Modular Supervisory Synthesis for Unknown Plant Models Using Active Learning

Fredrik Hagebring, Ashfaq Farooqui, and Martin Fabian

Department of Electrical Engineering,
Chalmers University of Technology,
Göteborg, Sweden

15th IFAC Workshop On Discrete Event Systems, 2020

Modular Supervisors

Modular Supervisors

Given a plant $G = \{G_1, G_2, \ldots, G_j\}$, let $K = \{K_1, K_2, \ldots K_i\}$ be a set of automata describing the *specifications* of the system. A set of supervisors $S = \{S_1, S_2, \ldots, S_i\}$ can then be calculated for each specification K_i in such a way that the synchronous composition of the supervisors results in a maximally permissive *controllable* supervisor[a].

[a] K. Åkesson, H. Flordal, and M. Fabian. "Exploiting modularity for synthesis and verification of supervisors". In: *IFAC Proceedings Volumes* 35.1 (2002). 15th IFAC World Congress, pp. 175 –180. ISSN: 1474-6670.

- Full synthesis
- Incremental synthesis

Problem

Given a set of specification (automata) $K = \{K_1, K_2, ..., K_n\}$, a PSH $H = \langle M, E, S \rangle$ and a simulation, all corresponding to a system:

- Can we learn a supervisor?
- What properties will the supervisor satisfy: Non-blocking? Controllable? Maximally Permissive?

Choosing modules to learn

- One supervisor per specification
- To guarantee maximally permissive system each specification needs to be combined with modules defined in the PSH that
  - directly share uncontrollable events with the specification
  - indirectly share uncontrollable events with the specification

Definition (Event Dependence)

Given an alphabet Σ' and a set $M = m_1, m_2, \ldots, m_i$ of modules, let

$$\mathrm{Dep}(M, \Sigma') = \{m_i \in M \mid E(m_i) \cap \Sigma' \neq \emptyset\}$$

Choosing modules to learn

Algorithm to define new modules

First initialize

$$\Sigma^{(1)} = \Sigma_u \cap \Sigma_{K_i}$$

$$M_{K_i}^{(1)} = \mathrm{Dep}(M, \Sigma^{(1)})$$

Then repeat the following statements until $\Sigma^{(n+1)} = \Sigma^{(n)}$.

$$\Sigma^{(n+1)} = \Sigma^{(n)} \cup (\Sigma_{M_{K_i}^{(n)}} \cap \Sigma_u)$$

$$M_{K_i}^{(n+1)} = \mathrm{Dep}(M_{K_i}^{(n)}, \Sigma^{(n)})$$

Defining supervisor modules (PSH + Spec)

- The supervisor learnt using the above PSH is maximally permissive.
- In some cases this could result in (inefficient) monolithic learning.

Incremental Learning and Synthesis

- Learn a supervisor for each specification and modules defined in the PSH that directly share uncontrollable events.
- Use existing tools to test if the obtained models are controllable, if not synthesize.
- For this, we need to learn additional modules: $M_G = M \setminus \bigcup_{K_i \in K} M_{K_i}$.

Updating E and S to handle specifications

$$E_{K_i} = \Sigma_{K_i} \cup \bigcup_{m \in M_{K_i}} E(m), \text{ and } S_{K_i} = \bigcup_{m \in M_{K_i}} S(m).$$

Defining supervisor modules (PSH + Spec)

New PSH $\langle M', E', S', K' \rangle$

$$M' = M_K \cup M_G$$

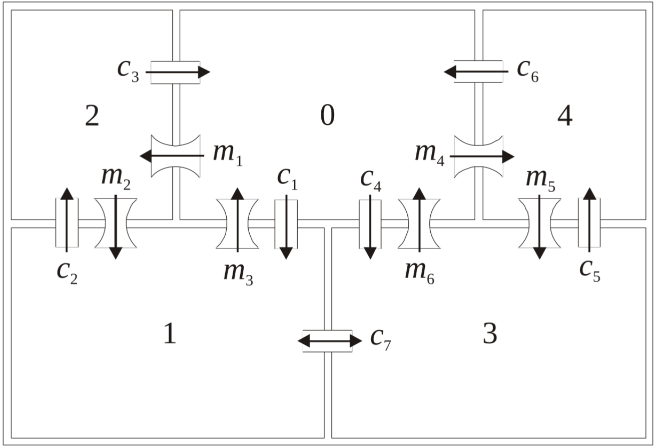
$$E'(m) = \begin{cases} E_{K_i}, & \text{if } m \in M_K \text{ and } m = M_{K_i} \\ E(m), & \text{if } m \in M_G \end{cases}$$

$$S'(m) = \begin{cases} S_{K_i}, & \text{if } m \in M_K \text{ and } m = M_{K_i} \\ S(m), & \text{if } m \in M_G \end{cases}$$

$$K'(m) = \begin{cases} K_i, & \text{where } m = M_{K_i} \\ \mathit{undefined}, & \text{otherwise.} \end{cases}$$

We can now apply the modular learning (with slight modifications) to this new PSH to obtain the set of supervisors.

Cat and Mouse example



Cat and Mouse example

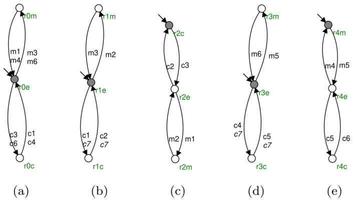


Fig. 2. Specifications for the different rooms ($Kr_0, Kr_1, Kr_2, Kr_3, Kr_4$, in that order) ensuring that only one of either the cat or the mouse can be present at a given time. Each state is identified using a unique name.

- $\Sigma_{Kr_0} = \{c_1, c_3, c_4, c_6, m_1, m_3, m_4, m_6\}$,
- $\Sigma_{Kr_1} = \{c_1, c_2, c_7, m_2, m_3\}$,
- $\Sigma_{Kr_2} = \{c_2, c_3, m_1, m_2\}$,
- $\Sigma_{Kr_3} = \{c_4, c_5, c_7, m_5, m_6\}$,
- $\Sigma_{Kr_4} = \{c_5, c_6, m_4, m_5\}$,

Cat and Mouse SUL/PSH

Simulation

- Let $\{R_0, R_1, R_2, R_3, R_4\}$ represent the different rooms.
- The simulator uses var_c and var_m to track the location of the animals.
- Initially, $var_c = R_2$ and $var_m = R_4$.

The PSH is defined as follows:

- $M = \{Cat, Mouse\}$,
- $E(Cat) = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$,
- $E(Mouse) = \{m_1, m_2, m_3, m_4, m_5, m_6\}$,
- $S(Cat) = \{var_c\}$,
- $S(Mouse) = \{var_m\}$,

Extending the PSH

$$M_{Kr_1} = \{Cat\}, \text{ and } M_{Kr_3} = \{Cat\};$$
$$E_{Kr_1} = \Sigma_{Kr_1} \cup E(Cat), \text{ and } E_{Kr_3} = \Sigma_{Kr_3} \cup E(Cat);$$
$$S_{Kr_1} = \{var_c, var_{Kr_1}\}, \text{ and } S_{Kr_3} = \{var_c, var_{Kr_3}\}.$$

$$M_K = \{M_{Kr_1}, M_{Kr_3}\}, \text{ and } M_G = \{Mouse\};$$

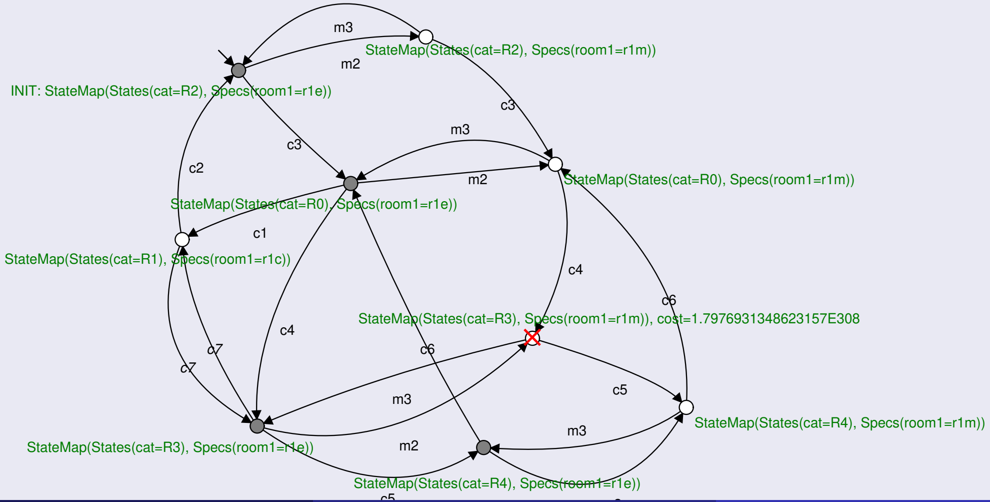
- The remaining specifications Kr_0, Kr_2, Kr_4 can be treated as supervisors.

Learning Algorithm

- Explore the state space in a breath-first search manner using the PSH to do so modularly.
- When building supervisors:
  - For each transition identified check if the specification allows it.
  - Specifications can be tracked using state variables.
  - If the identified transition is uncontrollable:
    - Mark the originating state as forbidden and check for controllability for all coreachable states.
    - Forbidden states will not be explored further.

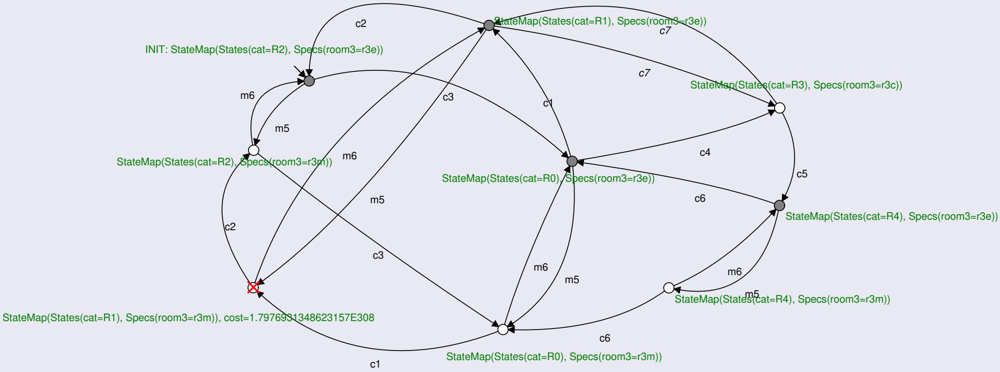
Cat and Mouse example

room 1 (M_{K1})



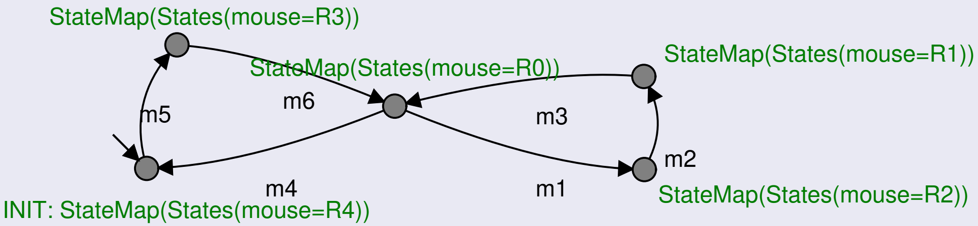
Cat and Mouse example

room 3 (M_{K3})



Cat and Mouse example

Mouse



The resulting supervisors (and plants) can then be used in existing tools to generate a maximally permissive controllable and non-blocking supervisor.

Notes on decomposable models and PSH

- Not all decomposable systems are learnable.
  - Consider a simulation which has a state variable for each room $\{R_0, R_1, R_2, R_3, R_4\}$
  - These variable can take the value *Cat* or *Mouse*
  - Such a simulation is valid but cannot result in a modular model using this technique

Concluding remarks

- This demonstrates the possibility to obtain supervisors for existing systems when models are not present.
- Larger systems like the AGV example can be solved within a short time.
- PSH is tricky to define, and needs to be in conformance with the simulation; hence, knowledge of the simulation helps in creating a meaningful PSH.
- Accuracy and performance of this method depends upon the PSH.

Future Work

- Study the application of these techniques to a diverse range of examples.
- Learning richer formalism, like EFA.
- Simplify, and (maybe) automate the creation of the PSH.
- Learning different perspectives of modeling.

Thank you!