Seminarpaper

Mario Wagner, 0730223

Graz, am September 27, 2016

Contents

List of Figures

1 Introduction

Nothing yet.

2 Preliminiary

2.1 AALERGIA

In this paper, the algorithm AALERGIA [1] and the provided MATLAB implementation[1] are used as a starting point for our work. In order to get a better idea of the way the implementation works, we will describe it in detail and a running example for better replicability is used throughout this section. The implementation of the algorithm consists of several parts, which are depicted in Figure 1.

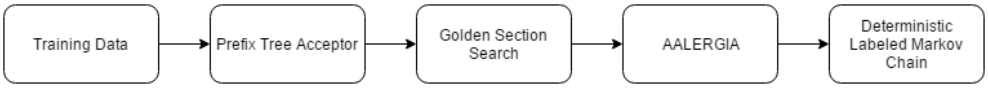

Figure 1: Flow Diagram AALERGIA

2.1.1 Training Data

The data used to learn the Deterministic Labeled Markov Chain (DLMC) is supplied by an external *.mat file, which contains MATLAB formatted data: The training set and the alphabet to be used. The alphabet consists of a finite 1xN-cell array, where each column contains one letter in the alphabet. The training set consist of a finite Nx1-cell array, where each row contains sequences of symbols generated from the model, separated by commas.

The running example we are going to use is a self-stabilizing ring network with 3 processes. Figure 2 shows the beginning of the trainings set, which contains 50 Sequences in total. In our case, each symbol in the training set represents the states in the ring at a given moment. For example, the symbol 001 shows that process 3 is in the state 1 and process 1 and 2 are in the state 0.

1	000,001,
2	000,011,100,010,101,010,001,100,010,101,
3	000,101,
4	000,111,010,001,110,011,101,010,001,110,011,
5	000,111,011,101,110,001,
6	000,101,010,101,110,011,100,011,101,110,

Figure 2: The trainings set

Figure 3 shows the alphabet, which consists of 8 symbols from 000 to 111.

[1] http://mi.cs.aau.dk/code/aalergia

	1	2	3	4	5	6	7	8
1	000	001	010	011	100	101	110	111
2								

Figure 3: The alphabet

2.1.2 Frequency Prefix Tree Acceptor

After loading the trainings set and the alphabet into the workspace, the Frequency Prefix Tree Acceptor (FPTA) is created. The frequency in this automaton shows how often an event occurs. A deterministic frequency finite automaton (DFFA) is a tuple $A = \langle \Sigma, Q, I_{\text{fr}}, F_{\text{fr}}, \delta_{\text{fr}}, \delta \rangle$ 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.

Algorithm 1 shows how the creation of the FPTA is implemented in the AALERGIA package. The code starts by sorting the trainings set and by removing double occurrences (line 3). After that, a for-loop iterates through the sorted strings and splits them at each comma (line 6 - 15). This creates all the prefixes from the trainings set and adds them to the cell array named *prefix*, as shown in Figure 4.

51	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,011,100,010,101,010,
52	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,011,100,010,101,
53	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,011,100,010,
54	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,011,100,
55	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,011,
56	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,100,
57	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,011,
58	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,100,
59	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,001,
60	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,010,
61	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,100,
62	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,011,
63	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,100,
64	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,001,
65	000,000,010,001,100,011,101,110,001,110,001,110,011,101,010,
66	000,000,010,001,100,011,101,110,001,110,001,110,011,101,
67	000,000,010,001,100,011,101,110,001,110,001,110,011,
68	000,000,010,001,100,011,101,110,001,110,001,110,
69	000,000,010,001,100,011,101,110,001,110,001,
70	000,000,010,001,100,011,101,110,001,110,
71	000,000,010,001,100,011,101,110,001,
72	000,000,010,001,100,011,101,110,
73	000,000,010,001,100,011,101,
74	000,000,010,001,100,011,

Figure 4: The Prefixes

Algorithm 1 Create the FPTA

1: **Input:** the $\sum$ and a set of strings S (training data)
2: **Output:** a DFFA A and a sorted set of strings U
3: $U \leftarrow \text{sort}(S)$
4: *prefixes* $\leftarrow U$
5: $F_{fr} \leftarrow \text{string\_count}(U)$
6: **for** `u` $\in$ `U` **do**
7: *index* $\leftarrow$ find positions of "," in u
8: **while** *index* > 0 **do**
9: *substr* $\leftarrow$ *substring(1:index)*
10: **if** *substr* $\notin$ *prefix* **then**
11: *prefixes(end+1)* $\leftarrow$ *substr*
12: **end if**
13: *index* $\leftarrow$ *index-1*
14: **end while**
15: **end for**
16: *prefixes* $\leftarrow \text{width\_first\_sort}(prefixes)$
17: *predecessor* $\leftarrow \text{find\_predecessor}(prefixes)$
18: **for** `p` $\in$ `prefixes` **do**
19: *sym* $\leftarrow$ *get_symbol(p)*
20: *pre* $\leftarrow$ *predecessor(p)*
21: *freq_trans_matrix*$\{1\}$ *(pre, sym)* $\leftarrow$ *predecessor(p)*
22: *frequency_transition* $\leftarrow$ *freq_trans_matrix*$\{2\}$ *(pre, sym)*
23: *frequency_transition* $\leftarrow$ *frequency_transition* $+$ *frequency(p)*
24: **while** *pre* **do**
25: *update_freq_trans_matrix(p, pre)*
26: *pre* $\leftarrow$ *predecessor(p)*
27: **end while**
28: **end for**
29: *init_states* $\leftarrow$ *freq_trans_matrix*$\{1\}$*(1, all)*
30: **return** *return A*

In our example, the dimension of *prefix* is 494x1 and contains all unique prefixes of the trainings set for further evaluation. The cell array *prefix* is sorted by width first sort (line 16). The shortest strings are at the beginning of the array and the largest at the end of the array, as shown in Figure 5.

The function *find_predecessor* searches through the *prefixes* array and saves all the predecessors in the corresponding array (line 16). In the end, a for-loop over all the prefixes is executed in order to find all transitions and their frequencies respectively. The values are saved into the *freq_trans_matrix* (line 18 - 28).

2	000,
3	000,000,
4	000,001,
5	000,010,
6	000,011,
7	000,100,
8	000,101,
9	000,110,
10	000,111,
11	000,000,010,
12	000,000,100,
13	000,000,111,
14	000,001,100,
15	000,001,110,

Figure 5: The Sorted Prefix Array

2.1.3 Golden Section Search

2.1.4 AALERGIA

2.1.5 Deterministic Labeled Markov Chain

3 Related Work

Nothing yet.

4 Meat

Nothing yet.

5 Evaluation

Nothing yet.

6 Conclusions

Nothing yet.

References

[1] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning probabilistic automata for model checking," in *2011 Eighth International Conference on Quantitative Evaluation of Systems (QEST)*, pp. 111–120.