APPENDIX

A. Taylor expansion in QSS

An example of using Taylor polynomial for approximating a transcendental function is shown below. Consider an ODE representing the velocity of a moving robot:

$$\dot{y} = \sin(\theta) \times v_1$$

where, y is the vertical position variable, θ is the orientation angle variable, and v_1 is a constant representing the velocity. Thus, $\dot{y}$ simply represents the vertical component of the robot's velocity. Let us denote $\theta(T_n)$ as the value of θ at time T_n. Then, the QSS approximation from the point T_n is

$$q_y(t) = \left| \int (\sin(\theta(T_n)) \times v_1)\ dt - q_y(T_n) \right| = \Delta q_y$$

where, $q_y(t)$ is the QSS approximation of $y(t)$, and $q_y(T_n) = y(T_n)$ is the initial value of y to this integral. Intuitively, the equation is the integral (of $\dot{y}$) minus its initial value at time T_n. By fundamental theorem of calculus, this simply indicates the change in the value, denoted as Δq_y. The trigonometric function part of the equation can be replaced by a Taylor polynomial. Finally, we obtain

$$q_y(t) = \left| \int ((\theta(T_n) + \ldots + (\theta(T_n)^9)/9!) \times v_1) d\tau - q_y(T_n) \right|$$

Thus, the Taylor polynomial can be used during the QSS approximation simply by replacing the transcendental functions inside the ODE with its respective Taylor series representation.

B. Stateflow model structure

A Stateflow is essentially a state machine called a *chart* (in Simulink terminolog), which is encapsulated by a *block* for the I/O interface. The *block* encapsulation is composed of:

- Name (String): name of the block instance.
- Inport (set of `Port`): set of input ports.
- Outport (set of `Port`): set of output ports.
- Machine (`Chart`): a finite state machine.

where `Port` is a two tuple: (Name (String), Index (int)). Note that, in the Stateflow model, there can be multiple state machines with the logical relationship of AND or OR operation. Such multiple state machines can be syntactically composed into a single state machine, as in statecharts [**?**]. Therefore, without loss of the generality, we consider that each Stateflow contains only one `Chart`. A `Chart` is composed of:

- Variables (set of `Data`): set of variables. `Data` is:
  - Name (String): name of the variable.
  - Type (String): the data type such as `int`.
  - Scope (String): it can be 1) input, 2) output, 3) local continuous, and 4) local discrete.
  - Initial (String): the initial value stored as a string.
- States (set of `State`): set of states. `State` contains a String which can be divided into separate Strings:
  - Name (String): name of the state.
  - Entry actions (set of Strings): set of mathematical equations. Each equation is stored as a string.
  - During actions (set of Strings): set of mathematical equations.
  - Exit actions (Set of Strings): set of mathematical equations.
- Transitions (set of `Transition`): set of state transitions. `Transition` consists of:
  - Source (`State`): the source state.
  - Dest (`State`): the destination state.
  - Condition (set of Strings): set of mathematical expressions. Each is stored as a string.
  - Assignment (set of Strings): set of equations.

Finally, the connection between Stateflow models is defined using a `Line`, which consists of:

- Source (String): name of the source block.
- Dest (String): name of the destination block.
- SourcePort (`Port`): the output port of the source block that this line is starting.
- DestPort (`Port`): the input port of the destination block

Input : Stateflow block $\mathcal{B} = \langle Name, Inport, Outport, Machine \rangle$
Output: Quantized State Hybrid Input Output Automaton (QSHIOA) $\mathcal{H}_q = \langle L, X, X_q, V, O, Init, f, f_q, h, Inv, E, G, R \rangle$

```
/* Extract the chart information                                   */
⟨Variables, States, Transitions⟩ ← readChart(B.Machine);
/* Translating the variables                                       */
foreach var ∈ Variables do
    if var.Scope == "Input" then
        V.append(var.Name);
    else if var.Scope == "Output" then
        O.append(var.Name);
        Init_O.append(var.Initial) // Initial value of O
    else
        X.append(var.Name);
        X_q.append(var.Name);
        Init_X.append(var.Initial) // Initial value of X
    end
end
/* Translating the states                                          */
foreach s ∈ States do
    L.append(s.Name); // Every state is a location
    Inv(s.Name) ← True // Invariant is always True
    foreach β ∈ s.During do
        if β.leftTermContains("_dot") then
            // Add the string equation as an ODE
            f(s.Name, V, X).addODE(β);
            f_q(s.Name, V, X).addQSS(β);
        else
            h(s.Name, X).addUpdate(β);
        end
    end
end
/* Translating the transitions                                     */
foreach tran ∈ Transitions do
    if tran.Source == Null then
        // This transition describes the
           initialization
        Init ← (tran.Dest, Init_X, Init_O);
    else
        e ← (tran.Source.Name, tran.Dest.Name);
        E.append(e);
        G(e) ← setGuard(tran.Condition);
        R(e, X, V) ← setReset(tran.Assignment);
    end
end
H_q ← ⟨L, X, X_q, V, O, Init, f, f_q, h, Inv, E, G, R⟩;
return H_q
```

Algorithm 1: Pseudo-code for syntactic translation of a Stateflow model into a QSHIOA.

that this line is ending.

These are the fundamental components of the Stateflow model, and there are also optional components such as "Matlab function", "Simulink function", and "truth table". These optional components are not considered in this paper. Note that the Entry actions and the Exit actions in `State` are essenatilly not different from the Assignment in `Transition`. For example, the Entry actions are executed when entering the state. This is exactly same as executing the Assignment of the transition leading to that state. Likewise, the Exit actions are identical to the Assignment of the transition exiting the state. Therefore, without the loss of generality, we demonstrate our approach by assuming the Entry and Exit actions are included as the Assignment on the transitions.

C. Compiling a single Stateflow chart into a QSHIOA

The syntactic translation to QSHIOA is described in Algorithm 1. The input to this algorithm is a Stateflow block, and the output is a QSHIOA. At the start, the components of the `Chart` in the Stateflow block are extracted (line 1). The loop on line 2 iterates over the variables of the Stateflow chart, and adds V, O, and X components of the generated QSHIOA depending on their scope. Note that both the local continuous and local discrete variables in the Stateflow chart are added to X. If the converted variable's scope is either output or local, the initial values are also stored in $Init_O$ and $Init_X$. The second loop on Line 14 converts each state into a location. Since the Stateflow model does not specify any constraints for the invariants, they are set `True` for all locations in the generated QSHIOA. The During actions in a statflow char state are mapped into either f or h in the generated QSHIOA depending on the function leftTermContains("_dot"). This function reads the left hand side variable from a Stateflow char equation, and returns `True` if its name includes the "_dot" keyword. This is a special keyword used in Stateflow to indicate the derivatives of continuous variables. If the left hand side of the equation is a derivative, then the equation should be an Ordinary Differential Equation (ODE), hence the equation is added to f, otherwise it is added to h in the generated QSHIOA. Lastly, the third loop on Line 26 converts the transitions into edges. There is a special transition, which has no source state, this transition is used to indicate the initial state in the Stateflow model. Hence, it is possible to set $Init$ with the initial values of $Init_O$ and $Init_X$. The other transitions, which have both the source and the destination states are mapped into an edge, and appended to the set of edges E. The "Condition" component of the Stateflow chart transition is converted into guards, and the "Assignment" is mapped into the reset relations in the generated QSHIOA. Finally, the created QSHIOA components form a QSHIOA instance, and it is returned by the algorithm. In a network of QSHIOAs, same named inputs and outputs are connected.