Seminar Paper Presentation

Mario Wagner

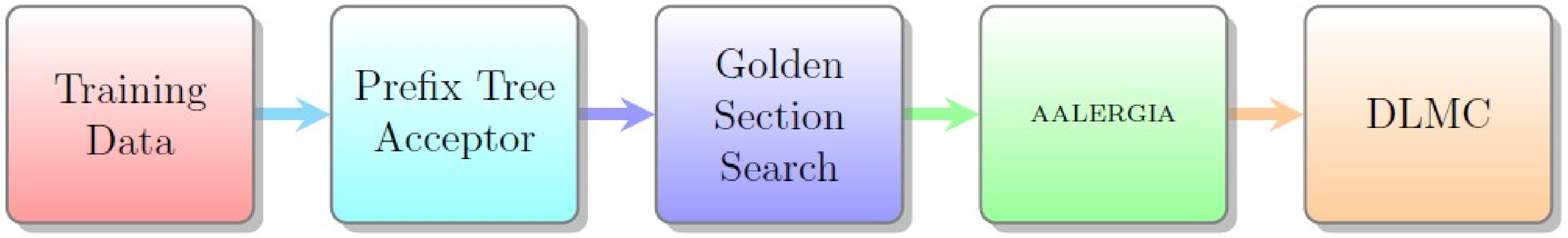
Problem/task

- Passive learning with AALERGIA
  - implement AALERGIA in python
- Is it possible to learn a probabilistic timed automaton with AALERGIA?
- Workflow
  - PTA → MDP→Traces→AALERGIA→DTMC→LTL model check

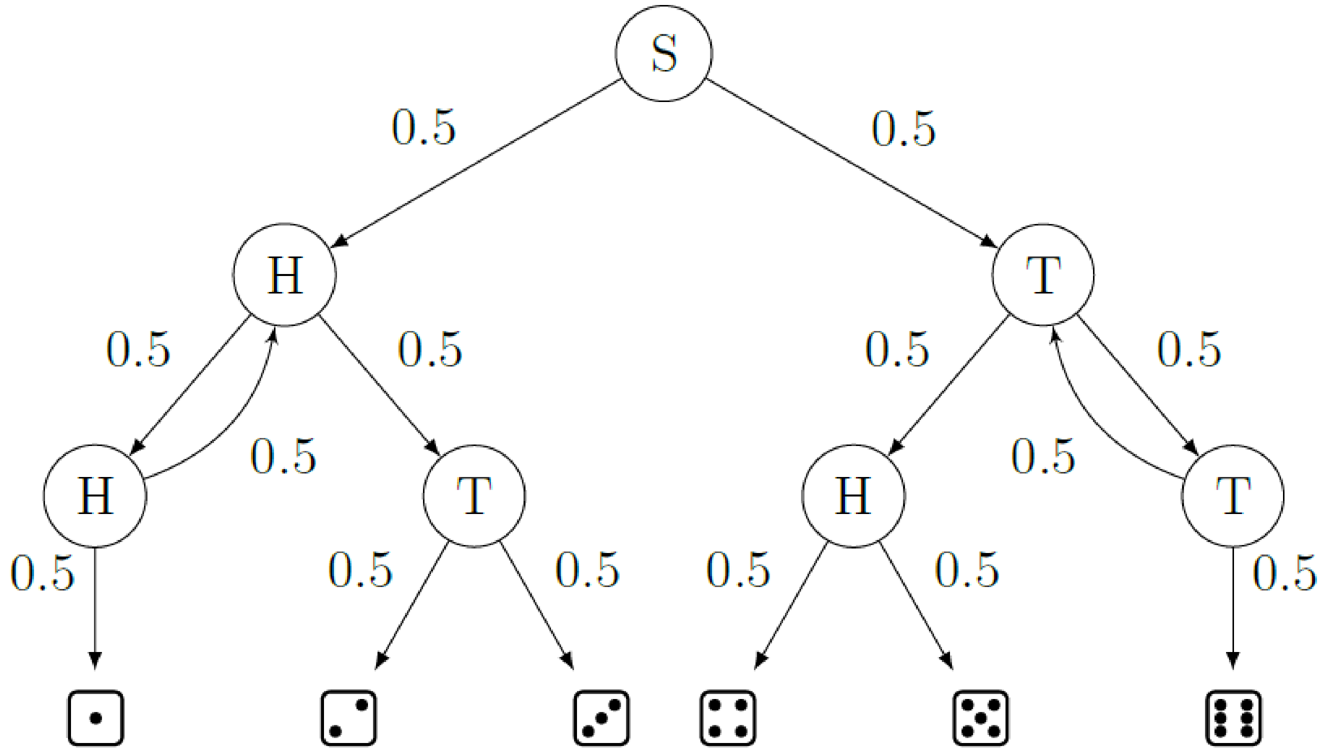
AALERGIA

- Obtaining accurate system models manually is a time-consuming process
- AALERGIA offers automatic construction of a system model from observations of the original system
- Learned model is a finite probabilistic automaton
- Mao et al., 2011: LTL properties of learned system has the same properties as the original
- Only Passive learning → no interaction with the system!

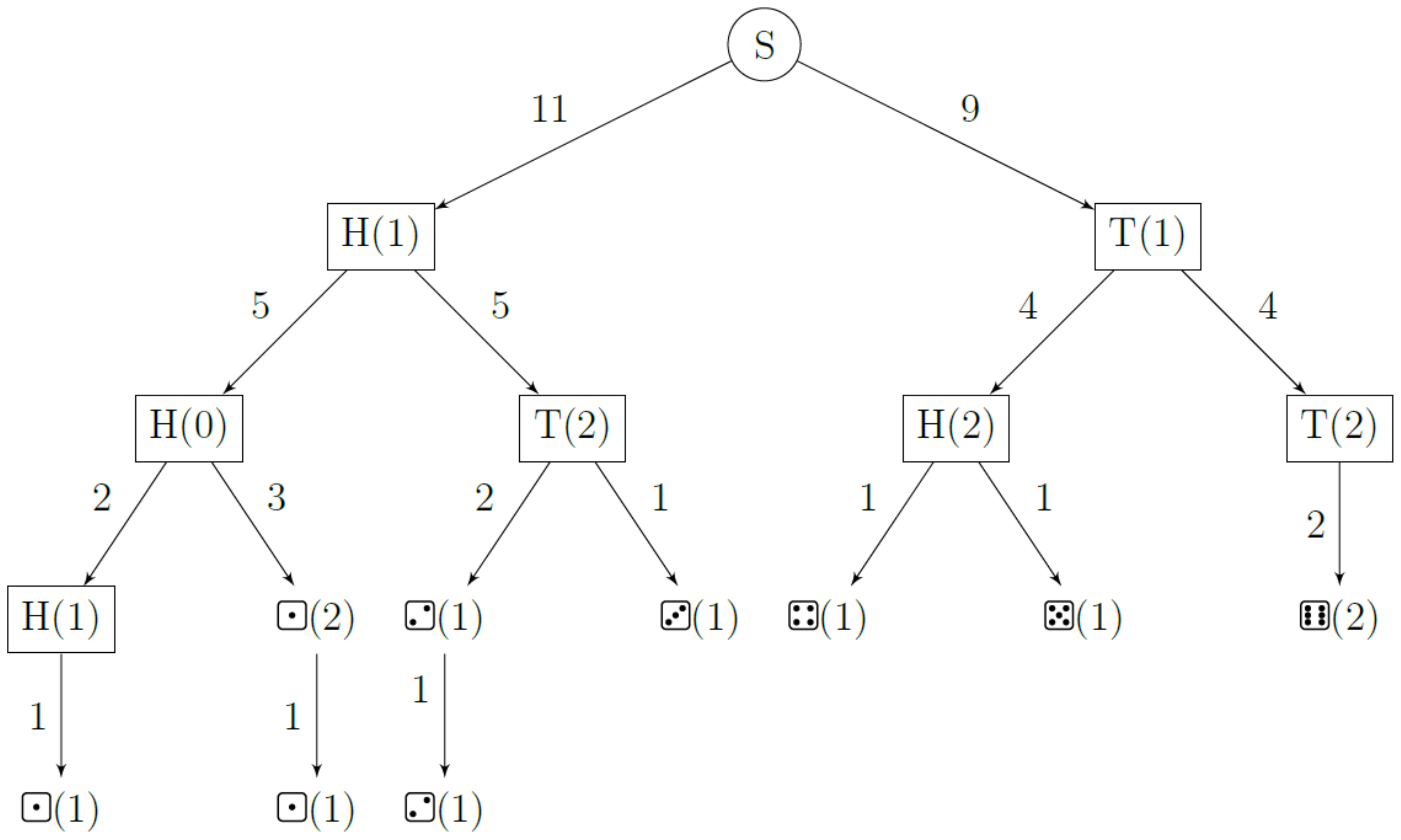
AALERGIA - Components



AALERGIA – FTPA (Dice Model) (1)



AALERGIA – FTPA (Dice Model) (2)



AALERGIA – Golden Section Search

- used to find an extremum by narrowing the range of values inside which the extremum is known to exist
- Before the Golden section search, a function searches for the left and right boarder of the extremum
- We use the Bayesian information criterion (BIC) score to rate the ε we get
- Once we have a range, the golden section search begins
- The algorithm suggets an ε with a high BIC score, in order to use it for AALERGIA

AALERGIA - Merging

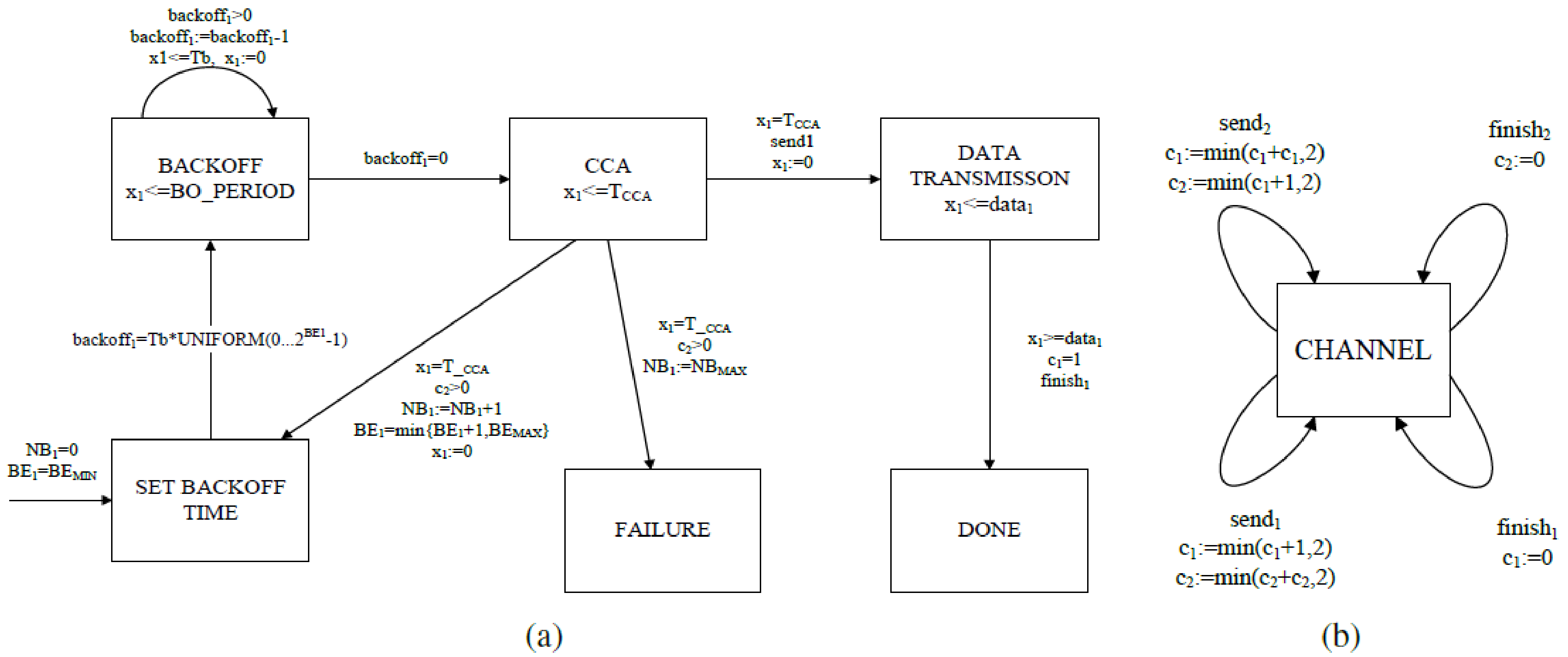
- RED is a set of states thats already been revised
- BLUE is a set of states that we need to look at
- Initial state is assigned to RED and successors are assigned to BLUE set
- We loop through the BLUE states and check if we can merge any of the elements
  - Merging is possible, if our compatibility criterion is satisfied
  - The ε-value we calculated with the golden section search is used to calculate the α-value for our compatibility criterion
  - If the merge happens, the size of the tree is reduced
  - If not, the state gets promoted to the RED set and is excluded from the BLUE set
- In the end, we get a merged DFFA

PRISM Model Checker

- Probabilistic model checker
- Formal modelling and analysis of random or probabilistic systems
- Provides an own, simple language: „PRISM language“
- Support for following probabilistic models:
  - DTMC, CTMC, MDP, PA, PTA,...
- LTL expressions can be used for model checking
- We use it to:
  - generate traces of the system we want to learn
  - model check the systems with LTL

Case study: IEEE 802.15.4 Model

- Source: Wu et al., 2014



Generating traces out of the system

- Generate traces with PRISM model
- New Python tool was needed in order to generate many traces and parse the output
- Alphabet is also created with the tool
- How are the states in the model represented in the output?

1	0000,1100,2100,3110,3110,3110,3210,3110,3110,3110,3110,3110,3110,3210,3110,3110,..
2	0000,1100,1100,1100,1100,1100,1200,1301,1301,2301,1301,2301,1301,2301,1301,2301,...
3	0000,1100,1100,1200,1301,1301,2301,1301,2301,1301,1301,1301,1301,1301,1301,2301,...

Evaluation – Compare results

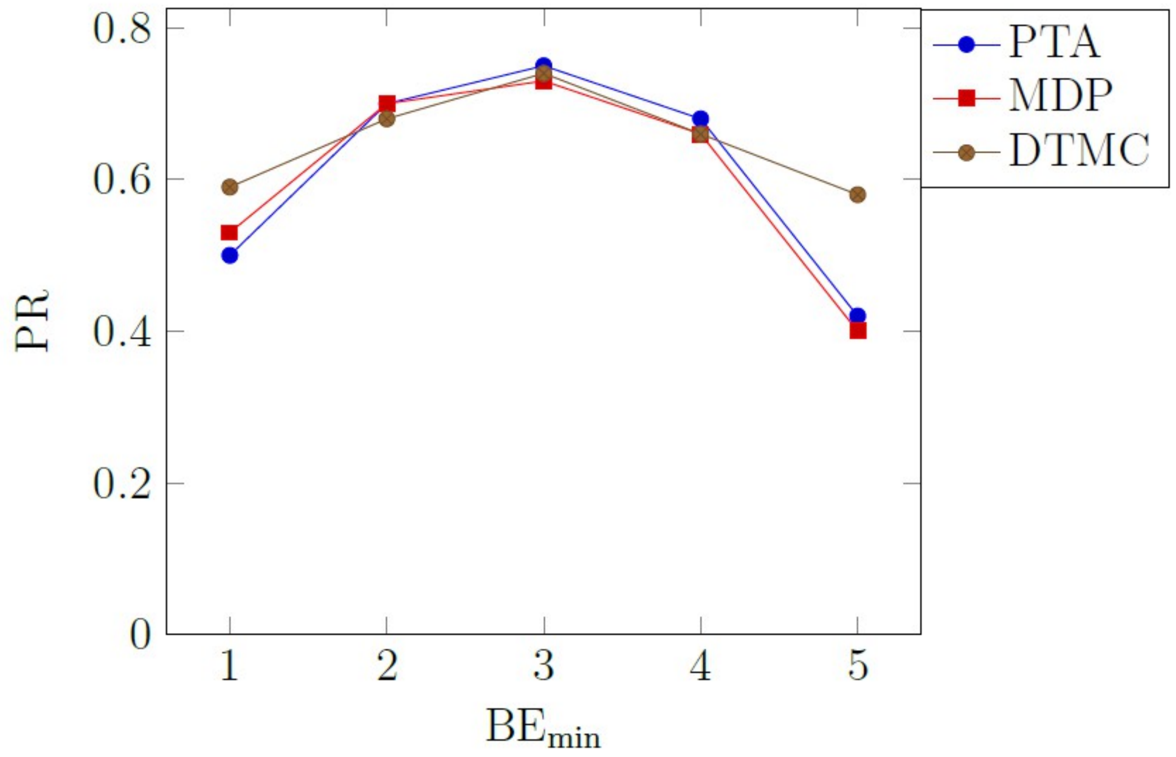
- Is the proposed network stable or not?
  - PR = The minimum probability of both stations successfully transmitting their packets within one sampling period
  - LTL formula:

$$\mathcal{P}_{\geq 1-p}\ [\mathrm{true}\ \mathcal{U}\ (\mathrm{done1} \wedge \mathrm{done2} \wedge \mathrm{z} \leq \mathrm{h})]$$

  - Sampling Period is set to 80Tb
  - Packet Length is set between 20-30Tb
- In order to test the system → change Parameters for BE_{min}, BE_{max} and Nb_{max}
- We used a count of 300 traces to generate the results

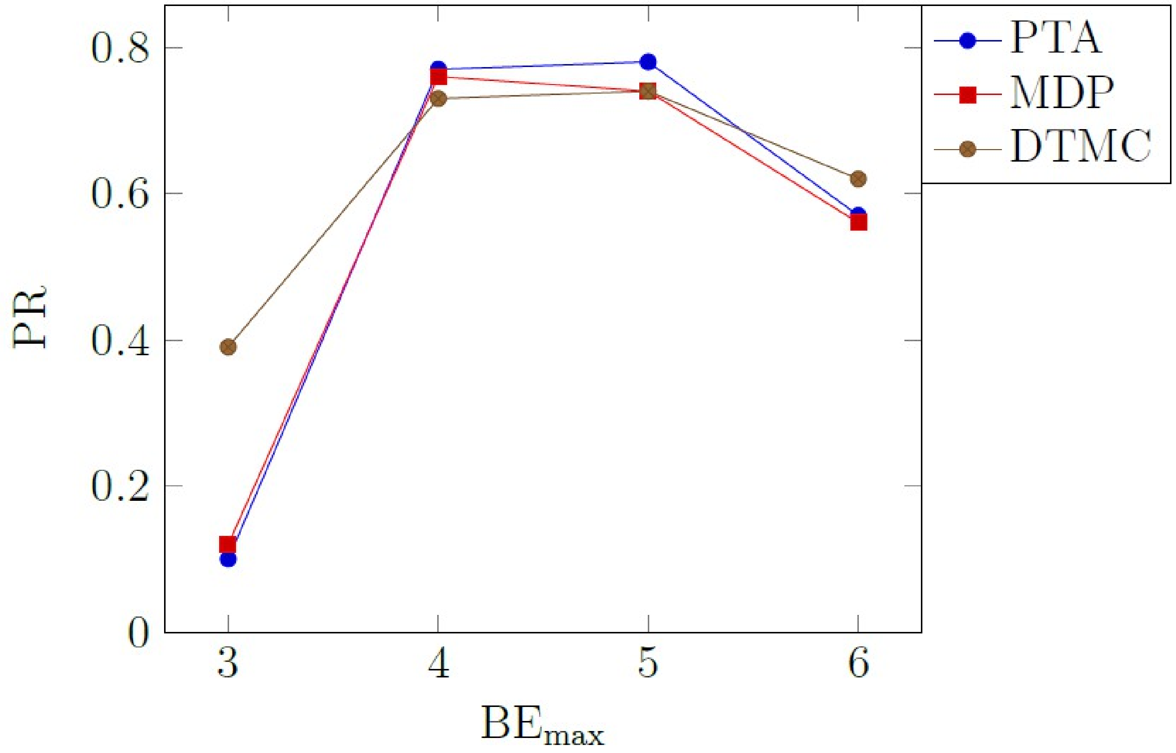
Evaluation – Compare results (1)

- $BE_{max} = 5$ and $NB_{max} = 4$
- BE_{min} is set between 1 and 5



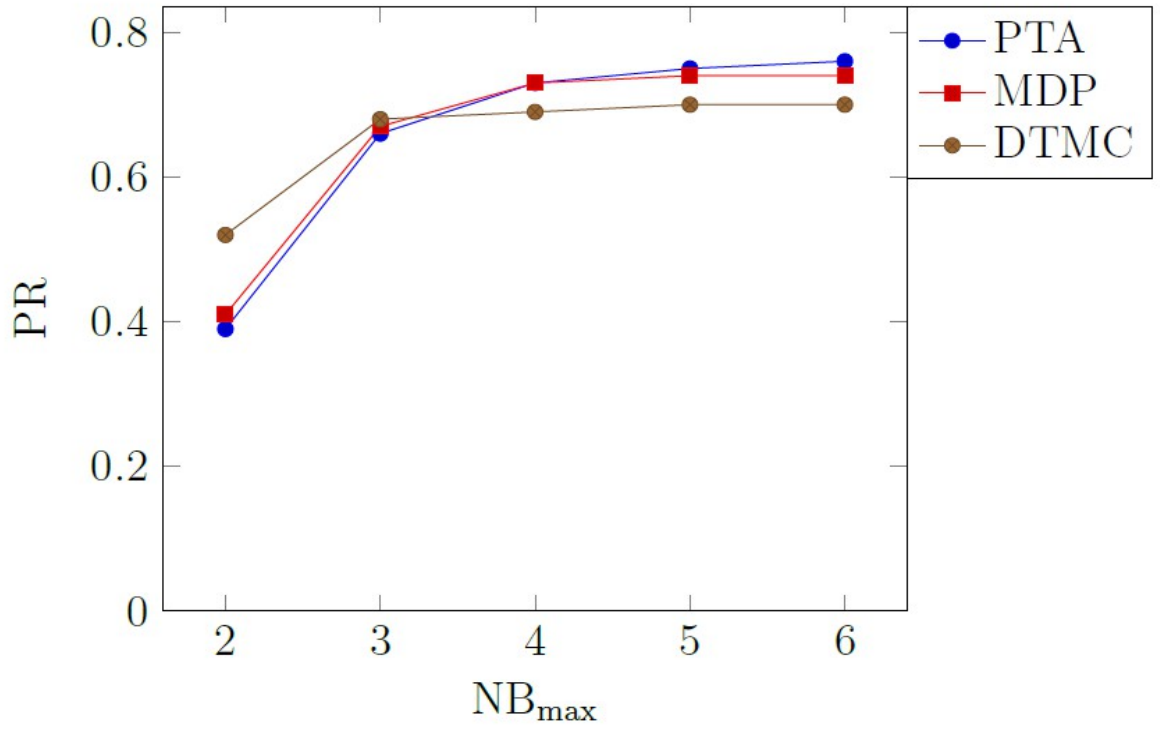
Evaluation – Compare results (2)

- BE_{min} = 3 and NB_{max} = 4
- BE_{max} is set between 3 and 6



Evaluation – Compare results (3)

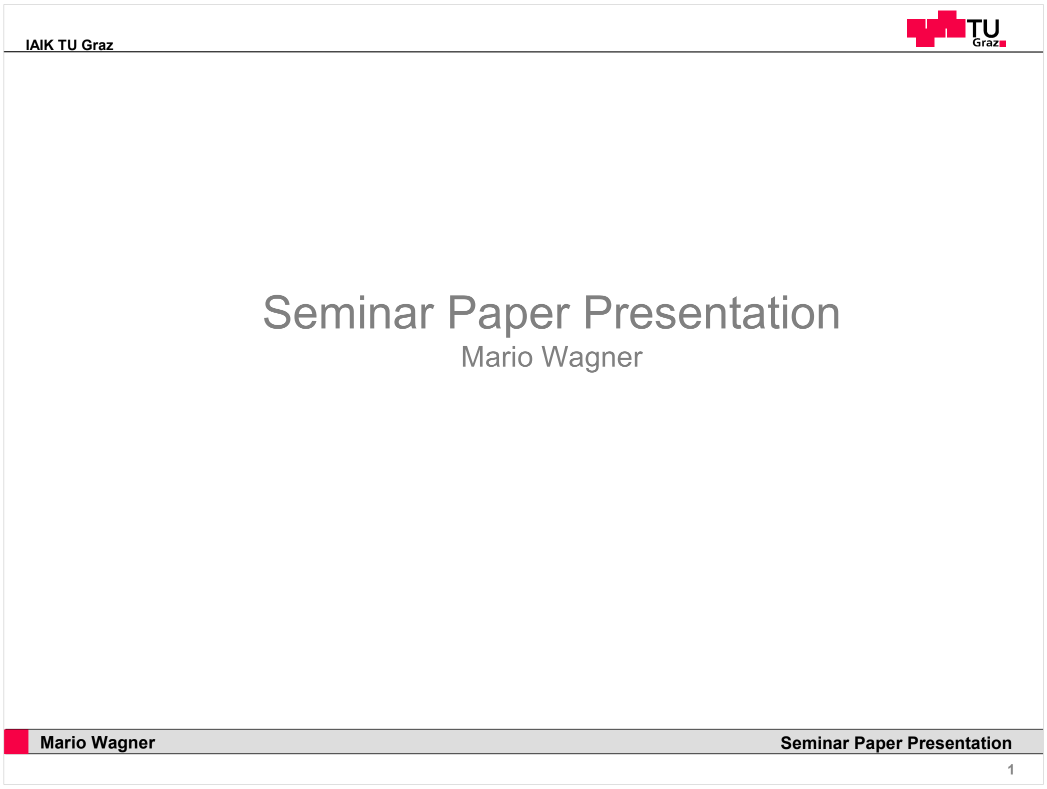
- $BE_{min} = 3$ and $BE_{max} = 5$
- NB_{max} is set between 2 and 6



Conclusion

- We managed to learn the original PTA model of the parallel composition of 2 senders and one channel
- When testing the LTL properties of the original system, results vary, some are very close to the original, some diverge from the original system
- → We do not know why this happens, therefore further research in the topic of sampling complexity is needed
- We also suffer from state-space explosion: our original models consist of <20 states and the learned models consists of 400-500 states

Thank you for your attention

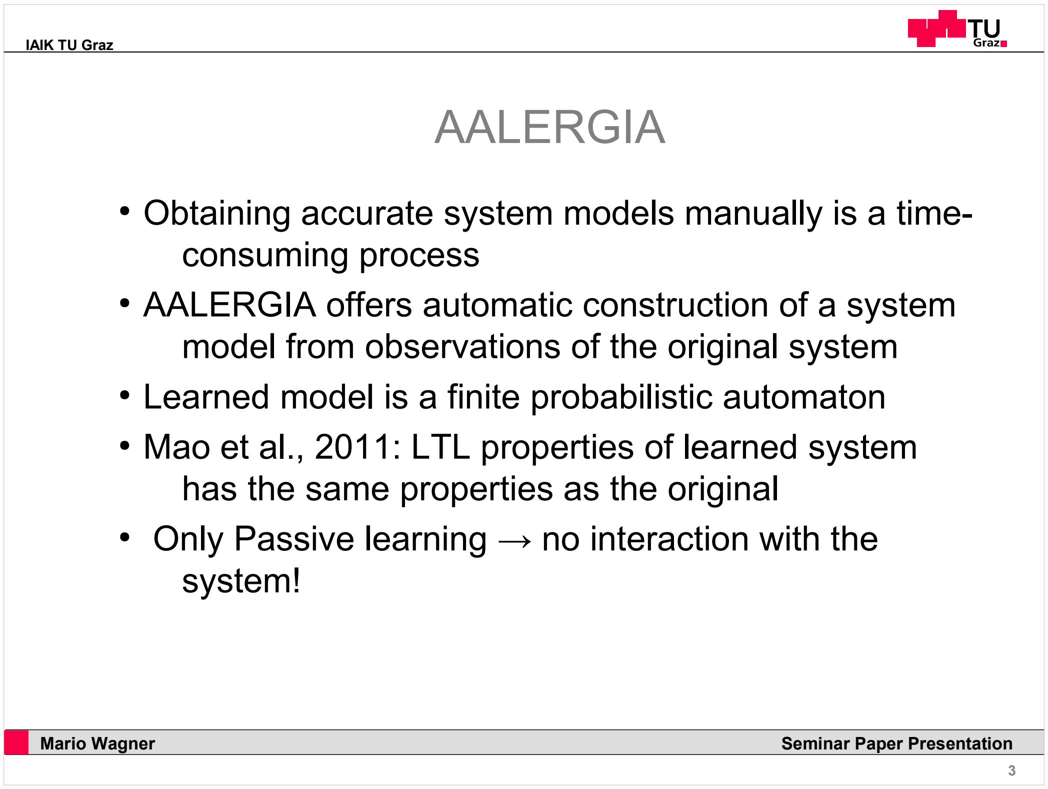


Seminar Paper Presentation

Mario Wagner

Problem/task

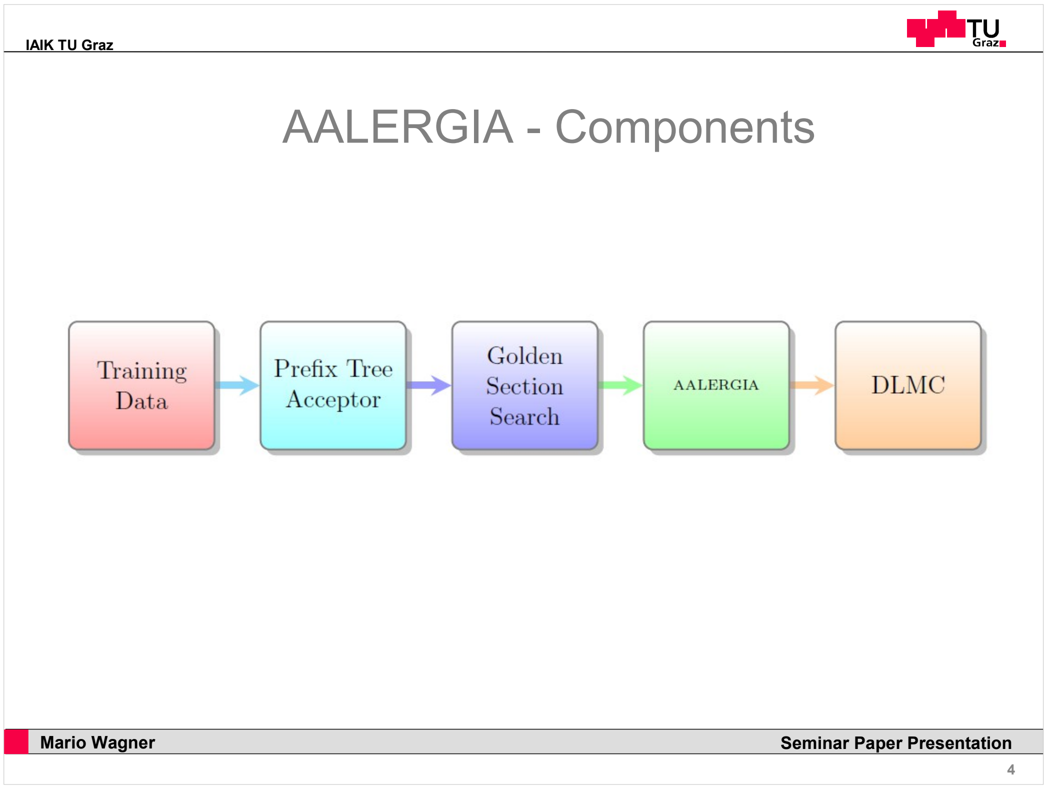
- Passive learning with AALERGIA
  - implement AALERGIA in python
- Is it possible to learn a probabilistic timed automaton with AALERGIA?
- Workflow
  - PTA → MDP→Traces→AALERGIA→DTMC→LTL model check



AALERGIA

- Obtaining accurate system models manually is a time-consuming process
- AALERGIA offers automatic construction of a system model from observations of the original system
- Learned model is a finite probabilistic automaton
- Mao et al., 2011: LTL properties of learned system has the same properties as the original
- Only Passive learning → no interaction with the system!

- Obtaining accurate system models for verification is a hard and time consuming process – this is a hindrance for the industry, because the techniques are otherwise very powerful and useful
  - useful for: documentation, simulation, model-checking, performance evaluation and prediction, test generation and so on.
- With AALERGIA, an accurate system model can be automatically constructed from observations of the (black box) system
- The learned final model is a finite probabilistic auntomata (a discrete time markov chain)
- probabilistic automata are prefered to deterministic automata, because in a realistic situation, they are more suitable to model an accurate model of a systems observable (and possibly non-deterministic) behavior
- As proven in the paper of Mao et al, in the large sample limit, the learned model of the original system has the same probabilities for linear temporal logic properties as the original system → it is possible to perform LTL model checking on the learned model and infer properties of the system
- We dont assume that the system is available for testing or interactive data generation (→ no active learning)
- better than statisctial model checking, as we abstract a real model and thus save us the cost of re-sampling the whole system and we are able to use the model for MDD tasks


IAIK TU Graz

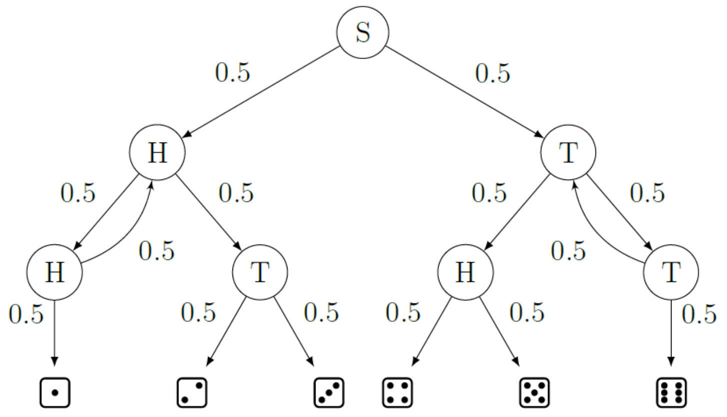
AALERGIA - Components

Mario Wagner Seminar Paper Presentation

4

- **Training data:**
  - supplied in 2 comma seperated value (CSV) file, one containing the alphabet, the other one containing the training data (the traces of the system) → the training data is provided by a PRISM model
- **Frequency Prefix Tree Acceptor**
  - After importing the data, a FTPA model is created
- **Golden Section Search**
  - used to find an alpha value that optimizes the BIC score of the learned model
- **AALERGIA**
  - the main part of the algorithm – Merges the states based on a compatibilitxy criterion
- DLMC
  - in the end we get a determinstic labeled markov chain (DLMC)

AALERGIA – FTPA (Dice Model) (1)

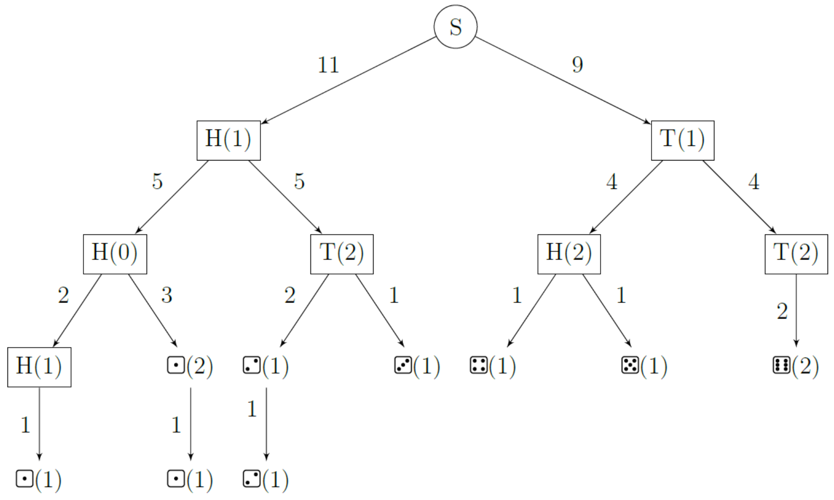


- Implemented in PRISM
- Toss a coin at each step, gives us a 50 percent chance of transitioning to the next states
- ends at the leafs of the tree, which is the dice value

->show on whiteboard what a typical path could look like S = {(SH,1), (SHHH,1), (SHH[1], 2)...} → first letters of the tuple show the path, 2nd number of the tuple shows the amount of times

IAIK TU Graz

AALERGIA – FTPA (Dice Model) (2)



Mario Wagner

Seminar Paper Presentation

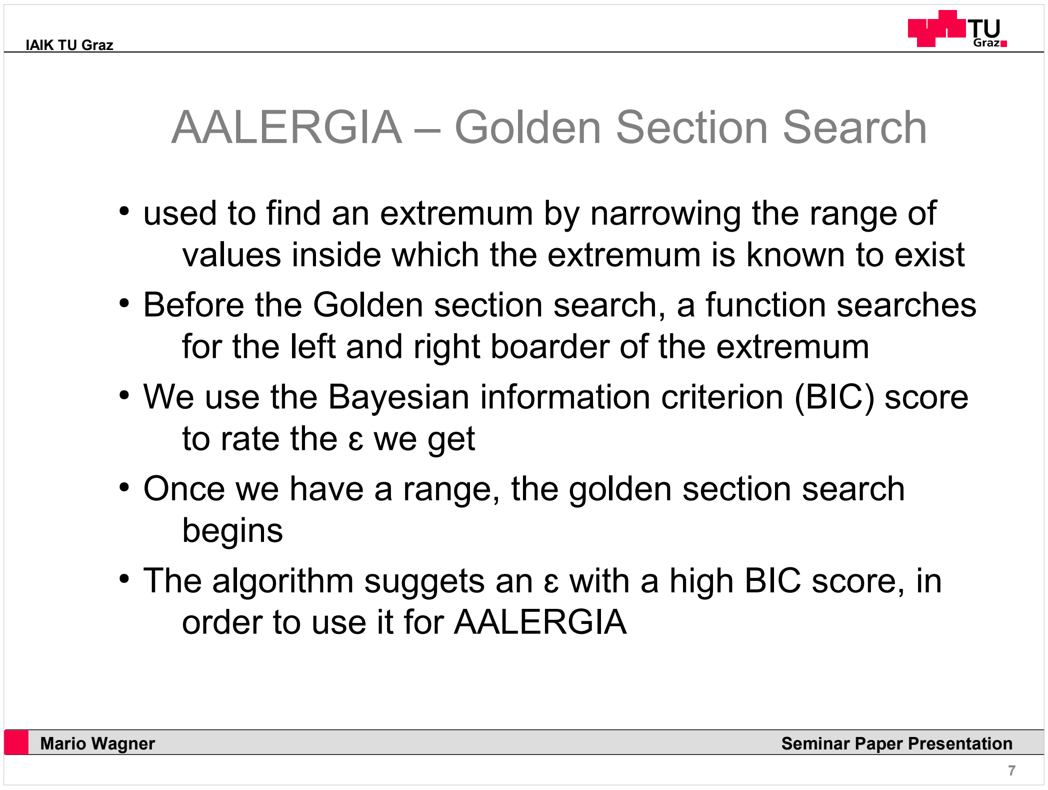
6

Tree Acceptor (FPTA) is created. A deterministic frequency finite automaton (DFFA) is a tuple $A = \langle \Sigma, Q, I_{\mathrm{fr}}, F_{\mathrm{fr}}, \delta_{\mathrm{fr}}, \delta \rangle$ where Σ = the finite alphabet, Q = the finite set of states, I_{fr} = the initial state frequencies, F_{fr} = the final state frequencies, δ_{fr} = the frequency transition function and δ = the transition function.

- show on whiteboard what a typical path could look like S = {(SH,1), (SHHH,1), (SHH[1], 2)...} → first letters of the tuple show the path, 2nd number of the tuple shows the amount of times
- All states combined show the amount of the total in the transition (e.g. for 11 → take all states below it and count them) → number of occurrences
- After this, a DFFA (deterministic frequency finite automaton) is created

The returned object is a *DFFA* class, representing the DFFA (line 30). It has the following members:

- a finite set of states
- the labels of the states
- the alphabet
- the initial state
- the initial state frequency
- the final state frequency
- the frequency transition matrix
- RED states: a finite set of states, which have already been determined to be in the final DLMC
- BLUE states: a finite set of states, which need to be tested for compatiblity


IAIK TU Graz

AALERGIA – Golden Section Search

- used to find an extremum by narrowing the range of values inside which the extremum is known to exist
- Before the Golden section search, a function searches for the left and right boarder of the extremum
- We use the Bayesian information criterion (BIC) score to rate the ε we get
- Once we have a range, the golden section search begins
- The algorithm suggets an ε with a high BIC score, in order to use it for AALERGIA

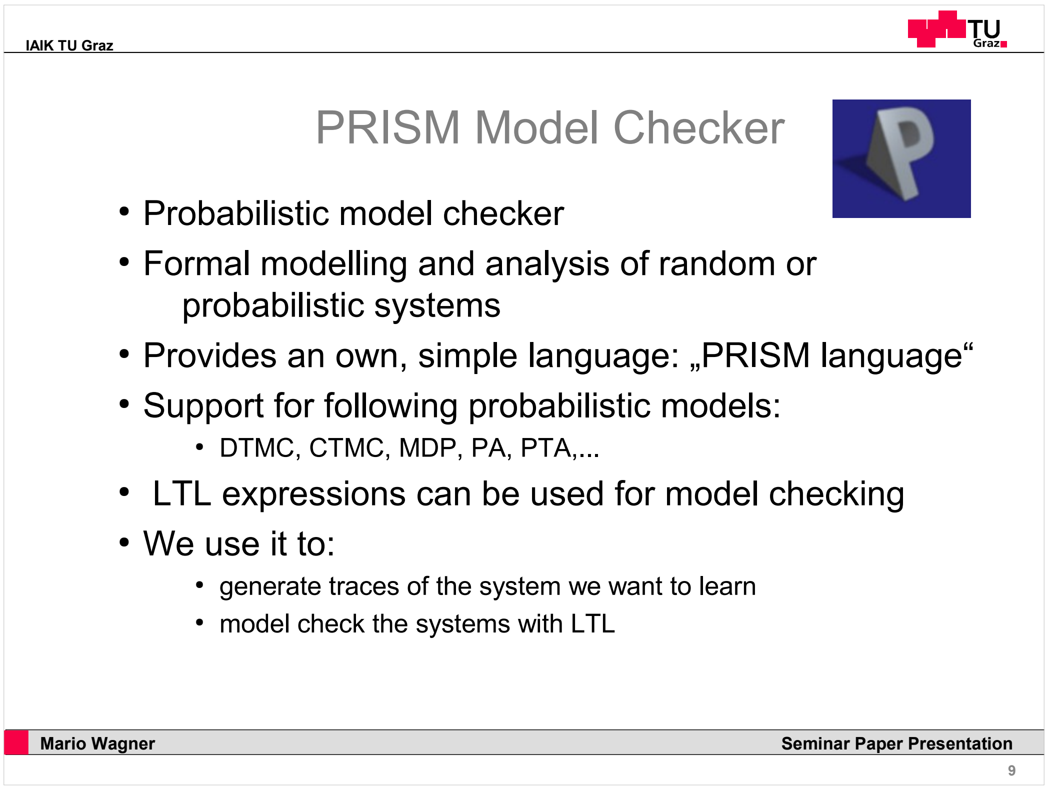
Mario Wagner | Seminar Paper Presentation

7

- First function sets a left and a right range for epsilon and calculates the BIC score → if score is higher, the range is changed and new epsilon values are calculated
- Once the range is found, the golden section search can begin
- algorithm tries to find the maxium between the 2 epsilon values
- if it exists, the section that contains the maxium is chosen as new region and the search starts again with a maximum recursion dept of 3
- in the end, a suggestion for a good epsilpon is proposed by the algorithm, it is used as input parameter for AALERGIA

AALERGIA - Merging

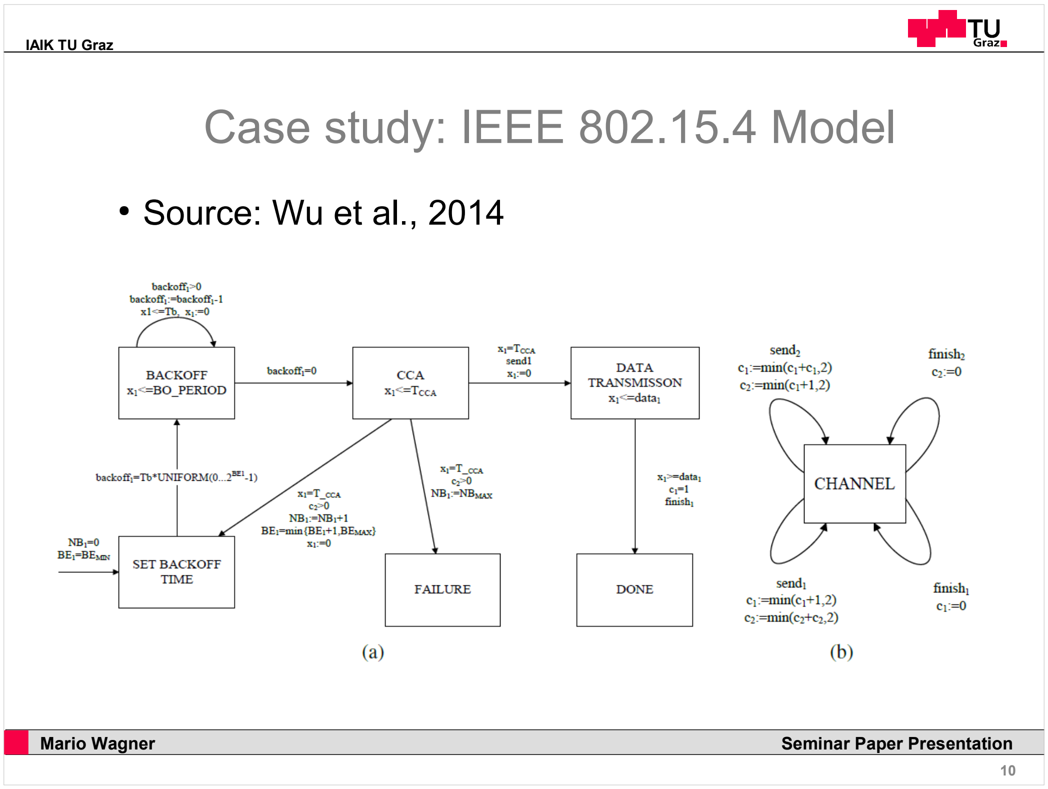
- RED is a set of states thats already been revised
- BLUE is a set of states that we need to look at
- Initial state is assigned to RED and successors are assigned to BLUE set
- We loop through the BLUE states and check if we can merge any of the elements
  - Merging is possible, if our compatibility criterion is satisfied
  - The ε-value we calculated with the golden section search is used to calculate the α-value for our compatibility criterion
  - If the merge happens, the size of the tree is reduced
  - If not, the state gets promoted to the RED set and is excluded from the BLUE set
- In the end, we get a merged DFFA



PRISM Model Checker

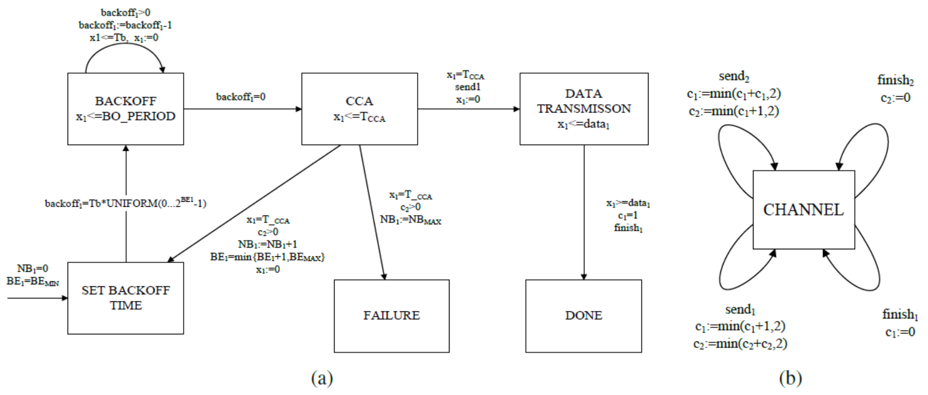
- Probabilistic model checker
- Formal modelling and analysis of random or probabilistic systems
- Provides an own, simple language: „PRISM language“
- Support for following probabilistic models:
  - DTMC, CTMC, MDP, PA, PTA,...
- LTL expressions can be used for model checking
- We use it to:
  - generate traces of the system we want to learn
  - model check the systems with LTL

- PRISM is a probabilistic model checker
- it is a tool for formal modelling and analysis of system that exhibit random or probabilistic behaviour
- following probabilistic models are supported:
  - discrete-time Markov chains (DTMCs)
  - continuous-time Markov chains (CTMCs)
  - Markov decision processes (MDPs)
  - probabilistic automata (PAs)
  - probabilsitic timed automoata (PTAs)
- Models are described via a PRISM language, a simple, state based language
- the property specification language incorporates the temporal logics
  - PCTL (probabilistic computation tree logic)
  - LTL (linear temporal logic)
- free and open source, licences under the GNU General Public License (GPL)
- we use it for generating traces/trainings data and to model check the learned system

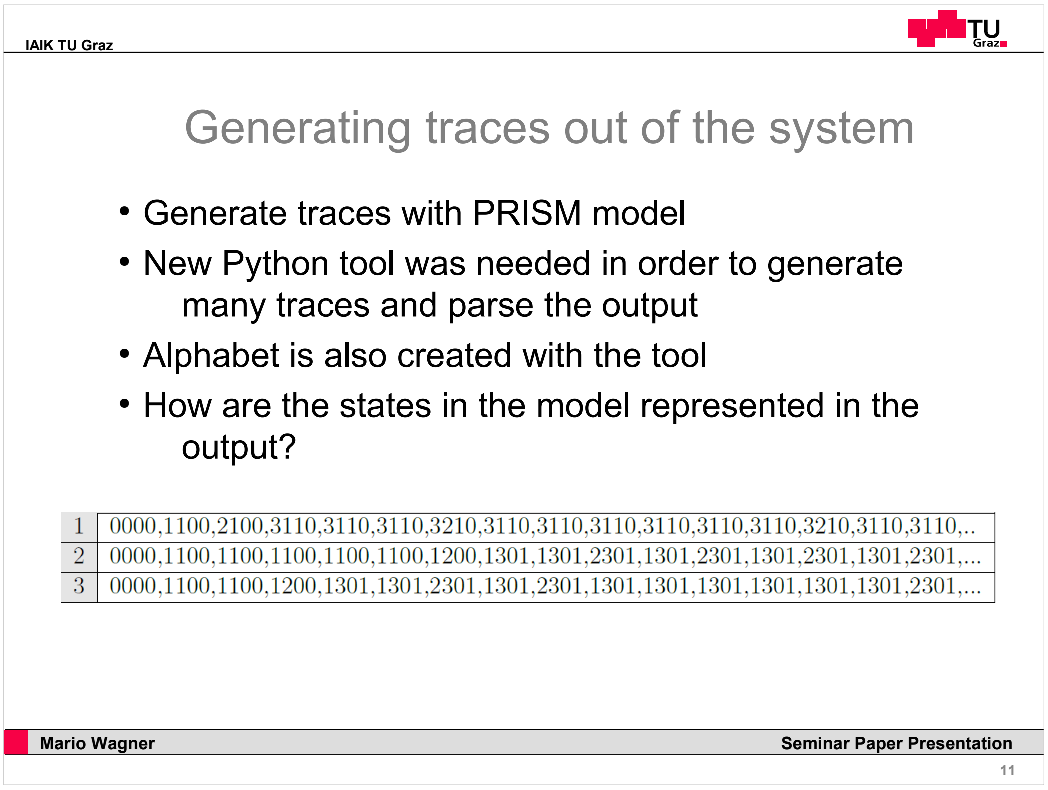


Case study: IEEE 802.15.4 Model

- Source: Wu et al., 2014



- Simple model of the IEEE 802.15.4 unslotted protocol
- taken from WU et al, 2014 → they introduced it as a PTA and we had to model it as MDP → not possible to generate traces out of a PTA at the moment in PRISM
- We used the digital clocks approach to convert the PTA to an MDP
- In this model, we have 2 senders and 1 channel in parallel composition
- Sending modules start their sending process at the same time
- The protocol used is the unslotted CSMA/CA protocol
- protocol uses a mechanism called exponential backoff, where a sender listens to the channel before attempting to send
- We work with discrete time, the smallest time unit in our model is T_b
- Start with initialising variables NB and BE => NB is number of times a backoff happened and is bounded by an upper bound (Bemax)
- BE is the backoff exponent and is related to the random amount of time the sender waits before it attempts to listen to the channel again, BE = Bemin and bounded by Bemax
- State CCA to listen to the channel, takes TB time
- explain further transitions...
- explain channel



Generating traces out of the system

- Generate traces with PRISM model
- New Python tool was needed in order to generate many traces and parse the output
- Alphabet is also created with the tool
- How are the states in the model represented in the output?

1	0000,1100,2100,3110,3110,3110,3210,3110,3110,3110,3110,3110,3110,3210,3110,3110,..
2	0000,1100,1100,1100,1100,1100,1200,1301,1301,2301,1301,2301,1301,2301,1301,2301,...
3	0000,1100,1100,1200,1301,1301,2301,1301,2301,1301,1301,1301,1301,1301,1301,2301,...

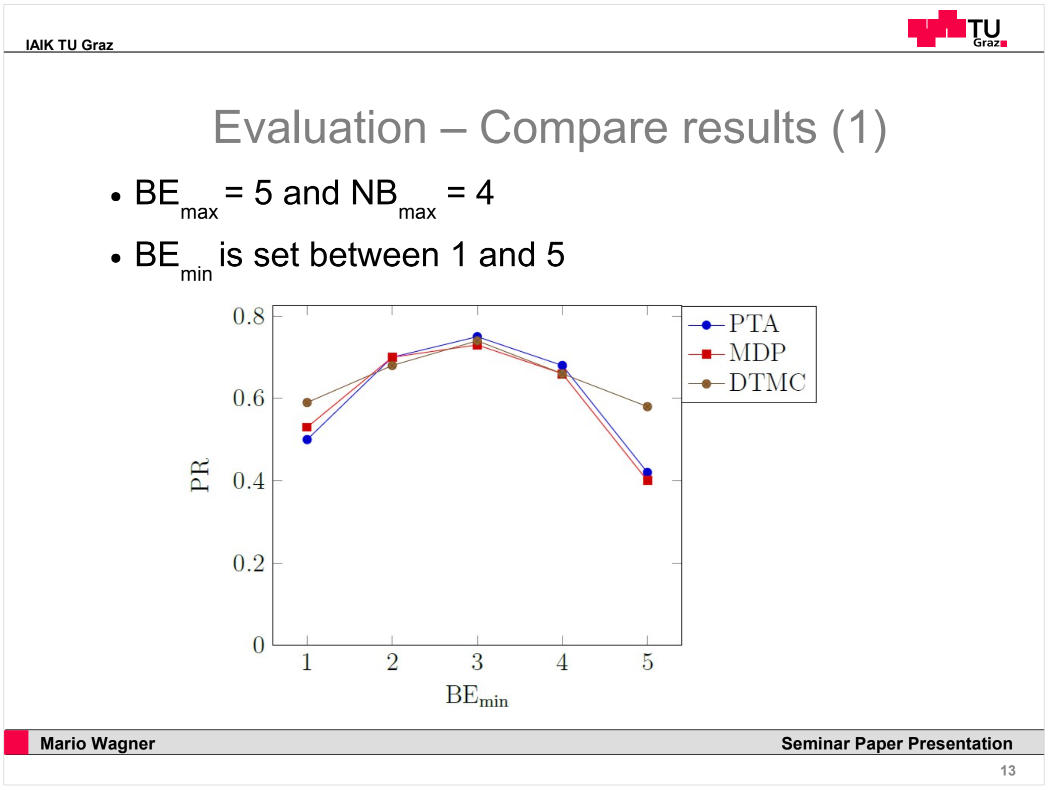
- traces are generated with PRISM
- A small python tool nhad to be implemented in order to generate the traces => with this, it was possible to generate many traces and it parsed the output to the desired CSV file
- explain what the numbers in the traces file mean and how they represent the state of the system

Evaluation – Compare results

- Is the proposed network stable or not?
  - PR = The minimum probability of both stations successfully transmitting their packets within one sampling period
  - LTL formula:

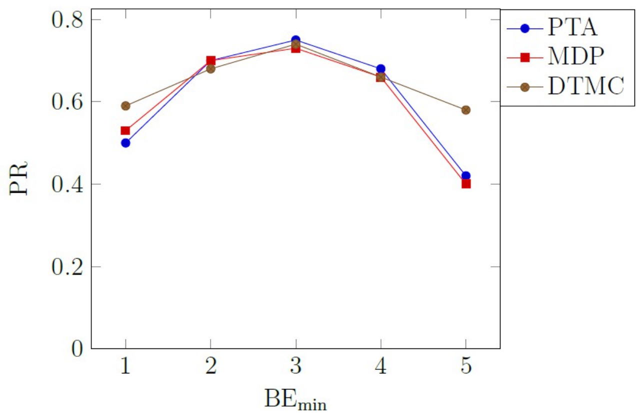
$$\mathcal{P}_{\geq 1-p}\ [\mathrm{true}\ \mathcal{U}\ (\mathrm{done1} \wedge \mathrm{done2} \wedge z \leq h)]$$

  - Sampling Period is set to 80Tb
  - Packet Length is set between 20-30Tb
- In order to test the system → change Parameters for BE_{min}, BE_{max} and Nb_{max}
- We used a count of 300 traces to generate the results

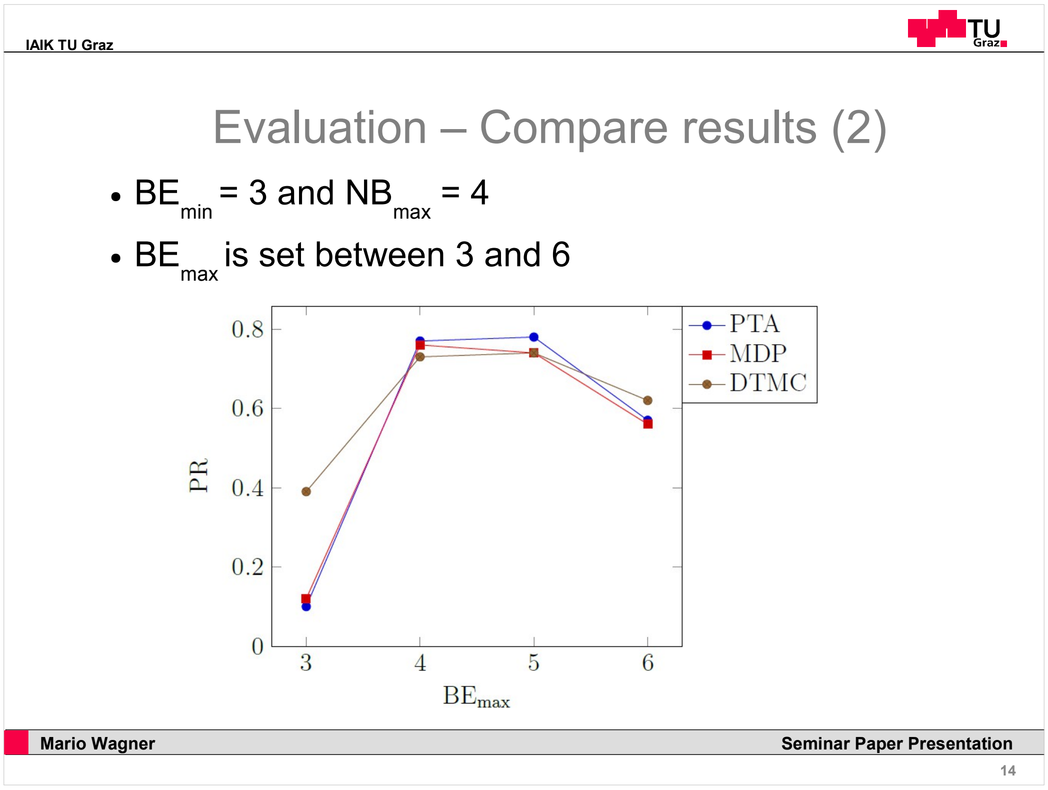


Evaluation – Compare results (1)

- BE_{max} = 5 and NB_{max} = 4
- BE_{min} is set between 1 and 5

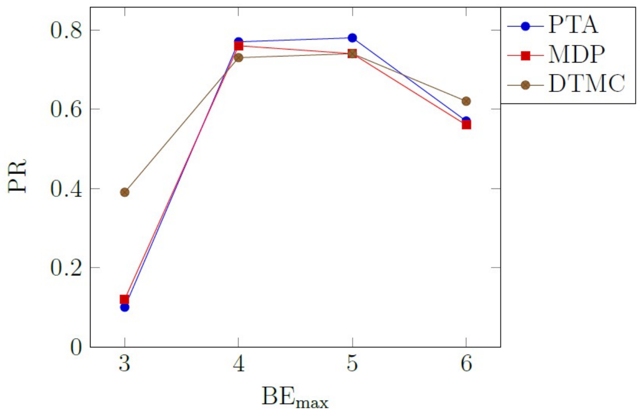


- Maximum PR is achieved when BE_min = 3,
- If Be_min is small, the backoff time will be short and it is likely that the channel is still busy when trying to send again, so NB_max is reached fast
- If BE_min is too lage, it is likely that the process cannot finish in the given timeframe

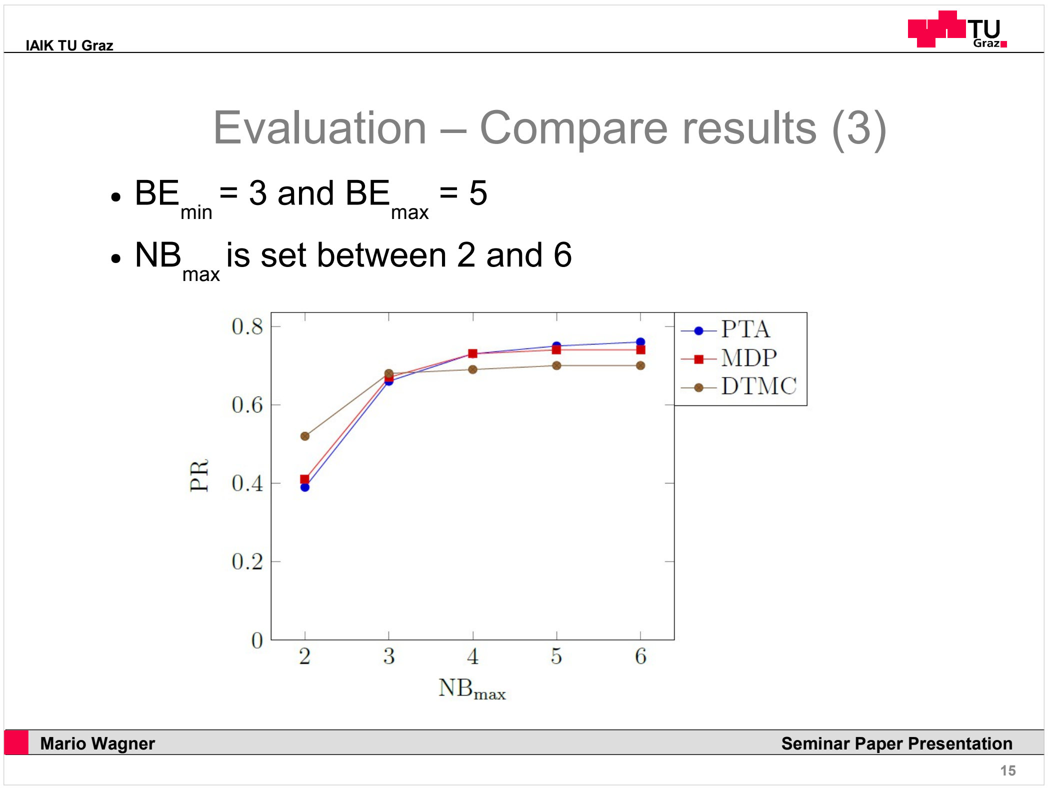


Evaluation – Compare results (2)

- BE_{min} = 3 and NB_{max} = 4
- BE_{max} is set between 3 and 6

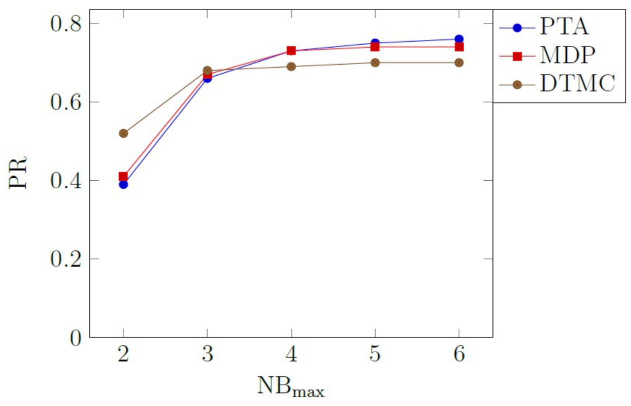


- Maximum PR is achieved with BE_max = 4 and 5
- similar situation to the picture before
- If BE_max is small, the backoff time is too short and there are too many tries → the channel is not finished sending and NB_max is reached too soon
- If BE_max is too big, it is likely that the sender waits too long and the channel cannot finish in the given timeframe



Evaluation – Compare results (3)

- BE_{min} = 3 and BE_{max} = 5
- NB_{max} is set between 2 and 6



- When NB_max is small, the maximum number of retries is reached very fast and the process fails
- IF NB_max is larger, the process levels off quickly, because the timit limit stays this limits the sending process

Conclusion

- We managed to learn the original PTA model of the parallel composition of 2 senders and one channel
- When testing the LTL properties of the original system, results vary, some are very close to the original, some diverge from the original system
- → We do not know why this happens, therefore further research in the topic of sampling complexity is needed
- We also suffer from state-space explosion: our original models consist of <20 states and the learned models consists of 400-500 states

Thank you for your attention