Timed symbolic transition system - Reference system model

The SPTG framework, being an extension of the DIVERSITY symbolic execution platform, uses its entry language to express models of timed symbolic transition systems. This language, called **XLIA** (**eXecutable Language for Interaction and Architecture**), is designed for specifying the behavior of component-based systems.

In XLIA, components are referred to as machines. These machines are communicating, hierarchical, and heterogeneous, and their evaluation semantics can be customized to support different analysis or execution contexts.

For timed symbolic transition systems, a single machine is typically used to represent the system under study, which interacts with its environment through well-defined communication interfaces.

Encoding of the timed symbolic transition system in XLIA

The automaton (\mathbb{G}) from Fig. \ref{fig:tiosts_symbex_tree} is encoded in the **XLIA** input language of the **DIVERSITY** symbolic execution platform as follows:

```
timed system Example01_Dummy {

@composite:
    statemachine Example01_Dummy {
    @public:
        port input In( urational );
        port output Out( urational );
        port output Done;
    @private:
        var urational sum;
        var urational x;
        var clock urational cl;
    @region:

        state<start> q0 {
            @init{
                sum := 0;
            }
            transition  tr1 --> q1 {
                input In( x );
                guard (1 <= x <= 10 );
                sum := sum+x;
                cl := 0;
            }
            transition  tr4 --> q2 {
                guard (sum >= 15 );
                output Done;
            }
        }
```

```
        state q1 {
            transition tr2 --> q0 {
                guard( x <= 5 && cl == 42-x );
                output Out( 0 );
            }
            transition tr3 --> q0 {
                guard( x > 5 && cl == 42-x );
                output Out( x );
            }
        }

        state q2;

    @com:
        connect< env >{
            input In;
            output Out;
            output Done;
        }
     }
}
```

1. General Structure

The **XLIA model** (entry language of the **DIVERSITY** symbolic execution platform) encodes the automaton by explicitly separating the **static part** (declarations of variables, clocks, and communication ports) from the **behavioral part** (states, transitions, and synchronization).
The `timed system` construct defines the **whole system**, while the nested `statemachine` block defines the actual automaton.

---

2. Static Part

The static part declares:

- **Ports** (inputs/outputs), which correspond to the external interactions of the automaton.
- **Variables** (data and clocks), which capture internal state and timing information.

From the XLIA model:

```
@public:
    port input In( urational );
    port output Out( urational );
    port output Done;
@private:
    var urational sum;
    var urational x;
    var clock urational cl;
```

Mapping to the automaton

XLIA element	Meaning	Automaton equivalent
`port input In(urational)`	**Input port** for receiving a rational value x	Incoming transition label $\text{In}(x)$
`port output Out(urational)`	**Output port** for sending a rational value	Outgoing transition label $\text{Out}(\dots)$
`port output Done`	**Output event** without data	Output transition labeled Done
`var urational sum`	**Data variable** tracking accumulated input	Automaton variable (used in **guards/actions**)
`var urational x`	**Data variable** holding the latest input	**Variable bound** by input message
`var clock urational c1`	**Clock variable** (measures elapsed time since last reset)	Automaton clock for **timed constraints**

Thus, the static part declares the **interface** and **internal state space** of the automaton.

Behavioral Part

The behavioral description is under `@region`, where states and transitions are defined.

```
@region:
    state<start> q0 { ... }
    state q1 { ... }
    state q2;
```

States

q0: Initial state (`<start>`), where sum is initialized.

q1: Intermediate state reached after an input.

q2: Final or terminal state when the condition `sum >= 15` holds.

Initialisation

```
@init {
    sum := 0;
}
```

Transitions from q0

```
transition tr1 --> q1 {
    input In( x );
    guard (1 <= x <= 10 );
    sum := sum+x;
    cl := 0;
}
transition  tr4 --> q2 {
    guard (sum >= 15 );
    output Done;
}
```

Transition Mapping

Transition	Automaton equivalent	Description
tr1	$$q_0 \xrightarrow{In?x, 1 \le x \le 10, \text{sum} := \text{sum} + x, c1 := 0} q_1$$	Receives input x, adds to sum, resets the clock $c1$.
tr4	$$q_0 \xrightarrow{\text{Done}, \text{sum} \ge 15} q_2$$	Produces Done output when accumulated $\text{sum} \ge 15$.

More on XLIA subset to encode timed symbolic transition system

```
// ============================================================
// Prologue - Header
// ============================================================
@xlia< system , 1.0 >:

// ============================================================
// System Definition
// ============================================================
timed system S {

    // ========================================================
    // Composite Part: State Machine Definition
    // ========================================================
    @composite:
    statemachine SM {
        @public:

            // -----------------------------------------------
            // Declaration of Ports
            // -----------------------------------------------
            port input  In;
            port output Done;
            port input  In1( urational );
            port input  In2( integer );
            port output Out( urational );

            // Declaration of N-ary Ports
```

```
        port input  In3( bool, integer, rational );
        port output Out2( integer, bool );

        // ------------------------------------------------
        // Declaration of Constants
        // ------------------------------------------------
        const integer N = 42;

    @private:

        // ------------------------------------------------
        // Declaration of Variables
        // ------------------------------------------------
        var urational sum;
        var urational x;
        var urational y;
        var integer   z;
        var bool      flag;
        var integer   fee;

        // ------------------------------------------------
        // Declaration of Clocks
        // ------------------------------------------------
        var clock urational cl;
        var clock urational cl2;

    // ====================================================
    // Behavioral Description: States and Transitions
    // ====================================================
    @region:

        // ------------------------------------------------
        // Initial State
        // ------------------------------------------------
        state<start> q0 {
            @init {
                sum  := 0;
                flag := false;
                guard( fee > 0 );
            }

            transition tr1 --> q1 {
                input In1( x );
                guard ( 1 <= x <= 10 );
                sum := sum + x;
                y   := sum;
                cl  := 0;
            }

            transition tr2 --> q1 {
                input In( x );
                guard ( 10 < x && x < N );
                {|,|
                    sum := sum + x;
```

```
                        y   := sum; // y receives the pre-increment sum value
                    }
                    cl2 := 0;
                }
            }

            // ---------------------------------------------
            // Secondary State
            // ---------------------------------------------
            state q1 {
                transition tr3 --> q0 {
                    guard( x <= 10 && cl == N - x );
                    output Out( sum - 1 );
                }

                transition tr4 --> q0 {
                    guard( x > 10 );
                    guard( cl <= 5 );
                    output Out2( fee, flag );
                    flag := true;
                    cl2 := 0;
                }

                transition tr5 --> q2 {
                    guard( sum >= 15 && cl2 <= 1 );
                    output Done;
                    cl2 := 0;
                }
            }

            // ---------------------------------------------
            // Terminal State
            // ---------------------------------------------
            state q2;

        // =====================================================
        // Communication Part: Port Connections
        // =====================================================
        @com:
        connect< env > {
            input  In;
            input  In1;
            input  In2;
            input  In3;
            output Done;
            output Out;
            output Out2;
        }
    }
}
```