An Approach for Validation of Semantic Composability in Simulation Models

Claudia Szabo and Yong Meng Teo
Department of Computer Science
National University of Singapore

email: claudias@comp.nus.edu.sg

June 23, 2009

Outline

- Introduction
- Objective
- Related Work
- CoDES Framework Overview
- Proposed Approach
- Example
- Conclusion

Composability

- *"Capability to select and assemble* ***simulation components*** *in various combinations to satisfy* ***user requirements****"* [4, 10]
- **Simulation component** - "*a reusable, self contained unit that is independently testable and usable in a variety of contexts. [It] interacts with its environment only through a well defined interface of inputs and outputs.*"[4]
- Various levels of composability [13]:
  - technical, **syntactic** (engineering), **semantic**, pragmatic, dynamic, conceptual …

Validation of Semantic Composability

- Does the composed model containing the reused simulation components produce semantically correct results?
- Key issues:
  - Not a closed operation [2, 4, 10]
  - Emergent properties [6]
  - Context [2, 13]
  - Orthogonal aspects
    - Logical [5, 7]
    - Temporal [12]
    - Formal [10]

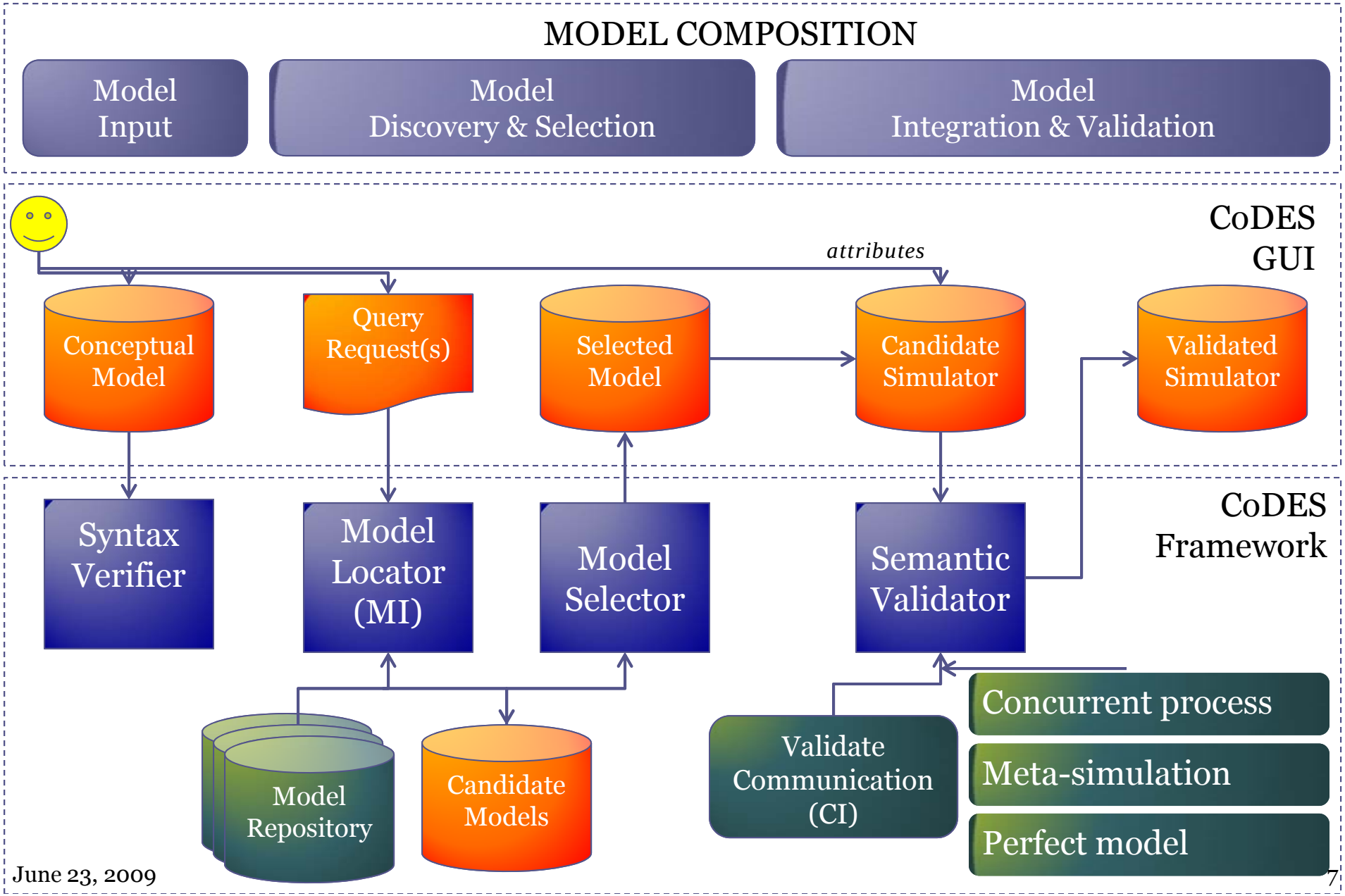
Objective

To design and develop techniques for the validation of semantic composability with formal guarantees and practical implementation potential.

Related Work

- Petty and Weisel's formal theory [10, 11]
  - Components statically represented as functions of integers; Composition = composition of mathematical functions; Validity = close enough to the simulation of a perfect model
  - Static; composition is based on the linear order of components; validation relation undefined
- Validation of DEVS models [14]
  - A DEVS model is represented a schema in the Z specification language; composition specification is validated using Z/EVES: inconsistencies, theorems, and syntax, type, and domain errors
  - Asynchronous; assumes knowledge about entire system
- Validation of BOM models [8]
  - Detailed user-specified scenario facilitates individual component discovery; candidate components executed in all possible combinations and validated against scenario
  - Informal

CoDES Framework Overview



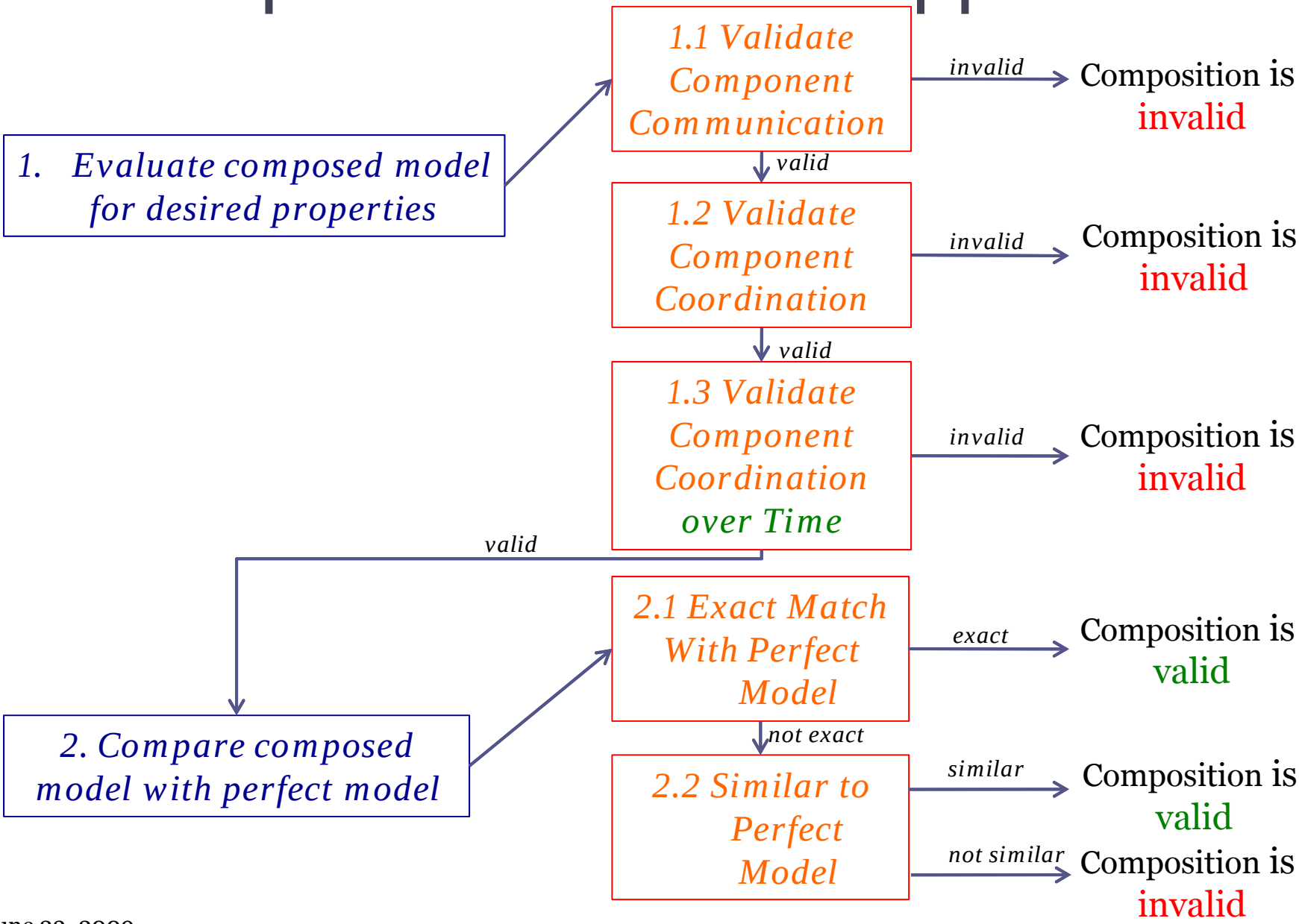
CoDES Component & Simulator

- Component: abstracted as a *black-box* with *in* and/or *out* communication channel, and represented as a *meta-component*:
  - mandatory & specific attributes
  - behavior:
    - External – constraints on input and output (destination, origin, data type, range, etc.)
    - Internal – a finite timed state automaton

$$[I_l]S_p[\Delta t] \xrightarrow{Cond_n} S_t[O_l][A_m]$$

- Simulator: represented by ***components*** (base, model) linked using ***connectors***

Proposed Validation Approach



Layered Approach

accuracy & complexity

Validate Communication

Validity = Communication is correct

- all communication is correct and meaningful

Concurrent Process Validation

Validity = Safety + Liveness (for all states)

For instantaneous transitions, for all possible interleaved executions:

- *no deadlocks*
- *valid end-states*
- *safety by invariance*
- *liveness by progress properties*

Meta-Simulation Validation

Validity = Safety + Liveness (**over time**)

For timed transitions, for all attributes

- *no deadlock*
- *liveness for all components throughout simulation run*

- sequence of valid possible transitions

Perfect Model Validation

Validity = L(M) V L(M*)

-***new time based formalism*** *to validate model against perfect model*

Concurrent Process Validation

- Validates the logical coordination of components with respect to *all* possible combinations of states

- 2 types of logical properties:
  - Safety – "Nothing bad ever happens" [9]
    - Entire composition: deadlock free, a property holds throughout the check
    - Individual components: a property holds throughout the check
  - Liveness – "Something good will always happen" [9]
    - Individual components: a specific state is always reached

- 3 Steps:
  1. Transform the composition into a specification of concurrent processes
  2. Specify safety and liveness
  3. Validate using a model checker

```
mtype {Job}; chan to1 = [10] of {mtype}; chan to2 = [10] of {mtype}; ...
hidden byte sourceIAMax = 10; byte sourceIATime; byte noJobsSource = 0;

proctype CON_ONE_TO_ONE(chan in, out){
do :: in ? Job -> out ! Job; od}

proctype SOURCE1(int id, noJobsMax; chan out){
do :: (sourceIATime == sourceIAMax) -> sourceIATime =0;
  if :: out ! Job -> progress: printf("[Source] Job sent\n"); noJobsSource ++; fiod }

proctype SourceCounter(){
do :: (sourceIATime < sourceIAMax) -> sourceIATime++; od}

proctype SINK1(int id; chan in){
S1: atomic{
if :: in ? Job -> printf("[Sink] Job received!\n"); goto S1; fi}}

proctype SERVER3(int id; chan  in, out){
bit busy;
S1: {
if :: in ? Job -> printf("[Server] Job received!\n"); busy=1;goto S2; fi}
S2: {
if :: out ! Job -> progress: printf("[Server] Job sent! \n"); busy=0; goto S1;}}
active proctype monitor(){assert (noJobsSource < noJobsMax);}
init{
run SourceCounter();
run SINK1(3, to3);…
run monitor();}
```

Liveness

Example: Single-Server Queue

Safety

Meta-Simulation Validation

- Guarantees composition safety and liveness through **time**

- 3 Steps:
  1. Transform composition into Java classes considering components' state machines through time
  2. Specify desired logical properties to be validated through time
     - Safety – Validity points - data that must pass through various connection points in the composition
       - VP1 = d1 {origin = Server, destination = Sink, range = 10; 35, type = double}
     - Liveness – Transient predicate - predicate that must become false during a given time interval after it becomes true.
       - transient(Server) = (busy == true)

  3. Validate by running the meta-simulation and observing the logical properties

Perfect Model Validation

- Proposes formal validation of model execution through time

5 Steps:

1. [**Formal Component Representation**] Represent a component as a function of states over time
2. [**Unfolding and Sampling]** Transform formal component representation using sampled time values
3. [**Composition**] Compose functions mathematically
4. [**Simulation**] Run the functions as a simulation and obtain LTS
5. [**Validation**] Compare the composed model with a perfect model

Perfect Model Validation (2)

1. [**Formal Component Representation**] Represent each component and perfect component as a function of states over time.

$$f_i : I_i \times S_i \times T_i \rightarrow O_i \times S_i \times T_i$$
$$f_i^* : I_i^* \times S_i^* \times T_i^* \rightarrow O_i^* \times S_i^* \times T_i^*$$

1. [**Unfolding and Sampling**] Sampling is done using the attribute values provided by the user.
2. [**Composition**] Verify that f_i are composable using domain and co-domain inclusion. Same for f^*.
   - Composability:

Given f_i and f_j that describe adjacent components, with f_j requiring input from f_i. f_i and f_j are composable and we can write $f_j(f_i)$ if each attribution of time moments for each unfolding of f_j when f_j requires input is greater or equal to the attribution for f_i when f_i produces output.

Perfect Model Validation (3)

4. [**Simulation**] Represent each simulation run as an LTS, L(M) and L(M ∗) where

$$\text{nodes}: \bigcup_{s \in S} s \text{ where S is the state of all components in the system}$$

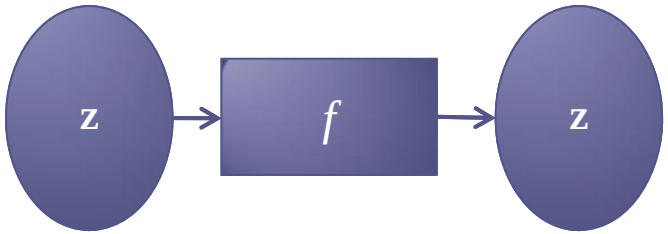
$$\text{edges}: \text{function calls } f_i \text{ and } f_i^* \text{ in the simulation run}$$

$$\text{labels}: \text{tuple } < \text{function\_name, duration, output} >$$

4. [**Validation**]
   a. Determine **strong** bisimulation relations between L(M) and L(M*)
   b. If no strong bisimulation relation exist, determine whether V_ε is a **weak** bisimulation relation

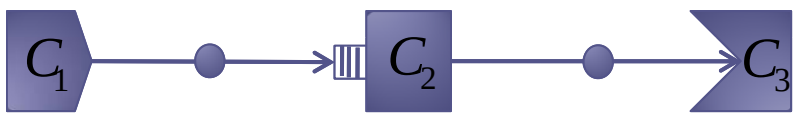
Petty & Weisel's Formal Theory

1. Formal component representation



-static
-unfeasible

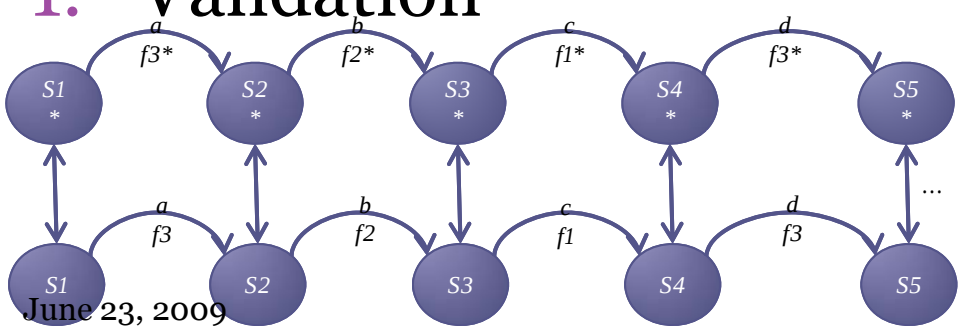
1. Composition



$$M = f_3 \circ f_2 \circ f_1$$

-component order

1. Validation



-less meaningful
-R: validation relation undefined

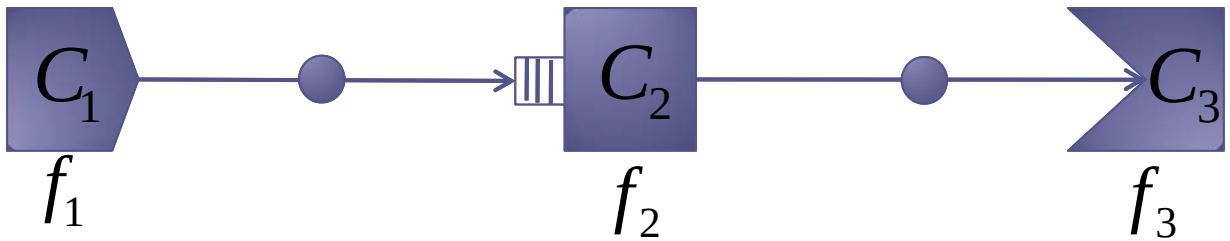
Our approach

-time-based
-dynamic

-execution order wrt time

-strong
-semantically close : V_ε

Example



1. + 2. [Formal Component Representation] + [Unfolding and Sampling]

$$f_1 : \varnothing \times S_1 \times T_1 \rightarrow \{O_1\} \times S_1 \times T_1$$

$$f_1(\varnothing, s_i, t) \rightarrow (O_1, s_i', t + \Delta t)$$

Unfold for $\tau = 3$

Sample with $mean = 4 : \Delta t = 6, \Delta t = 2, \Delta t = 4$

	Unfold	Δt	Formula	Meaning
f_1	1	6	$f_1\{0, s_1^1, 0\} \rightarrow \{O_1, s_2^1, 6\}$	from state s_1^1 with no input at time **0**, to state s_2^1 with output O_1 at time **6**
	2	2	$f_1\{0, s_2^1, 6\} \rightarrow \{O_1, s_3^1, 8\}$	from state s_2^1 with no input at time **6**, to state s_3^1 with output O_1 at time **8**
	3	4	$f_1\{0, s_3^1, 8\} \rightarrow \{O_1, s_4^1, 12\}$	from state s_3^1 with no input at time **8**, to state s_4^1 with output O_1 at time **12**
f_2	1	11	$f_2\{I_2, s_1^2, x > 0\} \rightarrow \{O_2, s_2^2, x + 11\}$	from state s_1^2 with input I_2 at time **x**, to state s_2^2 with output O_2 at time **x + 11**
	2	6	$f_2\{I_2, s_2^2, t > x + 11\} \rightarrow \{O_2, s_3^2, t + 6\}$	from state s_2^2 with input I_2 at time **t**, to state s_3^2 with output O_2 at time **t + 6**
	3	1	$f_2\{I_2, s_3^2, r > t + 6\} \rightarrow \{O_2, s_4^2, r + 1\}$	from state s_3^2 with input I_2 at time **r**, to state s_4^2 with output O_2 at time **r + 1**
f_3	1	1	$f_3\{I_3, s_1^3, x' > 0\} \rightarrow \{0, s_2^3, x' + 1\}$	from state s_1^3 with input I_3 at time **x'**, to state s_2^3 with no output at time **x' + 1**
	2	1	$f_3\{I_3, s_2^3, t' > x' + 1\} \rightarrow \{0, s_3^3, t' + 1\}$	from state s_2^3 with input I_3 at time **t'**, to state s_3^3 with no output at time **t' + 1**
	3	1	$f_3\{I_3, s_3^3, r' > t' + 1\} \rightarrow \{0, s_4^3, r' + 1\}$	from state s_3^3 with input I_3 at time **r'**, to state s_4^3 with no output at time **r' + 1**

Example (2)

3. [Composition] The most trivial constraints that can be defined are:

$$x \geq \Delta w_1, t \geq x + 11, t \geq 8 + \Delta w_2, r \geq t + 6, r \geq 12 + \Delta w_3$$

$$x' \geq x + 11 + \Delta w'_1, t' \geq x' + 1, t' \geq t + 6 + \Delta w'_2, r' \geq t' + 1, r' \geq r + 1 + \Delta w'_3$$

$$(x = 8, t = 19, r = 25), (x' = 23, t' = 28, r' = 29).$$

$$(x^* = 8, t^* = 19, r^* = 25), (x'^* = 23, t'^* = 28, r'^* = 29).$$

Example (3)

4. [Simulation]

$$f_1(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$$
$$f_1(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$$
$$f_2(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 19)$$
$$f_1(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$$
$$f_2(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$$
$$f_3(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$$
$$f_2(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$$
$$f_3(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$$
$$f_3(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$$

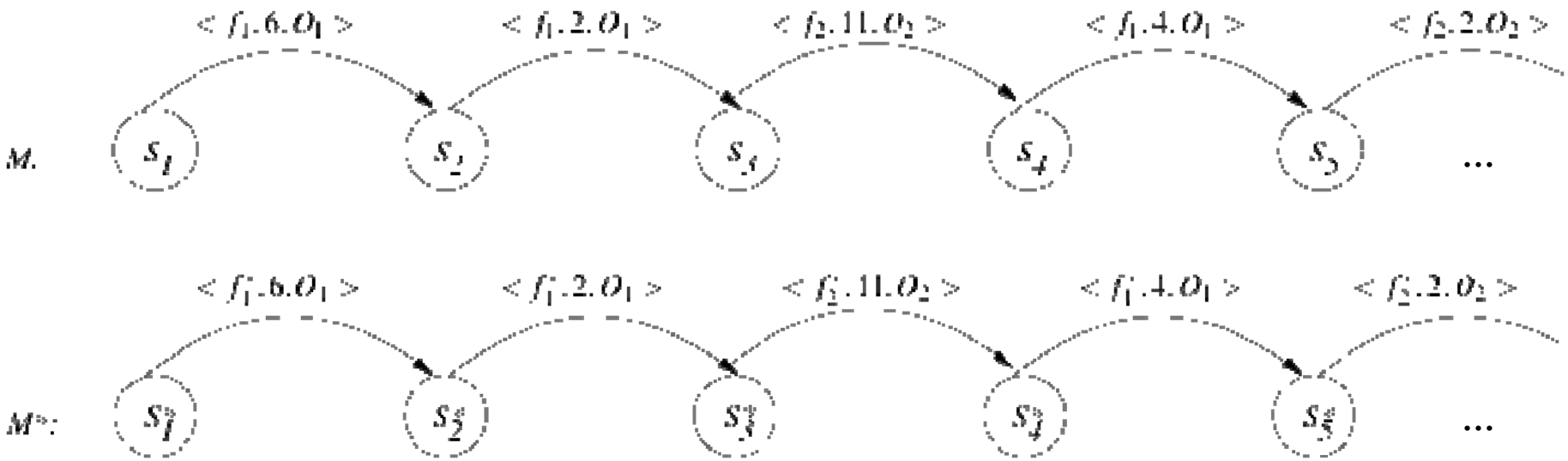
Composition

$$f_1^*(\emptyset, s_1^1, 0) \rightarrow (O_1, s_2^1, 6)$$
$$f_1^*(\emptyset, s_2^1, 6) \rightarrow (O_1, s_3^1, 8)$$
$$f_2^*(I_2, s_1^2, 8) \rightarrow (O_2, s_2^2, 19)$$
$$f_1^*(\emptyset, s_3^1, 8) \rightarrow (O_1, s_4^1, 12)$$
$$f_2^*(I_2, s_2^2, 19) \rightarrow (O_2, s_3^2, 25)$$
$$f_3^*(I_3, s_1^3, 23) \rightarrow (\emptyset, s_2^3, 24)$$
$$f_2^*(I_2, s_3^2, 25) \rightarrow (O_2, s_4^2, 26)$$
$$f_3^*(I_3, s_2^3, 28) \rightarrow (\emptyset, s_3^3, 29)$$
$$f_3^*(I_3, s_3^3, 29) \rightarrow (\emptyset, s_4^3, 30)$$

Perfect Composition

Example (4)

5. [Validation]



1. Strong bisimulation relation between M and M* validated by the CADP toolset [32]

Implementation

- Validation of composed model (Concurrent Process Validation)
  - Java program transforms the COML files into a Promela specification
  - The Spin model checker is called to validate the Promela specification

- Validation of composed model through time (Meta-Simulation Validation) (Java)
  - Java program transforms the COML files into a Java hierarchy
  - Threaded implementation using PipedInputStreams

- Validation against perfect model (Perfect Model Validation)
  - Java program transforms component COML files into functional representation and subsequently into LTS; uses the BISIMULATOR tool from the CADP toolset to determine strong equivalence
  - Java program determines related states according to proposed semantic metric relation; related states are subsequently validated by BISIMULATOR

Weaknesses

- Validation of composed model
  - State explosion
    - 3 components with 2 states/comp – 2 minutes
    - 10 components with 2 states/comp – 15 minutes
  - Can be limited by further abstracting the Promela representation – accuracy vs complexity
- Validation of composed model *through time*
  - Based on sampling - #samples vs computation cost
- Validation against perfect model
  - Based on sampling - #samples vs computation cost
  - Semantic metric relation – formal measure vs semantic meaning of validity

Conclusion

- Three-layer approach for semantic validation of compositions with increasing accuracy and complexity:
  - Validation of composed model:
    1. *Formal* guarantee of the *logical* component coordination
    2. *Safety* and *liveness* over *time*
  - Validation against perfect model:
    3. *Formal* guarantee of composition validation through *time*
- Implementation tested on simple and complex examples of open queueing networks

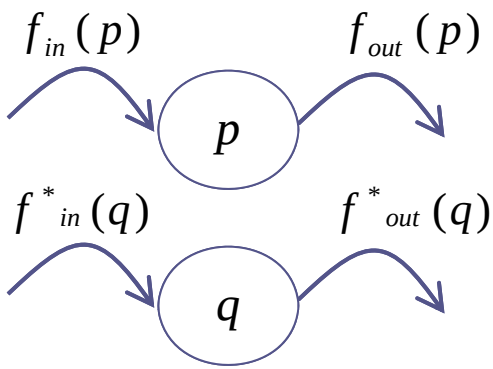
Thank you! Questions?

1) C. Szabo and Y.M. Teo, **On Syntactic Composability and Model Reuse**, Proceedings of the International Conference on Modeling and Simulation, pages 230–237, Phuket, Thailand, 2007 (invited paper).
2) Y.M. Teo and C. Szabo, **CODES: An Integrated Approach to Composable Modeling and Simulation**, Proceedings of the 41st Annual Simulation Symposium, pages 103–110, Ottawa, Canada, 2008.
3) C. Szabo and Y.M. Teo. **An Approach for Validation of Semantic Composability in Simulation Models**, Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation, pp. 3-10, IEEE Computer Society Press, New York, USA, Jun 22-25, 2009.
4) C. Szabo, Y.M. Teo and S. See. **A Time-based Formalism for the Validation of Semantic Composability**, Proceedings of the Winter Simulation Conference, Austin, USA, 2009 (to appear).

Semantic Parametric Metric Relation

$$p = [s(p), f_{in}(p), f_{out}(p)]$$

$$q = [s^*(q), f^*{}_{in}(p), f^*{}_{out}(p)]$$



$$V_\varepsilon(p,q) = \{(p,q) \in P \times Q \mid \|p-q\|_\sigma \le \varepsilon\}$$

$$\|p-q\|_\sigma = \frac{DS(s(p), s^*(q)) + \dfrac{DF(f_{in}(p), f^*{}_{in}(q)) + DF(f_{out}(p), f^*{}_{out}(q))}{2}}{2}$$

determine if composition states are related (attributes and values)

determine if the same component is executed