Formal Specification and Verification of Distributed Cyber-Physical Systems

Kyungmin Bae[1]

University of Illinois at Urbana-Champaign

December 31, 2013

[1]Collaboration with: Peter Ölveczky (University of Oslo), José Meseguer, Abdullah Al-Nayeem, Joshua Krisiloff, and Lui Sha (UIUC), and Steve Miller and Darren Cofer (Rockwell-Collins)

Distributed Cyber-Physical Systems (DCPS)

- collection of components that control physical entities
- complex interaction of embedded systems and real-time control
- e.g., avionics, automative, medical devices, ...

Distributed Cyber-Physical Systems (DCPS)

- safety-critical systems
- asynchronous communications
- hard real-time constraints
- often virtually synchronous
  - in each period, read input, perform transition, and produce output

Challenges (1)

- Hard to design correctly
  - race conditions
  - clock skews
  - network delays and execution times
- No fault found
  - hard to duplicate a (reported) failure
  - due to distributed nature

Challenges (2)

- Model checking
  - examines all possible behaviors from the initial states
  - provides a counterexample
- Hard to model check
  - real-time
  - state space explosion due to asynchrony

Our Approach

1. Multirate PALS
   - reduces design and verification of a DCPS to its synchronous version
2. Multirate Synchronous AADL
   - makes PALS available in avionics modeling standard AADL
   - formal semantics in (real-time) rewriting logic
3. The MR-SynchAADL tool
   - Eclipse plug-in for Multirate Synchronous AADL

Outline

Multirate PALS (1)

PALS: physically asynchronous logically synchronous

Reduces design/verification of DRTS to its synchronous version

- Relies on asynchronous bounded delay (ABD) network infrastructure
- Assumes underlying clock synchronization (IEEE 1588, etc.)

Multirate PALS (2)

PALS: physically asynchronous logically synchronous

Reduces design/verification of DRTS to its synchronous version

- Multirate PALS gives a transformation $(\mathcal{E}, T, \Gamma) \rightarrow \mathcal{MA}(\mathcal{E}, T, \Gamma)$

$\mathcal{E}$: multi-rate synchronous design
T: a rate function
Γ: bounds on network delay, execution time, and clock skew
$\mathcal{MA}(\mathcal{E}, T, \Gamma)$: the corresponding distributed asynchronous design

Multirate PALS (2)

PALS : physically asynchronous logically synchronous

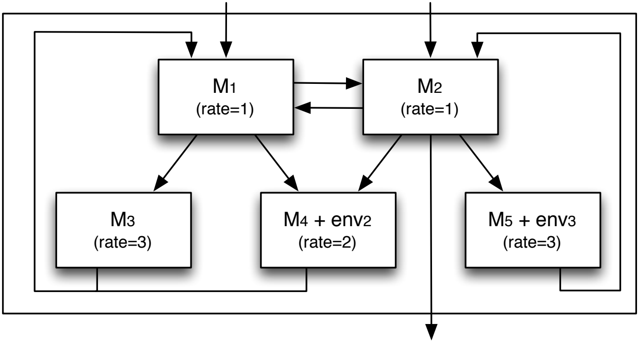
Reduces design/verification of DRTS to its synchronous version

- Multirate PALS gives a transformation $(\mathcal{E}, T, \Gamma) \rightarrow \mathcal{MA}(\mathcal{E}, T, \Gamma)$
- Correct by construction

$$\mathcal{E} \models \varphi \quad \text{if and only if} \quad \mathcal{MA}(\mathcal{E}, T, \Gamma) \models \varphi$$

- Verified formal architectural pattern
  - verification effort amortized over many systems!

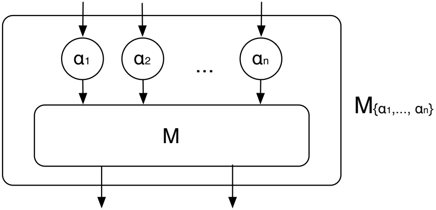
Synchronous composition of typed state machines



- Controller periods multiple of faster periods
- All components must perform in lock-step
  - "slow down" fast components by performing k ($=$ rate) transitions
  - input adaptors transform k-tuples to/from single values

Synchronous Model (2)

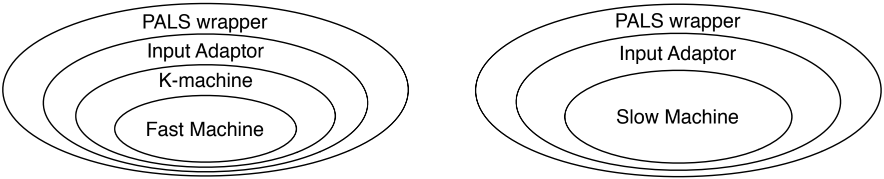
- Fast components perform k "internal transitions" in one step
  - reads/produces k-tuples of inputs/outputs
- Input adaptors transform k-tuples to/from single values



- Transformations/"formal patterns" define synchronous model
  - "k-step decelerated machine"
  - "input adaptor closure machine"

Asynchronous Model (1)

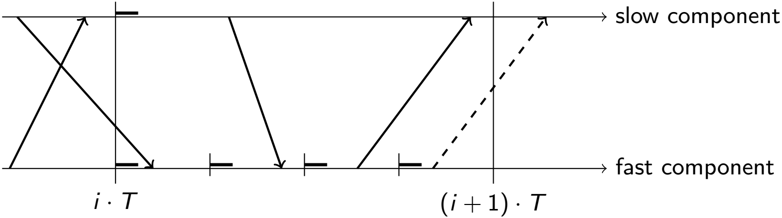
Add "wrappers" around each machine



- input buffers, output buffers, timers
- optimal PALS period: $\mu_{max} + 2 \cdot \epsilon + max(2 \cdot \epsilon - \mu_{min}, \alpha_{max})$
  - clock skew ϵ, execution time α_{max}, and network delay μ_{min}, μ_{max}

Asynchronous Model (2)

- Components perform at different rates



- Assumption: adaptors ignore inputs not received in time

Correctness of Multirate PALS

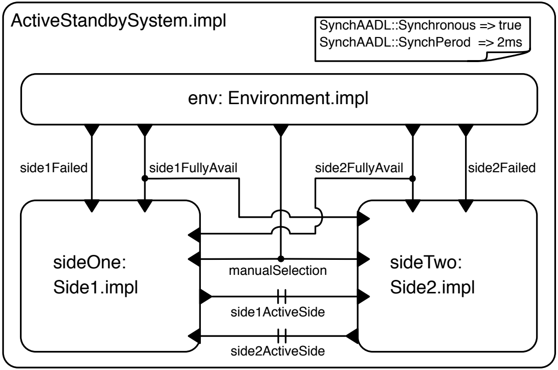
- Stable states of asynchronous models
  - virtually synchronized states
  - PALS wrapper: all input buffers full, all output buffers empty
- Correct by construction

$$\text{synchronous design} \models \varphi$$
$$\text{iff}$$
$$(\text{"stable-state"}) \text{ asynchronous design} \models \varphi$$

Case Study: Active Standby (1)

Integrated modular avionics example

- Which of two cabinets is active?
- Non-active side monitors active sides, failures, and pilot toggle



- By Steven Miller and Darren Cofer at Rockwell-Collins

Case Study: Active Standby (2)

System Requirements

- R_1: Both sides should agree on which side is active (provided neither side has failed, the availability of a side has not changed, and the pilot has not made a manual selection).
- R_2: A side that is not fully available should not be the active side if the other side is fully available (again, provided neither side has failed, the availability of a side has not changed, and the pilot has not made a manual selection).
- R_3: The pilot can always change the active side (except if a side is failed or the availability of a side has changed).
- R_4: If a side is failed the other side should become active.
- R_5: The active side should not change unless the availability of a side changes, the failed status of a side changes, or manual selection is selected by the pilot.

Case Study: Active Standby (3)

- Comparison with the simplest possible asynchronous model

Model	#States	Time
Synch.	185	0.1 s
Asynch. (0)	3047832	1249 s
Asynch. (1)	n/a	n/a

- 10! different message reception ordering in each round

Outline

Goal

Make Multirate PALS methodology and formal verification available to domain-specific modeling

Background: AADL

AADL: Industry standard for embedded systems modeling

- US Army, Honeywell, Airbus, Boeing, Dassault Aviation, EADS, ESA, Rockwell-Collins, Ford, Lockheed Martin, Raytheon, Toyota, U. S. Navy, ...
- Avionics, aerospace, medical devices, robotic, ...
- OSATE: Eclipse plug-ins for AADL

Multirate Synchronous AADL (1)

Goal:

Make Multirate PALS methodology and formal verification available to domain-specific modeling

1. Model synchronous design $\mathcal{E}$ in Multirate Synchronous AADL
2. Verify $\mathcal{E}$ using MR-SynchAADL OSATE plugin

Multirate Synchronous AADL (2)

- Subset of AADL to model synchronous PALS designs
  - identifies AADL models that can be considered as synchronous
  - extended with new annotations: property set `MR_SynchAADL`
  - provides predefined input adaptors
- Focus on behavioral and structuring subset of AADL
  - abstract from hardware and memory, etc.,

Multirate Synchronous AADL (3)

- AADL constructs in subset have the same meaning as before
  - easy to use for AADL modeler
  - same behaviors as in AADL, without the intermediate states introduced by asynchrony
- Formalized in real-time rewriting logic
  - Real-Time Maude: formal analysis tool for real-time systems

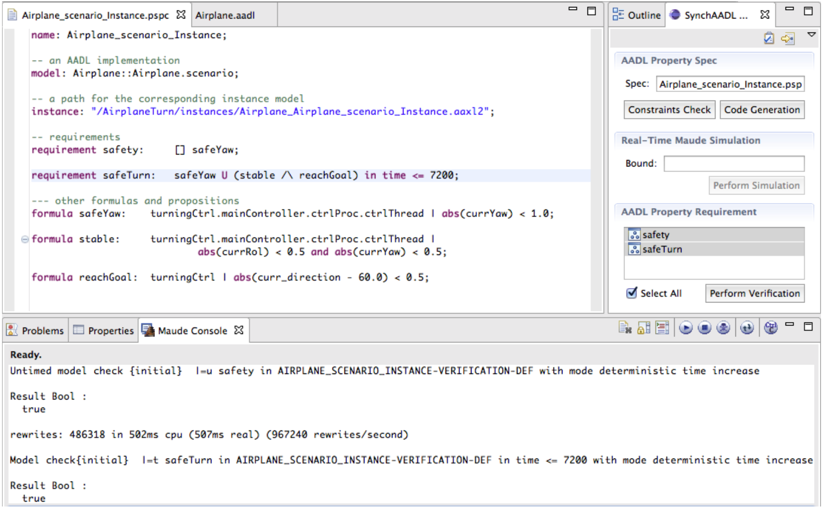
The MR-SynchAADL Tool (1)

- OSATE/Eclipse plug-in for Synchronous AADL
- Real-Time Maude model checking within OSATE
  - checks if given model is valid Multirate Synchronous AADL model
  - automatic synthesis of Real-Time Maude model
- Requirement specification language
  - easy to define system requirements as temporal formulas

The MR-SynchAADL Tool (2)

Modeling tools		Rewriting logic		Model checking (Real-Time Maude)
System design M	$\Longrightarrow$	Rewrite theory $\mathcal{R}_M$	$\Longrightarrow$	LTL time-bounded LTL Timed CTL Metric LTL ...
Property specification $spec$	$\Longrightarrow$	Logic formula φ_{spec}		

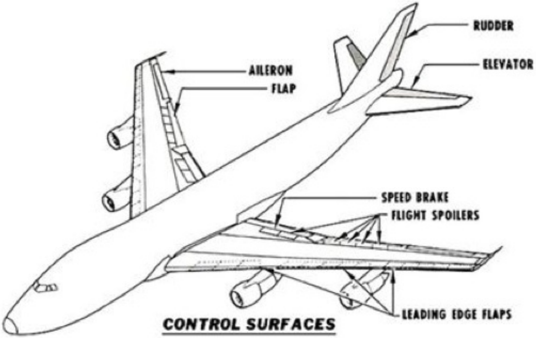
- MR-SynchAADL window in OSATE



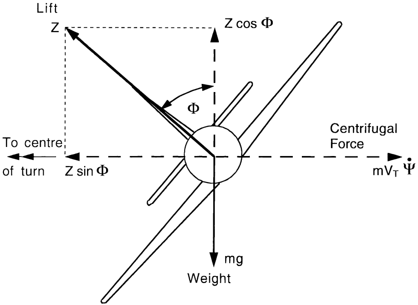
Outline

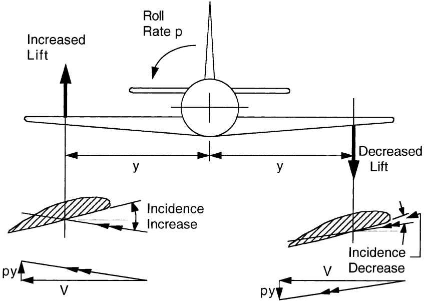
Problem: Design a Controller for Turning an Airplane

- aileron controllers (e.g. 67 Hz) and rudder controllers (e.g. 50 Hz)
- controller must ensure synchronization for turning aircraft



- Move ailerons to roll airplane for a turn
- Turning rate $d\psi = (g/v) * \tan\phi$ (roll angle ϕ)

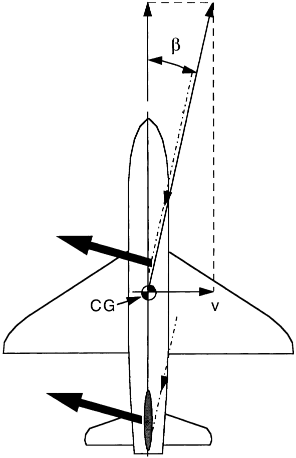




Turning an Airplane (2)

Rolling causes adverse yaw

- sideslip in wrong direction
- use rudder to avoid this
- yaw angle β should always be 0



Turning an Airplane (3)

Roll angle (ϕ) and yaw angle (β):

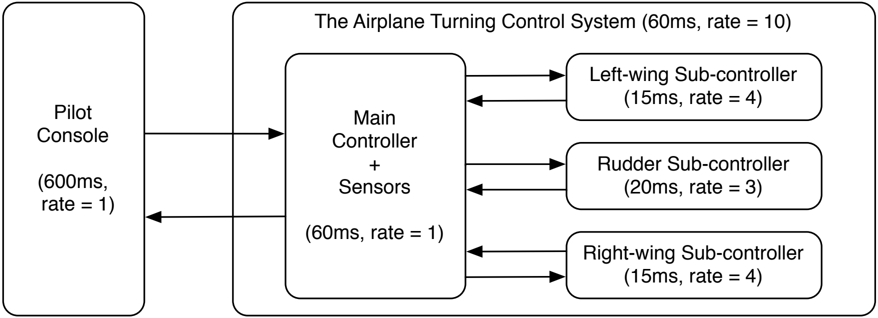
$$d\phi^2 = (Lift\ Right\ -\ Lift\ Left)\,/\,(Weight\ *\ Length\ of\ Wing) \tag{1}$$

$$d\beta^2 = Drag\ Ratio\ *\ (Lift\ Right\ -\ Lift\ Left)\,/\,(Weight\ *\ Length\ Wing) + Lift\ Vertical\,/\,(Weight\ *\ Length\ of\ Aircraft) \tag{2}$$

where

$$Lift = Lift\ constant\ *\ Angle \tag{3}$$

Architecture of the Airplane Turning Control System



Multirate Synchronous AADL Model (1)

```
system TurningController            -- "interface" of the turning controller
  features
    pilot_goal: in data port Base_Types::Float  {MR_SynchAADL::InputAdaptor => "use in first iteration";};
    curr_dr: out data port Base_Types::Float;
end TurningController;

system implementation TurningController.impl
 subcomponents
   mainCtrl: system Maincontroller.impl;    rudderCtrl: system Subcontroller.impl;
   leftCtrl: system Subcontroller.impl;     rightCtrl: system Subcontroller.impl;
 connections
   port leftCtrl.curr_angle   -> mainCtrl.left_angle    {Timing => Delayed;};
   port rightCtrl.curr_angle  -> mainCtrl.right_angle   {Timing => Delayed;};
   port rudderCtrl.curr_angle -> mainCtrl.rudder_angle  {Timing => Delayed;};
   port mainCtrl.left_goal    -> leftCtrl.goal_angle    {Timing => Delayed;};
   port mainCtrl.right_goal   -> rightCtrl.goal_angle   {Timing => Delayed;};
   port mainCtrl.rudder_goal  -> rudderCtrl.goal_angle  {Timing => Delayed;};
   port pilot_goal -> mainCtrl.goal_angle;
   port mainCtrl.curr_dr -> curr_dr;
 properties
   Period => 60 ms;
   Period => 15 ms applies to leftCtrl, rightCtrl;
   Period => 20 ms applies to rudderCtrl;
   Data_Model::Initial_Value => ("1.0") applies to     -- ailerons can move 1° in 15ms
     leftCtrl.ctrlProc.ctrlThread.diffAngle, rightCtrl.ctrlProc.ctrlThread.diffAngle;
   Data_Model::Initial_Value => ("0.5") applies to     -- rudder can move 0.5° in 20ms
     rudderCtrl.ctrlProc.ctrlThread.diffAngle;
   ...
end TurningController.impl;
```

Multirate Synchronous AADL Model (2)

```
system Subcontroller                    -- "interface" of a device controller
  features
    goal_angle: in data port Base_Types::Float
                {MR_SynchAADL::InputAdaptor => "use in first iteration";};
    curr_angle: out data port Base_Types::Float;
end Subcontroller;

thread implementation SubcontrollerThread.impl
  subcomponents
    currAngle : data Base_Types::Float {Data_Model::Initial_Value => ("0.0");};
    goalAngle : data Base_Types::Float {Data_Model::Initial_Value => ("0.0");};
    diffAngle : data Base_Types::Float;
  annex behavior_specification {**
    states
      init: initial complete state;       move, update: state;
    transitions
      init -[on dispatch]-> move;

      move -[abs(goalAngle - currAngle) > diffAngle]-> update {
       if (goalAngle - currAngle >= 0) currAngle := currAngle + diffAngle
       else currAngle := currAngle - diffAngle end if };

      move -[otherwise]-> update {currAngle := goal_angle};

      update -[ ]-> init {
       if (goal_angle'fresh) goalAngle := goal_angle end if; curr_angle := currAngle};
  **};
end SubcontrollerThread.impl;
```

System Requirements

- Key properties:
  - reach desired direction (+ no roll or yaw) in reasonable time
  - yaw angle always close to 0 during turn

- In the requirement specification language:

```
requirement safeTurn: safeYaw U (stable / reachGoal) in time <= 7200;

formula safeYaw:
    turnCtrl.mainCtrl.ctrlProc.ctrlThread | abs(currYaw) < 1.0;
formula stable:
    turnCtrl.mainCtrl.ctrlProc.ctrlThread |
        abs(currRol) < 0.5 and abs(currYaw) < 0.5;
formula reachGoal:
    turnCtrl | abs(curr_dr - 60.0) < 0.5;
```

Synch. vs. Asynch.

- Model checking with different pilot behaviors

Model	Env.	$T \leq 600\ ms$ states	time	$T \leq 1{,}800$ ms states	time(s)	$T \leq 3{,}000$ ms states	time(s)
Sync.	Det.	2	0.14	4	0.16	6	1.18
	3	4	0.16	28	0.33	202	1.55
	5	6	0.16	116	0.89	2,091	14.86
Async.	Det.	6,327	0.76	28,071	2.98	50,139	50.14
	3	17,469	2.26	381,213	73.13	2,547,423	2,884.81
	5	28,611	3.01	1,634,211	938.79	-	$>$ 10 hours

Concluding Remarks

- Multirate PALS
  - reduces design and verification of DRTS to its synchronous version
- Multirate Synchronous AADL
  - modeling synchronous designs in AADL
- MR-SynchAADL
  - simulation and model checking in OSATE

References (1)

K. Bae, P. C. Ölveczky, and J. Meseguer.
Definition, Semantics, and Analysis of Multirate Synchronous AADL.
Submitted to a conference, http://formal.cs.illinois.edu/kbae/MR-SynchAADL

K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky.
Designing and Verifying Distributed Cyber-Physical Systems using Multirate PALS: An Airplane Turning Control System Case Study.
Science of Computer Programming (2014), to appear, http://formal.cs.illinois.edu/kbae/airplane

K. Bae, J. Meseguer, and P. C. Ölveczky.
Formal patterns for multirate distributed real-time systems.
Science of Computer Programming (2013), in press, http://dx.doi.org/10.1016/j.scico.2013.09.010

K. Bae, P. C. Ölveczky, J. Meseguer, and A. Al-Nayeem.
The SynchAADL2Maude tool.
In *Proc. FASE'12*, volume 7212 of *LNCS*. Springer, 2012.

K. Bae, P. C. Ölveczky, A. Al-Nayeem, and J. Meseguer.
Synchronous AADL and its formal analysis in Real-Time Maude.
In *Proc. ICFEM'11*, volume 6991 of *LNCS*. Springer, 2011.

References (2)

K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky.
PALS-based analysis of an airplane multirate control system in Real-Time Maude.
In Proc. *FTSCS'12*, volume 105 of *EPTCS*, 2012.

K. Bae, J. Meseguer, and P. C. Ölveczky.
Formal patterns for multirate distributed real-time systems.
In Proc. *FACS'12*, volume 7684 of *LNCS*, pages 1–18. Springer, 2012.

Al-Nayeem, A., Sha, L., Cofer, D.D., Miller, S.M.
Pattern-based composition and analysis of virtually synchronized real-time distributed systems.
In Proc. ICCPS12. IEEE (2012)

P. C. Ölveczky and J. Meseguer.
Formalization and correctness of the PALS architectural pattern for distributed real-time systems.
Theoritical Computer Science 451, 1–37 (2012)

Al-Nayeem, A., Sun, M., Qiu, X., Sha, L., Miller, S.P., Cofer, D.D.
A formal architecture pattern for real-time distributed systems.
In Proc. 30th IEEE Real-Time Systems Symposium. IEEE (2009)

Thank you

Background: Rewriting Logic

Rewrite theory $\mathcal{R} = (\Sigma, E, R)$: a formal specification of concurrent systems

Σ : signature for logical terms $t \in T_\Sigma$

E : equations that define equalities $t =_E t'$

R : rewrite rules specifying labeled transitions $l : [t]_E \longrightarrow [t']_E$

1. naturally describes many concurrent systems
   - including their states and events
   - can be used as a universal system specification logic
2. executable under reasonable assumptions
3. Maude: high-performance rewriting logic language and tool

More Background: Real-Time Maude

Real-Time Maude: formal analysis tool for real-time systems

- expressiveness and ease of specification
- simulation and (LTL and timed CTL) model checking tool
- equational algebraic specification defines static parts
- rewrite rules define transitions
- suitable for object-oriented specification

Real-Time Maude Semantics of Synchronous AADL (1)

- Object-oriented semantics
  - "one-to-one" correspondence AADL model $\leftrightarrow$ Real-Time Maude term
- Example (the active standby)

```
< MAIN : System |
    features : none,
    properties : Synchronous(true) ; SynchPeriod(2),
    subcomponents : < env : System | ... >
                    < sideOne : System | ... >
                    < sideTwo : System | ... >,
    connections : sideOne . side1ActiveSide -->> sideTwo . side1ActiveSide ;
                  sideTwo . side2ActiveSide -->> sideOne . side2ActiveSide ;
                  ...
                  env . side2Failed --> sideTwo . side2Failed >
```

- Synchronous step formalized by rewrite rules

Real-Time Maude Semantics of Synchronous AADL (2)

Rewrite rule defining synchronous dynamics for each step:

```
crl [syncStepWithTime] :
    {SYSTEM}
   =>
    {applyTransitions(
          transferData(
               applyEnvTransitions(VAL, SYSTEM)))}
   in time period(SYSTEM)
 if VAL ; VALS := allEnvAssignments(SYSTEM) .
```

Real-Time Maude Semantics of Synchronous AADL (3)

Equation defining deterministic thread behaviors:

```
ceq applyTransitions(
      < O : Thread | properties : Deterministic(true) ; PROPS,
                     features : PORTS,     currState : L1,
                     completeStates : LS,  variables : VAL,
                     transitions : (L1 -[GUARD]-> L2 {SL}) ; TRANS >)
 = if L2 in LS then
      < O : Thread | features  : NEW-PORTS, currState : L2,
                     variables : NEW-VALUATION >
   else
      applyTransitions(< O : Thread | features  : NEW-PORTS,
                                      currState : L2,
                                      variables : NEW-VALUATION >) fi
 if evalGuard(GUARD, PORTS, VAL) = true
 /\ not someTransEnabled(TRANS, L1, VAL, PORTS)
 /\ transResult(NEW-PORTS, NEW-VALUATION) :=
        executeTransition(L1 -[GUARD]-> L2 {SL}, PORTS, VAL) .
```