Model Checking Parameterised Multi-Token Systems via the Composition Method

Benjamin Aminof (Technical University of Vienna),
Sasha Rubin (University of Naples)

IJCAR 2016

Distributed Systems are often Parameterised

- Swarm protocols (number of agents),
- Concensus algorithms (number of processes),
- Robot protocols (size of the environment),
- ...

What if one doesn't know the value of the parameter?

Distributed Systems are often Parameterised

- Swarm protocols (number of agents),
- Concensus algorithms (number of processes),
- Robot protocols (size of the environment),
- ...

What if one doesn't know the value of the parameter?

Verify the system for all values of the parameter.

Basic Model

- P = protocol/process-template (finite state).
- G = network graph representing communication links.
- P^G = product system.
  - Place a copy of P at each node of G
  - Adjacent nodes can communicate (e.g., by broadcast).
- Time is discreet, and operation is asynchronous.

We "parameterize" by considering (infinite) families $\mathbb{G}$ of possible graphs

Hence, in particular, the number of processes is not fixed.

Indexed Temporal Logic (ITL)

Take some temporal logic and modify it as follows:

- Index atoms by vertices in G.
  - p_x is **true** iff the atomic proposition p holds at the current state s of the process at vertex x.
- Add the ability to quantify over the process indices.

- Example: $\forall x \forall y \neq x . \mathbf{AG} \neg (c_x \wedge c_y)$.

Indexed Temporal Logic (ITL)

Take some temporal logic and modify it as follows:

- Index atoms by vertices in G.
  - p_x is **true** iff the atomic proposition p holds at the current state s of the process at vertex x.
- Add the ability to quantify over the process indices.

- Example: $\forall x \forall y \neq x . \mathbf{AG} \neg (c_x \wedge c_y)$.

- For us, **ITL** will be indexed-$\mathbf{CTL}^* \setminus X$ (or a fragment of it)

What is Parameterised Verification?

- Fix specification language **ITL**.
- Fix class of network graphs $\mathbb{G}$.
- Fix class of processes $\mathbb{P}$.

Verification

- Input: $P \in \mathbb{P}$, $G \in \mathbb{G}$, $\Phi \in$ **ITL**.
- Decide if $P^G \models \Phi$.

Parameterised Verification

- Input: $P \in \mathbb{P}$, $\Phi \in$ **ITL**.
- Decide if $P^G \models \Phi$ for all $G \in \mathbb{G}$.

Parameterised Verification

Quickly Undecidable!

Parameterised Verification

Quickly Undecidable!

- Suzuki 1988: undecidable on token-rings for the single formula $\exists i.\, \mathsf{E}\,\mathsf{F}\, q_i$.

  Idea: use ring as tape of TM. One process simulates the control, and all others simulate the contents of the tape. Use values on token to communicate head movement and read/write values.

Parameterised Verification

Quickly Undecidable!

- Suzuki 1988: undecidable on token-rings for the single formula $\exists i.\, \mathsf{E}\,\mathsf{F}\, q_i$.

  Idea: use ring as tape of TM. One process simulates the control, and all others simulate the contents of the tape. Use values on token to communicate head movement and read/write values.

- Examples of other undecidability results:
  - Broadcast, clique, termination (Esparza, Finkel, Meyer 1999)
  - No communication(!), indexed **LTL**$\setminus$**X** (Veith et al. 2013).

Deciding Parameterised Verification

To get decidability we must limit both the specification language and the process templates

Deciding Parameterised Verification

To get decidability we must limit both the specification language and the process templates

Prenex fragment of ITL

All process quantifiers must appear before any temporal and path quantifiers:

- $\forall x \forall y \neq x . \mathbf{AG} \neg (c_x \wedge c_y)$.

Deciding Parameterised Verification

To get decidability we must limit both the specification language and the process templates

Prenex fragment of ITL

All process quantifiers must appear before any temporal and path quantifiers:

- $\forall x \forall y \neq x . \mathbf{AG} \neg (c_x \wedge c_y)$.

Typical machinery:

- Vector addition systems, well-structured transition systems,
- Automata Theory,
- Composition Method.

Composition Method

Reduce reasoning about compound systems $A \square B$ to reasoning about constituent parts A, B.

- $\mathbf{FOL}_k(A \sqcup B)$ is determined by $\mathbf{FOL}_k(A)$ and $\mathbf{FOL}_k(B)$ (Feferman/Vaught 59).
- $\mathbf{ITL}(P^G)$ is determined by ...?

Composition Method for Verification

COMPOSITION PROPERTY. If $G \sim H$ then (for all P), $P^G \equiv_{\mathbf{ITL}} P^H$.

Conclude:

Reduce "$P^G \models \Phi$" to "$P^H \models \Phi$".

Composition Method for Verification

COMPOSITION PROPERTY. If $G \sim H$ then (for all P), $P^G \equiv_{\mathbf{ITL}} P^H$.

Conclude:

Reduce "$P^G \models \Phi$" to "$P^H \models \Phi$".

How can one use this for Parameterised Verification?

Composition Method for Parameterised Verification

FINITENESS PROPERTY. There are finitely many $\sim$-classes of graphs G.

Conclude:

"$P^G \models \Phi$ for all $G \in \mathbb{G}$"
reduces to
"$P^G \models \Phi$ for all $\sim$-representatives G from $\mathbb{G}$".

COMPOSITION & FINITENESS $\Longrightarrow$ PV decidable.

Positive Result Setting

Product Operator P^G:

- Processes communicate by sending/recieving tokens,
- Tokens have values,
- Multiple tokens,
- Graph G has directions (e.g., send token North),
- Fairness assumptions.

Logic for products **ITL**

- k-prenex indexed $\mathbf{CTL}^*_d \setminus \mathbf{X}$.
- $k \in \mathbb{N}$ bounds the number of quantifiers.
- $d \in \mathbb{N}$ bounds the nesting depth of path quantifiers.

Results

Theorem

For all $k, d \in \mathbb{N}$ and all $\mathbb{G}$: PV of token-passing systems for k-prenex $\mathbf{CTL}^*_d \backslash \boldsymbol{X}$ specifications is decidable.

Results

Theorem

For all $k, d \in \mathbb{N}$ and all $\mathbb{G}$: PV of token-passing systems for k-prenex $\mathbf{CTL}^*_d \backslash \boldsymbol{X}$ specifications is decidable.

Negative Result

The above is, in many ways, the best one can hope for, i.e., weakening any of the assumptions in the previous slide makes the problem undecidable.

How we do it

Reminder of the Method:

- **Composition Property:** $G \sim H \implies P^G \equiv_{k-\mathbf{CTL}^*_d \setminus \mathbf{X}} P^H$.
- **Finiteness Property:** $\sim$ has finite index.

How we do it

Reminder of the Method:

- **Composition Property:** $G \sim H \implies P^G \equiv_{k-\mathbf{CTL}^*_d\setminus\mathbf{X}} P^H$.
- **Finiteness Property:** $\sim$ has finite index.

We do composition modulo assignments $\bar{g}, \bar{h}$ in G, H.

We prove:

- $G[\![\bar{g}]\!] \equiv_{k-\mathbf{CTL}^*_d\setminus\mathbf{X}} H[\![\bar{h}]\!] \implies P^G[\bar{g}] \equiv_{k-\mathbf{CTL}^*_d\setminus\mathbf{X}} P^H[\bar{h}]$.
- $\equiv_{k-\mathbf{CTL}^*_d\setminus\mathbf{X}}$ has finite index over structures of the form $G[\![\bar{g}]\!]$.

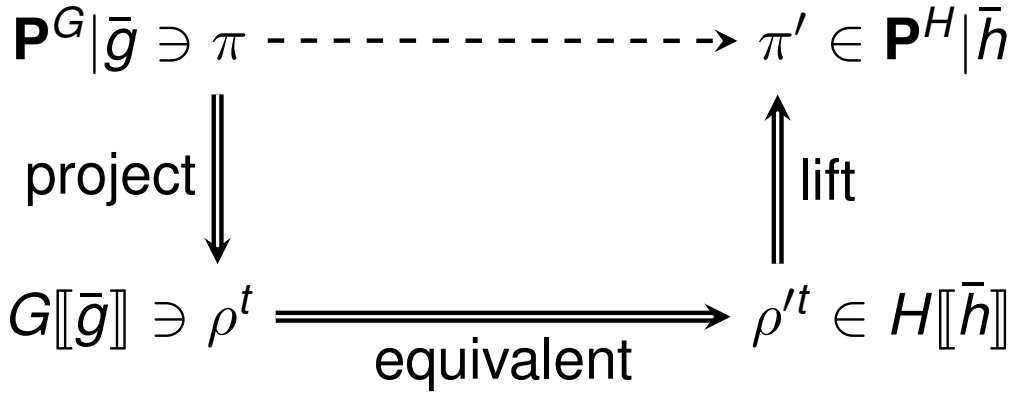
Why can we do this?

Since our formulas are in the prenex fragment:

- Evaluating the truth value of a formula $Q_1 x_1 \ldots Q_k x_k \varphi$ in P^G reduces to evaluating a boolean combination of the truth values of φ with respect to all possible assignments.
- The composition and finiteness properties imply that to consider all $G \in \mathbb{G}$ we have to evaluate boolean combinations over the same finite set of boolean variables.

$\exists x \mathbf{AF} c_x$

How we do it



Note: $\mathbf{CTL}^*_d \backslash \mathbf{X}$-equivalence is proved by stuttering bisimulation

Conclusion

... there is a tendency that the merits of the [composition method] are overlooked. This is unfortunate, because it excludes some interesting applications.

(Wolfgang Thomas).