Performance Analysis based on Finite Stochastic State Machines using Generative Radio Channel Models

Martin Ostermann, Herbert Steffan

Institute Communication Networks Aachen University of Technology
e-mail: ost@comnets.rwth-aachen.de
WWW: http://www.comnets.rwth-aachen.de/~ost

$Abstract$ — **Recent research results use finite stochastic state machines (FSSM) as a model to describe the error characteristics of radio channels. Certain classes of protocols on the sender and receiver side of a broadcasting system can be described as finite state machines (FSM) or FSSM as well. A complete scenario will yield a common finite stochastic system (CFSS).**

Common finite stochastic systems describe an interconnected aggregate of FSSM which communicate with each other. For such a system, the static state probabilities can be calculated by solving a huge system of linear equations. The generation of such a system of linear equations is known as the state space generation problem. The usual approach of state space generation needs to be optimized in order to minimize time and space requirements. This is especially true for CFSS of extended finite stochastic state machines.

Merging a CFSS into one common finite stochastic state machine in advance to the state space generation is one way to save time and space and to retain important ordering information useful for applying optimized sparse matrix algorithms on the resulting linear equations.

Besides an introduction into the algorithms involved, this paper discusses how these could be applied to the examination of an ARQ type protocol. A simple example is given as well.

I. Introduction

A. Objectives

In mobile radio communication environments, the quality of transmission is significantly influenced by fading of the signal envelope. Traditionally, burst error behaviour of radio channels is modeled by stochastic processes.

For mathematical performance analysis, there have been only few known methods describing the error behaviour of the radio channel. Stochastic simulation and mathematical analysis of communication links need better models of symbol error, which have been developed in previous works [1, 2, 3].

As we now have more appropriate models, with a structure that fits mathematical analysis as well as stochastical simulation, we try to extend the range of applications suitable for analysis.

B. Investigating Common Finite Stochastic Systems.

These models are useful for mathematical analysis based on stochastic finite state machines. In addition to the radio channel, the communication protocols of the radio stations can be modeled as (stochastic) finite state machines. It is now feasible to build a common finite stochastic system involving the channel and the two communicating stations. This common system can be mathematically analysed. In order to handle complex protocols, which may lead to systems with several thousands of separate states, it is necessary to develop rules suitable for machine processing and to explore all measures to reduce the number of states involved, e.g. an effective way of handling extended finite state machines. This is a new approach in contrast to algorithms that are based on matrix computations, which do not attempt to reduce the number of transitions involved [4, 5, 6].

II. Radio Channel Models

Channel state models are used to investigate error control algorithms. Each state is related to a certain symbol error. Calculating the transition probabilities of the state model requires some effort, but the main problem originates from the fact, that probabilities must be calculated again, when changing the scenario, e.g. the velocity of a mobile station. The fading processes can be approximated by state models. Using a Markov chain requires less numerical effort than using digital filters. Fig. 1 shows the respective principle [3].

A. Markov Models

Let $S = \{0, 1, 2, \ldots, K\}, K \in \mathbb{N}$ define a finite set of states and $\{S_n\}, n \in \mathbb{N}_0$ be a homogeneous Markov process. The transition probabilities $p_{i,j} = \Pr\{S_{n+1} = j | S_n = i\}$ are independent of index n. They can be determined by observing the original fading process $\{A(n)\}$. Let $A = \{A_1, A_2, A_3 \cdots, A_K\}$ define a set of thresholds with $A_i < A_{i+1}$. Thus, $\{S_n\}$ is said to be in state i at time instant k if $A_i < A(k) < A_{i+1}$ for $n \in \mathbb{N}_0$, $i = 1..(K-1)$. As a first approximation consecutive values are assumed to be neighbouring states. That means $p_{i,j} = 0$ for all $|i - j| > 1$. For exact derivations see [1] where level crossing rates are used for calculation. A better model can be build by deriving *all* transition probabilities.

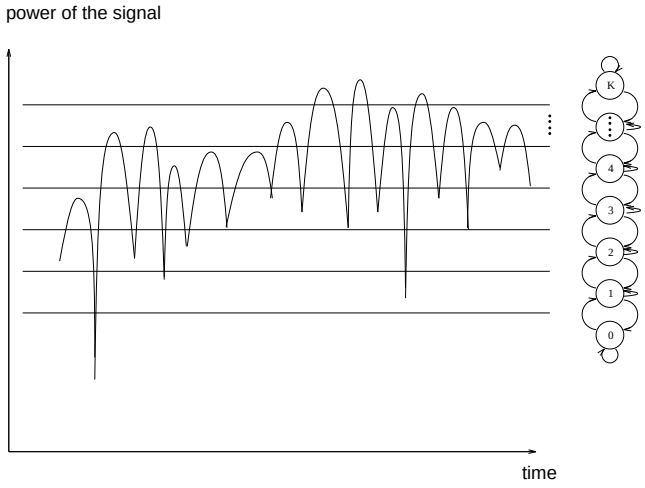

Fig. 1: Modeling Principle

The maximum likelihood estimator for the transition probabilities can be derived by observing the transitions of the original process. If $n_{i,j}$ denotes the number of observed transitions from state i to state j and n_i denotes the total number of transitions leaving state i, then the estimator is $\hat{p}_{i,j} = \frac{n_{i,j}}{n_i} \quad i,j = 1..K$.

III. Merging FSSMs

A. Definitions

Because we will give a rather informal description of the algorithm, only few formal definitions will be useful.

Definition 1 *A*$(\mathrm{X}, \mathrm{Y}, \mathrm{S}, \delta_d)$ *is called* ***(deterministic) state machine****, if*

- X, Y *and* S *are non-empty sets, and*
- δ_d *is a unique transformation of the product set* $\mathrm{X} \times \mathrm{S}$ *into the product set* $\mathrm{S} \times \mathrm{Y}$.

X *is called the* ***condition set****, and* Y *the* ***output set****.*

Definition 2 *An state machine is called* ***finite state machine****, if* X, Y *and* S *are finite sets.*

Definition 3 ***Stochastic State Machines.***

Since definitions for stochastic state machines is complex without being helpful for understanding, we will give an informal description only:

- it is similar to the deterministic finite state machine,
- now the transitions are only probable with $p \in (0 \ldots 1]$, and
- the sum of the probabilities of all transitions exiting one state for one condition is

$$\sum_{i=1}^{n} p_i = 1.$$

Definition 4 $\tau_s(s, x, s', y, p)$*, denoted as* $s \rightarrow s' : x/y; p$*, describes the* ***transition of a stochastic state machine*** τ_s*, if* $\tau_s \in \delta_s$.

Definition 5 Internal Inputs and Outputs *are those, which occur between the two machines to be merged. The data exchange can be classified by three types of communication:*

- **active internal communication**,
  if the transition needs an external (no internal) input, but an internal output,
- **passive internal communication**,
  if the transition contains an internal input; The type of the output does not matter, or
- **no internal communication (external communication)**,
  if the transition neither involves internal input, nor internal output.

Fig. 2 gives an example with five states and five transitions. The symbols a and d are external, while b and c are internal symbols. The transitions of the 'active internal communication' type are depicted with thick lines, the ones belonging to the 'passive internal communication' type use normal strength, and the external[1] one features dashed lines.

B. Algorithm

This section will give an informal description of the algorithm.

The merging is based on transitions, and involves two separate, subsequent steps. While building the common FSSM C out of two FSSMs A_1 and A_2, two phenomena have to be handled: **Concurrency** of transitions and **Chaining of transitions** (chained transitions).

We speak of **concurrency**, if

- A_1 contains $s_1 \rightarrow s_1' : x_1/y_1; p_1$
- A_2 contains $s_2 \rightarrow s_2' : x_2/y_2; p_2$
- $\Rightarrow C$ contains $s_1 * s_2 \rightarrow s_1' * s_2' : x/y; p$
  with $y = y_1 * y_2$ and $p = p_1 \cdot p_2$, if $x_1 = x_2 = x$

[1] Please note, that $s \rightarrow s' :; p$ is an abbreviation of $s \rightarrow s' : [clock]; p$, with $[clock]$ being an implicit external symbol.

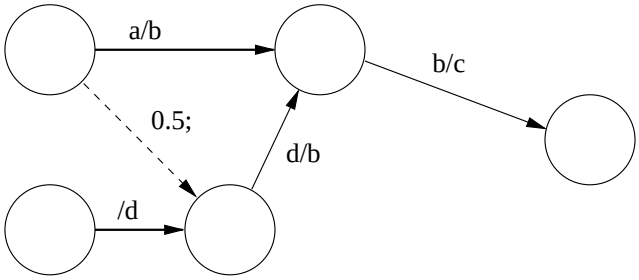

Fig. 2: Elimination of Transition Chains: Example Setting

for any combination of transitions contained in A_1 and A_2.

After that has been done, all remaining transitions with internal inputs are copied to the new FSSM C.[2] Please note, that this means that for each transition $\tau_1 \in \delta_1$ and every state $s_2 \in S_2$, there will be a transition τ_c.[3]

A **chain of transitions** is observed, if

- a transition τ_1 outputs an internal symbol x_1,
- which fits the condition y_2 of another transition τ_2, and
- the final state s_1' of the first transition matches the initial state s_2 of the second transition.

We can now substitute the first transition by a new one, which

- contains the condition x_1 of the first transition, and
- contains the outputs of both transitions y_1 and y_2, and
- transforms the initial state s_1 to the final state s_2'
- and is executed with the probability equal to the product $p = p_1 \cdot p_2$ of the original probabilities.

In our iteration process, we only substitute transition chains which contain 'active internal communication' type transitions. We continue this substitution process until there are no transitions of the 'active internal communication' type left. If this is done, we can also discard the transitions of the 'passive internal communication' type, as they are no longer triggered.

As you may have noticed, the result of this substitution process may no longer fit our definition of the state machine given in Definition 3, since only one input and output symbol was allowed. We can deal with this by introducing the combined symbol $y_1 * y_2$, and in addition we need to adjust the other state machines to make them react on the combined symbol instead of the original ones. However, if one of the original state machines contains a pair of transitions from the same state, each reacting on one of the two symbols y_1 and y_2, the result is no longer a stochastic state machine, but instead an indeterministic state machine, since we are not able to determine the probability for which of the two transitions will be taken.

But in this case, the indeterminism was a property of the original CFSS as well. The tool is not able to resolve this indeterminism but ought emit an error message instead.

C. Elimination Example

Let us reconsider the example given above (Fig. 2). During the first iteration, we are able to substitute two instances of chained transitions involving 'active internal communication' type transitions. (Fig. 3). In one case we get a transition of the 'external communication' type, the other case yields a result involving the 'active internal communication' type again. Thus we do another iteration (see Fig. 4).

[2]This step can be optimized by combining it with the first iteration of the elimination process that follows.

[3]The same is true for τ_2 and s_1.

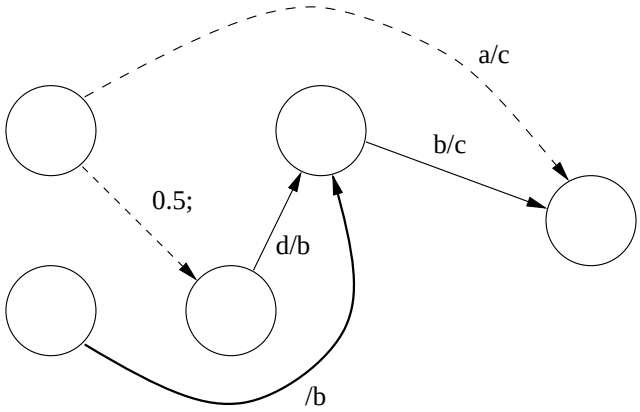

Fig. 3: Elimination of Transition Chains: Initial Step

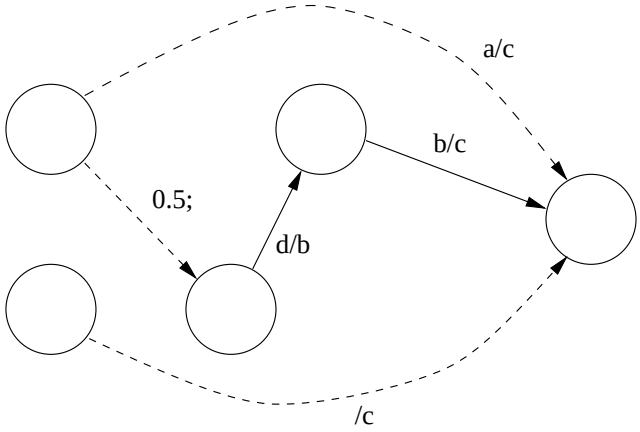

Fig. 4: Elimination of Transition Chains: 2nd Iteration

After this second step, there are no longer transitions of the 'active internal communication' type left. Since the internal symbols b and c do no longer appear as output symbols, those transitions of the 'passive internal communication' type will never be taken, and we can eliminate them (Fig. 5).

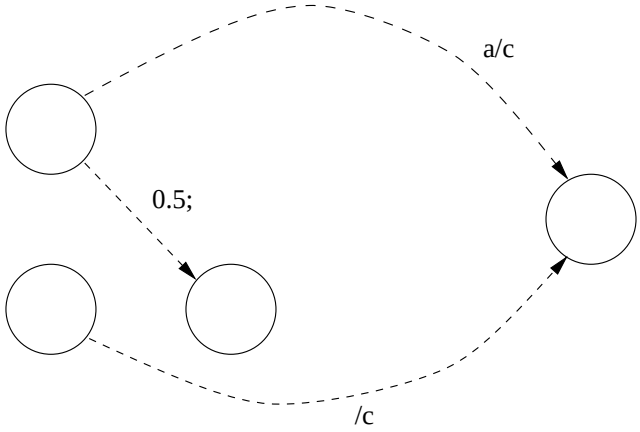

Fig. 5: Elimination of Transition Chains: Final Result

D. Steady State Probabilities

Once we have joined all FSSMs involved, we eliminated all input symbols except the implicit clock symbol. Thus, the final CFSSM is autonomous and can be described as a Markov chain. With the help of a standard Markovian analysis, we can calculate the steady state probabilities [3].

IV. Example

This section explains how to apply these algorithms to performance analysis problems. For the sake of simplicity, a simplified protocol description is used, to avoid the use of extended finite state machines. However, the same techniques could be applied to more complex protocols. A prototype tool has been developed to automate the merging process.

The general course of action is alway the same.

- Firstly, we need to obtain an adequate machine readable description of the protocol which is to be analysed in terms of a finite stochastic state machine. This is denoted as a set of transitions for each state machine involved.
- Using the tool, we can now combine two FSSMs at a time. Each combination process will yield another FSSM to be combined with the FSSMs left (see Fig. reftool). An iterative process involving all FSSM will yield the resulting CFSSM. The sequence of a combination process does not make a difference concerning the final result. However, it has an effect on the effort needed, and a carefully chosen sequence can greatly reduce the computational power needed. As a rule of thumb, it is only desirable to merge two FSSMs which actually communicate with each other (e.g. share input and output symbols). This is because unrelated state machines cannot be merged, but will only blow up the state space which is to be examined in later mergings.
- After all FSSM have been merged into on CFSSM, the steady state probabilities can be computed.
- The last step involves the evaluation of the results by the operator. Steady state probablities of a singe FSSM can be obtained by adding up the steady state probabilities of all states of the CFSSM which evolved from the FSSM in question.

To obtain a series of results, scripting tools can be applied.

Please note, that we have choosen this example for the sake of easy understanding, rather then to present realistic protocols and error ratings. Such are parameters, which could be easily adjusted.

A. Model

In order to examine 'Repeat Request'- and 'Selective Reject'-protocols (like HDLC[4]), it is possible to make certain simplifications[5] In order to reduce the total count of states, we need to select the information which is of importance to us.

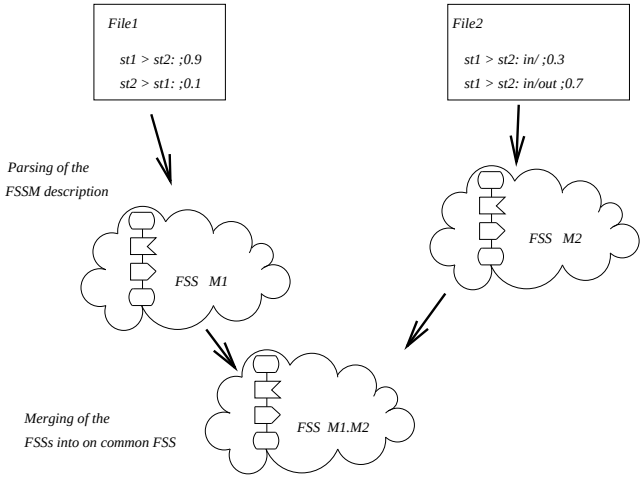

Fig. 6: Basic Merging Process

We are interested in the throughput of the protocols, whilst the data which is transmitted through the packets is not of interest. Thus, in the case of a 'Selective Reject' protocol it is possible to neglect which packets need retransmission and instead we only need to count how many have to be repeated. We can use the same model as we do for the 'Automatic Repeat Request' type. The type of the protocol then is just influenced by the probabilities assigned to the count of the packets to be retransmitted.

We examine an ARQ system which contains the following components:

- a *transmit buffer*, which holds the packets until the are enough to transmit[6],
- a *repeater*, which transfers the packet that need retransmission back to the transmit buffer,
- the *packet source*, which as well holds a buffer,
- a *timing device*, which controls the reloading of the transmission buffer, thus useful to adjust the load, and
- the *channel*, which has been joined with the *receiver*. Together they determine the count of packets which have to be repeated on a stochastical base.

The window size has been set to four as well. Please, refer to Fig. 7 and Fig. 8 for the state machine definitions.

B. Results

The following results have been obtained for a channel with just one state and a packet error rate of 5%, using a simple 'Reject' protocol. The actual screen output of the tool is shown. The output of the raw steady state probablities

[4] Highlevel Data Link Control

[5] It is desirable to let the tools do the simplifications on itself. But this needs further research.

[6] We assume an interleaving of four packets at a time on the channel.

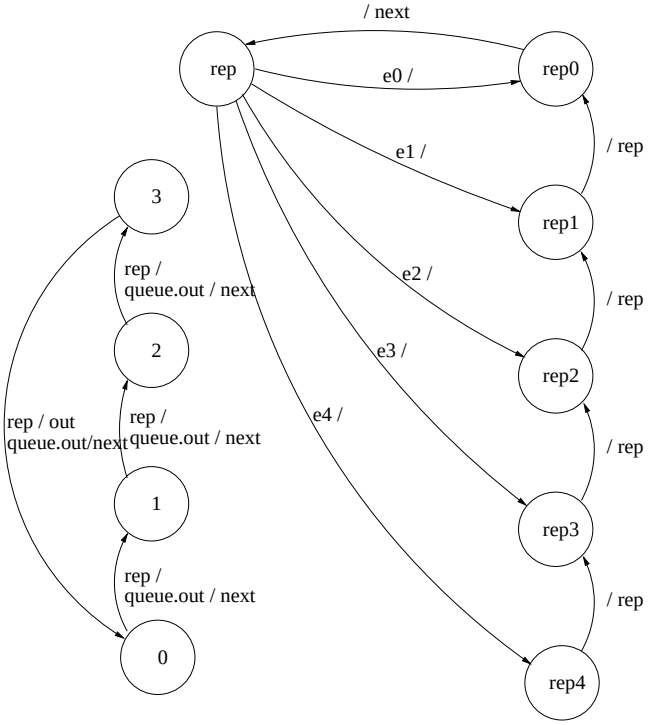

Fig. 7: Transmitter

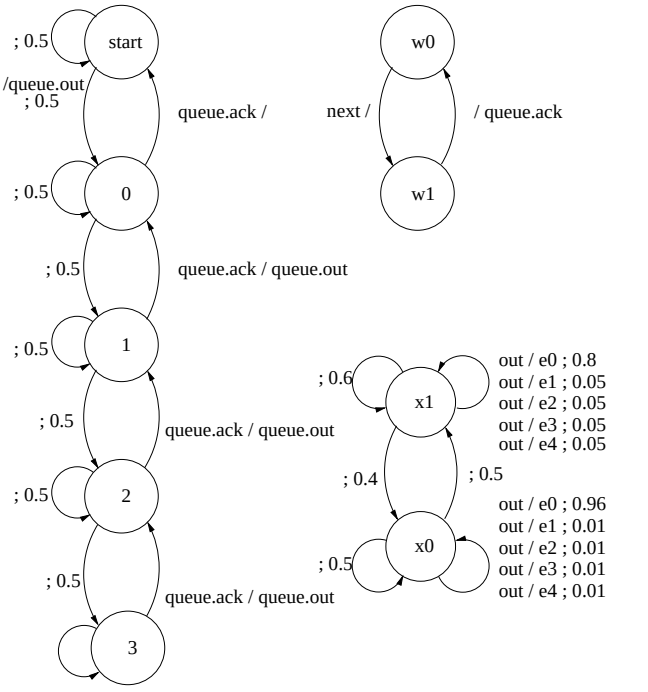

Fig. 8: Packet Source and Queue, Timer, Channel (2 states)

(of 65 different states left) of the CFSS have been omitted for the sake of brevity. In a realistic scenario, they are not useful for human analysis anyway.

Sorted according to the FSSMs, adding the separate states up, we get:

- Transmit Buffer

```
0 = 0.240155
1 = 0.246998
2 = 0.253376
3 = 0.259471
```

- Repeater Buffer

```
rep = 0.798436
rep0 = 0.134376
rep1 = 0.026875
rep2 = 0.020156
rep3 = 0.013438
rep4 = 0.006719
```

- Timer

```
w0 = 0.637186
w1 = 0.362814
```

- Source

```
start = 0.435622
0 = 0.473397
1 = 0.057784
2 = 0.027575
3 = 0.005621
```

C. Evaluation

This section is about obtaining the relevant data out of the simplified model.

Throughput. All packets that have been created but were not lost, get transmitted to the receiver. These are all packets, that were not created while the source buffer was in state 3. Thus, we add up all steady state probabilities which do not contain state 3 and we multiply the result with the probability for a packet created in this state, which has a value of 0.5. We get

$$n_p = 0,5 \cdot \sum_{s_p \neq 3} \mathcal{S}_\infty(s_p),$$

which accounts to $0,4971895$ packets per slot [7]. This is the throughput which is observed.

[7] The values have been put down with all available digits in order to enable the reader to follow this example easily. Of course, the actual precision depends on the accuracy of the radio channel model. Furthermore numerical stability problems must be considered.

Retransmitted Packets. Whenever the repeat buffer is neither in state rep nor in state $rep0$, it is retransmitting a packet on the channel. Thus, we get

$$n_w = \sum_{s_w \in R} \mathcal{S}_\infty(s_w) \text{ ; with } R = \{rep1 \ldots rep4\},$$

equal to $6,7188\%$ of the time. Together with the packets sent for the first time (equal to the packets generated and not lost), the channel is busy

$$n_b = n_p + n_w,$$

which is $56,43775\%$, of the time, which means it tries to transmit a packet. Furthermore, we yield the information that

$$n_v = \frac{n_w}{n_b},$$

equals to $11,904798\%$, of the packets transmitted on the channel were lost and need to be retransmitted.

V. Conclusions

So far, simple communication protocols have been successfully analysed using the method described above [7]. Ongoing work is focusing on resolving the problems involved with this method, especially concerning the use of extended finite state machines as a means to describe more complex communication protocols. Also the concept of time (for example, consider concurrency in contrast to parallelism) needs careful examination.

We believe that these methods are applicable on the physical and data-link layers of communication protocols, as well as on a broad range of other technical applications.

VI. References

[1] H. Steffan, "Generative Radio Channel Models for Analysis and Simulation," in *Proceedings IMACS Mathmod*, (Technical University Wien), pp. 753–759, 1994.

[2] H. Steffan, "Generative Radio Channel Models for Analysis and Simulation and their Properties," in *8. Aachener Kolloquium für Signaltheorie - Mobile Kommunikationssysteme*, (Aachen, Germany), pp. 149–153, 1994.

[3] H. Steffan, *Stochastische Modelle für den Funkkanal und deren Anwendung*. PhD thesis, RWTH Aachen, Aachen, June 1995.

[4] L. Kittel, "Eine Anwendung stochastischer Automatenmodelle zur Ermittlung günstiger Blocksynchronisationswörter für die digitale Funksignalisierung," *3. Aachener Kolloquium, RWTH*, pp. 125–128, 1979.

[5] L. Kittel, "Zur Modellierung diskreter Fehlerprozesse durch stochastische Automaten und Anwendung im Mobilfunk," *4. Aachener Kolloquium, RWTH*, pp. 135–138, 1981.

[6] L. Kittel, "Automatentheoretische Analyse von Übertragungssystemen für leitungscodierte Digitalsignale," *NTG-Fachtagung, Neue Aspekte der Informations- und Systemtheorie, Garmisch-Patenkirchen, NTG-Fachberichte 84*, pp. 155–169, Mar. 1983.

[7] M. Ostermann, "Entwicklung eines Werkzeuges zur Verwaltung stochastischer Automaten," Master's thesis, RWTH Aachen, Lehrstuhl Kommunikationsnetze, Aachen, 1995.