# Minimization of Stochastic Dynamical Systems

A Comparison of Models & Methods

## Software Requirement Specification & Design Documentation

by
Fabian Klopfer


Faculty of Sciences
Department of Computer and Information Science
Modelling of Complex Self-Organizing Systems Group

**Advisor:** Prof. Dr. Tatjana Petrov

Konstanz, 2020

# Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this Software Requirements Specification is to describe the features, constraints and demands of a tool for model reduction and minimization of stochastic dynamical systems in detail. This document is intended for both the supervisors and the developers of the tool.

## 1.2 Scope

This software system shall implement a tool for model reduction and minimization of stochastic dynamical systems, that provides support for differential equations and automata, equivalent benchmark examples for each, at least one algorithm for the reduction of each — namely [2] and [1]. An optional feature is the implementation of a conversion algorithm between automata and differential equations, when theoretically possible.

## 1.3 Definitions, Acronyms, Abbreviations

**Differential equation** (DE) An equation relating a function to its derivative with respect to one or more independent variable.

**Weighted automata** (WA) An automaton consisting of states and a transition function which associates a weight to each transition

**Possible DE extensions** Extensions of the differential equation framework. For details on the specifics, see **??**

**Isomorphism** A structure-preserving, revertable, i.e. bijective morphism or mapping from a domain to a codomain

## 1.4 Overview

In the second chapter several conditions, assumptions and circumstances will be mentioned, that help charachterizing the software's special use case. In the thrid chapter the concrete requirements are listed.

# 2 Overall description

## 2.1 Product Perspective

The software system shall be the practical and evaluative basis for the underlying theory, described in the previous report. Especially it shall provide an empirical basis for the analysis of equivalence of reduction techniques for DEs and the minimization of WAs, i.e. if reducing a DE using an algorithm, does one arrive at the same complexity as when minimising a weighted automaton representing the same system as the DE. For that it shall offer scalable benchmark examples, that is an example DE with an equivalent WA. Further it shall provide a base for implementing further algorithms or new methods to be developed (e.g. if an isomorphism is found between DEs and WAs a conversion tool).

## 2.2 Product Functions

The system shall take a DE or a WA as input and apply one selected algorithm onto those and then output the minimized model. Furhter it shall be able to compare the results of a minimisation in terms of model complexity and equivalence, i.e. answer the question weather DE reduction and WA minimisation yield different representations of the same model or if one model is more complex.

## 2.3 User Characteristics

For now the only user is going to be the developer in the course of the master project and thesis.

## 2.4 Constraints

- Run time is crucial for these algorithms to be feasible. The system shall not impose significant overhead and the implementations shall be as fast as possible
- Memory safety is important as inputs may induce infinite memory requirements that must not execute if so (else the system will crash)
- Correctness is the most important aspect of the implementation as the results will be unusable if minor errors are introduced that corrupt the reduction or minimization processes as well as comparisons and benchmarks

## 2.5 Assumptions and Dependencies

to be set when dependencies are fixed
CMake, C++, TCLAP, Catch2
Eigen, SuiteSparse, MAGMA

# 3 Requirement Specification

## 3.1 Software System requirements

The system shall be implemented with focus on the following traits:

S 1. Correctness

S 2. Extensibility

S 3. Maintainability

S 4. Lightweightness

S 5. As fast as possible

S 6. Memory safe for all inputs

S 7. Cross-compileable.

## 3.2 External Requirements

The system shall:

E 1. to be set when dependencies are fixed

## 3.3 Functional requirements

F 1. Control flow system
The system shall
   1. provide a mechanism for applying the methods depending on the specified model, i.e. apply WA validation checks, algorithm and evaluation for WA input and the respective equivalent for DEs.
   2. provide a file based input and output mechanism.
   3. provide a user interface based input and output mechanism.
   4. validate the inputs before algorithm execution

F 2. User interface
The system shall
   1. offer 3 different methods to be applied: WA minimization, DE reduction and Benchmarking.
   2. provide a way to specify which input method is to be used, either file-based or user interface-based.
   3. provide a way to specify which output method is to be used.
   4. provide the possibility to select an algorithm from a set of available ones for both models.
   5. display the input, the output and summaries and statistics provided by the model specific modules

F 3. WA minimization
The system shall
   1. specify an input format for WA.
   2. provide a validation machanism to ensure the well-formedness of the input.
   3. implement the algorithm specified in [2].
   4. use the algorithm above on an example specified in 5.1 and work deterministically correct with a fixed set of random vectors.
   5. summarize the results and statistics on the difference between input and output.

F 4. DE reduction
The system shall

1. specify an input format for arbitrary DEs and possible extensions, specifically stochastic differential, functional differential, integro-differential and differential-algebraic equations.
2. provide a validation machanism to ensure the well-formedness of the expression.
3. implement the algorithm specified in [1]
4. summarize the results and statistics on the difference between inputs and outputs

F 5. Benchmark The system shall

1. provide example WA and DEs, which are
    1. theoretically proven to be equivalent, see **??**
    2. scalable by a parameter, defining the number of states or the number of variables
2. implement a method to compare the results in terms of
    1. run time,
    2. memory
    3. resulting model complexity

F 6. Conversion or Equivalence The system shall

1. be able to convert the benchmark examples from one model to the other or to prove that instances of distinct models are equivalent
2. (Optional) be able to convert all instances of one model to the other model

# 4 Proposed Software Architecture

## 4.1 Overview

**Main Module** Entry point of the process, handles for the control flow.

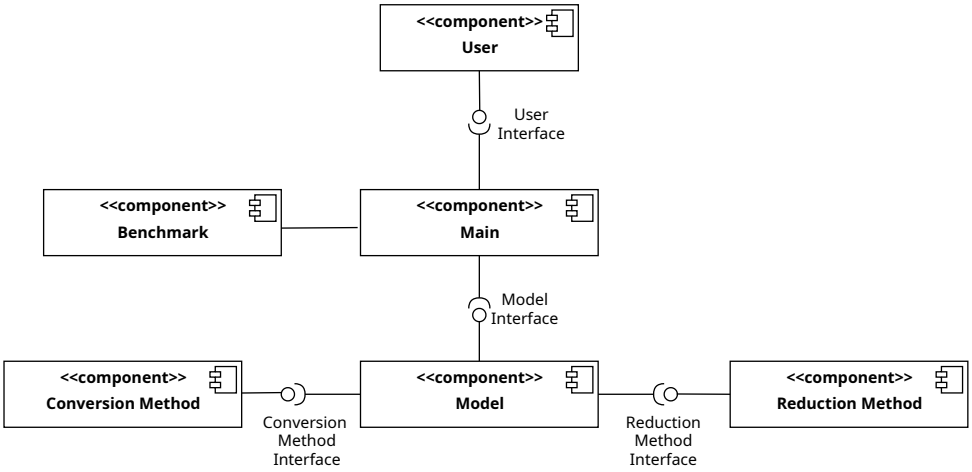
**User Interface** Lets the user control the main module.

**Model Interface** Defines a model including format.

**Reduction Method Interface** Defines a reduction method for a specific model.

**Conversion/Equivalence Interface** Defines the conversion or equivalence notion from a specific model to a specific model.

**Benchmark Module** Defines benchmark examples and generation processes for those.




## 4.2 Subsystem decomposition

**Main Module** The main module is the main entry point of the process, responsible for executing the user interface, the algorithms and utilities from the model specific modules, the conversions, as well as dealing with benchmark flows and loading the necessary examples. There are two mandatory basic control flows and one optional one that depends on the theoretical feasibility that is to be proven:

1. Standard model reduction for a single model with user specified input
2. Benchmark flow using the provided examples, resulting in two sequential model reduction executions. They can not be parallelized, as they would infere with the caches and slow each other down. After the execution it converts one instance to the other model preserving model complexity or checks if they are equivalent.

**User Interface** The User Interface let's the user control the main module, provides means of input to the main module and displays the output from the main module. Also contains usage information.
A text-based user interface is to be implemented (also called command-line interface). The user interface must contain a menu letting the user specify what control flow is to be executed, thus what task the user wants to perform. It must be able to either read the data from file or from the standard input of the terminal and display the input, output and summary on the standard output of the terminal, aswell as possible errors.
The UserInterface provides functionality to control the program and to deliver outputs to the user. Therefore most functions are parameterless and take their input from the user, so those must ensure the validity of the user input. The only special case is the actual input data which is validated by a model specific function and the output functions which do not take user input.

```
Interface UserInterface {
    public Task select_task() {
        ensure Task exists
    }

    public ModelInterface select_model() {
        ensure model exists
    }

    public ReductionMethodInterface select_method(ModelInterface model) {
        require model not null
        ensure method exists && method compatible to model
    }

    public IOMethod select_input_method() {
        ensure input method exists
    }

    public FILE* file_input() {
        ensure file exists && file is closed after read
    }

    public char* stdin_input() {
        require limited input
        ensure input is read completely or rejected
    }

    public IOMethod select_output_method() {
        ensure output method exists
    }

    public FILE* set_output_location() {
        ensure file exists or create new file && containing directory exists
    }

    public void display_exec_file(FILE* output_destination) {
        require output_destination not null
        assure file is closed after write
    }

    public void display_exec_stdout(char* output) {
        require output not null && output limited
        ensure output is less than default terminal buffer size || output is paged
    }
}
```

**Model Interface** The model interface specifies the details of a model including the representation format, input validation functionality, concrete instances will hold an array of function pointers of the reduction method interface type for the concrete model. Depending on the task that the user wants to perform either one or another instance is used to execute the control flow. The data flow from the user via the main module is passed to the model interface instance which does the actual processing using the defined functions and one of the functions from the array holding the

model reduction method pointers. Instances of this interface must also specify how to summarize and compute statistics over the result, i.e. the differences in the model and in model complexity. To be implemented are instances for DEs and WAs.

```
Interface ModelInterface {
   ModelRepresentation representation require not null
   ReductionInterface*[] ReductionMethodInterface[]
   ConversionMethodInterface*[] conversions

   public ModelRepresentation validate_model_instance(char* input) {
       require input not null
       ensure model is valid wrt. representation

   }

   public char* summarize_reduction(ModelRepresentation input, ModelRepresentation reduced)
       require input not null && reduced not null
   }
}
```

**Model Representation Interface** As the data needs to be validated, this interface shall specify a structure for holding an instance of the model.

```
Interface ModelRepresentationInterface {
   struct ModelRepresentation require not null
}
```

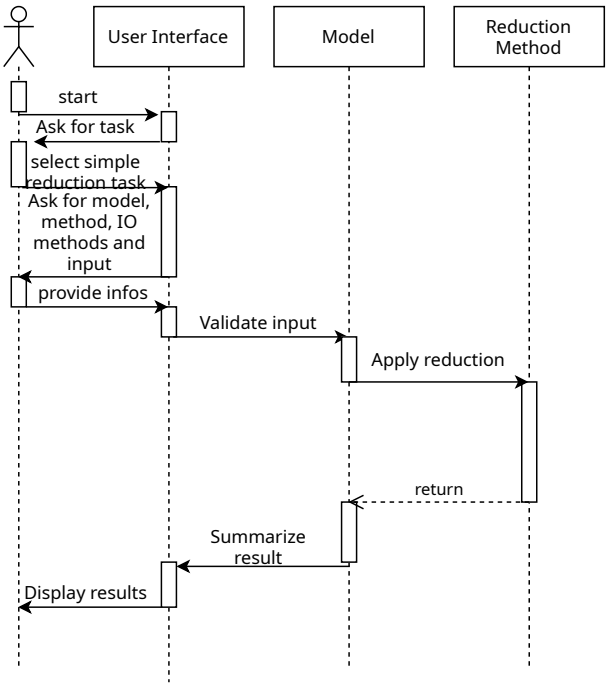
**Reduction Method Interface** This interface is used to implement concrete reduction methods for a specific model. For each models there may be multiple reduction methods. To be implemented are the reduction methods specified in [1, 2].

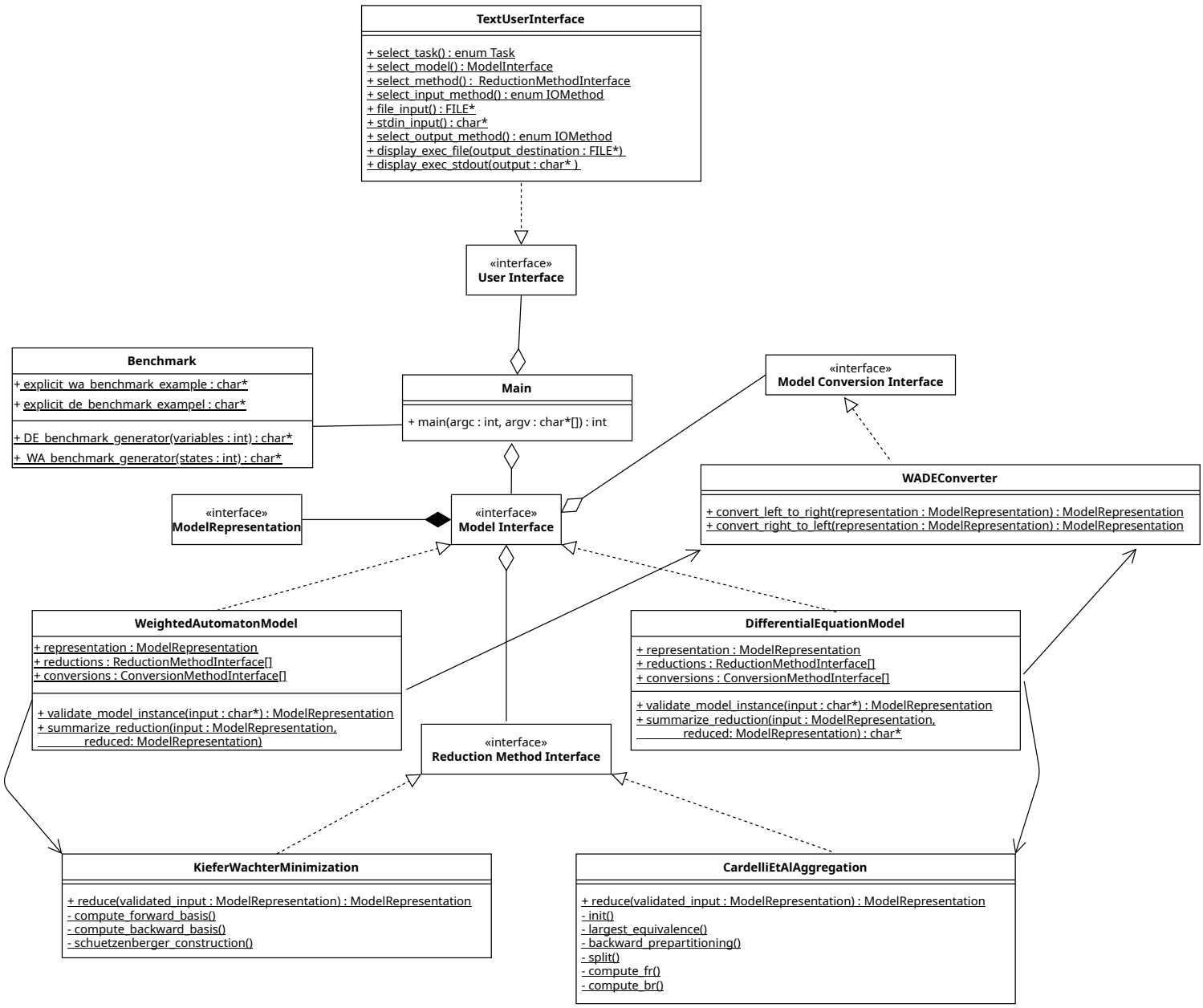
```
Interface ReductionMethodInterface {
   public ModelRepresentation reduce(ModelRepresentation input) {
       require input not null
       ensure return value valid and not null
   }198/1536 - 132/1024
}
```

**Conversion/Equivalence Interface** Using the model interfaces, implementations of this interface shall provide a method to convert an instance of a model to another model or to prove the equivalence of two instances of different models. As of now this is a question to be further investigated in terms of theory. One open question is if there exists an isomorphism between the to be implemented methods and if so how to do the conversion or equivalence check without also transforming the model complexity (e.g. from DEs to WAs, how not to produce more or less states than there are implicitly in the DE representation). The interface should optimally provide methods for conversion in both direction, i.e. a programatically equivalence proof and shall be attached to both models that it converts as pointer.

```
Interface ModelConversionInterface {
   public ModelRepresentation convert_left_to_right(ModelRepresentation left) {
       require left not null
       ensure output is valid in rights's representation
   }
```

```
    ModelRepresentation convert_right_to_left(ModelRepresentation right) {
        require right not null
        ensure output is valid in left's representation
    }
}
```



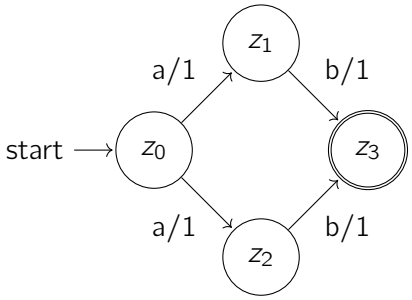

TextUserInterface
+ select_task() : enum Task
+ select_model() : ModelInterface
+ select_method() : ReductionMethodInterface
+ select_input_method() : enum IOMethod
+ file_input() : FILE*
+ stdin_input() : char*
+ select_output_method() : enum IOMethod
+ display_exec_file(output_destination : FILE*)
+ display_exec_stdout(output : char* )
«interface»
User Interface
Benchmark
+ explicit_wa_benchmark_example : char*
+ explicit_de_benchmark_exampel : char*
+ DE_benchmark_generator(variables : int) : char*
+ WA_benchmark_generator(states : int) : char*
Main
+ main(argc : int, argv : char*[]) : int
«interface»
Model Conversion Interface
WADEConverter
+ convert_left_to_right(representation : ModelRepresentation) : ModelRepresentation
+ convert_right_to_left(representation : ModelRepresentation) : ModelRepresentation
«interface»
ModelRepresentation
«interface»
Model Interface
WeightedAutomatonModel
+ representation : ModelRepresentation
+ reductions : ReductionMethodInterface[]
+ conversions : ConversionMethodInterface[]
+ validate_model_instance(input : char*) : ModelRepresentation
+ summarize_reduction(input : ModelRepresentation,
reduced: ModelRepresentation)
DifferentialEquationModel
+ representation : ModelRepresentation
+ reductions : ReductionMethodInterface[]
+ conversions : ConversionMethodInterface[]
+ validate_model_instance(input : char*) : ModelRepresentation
+ summarize_reduction(input : ModelRepresentation,
reduced: ModelRepresentation) : char*
«interface»
Reduction Method Interface
KieferWachterMinimization
+ reduce(validated_input : ModelRepresentation) : ModelRepresentation
- compute_forward_basis()
- compute_backward_basis()
- schuetzenberger_construction()
CardelliEtAlAggregation
+ reduce(validated_input : ModelRepresentation) : ModelRepresentation
- init()
- largest_equivalence()
- backward_prepartitioning()
- split()
- compute_fr()
- compute_br()

# 5 Algorithm Test Cases

## 5.1 Weighted Automata

Weighted automaton $\mathcal{A} = (n, \Sigma, \mu, \alpha, \eta)$ with

- $n = 3$,
- $\Sigma = \{a, b\}$,
- $\alpha = (1, 0, 0, 0)$,
- $\eta = (0, 0, 0, 1)$,
- $\mu(a) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$,
- $\mu(b) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.




All steps were performed using the Matlab script found in

`src/test/WA_kiefer.m`

.

**Restrictions:** Those are actually obvious from the automaton. To be on the safe side the "proofs" are stated explicitly here.

- We only need to evaluate words starting with "a" and ending with "b".

$$\alpha\mu(a) = (0, 1, 1, 0); \quad \alpha\mu(b) = (0, 0, 0, 0); \quad \mu(a)\eta = (0, 0, 0, 0)^T; \quad \mu(b)\eta = (0, 1, 1, 0)^T$$

- Words containing two consecutive letters do not need to be considered:

$$\mu(a)^2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad \mu(b)^2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Words may only contain "ab" as "ba" yields the zero matrix and matrix multiplication is associative (i.e. $(\mu(a)\mu(b))\mu(a) = \mu(a)(\mu(b)\mu(a)) = \mu(a) \cdot 0 = 0$

$$\mu(a)\mu(b) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mu(b)\mu(a) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Thus only words of length smaller or equal to 2 must be considered, staring with a or ending with b. This follows from the above, namely that only words starting with "a", not having two consecutive letters or "ba" in them yield non-zero vectors or matricies.

**Forward reduction,** i.e. applying [2] Figure 3: The only words that need to be considered due to the above restrictions are the words "a" and "ab".
Random vectors are generated using Wolframalpha and $K = 3$, i.e. $r^i \in \{1, \dots, 12\}^{2\times 4}$:

$$r^{(1)} = \begin{pmatrix} 9 & 5 & 5 & 7 \\ 6 & 11 & 2 & 1 \end{pmatrix}; \quad r^{(2)} = \begin{pmatrix} 2 & 3 & 1 & 2 \\ 12 & 3 & 9 & 4 \end{pmatrix} r^{(3)} = \begin{pmatrix} 2 & 7 & 9 & 10 \\ 1 & 11 & 2 & 6 \end{pmatrix} r^{(4)} = \begin{pmatrix} 4 & 5 & 2 & 10 \\ 5 & 9 & 5 & 5 \end{pmatrix}$$

As the two words are the same for all $v_1, \dots, v_n$ we have:

$$\rho(r^{(i)}) = \sum_{k=0}^{n} \sum_{w\in\Sigma^k} \alpha \sum_{j=1}^{|w|} \mu(w_j) r^{(i)}_{w_j j} = \alpha\mu(a) r^{(i)}_{a,1} + \alpha\mu(a)\mu(b) r^{(i)}_{a,1} r^{(i)}_{b,2}$$

$$= (0,1,1,0) r^{(i)}_{a,1} + (0,0,0,2) r^{(i)}_{a,1} r^{(i)}_{b,2}$$

Thus:

$$v_1 = 9 \cdot (0,1,1,0) + 9 \cdot 11 \cdot (0,0,0,2) = (0,9,9,198)$$
$$v_2 = 2 \cdot (0,1,1,0) + 2 \cdot 3 \cdot (0,0,0,2) = (0,2,2,12)$$
$$v_3 = 2 \cdot (0,1,1,0) + 2 \cdot 11 \cdot (0,0,0,2) = (0,2,2,44)$$
$$v_4 = 4 \cdot (0,1,1,0) + 4 \cdot 9 \cdot (0,0,0,2) = (0,4,4,72)$$

$\alpha$ needs to be stadcked atop of the row vectors.According to Matlab the maximally linearly independent subset is $\{\alpha, v_1, v_2\}$ and by that

$$\overrightarrow{F} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 9 & 9 & 198 \\ 0 & 2 & 2 & 12 \end{pmatrix}$$

Applying the forward reduction using Schützenberger construction as described in [2] Eq. 4.2 using the right inverse or the Moore-Penrose inverse:

$$\overrightarrow{F}\mu(\sigma) = \overrightarrow{\mu}(\sigma)\overrightarrow{F} \equiv \overrightarrow{\mu}(\sigma) = \overrightarrow{F}\mu(\sigma)\overrightarrow{F}^{-1}_{R}$$

Solving using Matlab gives:

$$\overrightarrow{\mu}(a) = \begin{pmatrix} 0 & \frac{-1}{24} & \frac{11}{16} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Regarding $\overrightarrow{\mu}(b)$ using the same procedure we get

$$\overrightarrow{\mu}(b) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{-9}{16} \\ 0 & \frac{1}{36} & \frac{-1}{8} \end{pmatrix}$$

Regarding $\overrightarrow{\eta}$ we get:

$$\overrightarrow{\eta} = \overrightarrow{F}\eta = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 9 & 9 & 198 \\ 0 & 2 & 2 & 12 \end{pmatrix} \cdot (0,0,0,1)^T = (0,198,12)^T$$

And finally $\overrightarrow{\mathcal{A}} = (3, \{a,b\}, \overrightarrow{\mu}, (1,0,0), (0,198,12)^T)$

Additional checks, that this is indeed the same automaton can be found in the output of the Matlab script using the

`probe_and_display()`

function.

**Backward reduction** The only words that need to be considered due to the above restrictions are the words "b" and "ab". The second reduction step of the algorithm is to be applied on the output automaton of the previous reduction, i.e. $\overrightarrow{\mathcal{A}}$
Random vectors are generated using Wolframalpha and $K = 3$, i.e. $r^i \in \{1, \ldots, 9\}^{2\times 3}$:

$$r^{(1)} = \begin{pmatrix} 1 & 6 & 2 \\ 4 & 5 & 4 \end{pmatrix}; \quad r^{(2)} = \begin{pmatrix} 8 & 1 & 8 \\ 7 & 8 & 2 \end{pmatrix} r^{(3)} = \begin{pmatrix} 1 & 6 & 5 \\ 9 & 2 & 1 \end{pmatrix}$$

As the two words are the same for all $v_1, \ldots, v_n$ we have:

$$\rho(r^{(i)}) = \sum_{k=0}^{n} \sum_{w\in\Sigma^k} \left( \sum_{j=1}^{|w|} \overrightarrow{\mu}(w_j) r^{(i)}_{w_j,j} \right) \overrightarrow{\eta} = \overrightarrow{\mu}(b)\overrightarrow{\eta}\, r^{(i)}_{b,1} + \overrightarrow{\mu}(a)\overrightarrow{\mu}(b)\overrightarrow{\eta}\, r^{(i)}_{a,1} r^{(i)}_{b,2}$$

$$= (0, 18, 4) r^{(i)}_{b,1} + (2, 0, 0) r^{(i)}_{a,1} r^{(i)}_{b,2}$$

Thus:

$$v_1 = 4 \cdot (0, 18, 4) + 1 \cdot 5 \cdot (2, 0, 0) = (10, 72, 16)$$

$$v_2 = 7 \cdot (0, 18, 4) + 8 \cdot 8 \cdot (2, 0, 0) = (128, 126, 28)$$

$$v_3 = 9 \cdot (0, 18, 4) + 1 \cdot 2 \cdot (2, 0, 0) = (4, 162, 36)$$

$\overrightarrow{\eta}$ needs to be stadcked ahead of the column vectors. According to Matlab the maximally linearly independent subset is $\{eta, v_1, v_2\}$ and by that

$$\overleftrightarrow{B} = \begin{pmatrix} 0 & 10 & 128 \\ 198 & 72 & 126 \\ 12 & 16 & 28 \end{pmatrix}$$

Applying the backward reduction using Schützenberger construction as described in [2] Eq. 4.2 using the left inverse or the Moore-Penrose inverse:

$$\overrightarrow{\mu}(\sigma)\overleftarrow{B} = \overleftarrow{B}\overleftrightarrow{\mu}(\sigma) \equiv \overleftrightarrow{\mu}(\sigma) = \overleftarrow{B}_L^{-1}\overrightarrow{\mu}(\sigma)\overleftarrow{B}$$

Solving using Matlab gives:

$$\overleftrightarrow{\mu}(a) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{-28}{221} & \frac{-49}{221} \\ 0 & \frac{16}{221} & \frac{28}{221} \end{pmatrix}$$

Regarding $\overleftrightarrow{\mu}(b)$ using the same procedure we get

$$\overleftrightarrow{\mu}(b) = \begin{pmatrix} 0 & 0 & 0 \\ \frac{64}{221} & 0 & 0 \\ \frac{-5}{221} & 0 & 0 \end{pmatrix}$$

Regarding $\overleftrightarrow{\alpha}$ we get:

$$\overleftrightarrow{\alpha} = \overleftarrow{B}\overrightarrow{\alpha} = \begin{pmatrix} 0 & 10 & 128 \\ 198 & 72 & 126 \\ 12 & 16 & 28 \end{pmatrix} \cdot (1, 0, 0) = (0, 10, 128)$$

Finally $\overleftrightarrow{\mathcal{A}} = (3, \{a, b\}, \overleftrightarrow{\mu}, (0, 10, 128), (1, 0, 0))$.

Additional checks, that this is indeed the same automaton can be found in the output of the Matlab script using the

`probe_and_display()`

function. ■

## References


[1] Luca Cardelli et al. "Maximal aggregation of polynomial dynamical systems". In: *Proceedings of the National Academy of Sciences of the United States of America* 114 38 (2017), pp. 10029–10034.

[2] Stefan Kiefer et al. "On the Complexity of Equivalence and Minimisation for Q-weighted Automata". In: *Logical Methods in Computer Science* 9 (2013).