (2000). https://doi.org/10.1103/PhysRevE.61.6987
35. Pnueli, A.: The temporal logic of programs. In: 18th Symposium on Foundations of Computer Science (FOCS). pp. 46–57. IEEE (1977). https://doi.org/10/dn8cpn
36. Sumpter, D.J., Blanchard, G.B., Broomhead, D.S.: Ants and agents: A process algebra approach to modelling ant colony behaviour. Bulletin of Mathematical Biology **63** (2001). https://doi.org/10.1006/bulm.2001.0252
37. Thomson, M., Murphy, K., Lukeman, R.: Groups clapping in unison undergo size-dependent error-induced frequency increase. Scientific Reports **8** (2018). https://doi.org/10.1038/s41598-017-18539-9
38. Tofts, C.M.N.: Describing social insect behaviour using process algebra. Transaction of the Society for Computer Simulation **9** (1992)
39. Winfree, A.T.: Biological rhythms and the behavior of populations of coupled oscillators. Journal of Theoretical Biology **16** (1967). https://doi.org/10/bhr4xf