Compositional Verification of Stigmergic Collective Systems

Luca Di Stefano[1] Frédéric Lang[2]

[1] University of Gothenburg, Sweden

[2] INRIA Grenoble, France

VMCAI'2023

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG Inria

Collective systems

- Collections of agents interacting with each other
- Found in CS, economics, biology…
- Interactions and feedback may lead to emergence of collective features
- Reasoning about emergence is hard. Model checking?

Collective systems

- Collections of agents interacting with each other
- Found in CS, economics, biology…
- Interactions and feedback may lead to emergence of collective features
- Reasoning about emergence is hard. Model checking?

Pros

- Can prove emergence, safety, etc. (arbitrary temporal properties)
- Push-button, no human guidance needed

Cons

- Requires user expertise for modelling, specification
- State space explosion as the number/complexity of agents increases

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

What LAbS is about

- High-level language to concisely specify systems/properties
- Focus on indirect, attribute-based interaction mechanisms
- Reuse of different existing verification technologies
  - E.g., CADP, which offers model-checking and compositional verification tools out of the box

In this talk

What LAbS is about

- High-level language to concisely specify systems/properties
- Focus on indirect, attribute-based interaction mechanisms
- Reuse of different existing verification technologies
  - E.g., CADP, which offers model-checking and compositional verification tools out of the box

Contributions

- Encoding of LAbS systems into parallel LNT programs
- Compositional verification workflow
- Sound value analysis to prune individual state spaces and speed up verification

Bully election on a stigmergy (1/2)

```
system {
  spawn = Node: N
}
stigmergy Election {
  link = true
  leader: N
}
```

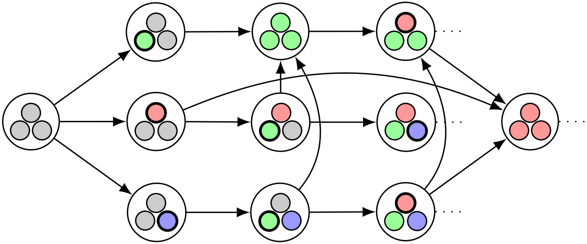
```
agent Node {
  stigmergies = Election
  Behavior =
    leader >= id ->
      leader <~ id;
      Behavior
}
```

- *N* nodes run for election, by storing their `id` in a *stigmergic* variable `leader`. If `leader` $<$ `id`, the node waits
- All communication is implicit
  - Nodes exchange their values of `leader`
  - Values are timestamped, “newer” ones replace “older”
  - `link = true` means broadcast messages
- Intuitively, they should converge to a state where all nodes set `leader` to the lowest `id` in the system

Bully election on a stigmergy (2/2)

Some feasible executions (with $N = 3$)

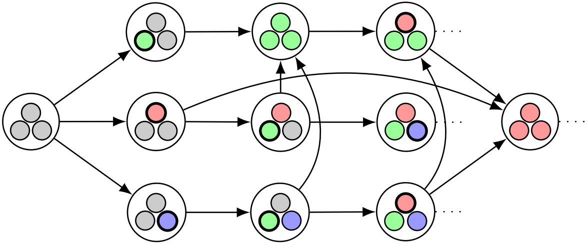
	leader
● (red)	0
● (green)	1
● (blue)	2
● (grey)	3



Some feasible executions (with $N = 3$)

leader

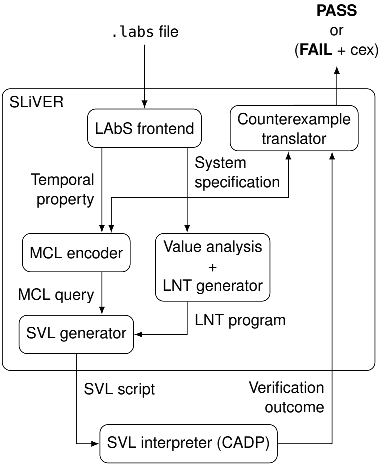
- 0 (red)
- 1 (green)
- 2 (blue)
- 3 (grey)



A property of interest

$$fairly_{\infty} \forall x : Node \bullet x.leader = 0$$

Along every fair execution, there are infinitely many states where all nodes have 0 as the leader



Based on CADP and its languages/formalisms:

LNT system description

MCL property specification (alternation-free μ-calculus with data)

SVL scripting of complex verification tasks (in our case: compositional state space generation + model checking)

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Program = *processes* communicating over *gates*

Send offer

$\mathtt{G}(v_1, \ldots, v_n)$

Offer values over gate G

Receive offer

$\mathtt{G(?x_1, \ldots, ?x_n)\ where}\ \varphi(\mathtt{x}_1, \ldots, \mathtt{x}_n)$

Receive *n* values over G and bind them to $\mathtt{x}_i$, $i = 1, \ldots, n$, *but* only if $\varphi(\mathtt{x}_1, \ldots, \mathtt{x}_n)$ holds.

Program = *processes* communicating over *gates*

Send offer

$\texttt{G}(v_1, \ldots, v_n)$

Offer values over gate G

Receive offer

$\texttt{G}(?\texttt{x}_1, \ldots, ?\texttt{x}_n)\ \texttt{where}\ \varphi(\texttt{x}_1, \ldots, \texttt{x}_n)$

Receive *n* values over G and bind them to $\texttt{x}_i$, $i = 1, \ldots, n$, *but* only if $\varphi(\texttt{x}_1, \ldots, \texttt{x}_n)$ holds.

Parallel composition

$\texttt{par}\ \texttt{G}_{11}, \ldots, \texttt{G}_{1j} \to P_1 \| \cdots \| \texttt{G}_{n1}, \ldots, \texttt{G}_{nk} \to P_n\ \texttt{end par}$

All processes with G in their set of gates must synchronize over it

LNT crash course

Program = *processes* communicating over *gates*

Send offer

$\mathtt{G}(v_1, \ldots, v_n)$

Offer values over gate G

Receive offer

$\mathtt{G}(?\mathtt{x}_1, \ldots, ?\mathtt{x}_n)\ \mathtt{where}\ \varphi(\mathtt{x}_1, \ldots, \mathtt{x}_n)$

Receive *n* values over G and bind them to $\mathtt{x}_i$, $i = 1, \ldots, n$, *but* only if $\varphi(\mathtt{x}_1, \ldots, \mathtt{x}_n)$ holds.

Parallel composition

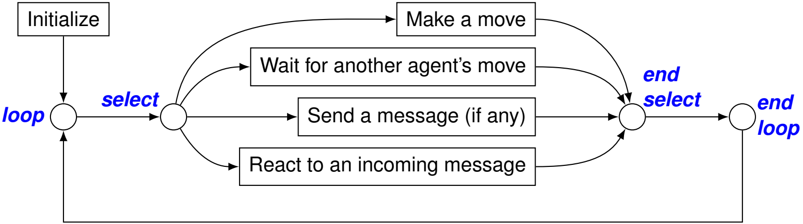
$\mathtt{par}\ \mathtt{G}_{11}, \ldots, \mathtt{G}_{1j} \rightarrow P_1 \| \cdots \| \mathtt{G}_{n1}, \ldots, \mathtt{G}_{nk} \rightarrow P_n\ \mathtt{end\ par}$

All processes with G in their set of gates must synchronize over it

Etc.

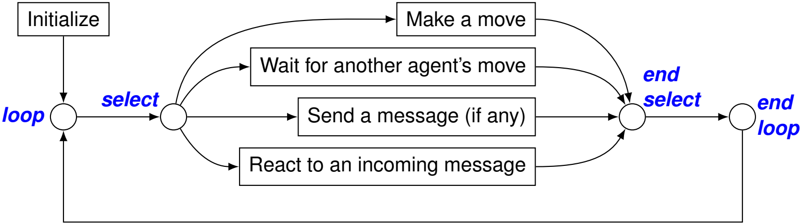
Loops, nondeterministic choice, conditionals, guards, . . .

Each agent is encoded as a process with this structure:



From LAbS to LNT

Each agent is encoded as a process with this structure:



- System = Parallel composition of all agents and additional processes (e.g., information about timestamps)
- Multi-party synchronization to resolve the agents' choices

Compositional state space generation

Given a tree of parallel processes S, generate the transition system $\mathit{lts}(S)$ by composing the (minimized) TSs of the "leaves" $P_1, \ldots, P_m \in S$

Compositional state space generation

Given a tree of parallel processes S, generate the transition system $lts(S)$ by composing the (minimized) TSs of the "leaves" $P_1, \ldots, P_m \in S$

Root-leaf reduction (modulo R)

- For every P_i generate $T_i = lts(P_i)$
- Minimize every T_i modulo R: $T_i' = min_R(T_i)$
- Generate $T = lts(S[T_i'/P_i])$
- Return $min_R(T)$

Compositional state space generation

Given a tree of parallel processes S, generate the transition system $lts(S)$ by composing the (minimized) TSs of the "leaves" $P_1, \ldots, P_m \in S$

Root-leaf reduction (modulo R)

- For every P_i generate $T_i = lts(P_i)$
- Minimize every T_i modulo R: $T_i' = min_R(T_i)$
- Generate $T = lts(S[T_i'/P_i])$
- Return $min_R(T)$

Drawback

When generating each T_i we do not know what messages we may receive from the other processes
E.g., in the bully election, *leader* may be any integer in $-128..127$

1. Compute abstract initial state ς_0 from specification $\mathbb{S}$
   - In this work we use powersets of intervals
2. Add ς_0 to a set σ
3. For every assignment *a* in $\mathbb{S}$ and every state ς in σ:
   - Evaluate *a* on ς
   - Add resulting state ς' to σ
4. Reach a fixed point
5. Merge all states in σ to obtain $\bar{\sigma}$

Value analysis of LAbS specifications

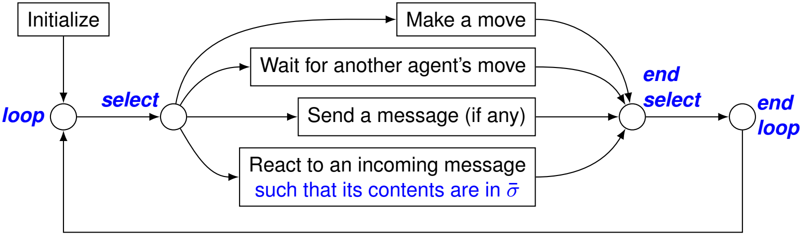
1. Compute abstract initial state ς_0 from specification $\mathbb{S}$
   - In this work we use powersets of intervals
2. Add ς_0 to a set σ
3. For every assignment *a* in $\mathbb{S}$ and every state ς in σ:
   - Evaluate *a* on ς
   - Add resulting state ς' to σ
4. Reach a fixed point
5. Merge all states in σ to obtain $\bar{\sigma}$

Example

In the bully election system we find out that *leader* $\in \{0, \ldots, N\}$ in every state

Pruning the state space

We use $\bar{\sigma}$ to prune out receptions of impossible messages:



(In practice we plug $\bar{\sigma}$ as a `where`-clause on receive offers)

Experimental evaluation

System	Baseline Time (s)	Baseline Memory (MiB)	Compositional Time (s)	Compositional Memory (MiB)	Parallel Time (s)	Parallel Memory (MiB)
`flock-rr`	**1875**	12000	4461	**11805**	4426	**11805**
`flock`	4787	30865	4071	**11113**	**4038**	**11113**
`formation-rr`	1670	**1657**	2511	1938	**1558**	5875
`leader5`	**10**	**41**	34	117	18	212
`leader6`	77	**147**	104	225	**65**	258
`leader7`	1901	2038	374	**404**	**326**	**404**
`twophase2`	**9**	**50**	67	93	34	210
`twophase3`	500	**209**	233	322	**131**	560

Baseline Previous (sequential) LNT encoding

Compositional Our work spaces on a dedicated core

Parallel What would happen if we split state space generation across multiple cores

`-rr` Round-robin scheduling

Experimental evaluation

System	Baseline Time (s)	Baseline Memory (MiB)	Compositional Time (s)	Compositional Memory (MiB)	Parallel Time (s)	Parallel Memory (MiB)
`flock-rr`	**1875**	12000	4461	**11805**	4426	**11805**
`flock`	4787	30865	4071	**11113**	**4038**	**11113**
`formation-rr`	1670	**1657**	2511	1938	**1558**	5875
`leader5`	**10**	**41**	34	117	18	212
`leader6`	77	**147**	104	225	**65**	258
`leader7`	1901	2038	374	**404**	**326**	**404**
`twophase2`	**9**	**50**	67	93	34	210
`twophase3`	500	**209**	233	322	**131**	560

- *B* wins on very small instances (no overhead)
- *C* scales better and has fewer issues with full interleaving
- *P* brings further gains wrt verification times but may be more memory hungry

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Conclusion

- Model checking enables verification of expressive properties in collective systems
- Compositional verification can palliate state space explosion
- Value analysis speeds up state space generation

Conclusion

- Model checking enables verification of expressive properties in collective systems
- Compositional verification can palliate state space explosion
- Value analysis speeds up state space generation

Future work

- Investigate tighter approximations (better abstract domains, better algorithm)
- Actually parallelize workflow across multiple cores/machines

References

1. De Nicola, Di Stefano, Inverso. Multi-agent systems with virtual stigmergy. Sci. Comput. Program. 187 (2020). DOI: `https://doi.org/10.1016/j.scico.2019.102345`
   General introduction to LAbS
2. Di Stefano, De Nicola, Inverso. Verification of Distributed Systems via Sequential Emulation. TOSEM 31 (2022). DOI: `https://doi.org/10.1145/3490387`
   Describes our general approach to LAbS verification
3. Di Stefano and Lang. Verifying Temporal Properties of Stigmergic Collective Systems Using CADP. In ISoLA2021. DOI: `https://doi.org/10.1007/978-3-030-89159-6_29`
   Baseline CADP-based verification workflow and benchmark description

Backup slides

Evaluation (x)

- Read $val(x)$
- Mark x for a `qry`-message

Stigmergic variables

Evaluation (`x`)

- Read $val(x)$
- Mark `x` for a `qry`-message

Assignment (`x <~ v`)

- Compute the current timestamp t
- Record $val(\mathtt{x}) \leftarrow v$, $ts(\mathtt{x}) \leftarrow t$
- Mark `x` for a `put`-message
- Unmark `x` for `qry`

Evaluation (`x`)

- Read $val(x)$
- Mark `x` for a `qry`-message

Assignment (`x <~ v`)

- Compute the current timestamp t
- Record $val(\mathtt{x}) \leftarrow v$, $ts(\mathtt{x}) \leftarrow t$
- Mark `x` for a `put`-message
- Unmark `x` for `qry`

Messaging

- Messages are sent asynchronously to all neighbours
- Neighbourhood is defined as a predicate on sender and potential receiver
- Different variables may have different predicates

Receiving $\langle \texttt{put}, \mathbf{x}, v, t \rangle$

- If $t > ts(\mathrm{x})$:
  1. Record $val(\mathrm{x}) \leftarrow v$, $ts(\mathrm{x}) \leftarrow t$
  2. Mark x for a `put`-message
- Otherwise, ignore the message

Receiving $\langle \texttt{qry}, \mathbf{x}, v, t \rangle$

- Mark x for a `put`-message
- If $t > ts(\mathrm{x})$, then record $val(\mathrm{x}) \leftarrow v$, $ts(\mathrm{x}) \leftarrow t$

Compositional vs parallel workflow

Table: Time and memory requirements for the *Compositional* and *Parallel* workflows.

	Compositional	Parallel
Time	$\sum_{Tasks} \mathit{time}(\mathcal{T})$	$\max_i \{\mathit{time}(\mathcal{T}_i)\} + \mathit{time}(\mathcal{T}_{\mathbb{P}}) + \mathit{time}(\mathcal{T}_{\models})$
Memory	$\max_{Tasks} \mathit{mem}(\mathcal{T})$	$\max \left\{\sum_i \mathit{mem}(\mathcal{T}_i), \mathit{mem}(\mathcal{T}_{\mathbb{P}}), \mathit{mem}(\mathcal{T}_{\models})\right\}$

Priorities

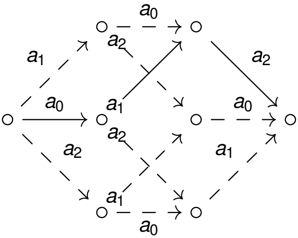


Figure: Example of a diamond when 3 agents perform independent actions a_0, a_1, and a_2. Dotted transitions are cut by applying the priority relation $a_0 \succ a_1 \succ a_2$.