I. Introduction

Reactive systems are computer systems that maintain a continuous interaction with their environment (Finkbeiner). They are often used to model and verify real-time systems, such as embedded systems, production lines, video-game consoles, etc2. Reactive systems can be described using extensions of classical automata theory, such as timed automata, hybrid automata, probabilistic automata, etc3. These extensions allow us to capture the temporal, continuous, and probabilistic aspects of reactive systems. Reactive systems are often so complex that their behavior cannot be neatly characterized. Only certain properties can be characterized (The modal μ-calculus, 2013)

II. Previous Work

In the earlier sections of this project, we converted each process of Big Data Analytics into a Labelled Transition System, evaluated their traces, simple trace equivalence, completed trace equivalence, and weak bi-simulation for each LTS. We then converted each LTS into a Deterministic Labelled Transition System and assessed their strong and weak bi-simulation. Further, we introduced inert in the LTS and evaluated strong bi-simulation. In this section of the project, we will proceed to convert each LTS into a Behavioral Benchmark Lemma and create a reactive system. Finally, we will code this reactive system into Open MPI/MPI.

III. LTS to BBL

DLTS-1

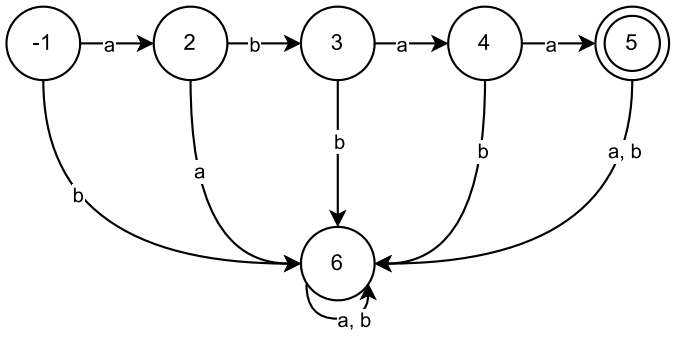

Figure 1: Finite Deterministic Labelled Transition System

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6

(b, aa, abb, abab, abaaa, abaab){a,b}*

After combination of all RE's we get one RE {a,b}*

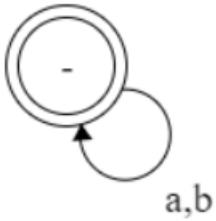

Figure 2: Finite State Automata of {a,b}*

DLTS-2

The graph of DLTS-2 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6

(b, aa, abb, abab, abaaa, abaab){a,b}*

After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-3

The graph of DLTS-3 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}

1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-4

The graph of DLTS-4 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-6

The graph of DLTS-6 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-8

The graph of DLTS-8 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-9

The graph of DLTS-9 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-11

The graph of DLTS-11 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-13

The graph of DLTS-13 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-15

The graph of DLTS-15 is show in Figure 1: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
5	{abaa}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 6
(b, aa, abb, abab, abaaa, abaab){a,b}*
After conversion of all RE's we get one RE {a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-5

The graph of DLTS-5 is show in Figure 3: Finite Deterministic Labelled Transition System

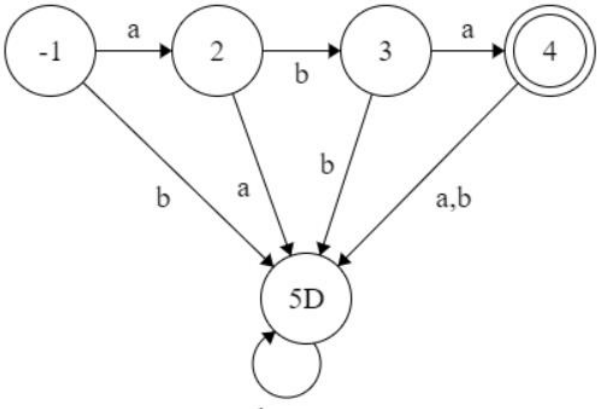

Figure 3: Finite Deterministic Labelled Transition System

Regular Expressions:

Final State's Node No.	Regular Expression
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 5
(b, aa, abb, abab, abaa){a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-7

The graph of DLTS-7 is show in Figure 3: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 5
(b, aa, abb, abab, abaa){a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-10

The graph of DLTS-10 is show in Figure 3: Finite Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 5
(b, aa, abb, abab, abaa){a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-12

The graph of DLTS-12 is shown in Figure 4: Deterministic Labelled Transition System

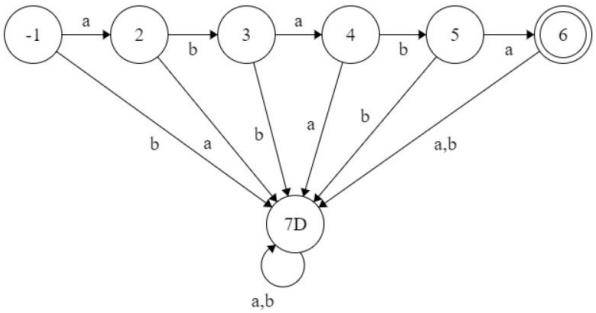

Figure 4: Deterministic Labelled Transition System

Regular Expressions:

Final State's Node No.	Regular Expression
6	{ababa}
5	{abab}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 5
(b, aa, abb, abaa, ababb, ababaa, ababab){a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

DLTS-14

The graph of DLTS-14 is shown in Figure 4: Deterministic Labelled Transition System.

Regular Expressions:

Final State's Node No.	Regular Expression
6	{ababa}
5	{abab}
4	{aba}
3	{ab}
2	{a}
1	{π}

When Final State's Node is 5
(b, aa, abb, abaa, ababb, ababaa, ababab){a,b}*
After combining all RE's we get one RE {a,b}* as shown in Figure 2: Finite State Automata of {a,b}*

After converting all DLTS's to BBL, we convert all BBL's to a reactive system. Since all BBL's are (a,b)*, so the reactive system is (a,b)*.

IV. Coding Reactive System (MPI)

```
#include <iostream>
#include <mpi.h>
using namespace std;
bool containsOnlyAB(std::string str) {
   if (str[0] == ' ' && str.length() > 1) {
      return false;
   }
   for (int i = 1; i < str.length(); i++) {
      char c = str[i];
      if (c != 'a' && c != 'b') {
         return false;
      }
   }
   return true;
}

int main()
{
        MPI_Init(NULL, NULL);

        string lex = "abaaabaababababa";

   bool res = containsOnlyAB(lex);
   if (res == true) {
      cout << "Accepted by (a,b)* ";
   }
   else {
      cout << "Not Accepted by (a,b)*";
   }

        MPI_Finalize();
        return 0;
}
```

V. Reactive System in Open MPI

Edrlib.h

```
#ifdef __cplusplus
#define EXPORT extern "C" __declspec (dllexport)
#else
#define EXPORT __declspec (dllexport)
#endif
EXPORT TCHAR lex[] = TEXT("abababab");
```

```
EXPORT BOOL CALLBACK EdrCenterTextA(HDC, PRECT, PCSTR);
EXPORT BOOL CALLBACK EdrCenterTextW(HDC, PRECT, PCSTR);
#ifdef UNICODE
#define EdrCenterText EdrCenterTextW
#else
#define EdrCenterText EdrCenterTextA
#endif
```

Edrlib.c

```
#include <windows.h>
#include "EDRLIB.h"
int WINAPI DllMain(HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved) {
	return TRUE;
}
EXPORT BOOL CALLBACK EdrCenterTextA(HDC hdc, PRECT prc, PCSTR pString) {
  int  iLength;
  SIZE size;
  iLength = lstrlenA(pString);
  GetTextExtentPoint32A(hdc, pString, iLength, &size);
  return TextOutA(hdc, (prc->right - prc->left - size.cx) / 2,
    (prc->bottom - prc->top - size.cy) / 2,
    pString, iLength);
}
EXPORT BOOL CALLBACK EdrCenterTextW(HDC hdc, PRECT prc, PCWSTR pString) {
  int  iLength;
  SIZE size;
  iLength = lstrlenW(pString);
  GetTextExtentPoint32W(hdc, pString, iLength, &size);
  return TextOutW(hdc, (prc->right - prc->left - size.cx) / 2,
    (prc->bottom - prc->top - size.cy) / 2,
    pString, iLength);
}
```

Edrtest.c

```
#include <windows.h>
#include "EDRLIB.h"
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow) {
	static TCHAR szAppName[] = TEXT("StrProg");
	HWND hwnd;
	MSG msg;
	WNDCLASS wndclass;
	wndclass.style = CS_HREDRAW | CS_VREDRAW;
	wndclass.lpfnWndProc = WndProc;
	wndclass.cbClsExtra = 0;
	wndclass.cbWndExtra = 0;
	wndclass.hInstance = hInstance;
	wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
	wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
	wndclass.lpszMenuName = NULL;
	wndclass.lpszClassName = szAppName;
	if (!RegisterClass(&wndclass)) {
		MessageBox(NULL, TEXT("This Program requires Windows NT!"), szAppName, MB_ICONERROR);
		return 0;
	}
	hwnd = CreateWindow(szAppName, TEXT("DLL Demonstration Program"), WS_OVERLAPPED, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance, NULL);
	ShowWindow(hwnd, iCmdShow);
	UpdateWindow(hwnd);
	while (GetMessage(&msg, NULL, 0, 0)) {
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return msg.wParam;
}
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {
	HDC hdc;
	PAINTSTRUCT ps;
	bool validLex = true;
	RECT rect;
	switch (message) {
	case WM_PAINT:
		hdc = BeginPaint(hwnd, &ps);
		GetClientRect(hwnd, &rect);
		// isValid = true;
		for (int i = 0; i < strlen(lex); i++) {
			if (strncmp(lex, " ", 1) == 0) {
				break;
			}
			else if (lex[i] != 'a' && lex[i] != 'b') {
```

```
                                validLex = false;
                                break;
                        }
                }
                if (validLex) {
                        EdrCenterText(hdc, &rect,
                        TEXT("Your input is valid."));
                }
                else
                {
                        EdrCenterText(hdc, &rect,
                                TEXT("Your input is not
valid."));
                }
                EndPaint(hwnd, &ps);
                return 0;
        case WM_DESTROY:
                PostQuitMessage(0);
                return 0;
        }
        return      DefWindowProc(hwnd,      message,
wParam, lParam);
}
```