

# CS6133 Fall 2011 Software Specification and Verification

## Lecture 6 Template Semantics

## Formal Methods

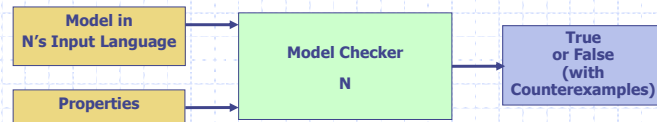
- ◆ Errors in critical systems can cause loss of life
- ◆ Confidence in software can be obtained using formal methods
- ◆ Formal methods are rigorous means to specify and verify computer-based systems
  - Formal notations
  - Powerful tools (e.g., model checkers and theorem provers)

CS6133

2

## Formal Analysis Tools

- ◆ Many powerful tools have been increasingly applied in reasoning about computer-based systems, e.g.,  
SMV, SPIN, LTSA, Concurrency Workbench, SAT Solver

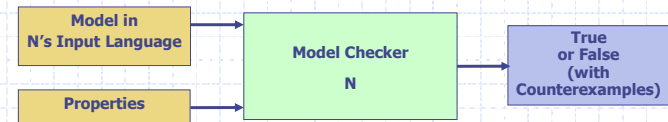


CS6133

3

## Input Language to Formal Analysis Tools

- ◆ Input languages to many analysis tools are close to low-level computation models
  - Kripke structures
  - BDDs
  - Logic formulae



CS6133

4

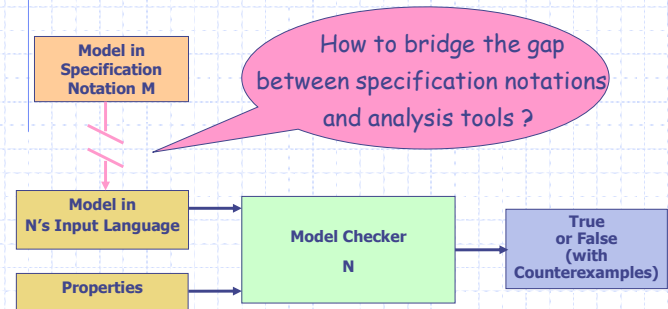
## Formal Specification Notations

- ◆ Specification notations describe systems' behavior
  - Process algebras, e.g., CCS, FSP
  - State-based notations, e.g., statecharts
- ◆ Model-based notations are suitable and flexible for modeling large reactive systems
  - Step semantics are intuitive for software practitioners
  - Composition mechanisms provide facilities to represent concurrency, communication, and synchronization
- ◆ Specification errors can be easier and cheaper to fix in the early stages - using analysis tools

CS6133

5

## Analyzing Specification Notations

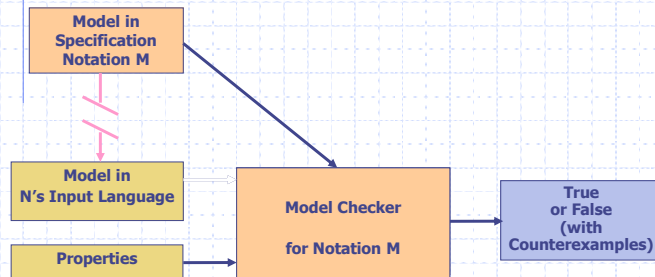


CS6133

6

## Current Approaches for Analyzing Specification

- ◆ Construct a specific tool for a specification notation [Cleaveland & Parrow & Steffen, Harel & Naamad]

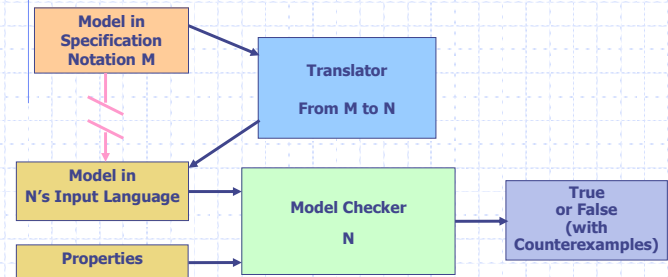


CS6133

7

## Current Approaches for Analyzing Specifications

- ◆ Write a translator from a notation to the input language of an analyzer [Atlee & Gannon, Cheng et al., Chan et al.]

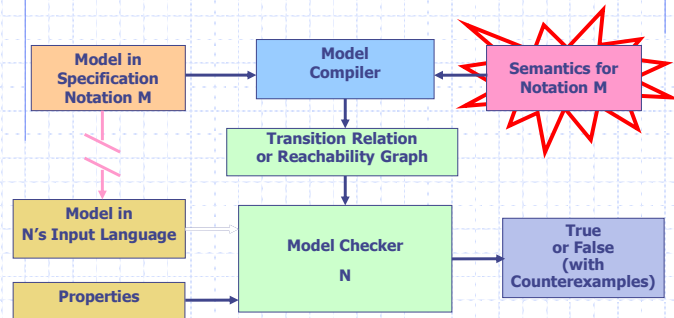


CS6133

8

## Semantics-Based Model Compilation

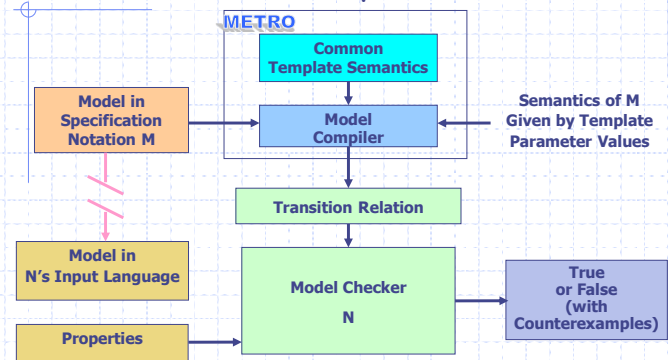
[Day & Joyce, Dillion & Stirewalt, Pezze & Young]



CS6133

9

## Parameterized Semantics-Based Model Compilation



CS6133

10

## Template Semantics

### ◆ Template semantics

- A new approach to structure the semantics of model-based specification notations that captures the common behavior among notations and parameterizes their differences
- Separation of concerns among aspects of a notation's semantics, e.g., step semantics, composition operators

### ◆ METRO: a template-semantics-based model compiler

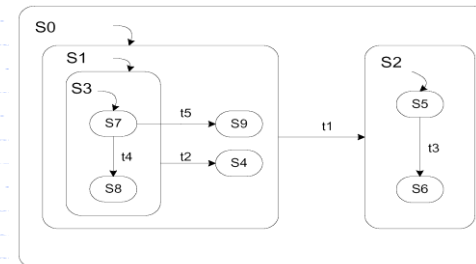
- Transformation of new notations or notation variants to analysis tools

CS6133

11

## Computation Model

- ◆ Hierarchical transition system (HTS) is a hierarchical, extended finite state machine without concurrency

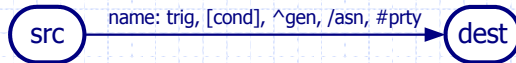


CS6133

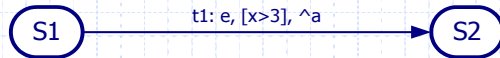
12

## Computation Model --- Syntax

- ◆ An HTS contains
  - States and a state hierarchy
  - Internal and external events
  - Variables
  - Transitions



Example:



CS6133

13

## Semantics of HTS --- Snapshot

- ◆ Snapshot: observable point in execution

Basic Elements	current states current internal events current variable values current generated events	
Auxiliary Elements	auxiliary states auxiliary internal events auxiliary variable values auxiliary external events	} used to determine which transitions are enabled

CS6133

14

## Common Semantics of HTS

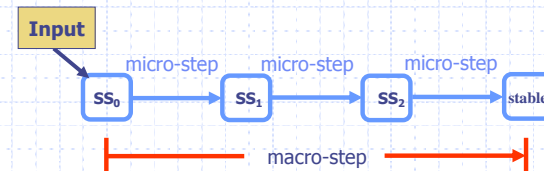
- ◆ Behavior of an HTS is described by the possible execution steps it can take
  - Which transition is enabled
    - enabling states
    - enabling events
    - enabling variables
  - How the HTS is affected by executing a transition
- ◆ Step relates the current snapshot and the next snapshot of an HTS

CS6133

15

## Step Semantics

- ◆ Micro-step: one transition execution
- ◆ Macro-step: a sequence of micro steps until the system is stable



CS6133

16

## Step Semantics

- ◆ Three types of macro-steps
  - Simple diligent: e.g., CCS
    - take a micro-step if a transition is enabled
  - Simple non-diligent: e.g., BTS
    - take a micro-step or stay idle
  - Stable: e.g., statecharts
    - a maximal sequence of micro-steps until no enabled transitions for the set of inputs

CS6133

17

## Common Template Definitions

Common semantics	Template parameters
reset	<ul style="list-style-type: none"> <li>• reset state info</li> <li>• reset event info</li> <li>• reset variable info</li> </ul>
enabled transitions	<ul style="list-style-type: none"> <li>• enabling states</li> <li>• enabling events</li> <li>• enabling variable values</li> </ul>
apply	<ul style="list-style-type: none"> <li>• change state</li> <li>• generate events</li> <li>• change variable values</li> </ul>

CS6133

18

## Common Templates

- **micro-step** ( $ss, \tau, ss'$ ): transition  $\tau$  is enabled in snapshot  $ss$ , and its actions may produce next snapshot  $ss'$
- **enabled\_trans** ( $ss, T$ ): the subset of transitions in  $T$  that are enabled by snapshot  $ss$ 's states, events, and variable values
- **apply** ( $ss, \tau, ss'$ ): applying transition  $\tau$ 's actions (variable assignments, generated events) to snapshot  $ss$  may produce snapshot  $ss'$
- **macro-step** ( $ss, I, ss'$ ): there is a sequence of micro-steps that starts from snapshot  $reset(ss, I)$  and ends the macro-step in snapshot  $ss'$
- **reset** ( $ss, I$ )= $ss''$ : resetting snapshot  $ss$  at the start of a macro-step with inputs  $I$  produces snapshot  $ss''$

CS6133

19

## Template Parameters

how reset at beginning of macro-step

RESET

how changed when a transition is taken

NEXT

current state		
current int_ev		
current var_val		
current output		
auxiliary state		
auxiliary int_ev		
auxiliary var_val		
auxiliary ext_ev		
en_states		
en_events		
en_cond		

how used to enable a transition

CS6133

20

## Event-Related Template Parameters

		RESET	NEXT
current state	CS		
current int_ev	IE		
current var_val	AV		
current output	O		
auxiliary state	CSa		
auxiliary int_ev	IE		
auxiliary var_val	AVa		
auxiliary ext_ev	Ia		
en_states			
en_events			
en_cond			

how reset at beginning of macro-step

how changed when a transition is taken

how used to enable a transition

Enabling Events

CS6133

21

## Enabling Events

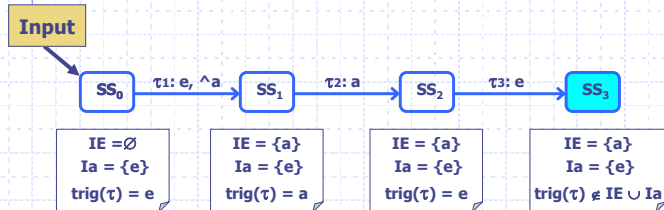
- ◆ Enabling internal events
  1. Events generated since the beginning of the macro-step
  2. Events generated in the previous micro-step
- ◆ Enabling external events
  1. Events from the input are persistent through macro-step
  2. Events from the input are enabling only at the first micro-step

CS6133

22

## Enabling Events

- ◆ Enabling internal events
  1. Events generated since the beginning of the macro-step
- ◆ Enabling external events
  1. Events from the input are persistent through macro-step

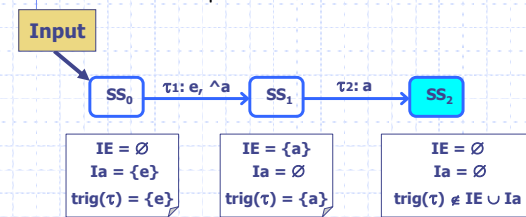


CS6133

23

## Enabling Events

- ◆ Enabling internal events
  2. Events generated in the previous micro-step
- ◆ Enabling external events
  2. Events from the input are enabling only at the first micro-step



CS6133

24

## Template Parameters for STATEMATE

how reset at beginning of macro-step



how changed when a transition is taken

current state	CS		
current int_ev	IE	$\emptyset$	$IE' = \text{gen}(\tau)$
current var_val	AV		
current output	O		
auxiliary state	CSa		
auxiliary int_ev	IE	n/a	n/a
auxiliary var_val	AVa		
auxiliary ext_ev	Ia	$Ia' = \text{ss.Ia}$	$Ia' = \emptyset$
en_states			
en_events		$\text{trig}(\tau) \subseteq \text{ss.IE} \cup \text{ss.Ia}$	
en_cond			

Enabling Events

how used to enable a transition

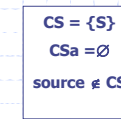
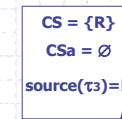
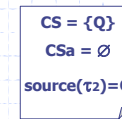
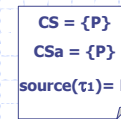
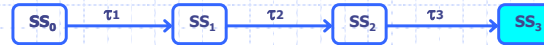
CS6133

25

## Enabling States

### Current states

en\_states :=  
source( $\tau$ )  $\in$  CS



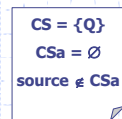
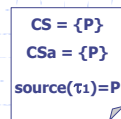
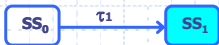
CS6133

26

## Enabling States

### States at the beginning of the macro-step

en\_states :=  
source( $\tau$ )  $\in$  CSa



CS6133

27

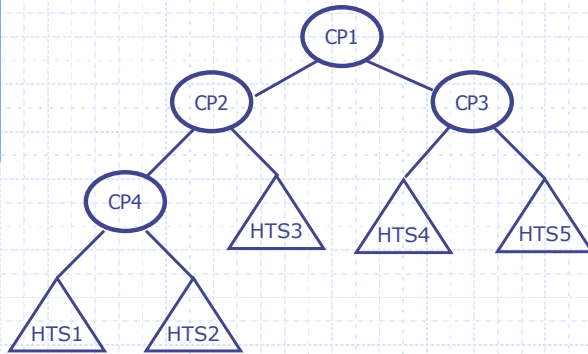
## Composition Operators

- ◆ Compose multiple HTSs into a system, and represent
  - Concurrency
  - Communication
  - Synchronization
- ◆ Constrain
  - Which component to execute
  - When to transfer control to each other
  - How to exchange events and data
- ◆ Rely on template parameters to ensure their semantics are consistent with the step semantics

CS6133

28

## Composition Operators



CS6133

29

## Composition Operators

- ◆ Parallel
- ◆ Interleaving
- ◆ Synchronization
  - Environmental synchronization
  - Rendezvous synchronization
- ◆ Interrupt
- ◆ Sequence
- ◆ Choice

CS6133

30

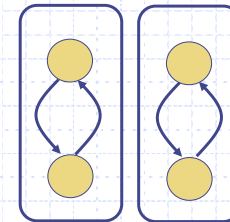
## Composition Operators

- ◆ **Parallel**
- ◆ Interleaving
- ◆ Synchronization
  - **Environmental synchronization**
  - Rendezvous synchronization
- ◆ **Interrupt**
- ◆ Sequence
- ◆ Choice

CS6133

31

## Parallel Composition

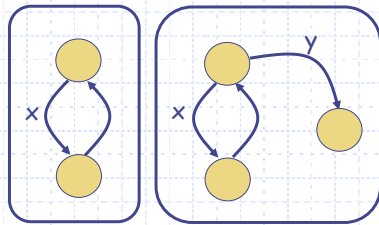


Components execute in parallel

CS6133

32

## Environmental Synchronization

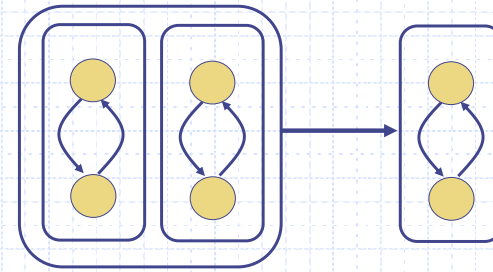


Components synchronize on an external synchronization event

CS6133

33

## Interrupt



Transfer control from one component to the other

CS6133

34

## Interrupt --- Formal Definition

$$\begin{aligned}
 & \text{startcomp}(\bar{s}, \bar{\tau}, \bar{s}') \equiv \\
 & \text{ent\_comp}(\bar{s}, \bar{\tau}, \bar{s}', CS) \wedge \text{next\_CS}_0(\bar{s}, \bar{\tau}, \bar{s}', CS_0) \wedge \text{next\_JE}(\bar{s}, \bar{\tau}, \bar{s}', JE) \wedge \text{next\_JE}_0(\bar{s}, \bar{\tau}, \bar{s}', JE_0) \wedge \\
 & \text{next\_L}_0(\bar{s}, \bar{\tau}, \bar{s}', L_0) \wedge \text{next\_O}(\bar{s}, \bar{\tau}, \bar{s}', O) \\
 \\
 & \text{N}^{\text{interrupt}}_{\text{micro}}((\bar{s}_1, \bar{s}_2), (\bar{\tau}_1, \bar{\tau}_2), (\bar{s}'_1, \bar{s}'_2)) T_{\text{interrupt}} \equiv \\
 & [ \bar{s}_1, CS \neq \emptyset \wedge \text{higher\_pri}(\bar{\tau}_1, \text{pri}(\text{enabled\_trans}(\bar{s}_1, T_{\text{interrupt}}))) \wedge \text{comp1steps}(\bar{s}_1, \bar{s}_2), (\bar{\tau}_1, \bar{\tau}_2), (\bar{s}'_1, \bar{s}'_2) ] \text{ ("component 1 steps")} \\
 & \vee \exists \tau. \left[ \begin{array}{l} \tau \in \text{pri}(\text{enabled\_trans}(\bar{s}'_1, T_{\text{interrupt}})) \wedge \text{higher\_pri}(\tau, \text{pri}(\text{enabled\_trans}(\bar{s}_1, \bar{\tau}_1))) \\ \bar{\tau}_1 = \{\tau\} \wedge \bar{\tau}_2 = \emptyset \wedge \text{startcomp}(\bar{s}'_2, \tau, \bar{s}'_2) \end{array} \right] \text{ ("transition to component 2")} \\
 & \vee \left[ \begin{array}{l} \text{update}(\bar{s}'_1, \bar{s}'_2, \{\tau\}, \bar{s}'_1) \wedge \text{communicate\_vars}((\bar{s}_1, \bar{s}_2), (\tau, \emptyset), (\bar{s}'_1, \bar{s}'_2)) \\ \text{("symmetric cases of the above two cases, replacing 1 with 2 and 2 with 1")} \end{array} \right]
 \end{aligned}$$

CS6133

35

## Validation of Template Semantics

- ◆ Template semantics is
  - Succinct
    - Description of a notation is an instantiation of parameters
  - Expressive
    - CCS, CSP, LOTOS, statecharts variants, and BTS
    - SCR, Petri Net, and SDL
  - Extensible
    - History states, negated events, and SDL timers
- ◆ Template semantics eases a user's effort in understanding and comparing specification notations

CS6133

36



## METRO : Parameterized Model Compiler

- ◆ Template semantics forms the theoretical foundation for **METRO**
  - Parameterized template definitions represent common allowable execution steps of model-based notations
  - Parameters describe a notation's distinct semantics
  - Composition operators, defined as a separate concern, constrain how multiple HTSs execute concurrently
  - A notation's semantics is expressed as a set of parameter values and composition operators

CS6133

41

## METRO : Parameterized Model Compiler

- ◆ **METRO** takes as input a specification and its template semantics description (a set of template-parameter values and composition operators)
- ◆ **METRO** produces for a specification a transition relation, which can be used as an input to formal analysis tools (e. g., model checker)
- ◆ **METRO** eases effort required for mapping multiple notations to analysis tools
  - As a notation evolves, a user only needs to modify parameter values to reflect the notation's changes

CS6133

42

## Summary

- ◆ Template semantics: structure a notation's semantics
  - Capture the common behaviors among notations and parameterizes their differences
  - Separate concerns among aspects of a notation's semantics
  - Ease a specifier's effort in understanding and comparing different model-based notations
- ◆ **METRO**: map specification notations to analysis tools
  - Compile a specification into a transition relation based on the template-semantics description of the notation
  - Facilitate the transformation of new notations or notation variants to analysis tools

CS6133

43