Time to Learn – Learning Timed Automata from Tests

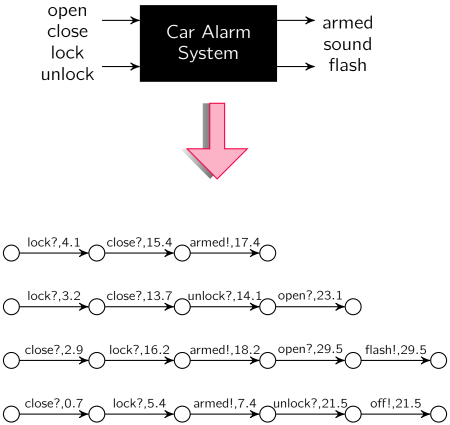
Martin Tappler Bernhard K. Aichernig Kim Guldstrand Larsen
Florian Lorber

Institute of Software Technology, Graz University of Technology, Austria
Department of Computer Science, Aalborg University

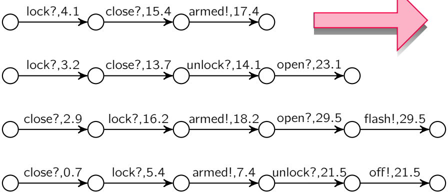
August 28^{th}, 2019

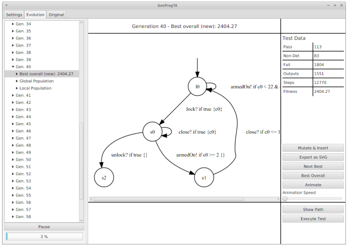
Learning a Car Alarm System

Learning a Car Alarm System

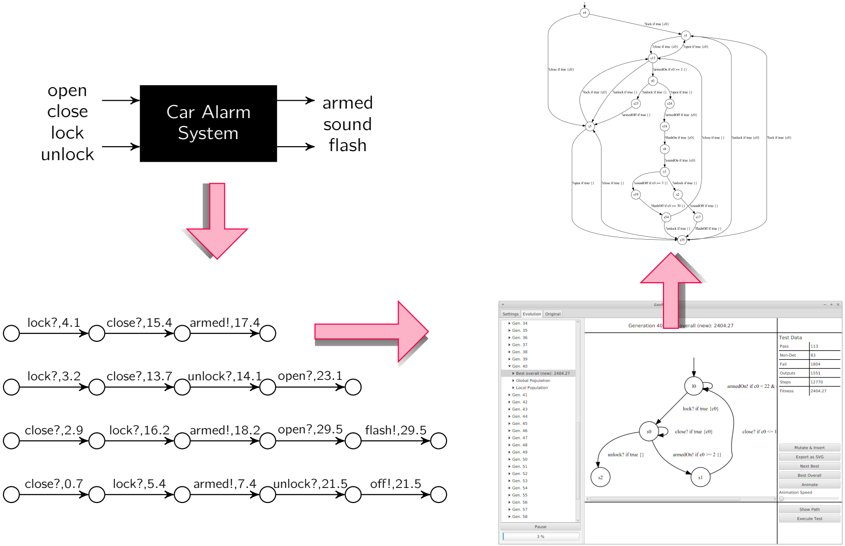


Learning a Car Alarm System





Learning a Car Alarm System



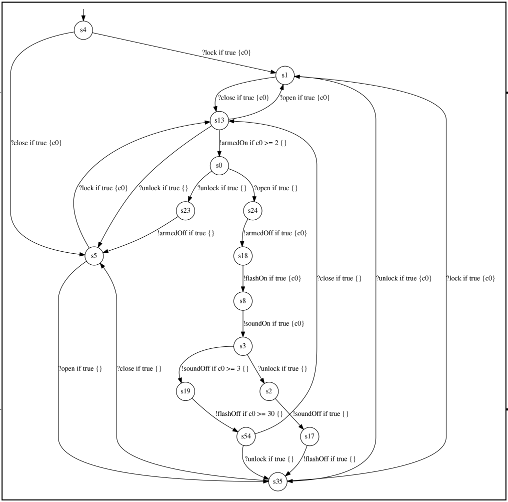
Learning a Car Alarm System

open

close

lock

unlock

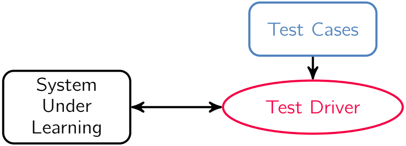


armed

sound

flash

Motivation – Learning-Based Verification

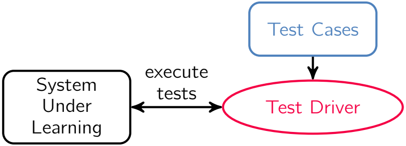


$$out_{\mathrm{SUT}}(Ping) =$$

$$out_{\mathrm{SUT}}(Con) =$$

$$out_{\mathrm{SUT}}(Con \cdot Ping) =$$

Motivation – Learning-Based Verification

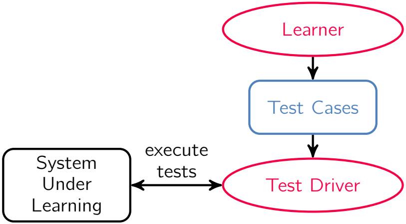


$$out_{\mathrm{SUT}}(Ping) = ConC$$

$$out_{\mathrm{SUT}}(Con) = ConAck$$

$$out_{\mathrm{SUT}}(Con \cdot Ping) = ConAck \cdot Pong$$

Motivation – Learning-Based Verification

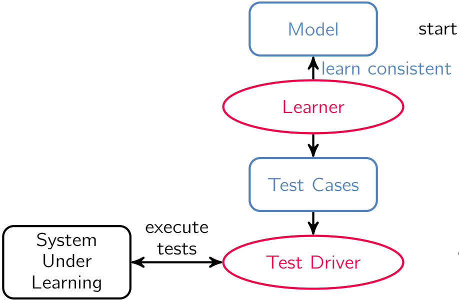


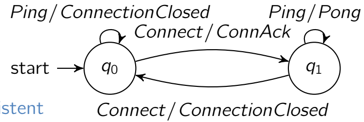
$$out_{\mathrm{SUT}}(Ping) = ConC$$

$$out_{\mathrm{SUT}}(Con) = ConAck$$

$$out_{\mathrm{SUT}}(Con \cdot Ping) = ConAck \cdot Pong$$

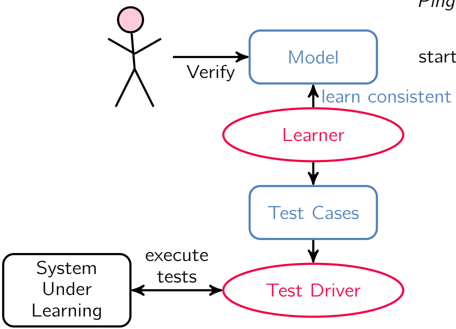
Motivation – Learning-Based Verification

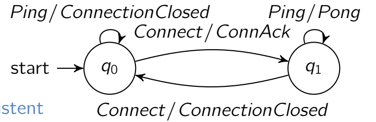




$$out_{\mathrm{M}}(Ping) = ConC$$
$$out_{\mathrm{SUT}}(Ping) = ConC$$
$$out_{\mathrm{M}}(Con) = ConAck$$
$$out_{\mathrm{SUT}}(Con) = ConAck$$
$$out_{\mathrm{M}}(Con \cdot Ping) = ConAck \cdot Pong$$
$$out_{\mathrm{SUT}}(Con \cdot Ping) = ConAck \cdot Pong$$

Motivation – Learning-Based Verification





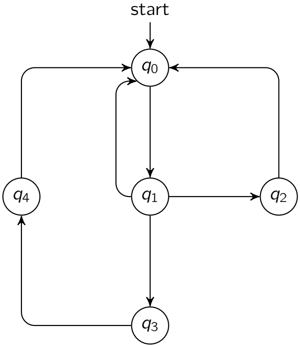
$$out_{\mathrm{M}}(Ping) = ConC$$
$$out_{\mathrm{SUT}}(Ping) = ConC$$
$$out_{\mathrm{M}}(Con) = ConAck$$
$$out_{\mathrm{SUT}}(Con) = ConAck$$
$$out_{\mathrm{M}}(Con \cdot Ping) = ConAck \cdot Pong$$
$$out_{\mathrm{SUT}}(Con \cdot Ping) = ConAck \cdot Pong$$

Verification

- Model checking [Fiterau-Brostean et al., 2016], comparison of models [Aarts et al., 2012, Tappler et al., 2017]
- Issue: "we had to eliminate timing based behavior as well as re-transmissions" [Fiterau-Brostean et al., 2016]

Timed Automata

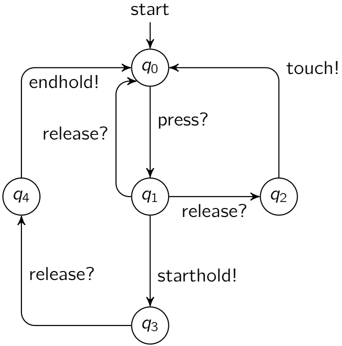
- Finite automata ...
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



A Lamp Touch Sensor

Timed Automata

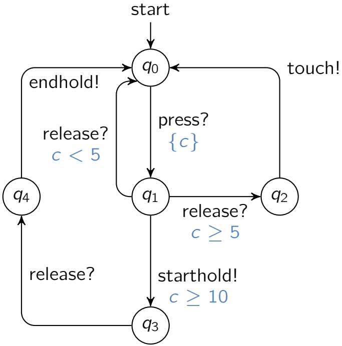
- Finite automata . . .
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



A Lamp Touch Sensor

Timed Automata

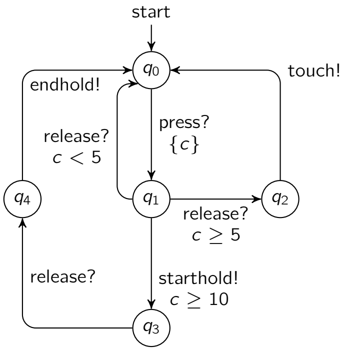
- Finite automata ...
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



A Lamp Touch Sensor

Timed Automata

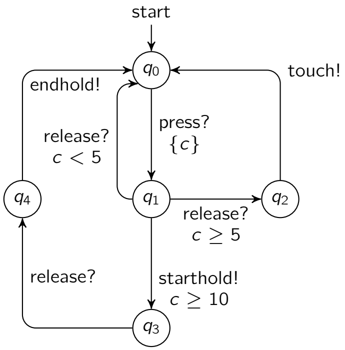
- Finite automata ...
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



A Lamp Touch Sensor

Timed Automata

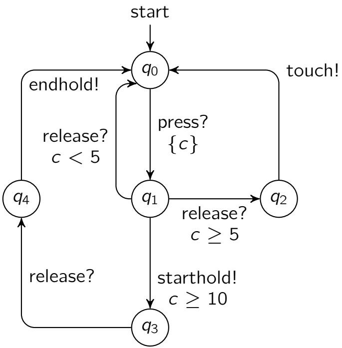
- Finite automata ...
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



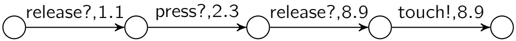
A Lamp Touch Sensor

Timed Automata

- Finite automata ...
- with inputs and outputs
- extended with real-valued clocks
  - used in guards
  - reset upon transitions
- constraints limiting sojourn time
- Assumptions for testing [Hessel et al., 2003]:
  - output urgent: outputs fire as soon as possible
  - input enabled: inputs must be accepted
  - deterministic



A Lamp Touch Sensor



Automata Learning for Timed Systems

- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

Automata Learning for Timed Systems

- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]:

  $\rightarrow$ does not distinguish inputs and outputs
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]

  $\rightarrow$ high runtime complexity
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

Automata Learning for Timed Systems

- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]:

  → does not distinguish inputs and outputs
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]

  → high runtime complexity
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

Automata Learning for Timed Systems

- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]:

  → does not distinguish inputs and outputs
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]

  → high runtime complexity
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

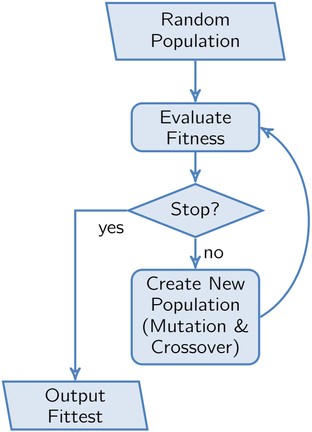
Automata Learning for Timed Systems

- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]:
  - → does not distinguish inputs and outputs
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]
  - → high runtime complexity
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

Automata Learning for Timed Systems

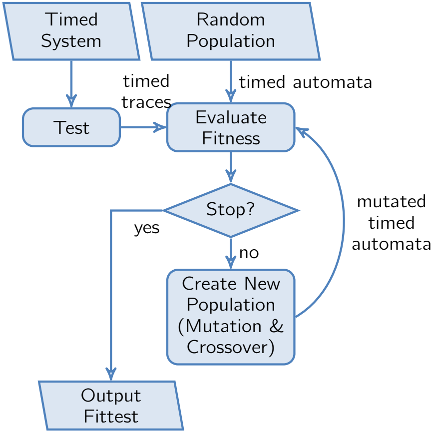
- Motivation: Model-based analysis for black-box timed systems
- Existing approaches:
  - Passive learning of real-time automata [Verwer et al., 2010, Verwer et al., 2012]:
  - $\rightarrow$ does not distinguish inputs and outputs
  - Active learning of event-recording automata [Grinchtein et al., 2010, Grinchtein et al., 2006]
  - $\rightarrow$ high runtime complexity
  - Both: restrictions on clock resets
- Promising results of genetic programming in program synthesis (e.g. mutual exclusion algorithms) [Katz and Peled, 2017]
- Apply genetic programing for timed automata
- Focus: generate models for testing
  - input-enabled, arbitrary clock resets

Genetic Programming



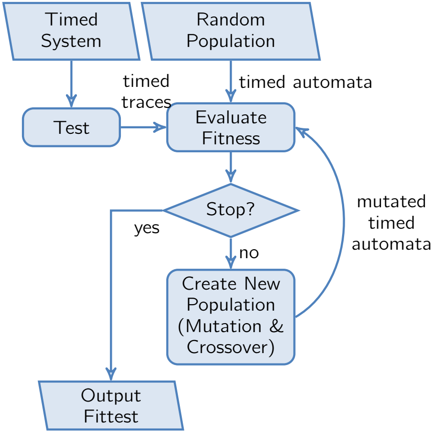
1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

Genetic Programming of TA – Basic



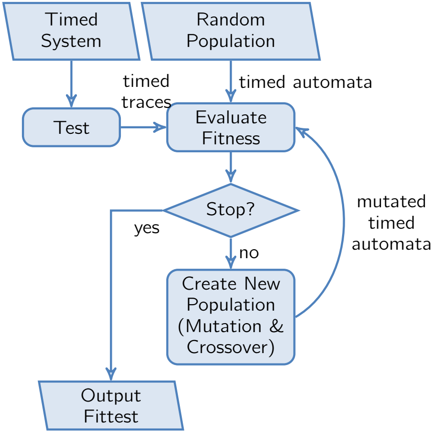
1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

Genetic Programming of TA – Basic



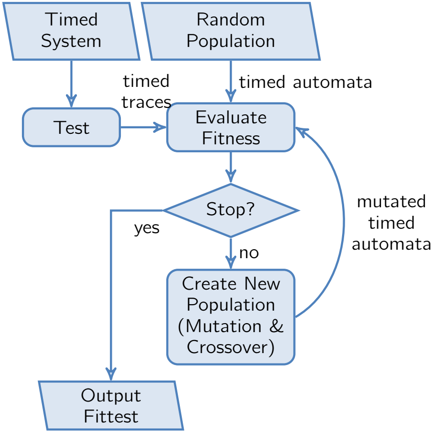
1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

Genetic Programming of TA – Basic



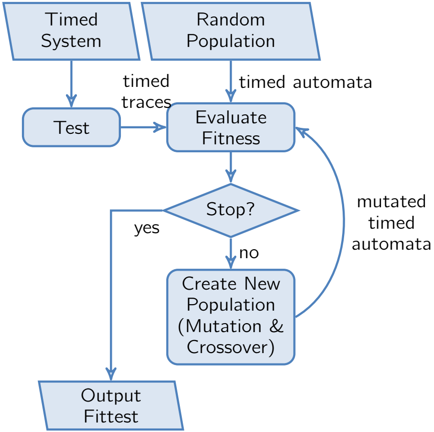
1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

Genetic Programming of TA – Basic



1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

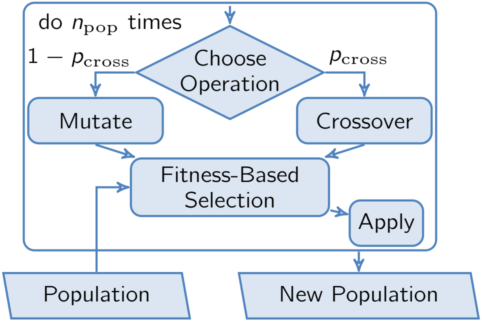
Genetic Programming of TA – Basic



1. Create small random timed automata
2. Test system to get timed traces
3. Fitness: simulate automata
   - # accepted traces
   - # outputs of accepted traces
   - determinism
   - penalty for model size
4. New population: mutate & crossover
5. Stop if all traces accepted or max. # rounds

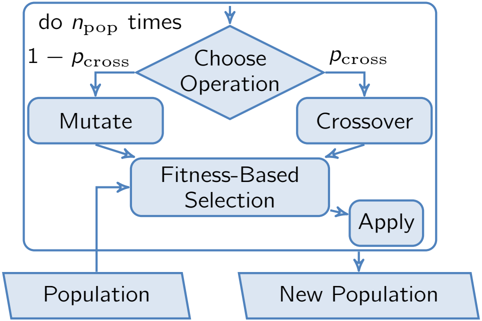
Creating a New Population – Detailed

- Probabilistic choice between mutation and crossover
- Fitness-based selection of parents from population
- Repeat n_{pop} times

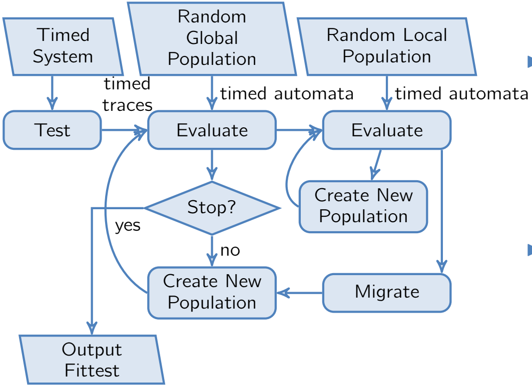


Creating a New Population – Detailed

- Probabilistic choice between mutation and crossover
- Fitness-based selection of parents from population
- Repeat n_{pop} times



Genetic Programming of TA – Optimized



- Two populations:
  1. Global search: all timed traces
  2. Local search: timed traces not accepted by global population
- From local search to global search:
  - Migration
  - Crossover

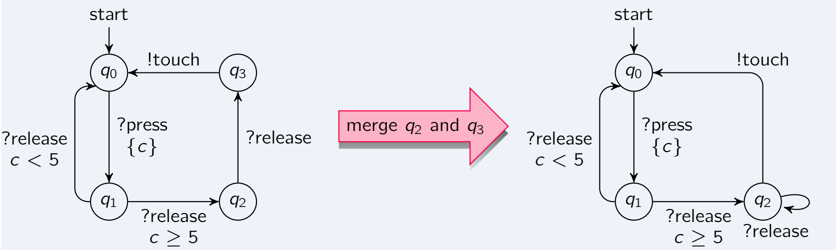
Mutation & Crossover (1)

- Mutation operators for changing all aspects of timed automata
- Chosen at random
- An operator inspired by passive automata learning: merge location

Mutation & Crossover (1)

- Mutation operators for changing all aspects of timed automata
- Chosen at random
- An operator inspired by passive automata learning: merge location

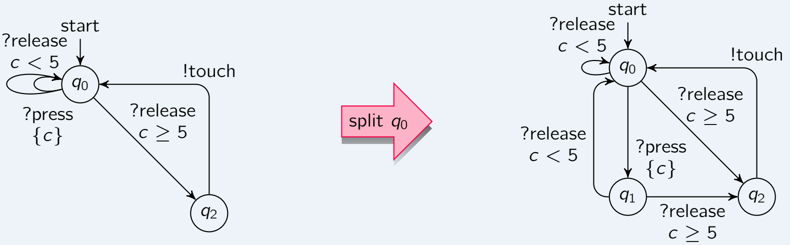
Merge Location Example



Mutation & Crossover (2)

- An operator inspired by active automata learning: split location

Split Location Example

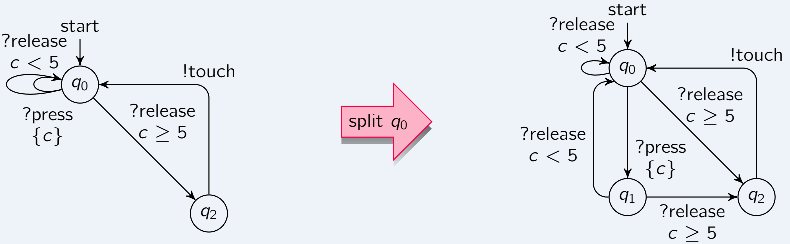


- Crossover: randomised product
  - explore parents and synchronise on labels
  - random combination of parents' edges

Mutation & Crossover (2)

- An operator inspired by active automata learning: split location

Split Location Example



- Crossover: randomised product
  - explore parents and synchronise on labels
  - random combination of parents' edges

Challenge: Parameter Configuration

- Lots of Parameters
  - # clocks, clock-bound range
  - weights for fitness computation
  - # tests, population size, # generations, test length
  - crossover probability

→ We have guidelines

- Some are fixed

Challenge: Parameter Configuration

- Lots of Parameters
  - # clocks, clock-bound range
  - weights for fitness computation
  - # tests, population size, # generations, test length
  - crossover probability

$\rightarrow$ We have guidelines

- Some are fixed

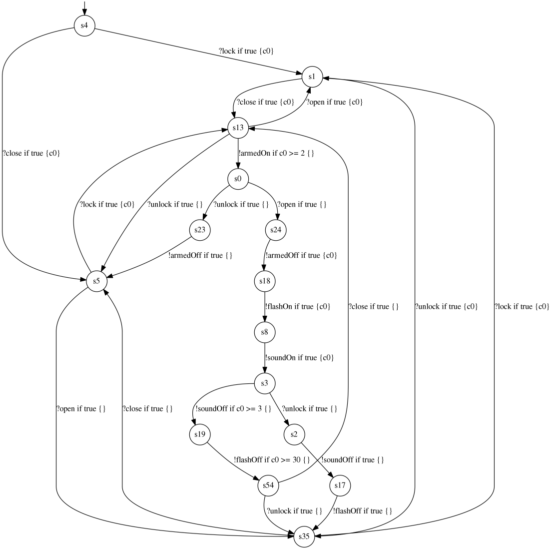
Challenge: Parameter Configuration

- Lots of Parameters
  - # clocks, clock-bound range
  - weights for fitness computation
  - # tests, population size, # generations, test length
  - crossover probability

$\rightarrow$ We have guidelines

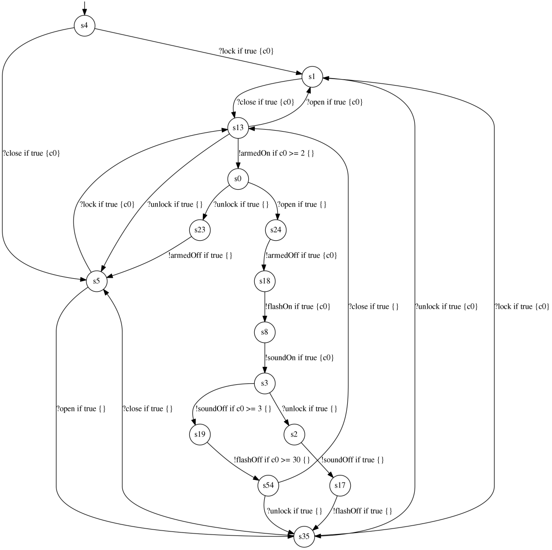
- Some are fixed

Experiments – a Learned Model (1)



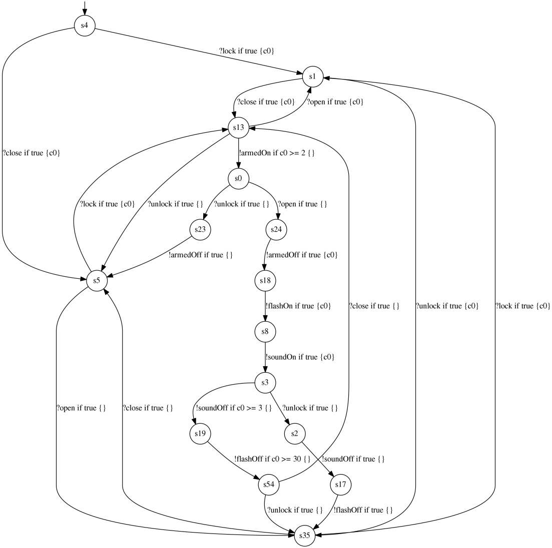
- Automatic generation of human-readable models
- Experiments with:
  - 40 random TA
  - 4 TA from the literature
  - up to 26 locations and 1 clock
  - up to 10 locations and 2 clocks
- Evaluation
  1. learn from training data
  2. simulate on test data

Experiments – a Learned Model (1)



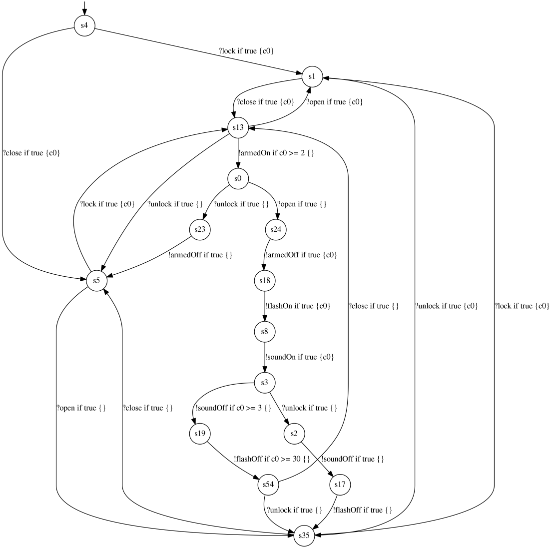
- Automatic generation of human-readable models
- Experiments with:
  - 40 random TA
  - 4 TA from the literature
  - up to 26 locations and 1 clock
  - up to 10 locations and 2 clocks
- Evaluation
  1. learn from training data
  2. simulate on test data

Experiments – a Learned Model (1)



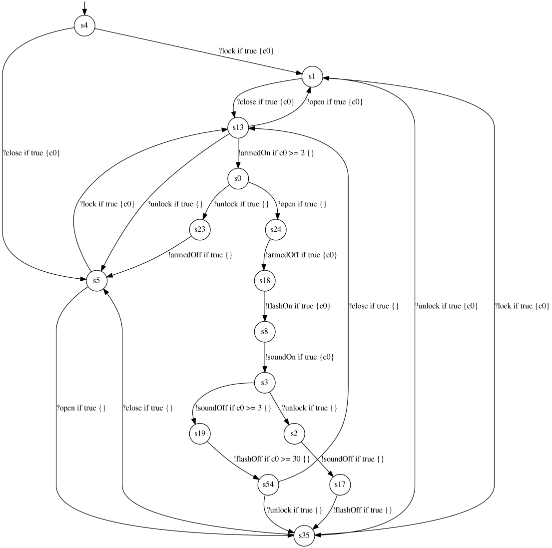
- Automatic generation of human-readable models
- Experiments with:
  - 40 random TA
  - 4 TA from the literature
  - up to 26 locations and 1 clock
  - up to 10 locations and 2 clocks
- Evaluation
  1. learn from training data
  2. simulate on test data

Experiments – a Learned Model (2)



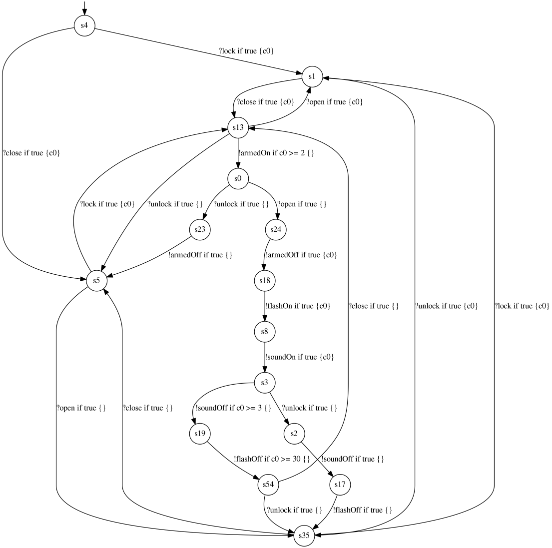
- Results:
  - successfully learned all 44 models
  - consistent with given training data
  - high accuracy on test data
- Runtime:
  - several minutes and up to 20 hours
  - not yet parallelized
- GUI demo: link in paper

Experiments – a Learned Model (2)



- Results:
  - successfully learned all 44 models
  - consistent with given training data
  - high accuracy on test data
- Runtime:
  - several minutes and up to 20 hours
  - not yet parallelized
- GUI demo: link in paper

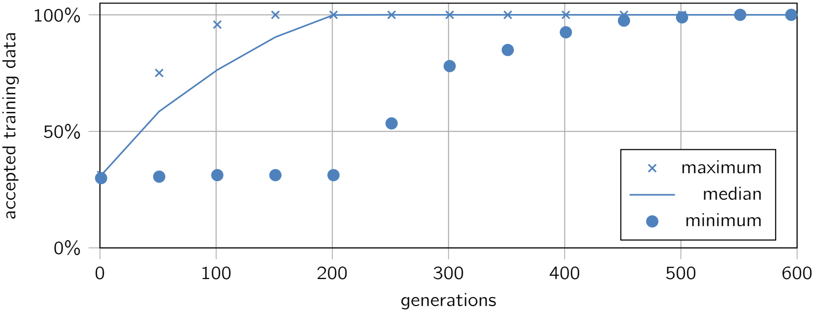
Experiments – a Learned Model (2)



- Results:
  - successfully learned all 44 models
  - consistent with given training data
  - high accuracy on test data
- Runtime:
  - several minutes and up to 20 hours
  - not yet parallelized
- GUI demo: link in paper

Experiments – Evolution of Fitness

Fitness of Car Alarm System Models

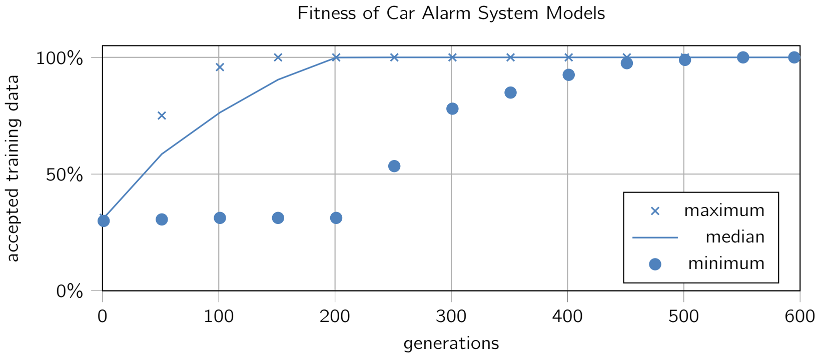


- Early generations accept only initial inputs
- Further behaviour continuously added

→ Random generation infeasible

- Final generations decrease model size

Experiments – Evolution of Fitness

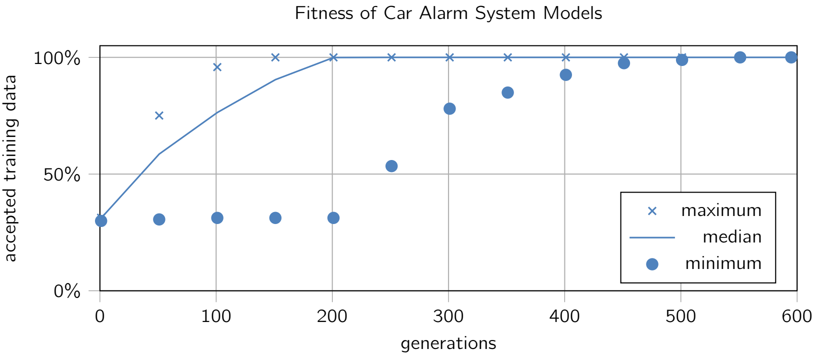


- Early generations accept only initial inputs
- Further behaviour continuously added

→ Random generation infeasible

- Final generations decrease model size

Experiments – Evolution of Fitness



- Early generations accept only initial inputs
- Further behaviour continuously added

→ Random generation infeasible

- Final generations decrease model size

Concluding Remarks

Summary

- Genetic Programming for timed automata including mutation, crossover, subpopulations, and fine-grained fitness computation
- Evaluated on 44 timed automata used as black boxes
  - up to 26 locations
  - up to two clocks with arbitrary resets
- Implemented in a tool

Concluding Remarks

Summary

- Genetic Programming for timed automata including mutation, crossover, subpopulations, and fine-grained fitness computation
- Evaluated on 44 timed automata used as black boxes
  - up to 26 locations
  - up to two clocks with arbitrary resets
- Implemented in a tool

Conclusion

- Successfully learned medium-sized models from tests
- Future work:
  - active learning
  - relaxing assumptions
  - synthesis via model-checking-based fitness computation

Concluding Remarks

Summary

- Genetic Programming for timed automata including mutation, crossover, subpopulations, and fine-grained fitness computation
- Evaluated on 44 timed automata used as black boxes
  - up to 26 locations
  - up to two clocks with arbitrary resets
- Implemented in a tool

Conclusion

- Successfully learned medium-sized models from tests
- Future work:
  - active learning
  - relaxing assumptions
  - synthesis via model-checking-based fitness computation

Thank you!

Aarts, F., Kuppens, H., Tretmans, J., Vaandrager, F. W., and Verwer, S. (2012).
Learning and testing the bounded retransmission protocol.
In Heinz, J., de la Higuera, C., and Oates, T., editors, *Proceedings of the Eleventh International Conference on Grammatical Inference, ICGI 2012, University of Maryland, College Park, USA, September 5-8, 2012*, volume 21 of *JMLR Proceedings*, pages 4–18. JMLR.org.

Fiterau-Brostean, P., Janssen, R., and Vaandrager, F. W. (2016).
Combining model learning and model checking to analyze TCP implementations.
In Chaudhuri, S. and Farzan, A., editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pages 454–471. Springer.

Grinchtein, O., Jonsson, B., and Leucker, M. (2010).
Learning of event-recording automata.

Theor. Comput. Sci., 411(47):4029–4054.

Grinchtein, O., Jonsson, B., and Pettersson, P. (2006).
Inference of event-recording automata using timed decision trees.
In Baier, C. and Hermanns, H., editors, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 435–449. Springer.

Hessel, A., Larsen, K. G., Nielsen, B., Pettersson, P., and Skou, A. (2003).
Time-optimal real-time test case generation using UPPAAL.
In *FATES 2003*, volume 2931 of *LNCS*, pages 114–130. Springer.

Katz, G. and Peled, D. (2017).
Synthesizing, correcting and improving code, using model checking-based genetic programming.
STTT, 19(4):449–464.

Tappler, M., Aichernig, B. K., and Bloem, R. (2017).
Model-based testing IoT communication via active automata learning.
In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 276–287. IEEE Computer Society.

Verwer, S., de Weerdt, M., and Witteveen, C. (2010).
A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data.
In Sempere, J. M. and García, P., editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, pages 203–216. Springer.

Verwer, S., de Weerdt, M., and Witteveen, C. (2012).
Efficiently identifying deterministic real-time automata from labeled data.

Machine Learning, 86(3):295–333.