Active Learning of Modular Plant Models

Ashfaq Farooqui, Fredrik Hagebring and Martin Fabian

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

15th IFAC Workshop On Discrete Event Systems, 2020
November 2020

Topic

Why learn models?

- Model based methods are being embraced within the industry.
- Tools that help with model based design are easily available.
- Bottleneck: Where do we get the models from?
- Manually building models is a challenge, time consuming, and prone to errors.
- Building models of legacy systems requires reverse engineering skills.
- If there already exists a system (simulation or physical), can we extract the behavioral model automatically?

Models

By model we mean the discrete behavior of the system represented by one or more deterministic automata.

Monolithic Model

$G = <Q, \Sigma, \delta, q_0>$

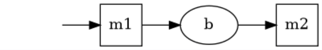
- Q is the set of *states*
- Σ is the *alphabet* containing the events
- $\delta: Q \times \Sigma \to Q$ is the partial *transition function*
- $q_0 \in Q$ is the *initial state* of the system

Modular Model

$G = G_1 || G_2 || \ldots || G_n$

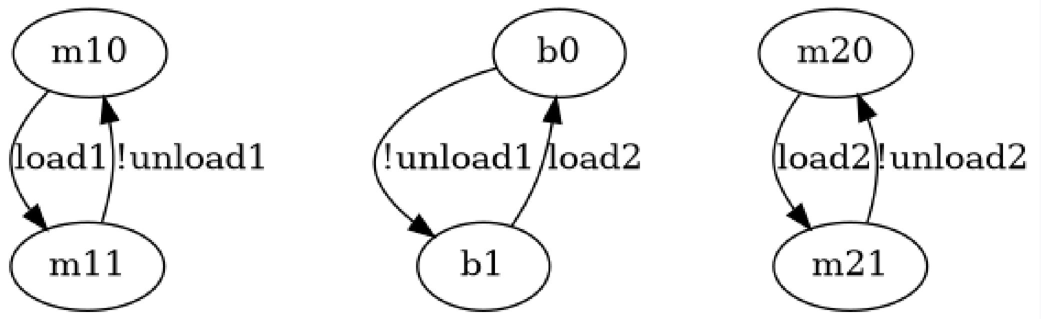
Machine Buffer Machine

Product flow



System Behavior

$\Sigma = \{load1, load2, unload1, unload2\}$

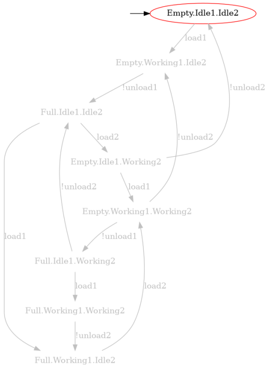


Machine Buffer Machine

To learn a model we require:

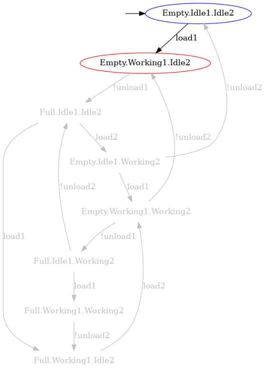
- Knowledge about the events.
- Possibility to interact and observe the internal state of the system.

Brute Force



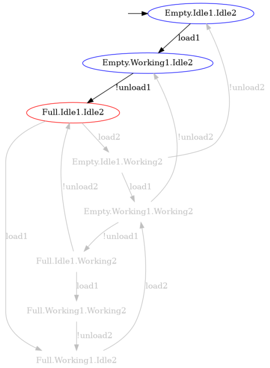
Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force



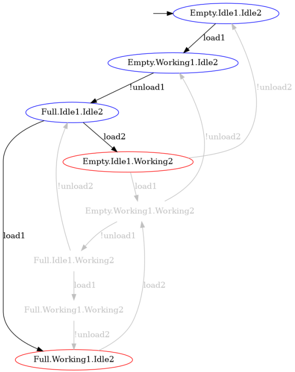
Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force



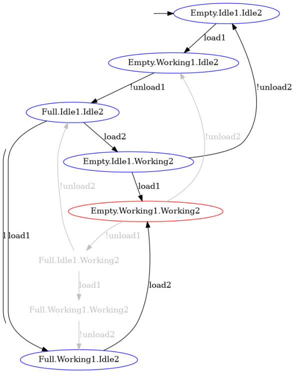
Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force

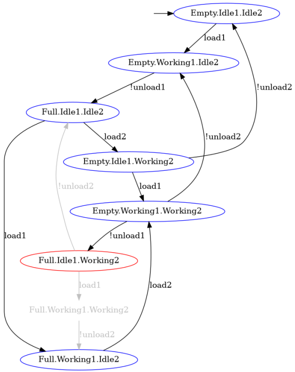


Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force

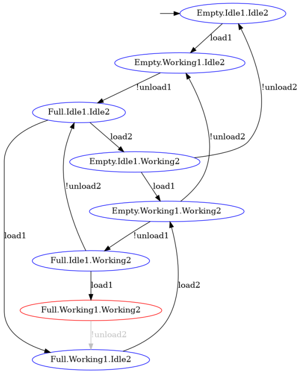

Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
load1
!unload2
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force



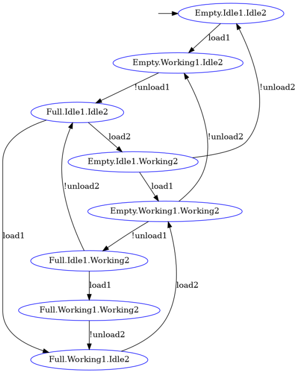
Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force



Empty.Idle1.Idle2
load1
Empty.Working1.Idle2
!unload1
!unload2
Full.Idle1.Idle2
load2
!unload2
Empty.Idle1.Working2
!unload2
load1
Empty.Working1.Working2
load1
!unload1
Full.Idle1.Working2
load1
load2
Full.Working1.Working2
!unload2
Full.Working1.Idle2

Brute Force



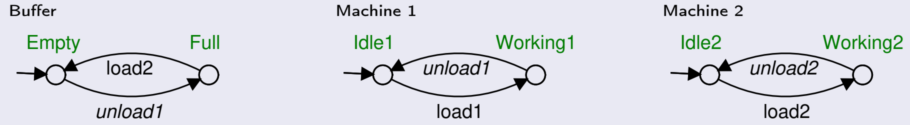
Can we instead learn smaller modules that together make up the complete system?

Topic

Aim

- Our work aims to alleviate the state-space explosion problem by exploring a smaller state-space rather than the monolithic one.
- This is done by exploiting the structural knowledge of the system.

Can we learn the MBM like so:



System Under Learning

We assume an interface to a simulation or the actual production code(in case of software) of the target. More importantly, it should be possible to interface with the system and

- run the discrete system by calling it externally.
- access to the set of state variables.
- be able to read and write these state variables.

System Under Learning

MBM Example

- State variables $= \{varB, varM_1, varM_2\}$
- Domain for the machines $\{idle, working\}$;
- Domain for the buffer $\{empty, full\}$
- State: <full,idle1,idle2>

Plant Structure Hypothesis

Provides structural information about the system and how the modules should be constructed.

PSH

Formally, the PSH is a 3-tuple $H = \langle M, E, S \rangle$, where:

- M is a set of identifiers for the modules;
- $E : M \rightarrow 2^{\Sigma}$ is the *event mapping*;
- $S : M \rightarrow 2^{V}$ is the *state mapping*;

Plant Structure Hypothesis

Example

- $M = \{M_1, M_2, Buffer\}$
- $E(M_1) = \{load_1, unload_1\}$
- $E(M_2) = \{load_2, unload_2\}$
- $E(B) = \{unload_1, load_2\}$
- $S(M_1) = \{varM_1\}$
- $S(M_2) = \{varM_2, varB\}$
- $S(B) = \{varB\}$

States and Projected States

- State is a unique valuation of state variables.
- Unique valuation of a subset of the state variables gives a projected state.

Projected States

- Global State: s = <full,idle1,idle2>;
- $P_{varB}(s) = <full>$

Topic

Initial State

M_1

M_2

Buffer

PSH

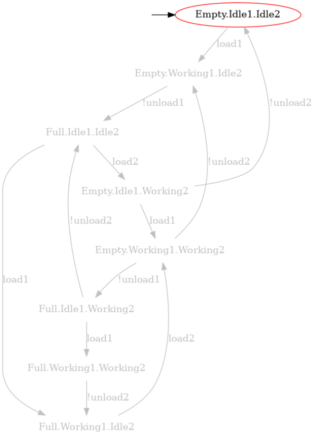
M={M_1,M_2,Buffer}

E(M_1)= {load$_1$, unload$_1$} S(M_1) = {$varM_1$ }

E(M_2)= {load$_2$, unload$_2$} S(M_2) = {$varM_2$,$varB$ }

E(B)= {unload$_1$, load$_2$} S(B) = {$varB$ }

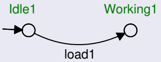
Example (Simulation)



Learning a modular plant – Step 1

Step 1

M_1



M_2

Buffer

PSH

M={M_1,M_2,Buffer}

E(M_1)= {load$_1$, unload$_1$}

E(M_2)= {load$_2$, unload$_2$}

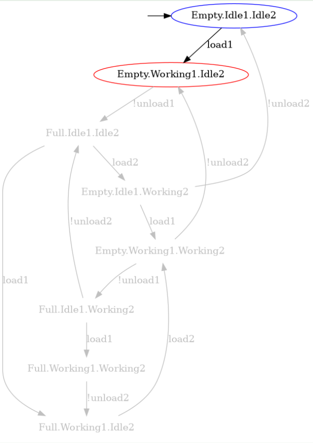
E(B)= {unload$_1$, load$_2$}

S(M_1) = {$varM_1$ }

S(M_2) = {$varM_2$,$varB$ }

S(B) = {$varB$ }

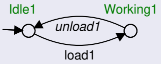
Example (Simulation)



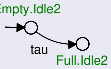
Learning a modular plant – Step 2

Step 2

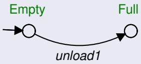
M_1



M_2



Buffer



PSH

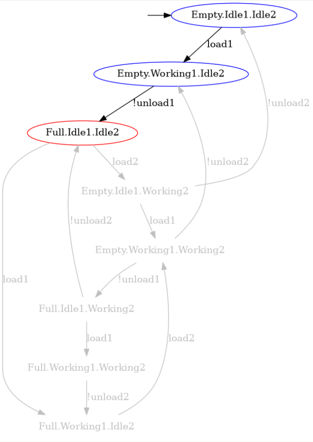
$\mathrm{M}=\{M_1, M_2, \mathrm{Buffer}\}$

$\mathrm{E}(M_1)=\{\mathrm{load}_1, \mathrm{unload}_1\}$ $\quad$ $\mathrm{S}(M_1)=\{varM_1\}$

$\mathrm{E}(M_2)=\{\mathrm{load}_2, \mathrm{unload}_2\}$ $\quad$ $\mathrm{S}(M_2)=\{varM_2, varB\}$

$\mathrm{E}(B)=\{\mathrm{unload}_1, \mathrm{load}_2\}$ $\quad$ $\mathrm{S}(B)=\{varB\}$

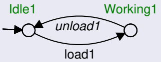
Example (Simulation)



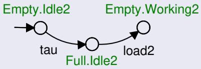
Learning a modular plant – Step 3

Step 3

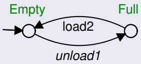
M_1



M_2



Buffer



PSH

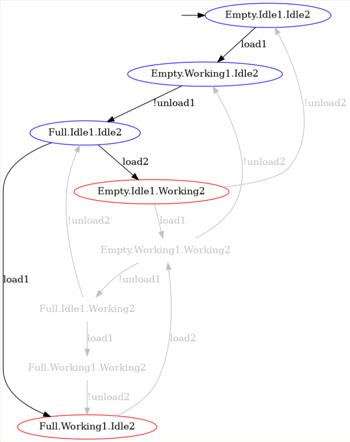
$$\mathbf{M}=\{M_1, M_2, \mathbf{Buffer}\}$$

$\mathbf{E}(M_1) = \{\mathbf{load}_1, \mathbf{unload}_1\}$ $\mathbf{S}(M_1) = \{varM_1\}$

$\mathbf{E}(M_2) = \{\mathbf{load}_2, \mathbf{unload}_2\}$ $\mathbf{S}(M_2) = \{varM_2, varB\}$

$\mathbf{E}(B) = \{\mathbf{unload}_1, \mathbf{load}_2\}$ $\mathbf{S}(B) = \{varB\}$

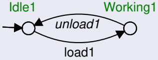
Example (Simulation)



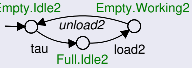
Learning a modular plant – Step 4

Step 4

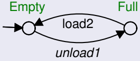
M_1



M_2



Buffer



PSH

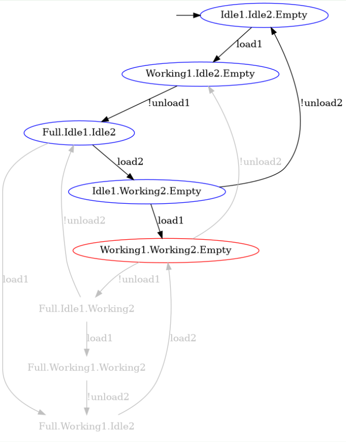
M={M_1,M_2,Buffer}

E(M_1)= {load$_1$, unload$_1$} **S(M_1) = {$varM_1$ }**

E(M_2)= {load$_2$, unload$_2$} **S(M_2) = {$varM_2$,$varB$ }**

E(B)= {unload$_1$, load$_2$} **S(B) = {$varB$ }**

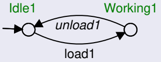
Example (Simulation)



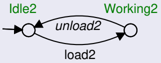
Learning a modular plant – Termination

Termination

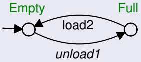
M_1



M_2



Buffer



PSH

M={M_1,M_2,Buffer}

E(M_1)= {load$_1$, unload$_1$}

E(M_2)= {load$_2$, unload$_2$}

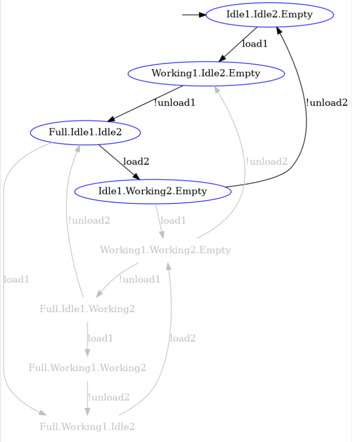
E(B)= {unload$_1$, load$_2$}

S(M_1) = {$varM_1$ }

S(M_2) = {$varM_2$,$varB$ }

S(B) = {$varB$ }

Example (Simulation)



Topic

Limitations

To be able to learn the system modularly we are limited by:

- A deterministic system.
- Knowledge about the events and state variables.
- The discrete simulation of the system. With the possibility to set the state variables in the simulation, execute events, and observe the updated state variables.
- Definition of Plant Structure Hypothesis (PSH).
- Decomposable system as defined by the PSH.

Topic

Conclusion

- It was possible to learn a modular plant of a system given its simulation.
- Successfully applied this algorithm to learn a model of a sub-component in an autonomous car[1].
- The accuracy and performance of this method depends upon the defined PSH.
- Given specifications can we directly learn a modular supervisor?

[1] Yuvaraj Selvaraj et al. "Automatically Learning Formal Models: An Industrial Case from Autonomous Driving Development". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '20. Virtual Event, Canada: Association for Computing Machinery, 2020. ISBN: 9781450381352.

Thank You!