Matlab Function Library of Finite State Automata Operations

Bruno Lacerda

Abstract—The purpose of this text is to provide a manual for the utilization of a number of Matlab functions that implement operations over Finite State Automata that are relevant for the modeling and analysis of Discrete Event Systems.

I. INTRODUCTION

Finite State Automata (FSA) are widely used in the modeling of Logical Discrete Event Systems (DES). Hence, it is useful to have a library of Matlab functions that implements some operations over FSA that are useful in the scope of DES. These functions will allow, for example, to synthesize a supervisor for a given DES and to answer several analysis problems, such as safety, blocking, state estimation and diagnostics.

The text is divided in three main sections. First, we define FSA and the operations that will be implemented and how they can be used to synthesize a supervisor and answer the analysis problems. Then, we describe the data structure used to code an FSA. Finally, a description of how to use the functions is provided.

II. DEFINITIONS

We start by giving the notion of Deterministic Finite State Automaton, and Non-Deterministic Finite State Automaton with ε transitions and defining generated and marked language.

Definition 1 (Deterministic Finite State Automaton) A Deterministic Finite State Automaton (DFA) is a six-tuple $G = (X, E, f, \Gamma, x_0, X_m)$ where:

- $X$ is the finite set of states
- $E$ is the finite set of events
- $f : X \times E \rightarrow X$ is the (possibly partial) transition function
- $\Gamma : X \rightarrow 2^E$ is the active event function
- $x_0 \in X$ is the initial state
- $X_m \subseteq X$ is the set of marked states

Definition 2 (Non-Deterministic Finite State Automaton) A Non-Deterministic Finite State Automaton with ε transitions ($\varepsilon$-NFA) is a six-tuple $N = (X, E \cup \{\varepsilon\}, f, \Gamma, X_0, X_m)$ where:

- $X$ is the finite set of states
- $E$ is the finite set of events
- $f : X \times (E \cup \{\varepsilon\}) \rightarrow 2^X$ is the (possibly partial) transition function
- $\Gamma : X \rightarrow 2^E$ is the active event function
- $x_0 \in X$ is the initial state
- $X_m \subseteq X$ is the set of marked states

Definition 3 (Generated Language) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be an FSA. We define the language generated by $G$ as $L(G) = \{s \in E^* : f(x_0, s) \text{ is defined}\}$

Definition 4 (Marked Language by a DFA) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. We define the language marked by $G$ as $L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}$

Now, we define the operations over FSA that will be implemented.

Definition 5 (Accessible Part) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. The accessible part of $G$ is the DFA $Ac(G) = (X_{ac}, E, f_{ac}, \Gamma_{ac}, x_0, X_{ac,m})$ where:

- $X_{ac} = \{x \in X : \exists s \in E^* \text{ such that } f(x_0, s) = x\}$
- $X_{ac,m} = X_m \cap X_{ac}$
- $f_{ac} = f|_{X_0 \times E \rightarrow 2^X_{ac}}$
- $\Gamma_{ac} = \Gamma|_{X_{ac} \rightarrow 2^E}$

Definition 6 (Co-Accessible Part) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. The accessible part of $G$ is the DFA $CoAc(G) = (X_{coac}, E, f_{coac}, \Gamma_{coac}, x_{coac,0}, X_m)$ where:

- $X_{coac} = \{x \in X : \exists s \in E^* \text{ such that } f(x, s) \in X_m\}$
- $x_{coac,0} = \{x_0 \text{ if } x_0 \in X_{coac} \text{ undefined otherwise}\}$
- $f_{coac} = f|_{X_0 \times E \rightarrow 2^X_{coac}}$
- $\Gamma_{coac} = \Gamma|_{X_{coac} \rightarrow 2^E}$

Definition 7 (Trim) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. We define the trim operation as $trim(G) = CoAc(Ac(G)) = Ac(CoAc(G))$

Definition 8 (Complement) Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. We define the complement automaton of $G$ as the automaton $comp(G)$ such that $L_m(comp(G)) = E^* \setminus L(G)$

Definition 9 (Product Composition) Let $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{1m})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{2m})$ be two DFA. The product composition of $G_1$ and $G_2$ is the DFA $G_1 \times G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_1 \times \{x_{01}, x_{02}\}, (X_{1m} \times X_{2m}))$ where $f((x_1, x_2), e) = \begin{cases} \{(f_1(x_1), f_2(x_2)) \text{ if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \} & \text{undefined otherwise} \\ \text{undefined} & \text{undefined otherwise} \end{cases}$

Definition 10 (Parallel Composition) Let $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{1m})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{2m})$ be two DFA. The parallel composition of $G_1$ and $G_2$ is the DFA $G_1 \parallel G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\parallel 2}(x_{01}, x_{02}), (X_{1m} \times X_{2m}))$ where $f((x_1, x_2), e) = \begin{cases} \{(f_1(x_1), f_2(x_2)) \text{ if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \} & \text{undefined otherwise} \\ \{(f_1(x_1), x_2) \text{ if } e \in \Gamma_1(x_1) \cap E_2 \} & \text{undefined otherwise} \\ \{(x_1, f_2(x_2)) \text{ if } e \in \Gamma_2(x_2) \cap E_1 \} & \text{undefined otherwise} \end{cases}$

Definition 11 (Observer Automaton) Let $N = (X, E \cup \{\varepsilon\}, f, \Gamma, X_0, X_m)$ be an $\varepsilon$-NFA. The observer automaton is the DFA $G$ such that $L(G) = L(N)$ and $L_m(G) = L_m(N)$
Next, we state some analysis problems that can be solved using these functions and explain how to find the solution. We also state how to synthesize a supervisor.

- **Safety:** Reachability from \( x \) of an undesired or unsafe state \( y \): Take the \( Ac \) operation, with \( x \) declared as the initial state and look for state \( y \) in the result.

- **Presence of certain undesirable strings or sub-strings in the generated language:** Try to execute the substring from all the accessible states in the automaton.

- **Inclusion of the generated language \( A \) in a legal or admissible language \( B \):** testing \( A \subseteq B \) is equivalent to testing \( A \cap B^c = \emptyset \). Hence, we can test if \( L(G \times G^c) = \emptyset \), where \( L(G) = A \) and \( L(G^c) = B \).

- **Blocking:** \( G \) is blocking if and only if the Co\( Ac \) operation over \( Ac(G) \) deletes one or more states.

- **State Estimation:** Let \( G = (X, E, \ell, \Gamma, x_0, X_m) \) be a DFA, where \( E = E_o \cup E_uo \) (\( E_uo \) is the set of unobservable events). We perform state estimation by building the observer automaton of the \( \epsilon \)-NFA \( N = (X, E_o \cup \{ \}, f, \Gamma, x_0, X_m) \), where, for all \( e \in E_uo \), we replace \( e \) by \( \epsilon \).

- **Diagnosable:** Let \( G = (X, E, f, \ell, \Gamma, x_0, X_m) \) be a DFA, where \( E = E_o \cup E_uo \) and let \( e \in E_uo \). We are interested to determine if \( e \) could have occurred or has occurred with certainty. We build the diagnoser automaton, which is a modified observer, where each state is labeled only with "Y"s if \( e \) has occurred, only with "N"s if \( e \) has not occurred and with both "Y"s and "N"s if there is no certainty.

- **Supervisor Synthesis:** Let \( G = (X, E, f, \ell, \Gamma, x_0, X_m) \) and \( H_{spec} = (X', E', f', \Gamma', x_0', X_m') \) be two DFA, where \( E' \subseteq E \). We synthesize the supervisor \( S \) by calculating \( S = G \times H_{spec} \) or \( S = G \parallel H_{spec} \) as required by the supervision problem.

### III. The Matlab Data Structure

In this section, we define the data structure that is used to codify an FSA. An FSA is a Matlab struct, where (assuming the variable name is \( aut \)):

- **\( aut.S \) = 1 : n, where \( n \) is the number of states of the automaton is the set of states**
- **\( aut.S0 = i \), 1 \( \leq i \leq n \) is the initial state**
- **\( aut.F = [i \ j \ k...] \subseteq aut.S \) is the set of final states**
- **\( aut.E = \{p1', \ p2', ... \} \) set of transitions**
- **\( aut.U = \{pi', \ pj', \ pk' \} \subseteq aut.E \) set of unobserved events**
- **\( aut.trans = cell(n) \) where \( cell(i,j) = \{p'k', \ p'i', \ p'm' \} \) is the set of \( E \)**
- **\( aut.labels \) is an output of some functions. It is only used to help the understanding of how the operation was done, no functions needs it as input.**

### Example 1: We present a simple automaton:
- **\( aut.S = 1 \times 3 \)**
- **\( aut.S0 = 1 \)**
- **\( aut.F = [1 \ 3] \)**
- **\( aut.E = \{ 'a', 'b', 'g' \} \)**
- **\( aut.U = \{ 'g' \} \)**
- **\( aut.trans = cell(3) \)**

### IV. Functions Description

In this Chapter, we describe the implemented functions. First we describe the implementations of automata operations. Let \( aut1 \) and \( aut2 \) be two Matlab automata structs:

- **\( accessible.part(aut1,0) = Ac(aut1) \)**
- **\( accessible.part(aut1,1) = \{ \text{set of states in } aut1 \text{ that are not accessible} \} \)**
- **\( co.accessible.part(aut1,0) = CoAc(aut1) \)**
- **\( co.accessible.part(aut1,1) = \{ \text{set of states in } aut1 \text{ that are not coaccessible} \} \)**
- **\( trim(aut1) = trim(aut1) \)**
- **\( aut.complement(aut1) = comp(aut1) \)**
- **\( product.composition(aut1,aut2) = aut1 \parallel aut2 \)**

- **In this case, the \( aut.labels \) in the output automaton \( aut \) is such that \( aut.labels(i) \) are the corresponding states in \( aut1 \) and \( aut2 \) to state \( i \) in the output automaton**
- **\( parallel.composition(aut1,aut2) = aut1 \parallel aut2 \)**

- **In this case, the \( aut.labels \) in the output automaton \( aut \) is such that \( aut.labels(i) \) are the corresponding states in \( aut1 \) and \( aut2 \) to state \( i \) in the output automaton**

Now, we describe the functions used to perform automata analysis and supervision:

- **\( is.reachable(aut1,i,j) = 1 \text{ if and only if state } j \text{ is reachable fro state } i \text{ in } \text{FSA aut1} \)**
- **\( is.substring(string,aut1) = 1 \text{ if and only if string can be a substring of a generated string of aut1. We represent string as } \{p'i', \ p'j', \ p'k', ... \} \subseteq aut1.E \)**
- **\( is.language.included(aut1,aut2) = 1 \text{ if and only if } L(aut1) \subseteq L(aut2) \)**
- **\( is.blocking(aut1) = 1 \text{ if and only if aut1 is blocking} \)**
- **\( observer(aut1) \) outputs the state estimation automaton for aut1 \)**

- **In this case, the \( aut.labels \) in the output automaton \( aut \) is such that \( aut.labels(i) \) is the set of possible states \( aut1 \) can be in when \( aut \) is in state \( i \).**
- **\( diagnoser(aut1,e) \) outputs the diagnostics automaton for \( aut1 \) and event \( e \)**

In this case, the \( aut.labels \) in the output automaton \( aut \) is such that \( aut.labels(i) \) is the set of possible states \( aut1 \) can be in when \( aut \) is in state \( i \), each one labeled with 'Y' if \( e \) is included with certainty, 'N' if \( e \) has not occurred with certainty and both 'Y' and 'N' if one cannot be sure.

- **\( build.supervisor(aut1,aut2,0) \) builds the supervisor \( aut1 \parallel aut2 \), where \( aut1 \) is the plant and \( aut2 \) is the specification**
- **\( build.supervisor(aut1,aut2,1) \) builds the supervisor \( aut1 \parallel aut2 \), where \( aut1 \) is the plant and \( aut2 \) is the specification**

The file examples.m, available in the function library has examples of representation of automata and the application
of the implemented functions. If any doubts subsist after reading this text, one should examine those examples.

REFERENCES