Automatically Learning Formal Models: An Industrial Case from Autonomous Driving Development

Yuvaraj Selvaraj[1,2] Ashfaq Farooqui [2]
Ghazaleh Panahandeh [1] Martin Fabian [2]

[1]Zenseact AB

[2]Chalmers University of Technology

MASE'20
16 October 2020

Introduction

Background

- Industrial PhD project
  - Formal verification for Autonomous Driving (AD) software development
  - Collaboration between Zenseact and Chalmers University of Technology
- Zenseact[1]
  - Software company for AD and Advanced Driver Assistance Systems (ADAS)
  - Offices in Sweden and China
  - Robustness and safety are top priorities

[1] formerly known as Zenuity

Introduction

Motivation

- Formal verification needs a model of the system
- Experience[2] shows manual model construction is an obstacle

Manual model construction

- Potentially error prone
- Intractable for large systems

[2] Yuvaraj Selvaraj, Wolfgang Ahrendt, and Martin Fabian. "Verification of Decision Making Software in an Autonomous Vehicle: An Industrial Case Study". In: *Formal Methods for Industrial Critical Systems*. LNCS 11687. Springer, 2019.

Research Question

Research Question

How can we address the challenge of manual model construction?

Approach

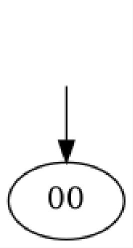
- Learn an automata model of the existing program code
- Active automata learning to automatically obtain formal models
- Two learning algorithms are evaluated:
  - L* algorithm[3]
  - Modular Plant Learner (MPL)[4]

[3] Ashfaq Farooqui and Martin Fabian. "Synthesis of Supervisors for Unknown Plant Models Using Active Learning". In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. Vancouver, BC, Canada: IEEE, 2019, pp. 502–508.

[4] Ashfaq Farooqui, Fredrik Hagebring, and Martin Fabian. "Active Learning of Modular Plant Models". In: WODES 2020.

Learns a model consisting of interacting components

MPL

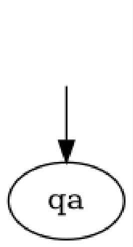


Learning Algorithms

L*

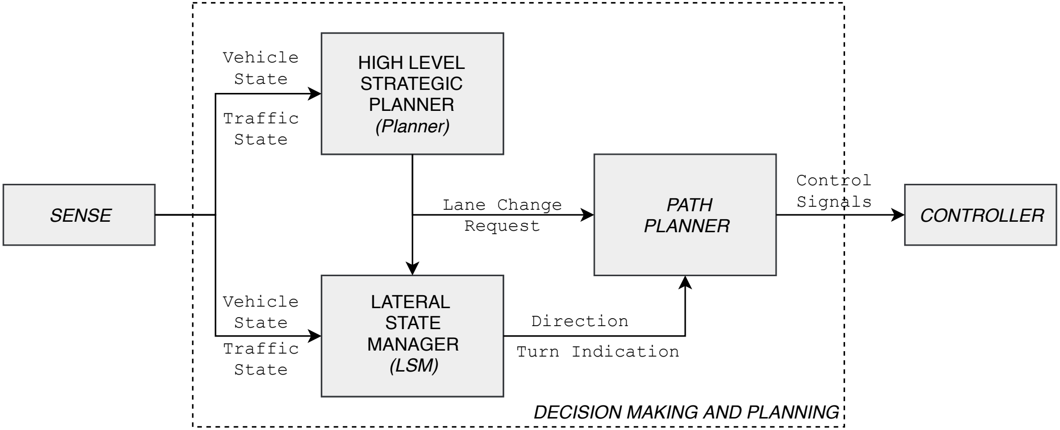
Learns a global regular language

L*



System Under Learning (SUL)

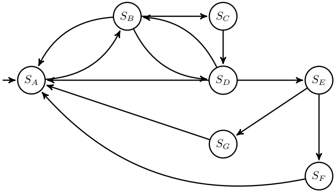
Lateral State Manager (LSM)



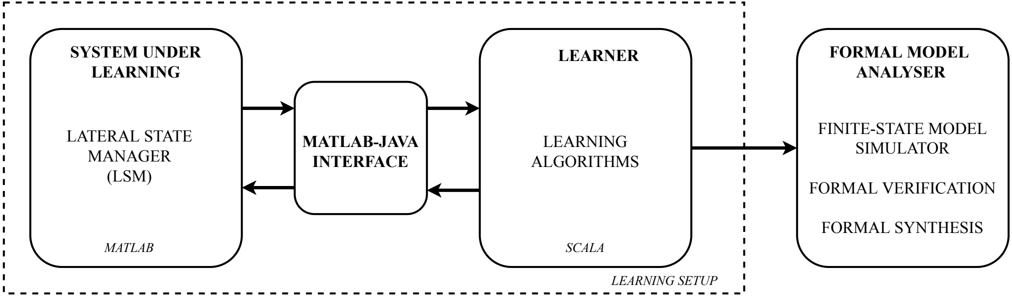
System Under Learning (SUL)

LSM

- LSM is programmed in MATLAB
- Inputs from other sub-components are abstracted
- Functionality to run and observe LSM by the learner is introduced



Learning Setup



- The interface provides a standard API to run and observe the SUL
- Provide information to the learner to interpret the observed information

Outcomes

- L* algorithm did not terminate and hence failed to learn a complete model
- MPL manages to learn a model that was validated with good confidence
- The obtained model was simulated alongside the original code to check correctness

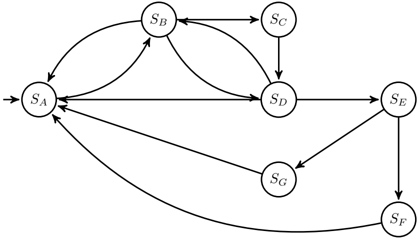


Figure: A meta-level finite-state abstraction of the LSM

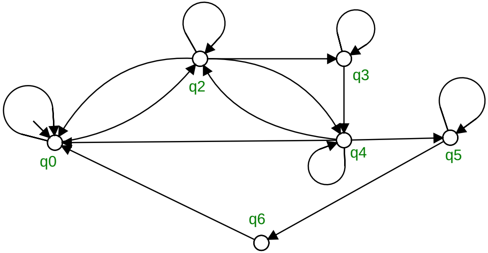


Figure: Language Minimised model

Insights

- Practical challenges
  - Interface between SUL and learner crucial for scaling up
  - Possibility to use stand-alone libraries and/or automatic abstraction
  - Trade-off between states and events
- Towards formal software development
  - Proof of concept for successfully learning formal models
  - Approach is independent of semantics of implementation language
  - Enables the use of formal methods for continuous software development and software reverse engineering

Future Work

- Study the application of these techniques to a diverse range of examples
- Learning richer formalism, like EFA
- Explore possibilities of incorporating this approach in day to day development

Thank you for listening!