Formal Composition for

Time-Triggered Systems

John Rushby and Ashish Tiwari

Rushby,Tiwari@csl.sri.com

Computer Science Laboratory
SRI International
Menlo Park CA 94025
Objective: Specific

Stateflow/Simulink + TT Tools

- Analyze models
- Verify transformations
- Verify TT services
- To yield assurance for final system

Partitioning
Safety/FT

RTW/Beacon

TTA
Analysis Techniques

Effort

Assurance for system
Analysis Techniques

Simple Typechecking
Static Analysis

Assurance for system

Effort

Rushby, Tiwari, SR I
Formal Composition for Time-Triggered Systems: 4
Analysis Techniques

Assurance for system

Effort

Invariant Checking / Typechecking

Rushby, Tiwari, SR I

Formal Composition for Time-Triggered Systems: 5
Analysis Techniques

Assurance for system

Effort

Invariant Generation
Reachability
Abstraction

Rushby, Tiwari, SR I

Formal Composition for Time-Trigged Systems: 6
Analysis Techniques

Assurance for system

Exhaustive State Space Exploration

Effort

Rushby, Tiwari, SR I

Formal Composition for Time-Triggered Systems: 7
Analysis Techniques

- Local Analysis
- Global Analysis

Assurance for system

Effort

Rushby, Tiwari, SR I

Formal Composition for Time-Triggered Systems: 8
SAL models transition systems and supports

- Transitions: Definitions and guarded commands
- Modules: input, output, local, global variables
- Composition of modules

Supported by theorem-provers, model-checkers, and program analyzers
SAL: Tool Suite

- Simple typechecking
- Symbolic Simulation
- Invariant Checking
- Invariant Generation
- Abstraction

All of these tools work on modules. Module could represent individual components of the system, or the full system.
Benefit to Development Process

- Early detection of errors: models can be typechecked and verified in the design phase
- Reduction in the development cycle time
- Provably correct transformation and mapping onto target architecture
- Extra information generated in the verification process may be used for efficient code generation
Verification tool—

**Input**: Stateflow-Simulink, or SAL language

**Intermediate Representation**: SAL (XML)

**Output**: SAL Theorems

We have a translator from Stateflow-Simulink abstract (logical) syntax to SAL.
SAL is designed for easy integration with other verification tools.

- SAL concrete syntax is XML based.
- SAL analysis capabilities comprise of a collection of independent tools.
- Different tools communicate through XML and a tool bus management software is under development.
The ETC Example in SAL

ETC : CONTEXT =
BEGIN
    Driver : MODULE = . . .
    Actuator : MODULE = . . .
    Controller : MODULE = . . .
    HumanController : MODULE = . . .
    Plant : MODULE = . . .
END
Driver : MODULE =
    BEGIN
        INPUT duty : REAL
        LOCAL lduty, cnt : REAL
        LOCAL mode : BOOLEAN
        OUTPUT pwm : REAL
        INITIALIZATION . . .
        TRANSITIONS . . .
        END;

Given duty s.t. 0 < duty < 100, output a pwm signal.
ETC: Driver Specification

TRANSITIONS

\[
\begin{align*}
\text{mode} &= F \land \text{cnt} = 0 \land \text{duty} > 0 \land \text{duty} < 100 \rightarrow \\
& \quad \text{l duty}' = \text{duty}; \quad \text{mode}' = T; \\
& \quad \text{pwm}' = 1; \quad \text{cnt}' = 100 \\
\end{align*}
\]

\[
\begin{align*}
\text{mode} &= T \land \text{cnt} < \text{l duty} \rightarrow \\
& \quad \text{mode}' = F; \quad \text{pwm}' = 0 \\
\end{align*}
\]

\[
\begin{align*}
(\text{mode} = F \land \text{cnt} > 0) \lor (\text{mode} = T \land \text{cnt} \geq \text{l duty}) \rightarrow \\
& \quad \text{cnt}' = \text{cnt} - 1
\end{align*}
\]
Driver: Symbolic Propagation

sal(45): (propagate-up ’ETC ’Driver)
sal(48): (widen ’ETC ’Driver “…”)
   The widening is correct.
sal(49): (propagate-up ’ETC ’Driver)
sal(52): (widen ’ETC ’Driver “…”)
   The widening is correct.
sal(53): (propagate-up ’ETC ’Driver)

The formula

\[(\text{mode} = T \land \text{pwm} = 1 \land 0 < \text{lduty} < 100 \land \text{lduty} - 1 \leq \text{cnt} \leq 100) \lor \]
\[(\text{mode} = F \land \text{pwm} = 0 \land 0 < \text{lduty} < 100 \land -1 < \text{cnt} < \text{lduty})\]

is an invariant.
Driver: Assigning Types

Variable \textit{lduty} can be declared to be of type:

\[ \{x: \text{INT} \mid 0 < x \land x < 100\}. \]

Similarly, variable \textit{cnt} is of type:

\[ \{x: \text{INT} \mid \text{if} \quad \text{mode} \quad \text{then} \quad \text{lduty} - 1 \leq x \leq 100 \]
\[ \text{else} \quad 0 \leq x < \text{lduty} \} \]

Typechecking establishes correctness. Typechecking involves one step of \textit{symbolic simulation}. 

Rushby, Tiwari, SR I, Formal Composition for Time-Triggered Systems: 18
ETC: Actuator

Actuator : MODULE =

BEGIN

  INPUT pwm_state : BOOLEAN
  LOCAL Vc, i : REAL
  OUTPUT Trq_throttle : REAL

  INITIALIZATION . . .
  TRANSITIONS . . .

END;

Actuator outputs \textit{Trq\_throttle} based on the input pwm signal.
ETC: Actuator Specification

TRANSITIONS

\[
\begin{align*}
\text{pwm\_state} &= T \\ Vc' &= Vc + 2/9 \times (24 - i - 2Vc); \\
i' &= i + (1/15) \times (120 - 22i); \\
Trq\_throttle' &= 3/250\times i
\end{align*}
\]

\[
\begin{align*}
\text{pwm\_state} &= F \\ Vc' &= Vc - 2/3 \times i; \\
i' &= i + 2/15 \times (5Vc - 16i); \\
Trq\_throttle' &= 3/250\times i
\end{align*}
\]
ETC: Actuator Analysis

Using the same technique, we can show that when $\text{pwm\_state}$ is TRUE

$$Trq_{\text{throttle}} = \frac{3}{250} \times i \land Vc = \frac{102}{11} \land i = \frac{60}{11}$$

is a stable solution, and when $\text{pwm\_state}$ is FALSE, it is

$$Trq_{\text{throttle}} = \frac{3}{250} \times i \land Vc = 0 \land i = 0.$$
Properties of individual components help in getting an abstract system.

Replace the driver and actuator modules by a simplified module: given duty 0 \( \leq d \leq 1 \), \( Trq\_throttle \) is 0.065 for \( d \)-fraction of the time, and 0 for \( (1-d) \)-fraction of the time.
System : MODULE =
BEGIN
    INPUT desired : REAL
    LOCAL alpha, omega : REAL
    LOCAL mode : BOOLEAN

Discrete transition triggers:

\[
|160*(alpha - desired)| - 3
\]
\[
|40*(alpha - desired)| - 1
\]
omega
\[
(alpha - desired)*30 + omega
\]

...
Each new symbolic state is obtained using

- simulation of current symbolic state
- widening the reached symbolic state

Thus, we have a tool suite for analysis ranging from typechecking to complete verification via invariant generation, abstraction, and model-checking.