Part 3: Models of Computation

- FSMs
- Discrete Event Systems
- CFSMs
- Data Flow Models
- Petri Nets
- The Tagged Signal Model
- Synchronous Languages and De-synchronization
- Heterogeneous Composition: Hybrid Systems and Languages
- Interface Synthesis and Verification
- Trace Algebra, Trace Structure Algebra and Agent Algebra

1 EE249Fall07

Design

- From an idea…
- … build something that performs a certain function
- Never done directly:
  - some aspects are not considered at the beginning of the development
  - the designer wants to explore different possible implementations in order to maximize (or minimize) a cost function
- Models can be used to reason about the properties of an object

2 EE249Fall07

Formalization

- Model of a design with precise unambiguous semantics:
  - Implicit or explicit relations: inputs, outputs and (possibly) state variables
  - Properties
  - "Cost" functions
  - Constraints

Formalization of Design + Environment =
closed system of equations and inequalities over some algebra.

3

EE249Fall07

Models of Computation: And There are More...

- Continuous time (ODEs)
- Spatial/temporal (PDEs)
- Discrete time
- Rendezvous
- Synchronous/Reactive
- Dataflow
- ...

Each of these provides a formal framework for reasoning about certain aspects of embedded systems.

Tower of Babel, Bruegel, 1563

4

EE249Fall07

Model Of Computation

Definition: A mathematical description that has a syntax and rules for computation of the behavior described by the syntax (semantics). Used to specify the semantics of computation and concurrency.

Examples: Finite State Machine, Turing Machine, differential equation

An MoC allows:

- To capture unambiguously the required functionality
- To verify correctness of the functional specification wrt properties
- To synthesize part of the specification
- To use different tools (all must “understand” the model)
- MOC needs to
  - be powerful enough for application domain
  - have appropriate synthesis and validation algorithms

5

EE249Fall07

Usefulness of a Model of Computation

- Expressiveness
- Generality
- Simplicity
- Compilability/ Synthesizability
- Verifiability

The Conclusion

One way to get all of these is to mix diverse, simple models of computation, while keeping compilation, synthesis, and verification separate for each MoC. To do that, we need to understand these MoCs relative to one another, and understand their interaction when combined in a single system design.

6

EE249Fall07

Common Models of Computation

- Finite State Machines
  - finite state
  - no concurrency nor time
- Data-Flow
  - Partial Order
  - Concurrent and Determinate
  - Stream of computation
- Discrete-Event
  - Global Order (embedded in time)
- Continuous Time

The behavior of a design in general is described by a composition

7

EE249Fall07

Control versus Data Flow

- Fuzzy distinction, yet useful for:
  - specification (language, model, ...)
  - synthesis (scheduling, optimization, ...)
  - validation (simulation, formal verification, ...)
- Rough classification:
  - control:
    - don't know when data arrive (quick reaction)
    - time of arrival often matters more than value
  - data:
    - data arrive in regular streams (samples)
    - value matters most

8

EE249Fall07

Control versus Data Flow

- Specification, synthesis and validation methods emphasize:
  - for control:
    - event/reaction relation
    - response time
      (Real Time scheduling for deadline satisfaction)
    - *priority* among events and processes
  - for data:
    - functional dependency between input and output
    - memory/time efficiency
      (Dataflow scheduling for efficient pipelining)
    - all events and processes are *equal*

9

EE249Fall07

The vending machine

- A machine that sells coffee
  - Accepts one dollar (d1) bills
  - Maximum two dollars
  - Quarters change
  - Sells two products
    - Small coffee for $1
    - Large coffee for $1.25

10

EE249Fall07

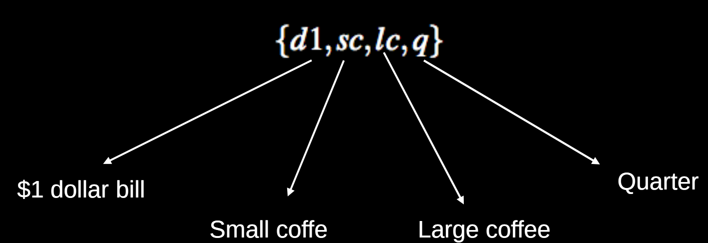
Denotational description basics

Denotational descriptions are “implicit” in the sense that they describe the properties that the system must have. They often are given as a system of equalities and inequalities that must be satisfied by the system.

- The controller is denoted by a set of traces of symbols from an alphabet
- Non all-capital letters names belong to the alphabet of a process
- Capital letters names denote processes (CTRL is the controller process)
- A process is a letter followed by a process: P = x → Q
- SKIP is a processes that successfully completes execution (it does nothing, it just completes the execution)
- If P and Q are processes then Z = P ; Q is a process that behaves like P until it completes and then like Q
- If P and Q are processes then P | Q denotes a choice between P and Q

Vending machine description

- Alphabet



Vending machine description

• Vending machine process

$$VM = (SMALL | LARGE); VM$$

Behaves as (small "choice" large) until successful completion and then like VM

• It's a recursive definition of the form $X = F(X)$

• For a large coffee:

$$LARGE = d1 \rightarrow (d1 \rightarrow (lc \rightarrow CHANGE3))$$

$$CHANGE3 = q \rightarrow (q \rightarrow (q \rightarrow STOP))$$

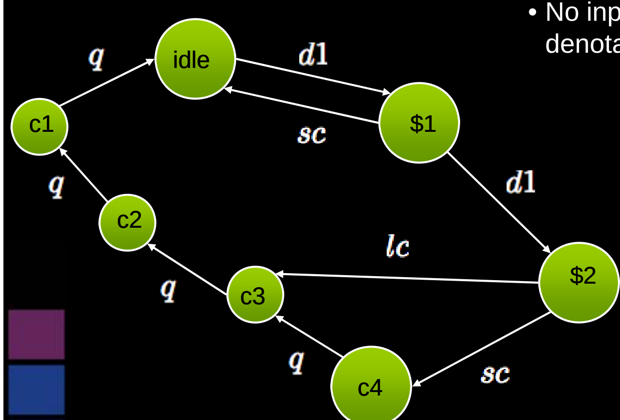
13

EE249Fall07

Vending machine FSM

• The encoding of the behaviors with a labeled directed graph

• No inputs/outputs yet (as in the denotational description)

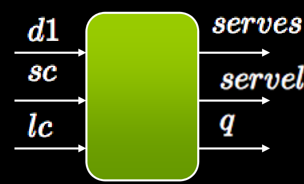


14

EE249Fall07

Vending machine I/O description

$$
\begin{aligned}
I &= \{d1, sc, lc\}\\
O &= \{serves, servel, q\}\\
S &= \{idle, \$1, \$2, c1, c2, c3, c4\}
\end{aligned}
$$



(deterministic description)

$$
\begin{aligned}
\delta &: I \times S \rightarrow S \quad \text{State transition function}\\
\lambda &: I \times S \rightarrow O \quad \text{Output function}
\end{aligned}
$$

Examples: $\delta(d1, idle) = \$1$ $\longrightarrow$ If waiting and one dollar is inserted change state to \$1 credit

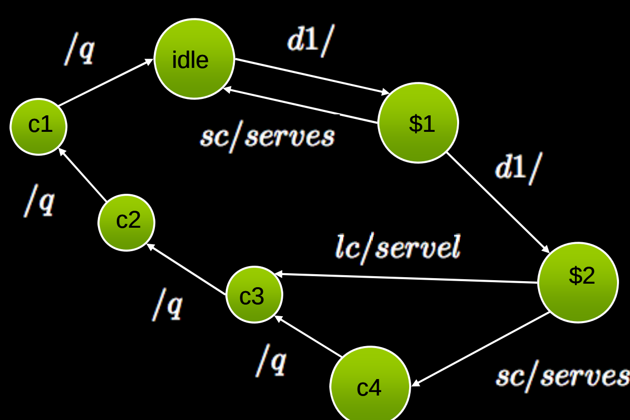
$$
\begin{aligned}
\delta(sc, \$1) &= idle\\
\lambda(sc, \$1) &= serves
\end{aligned}
$$

$\longrightarrow$ If \$1 credit and small coffee is requested, change state to idle and serve the coffee

15 EE249Fall07

Vending machine I/O description

Lables: $input/output$, where input and output can both be empty

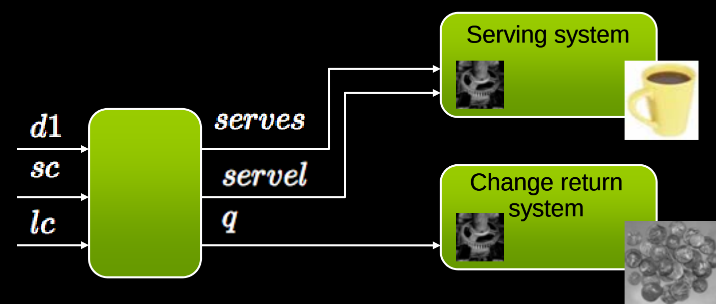


16 EE249Fall07

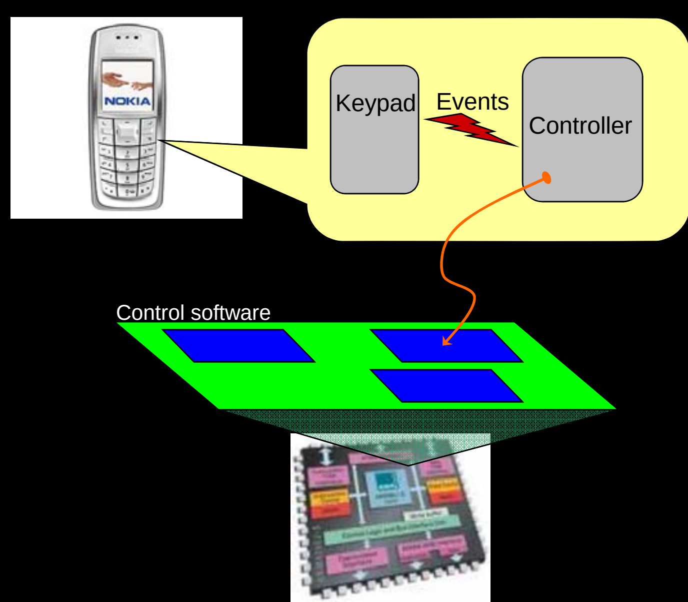
Communication with the rest of the system

Our state machine does not live in isolation

- What is the communication semantics?
- The serving system and the change return are electromechanical system with their own evolution dynamics



The Nokia 3120 User Interface



Controller description: Denotational

- The controller is denoted by a set of traces of symbols from an alphabet
- Non all-capital letters names belong to the alphabet of a process
- Capital letters names denote processes (CTRL is the controller process)
- A process is a letter followed by a process: P = x → Q
- SKIP is a process that successfully completes execution (it does nothing, it just completes the execution)
- If P and Q are processes then Z = P ; Q is a process that behaves like P until it completes and then like Q
- *P is a finite number of repetition of process P

19

EE249Fall07

Controller description: Denotational

To lock or unlock a Nokia phone press “Menu” followed by the Star key

$$LKUNLK = Menu \rightarrow Star \rightarrow SKIP$$

Process — Letter of the alphabet — Successful

Once unlocked, pick something from the menu and perform some action (for instance, choose “Contacts->Find->Alberto) and perform the action “Call”

$$SELECTION = Menu \rightarrow (CHOICE; ACTION)$$

Sequential composition

$$CHOICE = (1 \rightarrow SKIP)|(2 \rightarrow SKIP)|...$$

A complete operation is an unlock followed by a selection followed by a lock

$$OP = LKUNLK; SELECTION; LKUNLK$$

A controller is a finite (the phone breaks at some point) sequences of operations

$$CRTL = {}^{*}OP$$

EE249Fall06

Controller description: Denotational Implicit

A tuple is the mathematical object that denotes the controller

$$(I, O, S, \delta, \lambda, s_0)$$

$$\begin{aligned} I &= (Menu, Star, 1, 2 ...) \\ O &= (Call, SMS, ...) \\ S &= (Lk, Lk\_Menu, UnLk, MainMenu, Contacts, ...) \end{aligned}$$

These two functions encode the possible traces

$$\begin{aligned} \delta &: 2^I \times S \rightarrow S \\ \lambda &: 2^I \times S \rightarrow O \end{aligned}$$

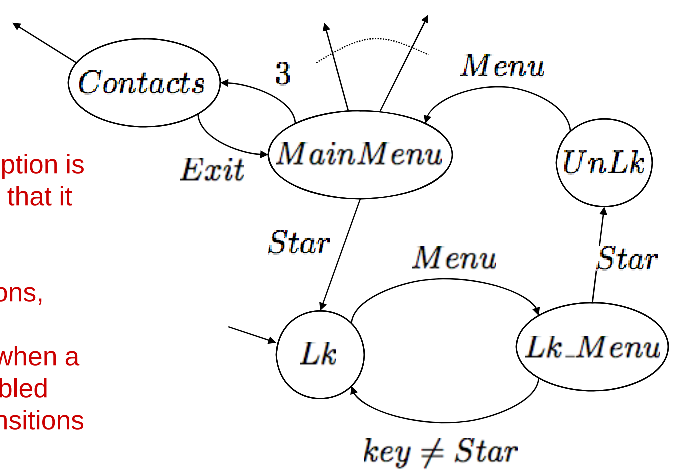
Example: To describe the unlock sequence

$$\begin{aligned} \delta(Menu, Lk) &= Lk\_Menu \\ \delta(Star, Lk\_Menu) &= UnLk \end{aligned}$$

EE249Fall06

Controller Description: Operational

State transition graph



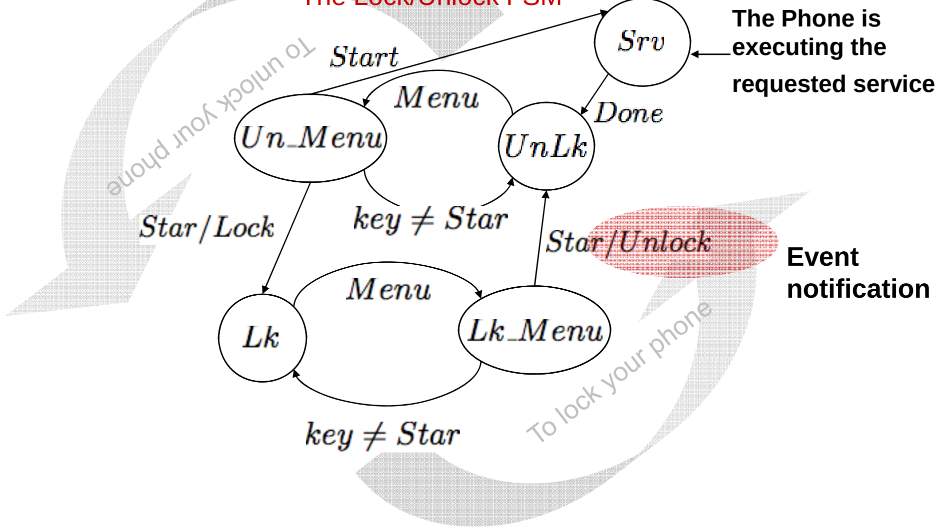
An operational description is “explicit” in the sense that it defines:

- The meaning of enabled transitions, events etc.
- What happens when a transitions is enabled
- How a state transitions is accomplished

EE249Fall06

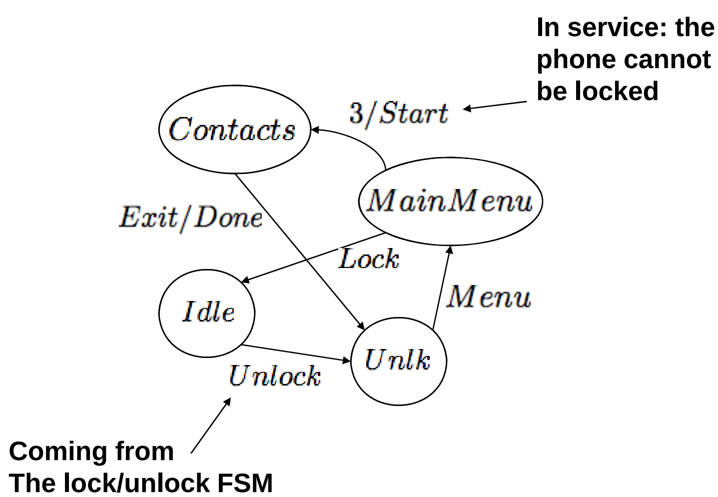
Composition with synchronization labels

The Lock/Unlock FSM



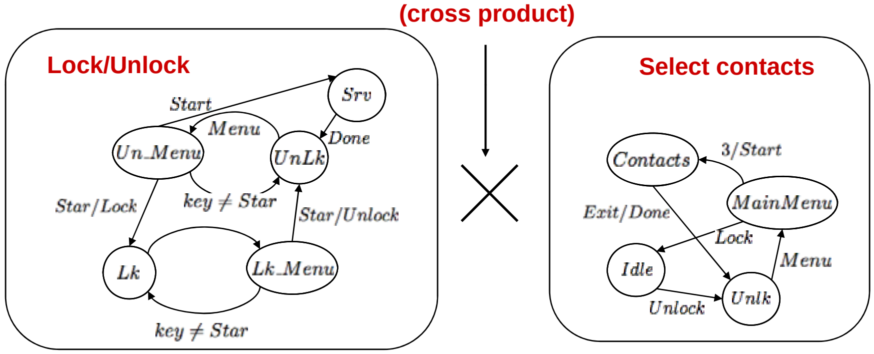
An example of service

The Select Contacts FSM



Communication by synchronization

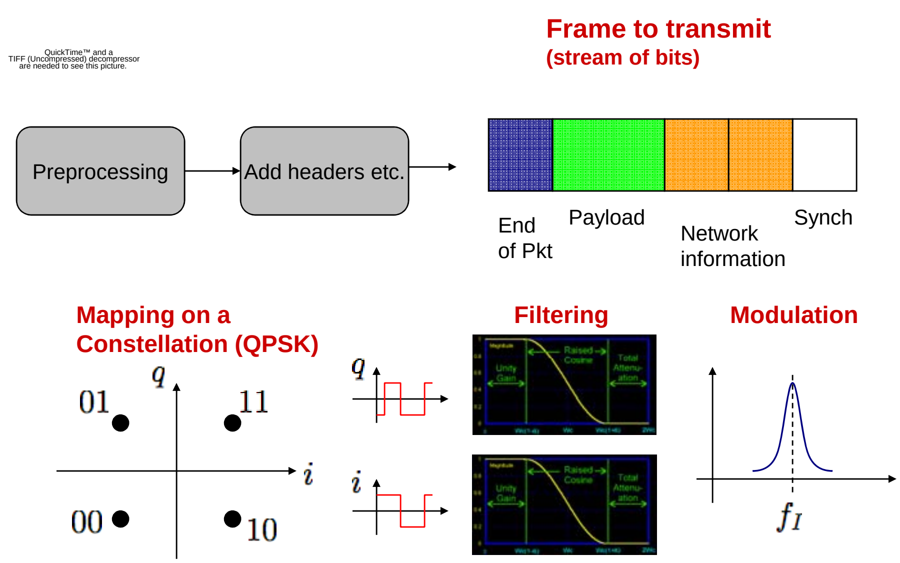
Operation of composition (cross product)



Transitions with the same synchronization labels must happen concurrently. There is no notion of time.

EE249Fall06

Base-band Processing



EE249Fall06

Base-band Processing: Denotation

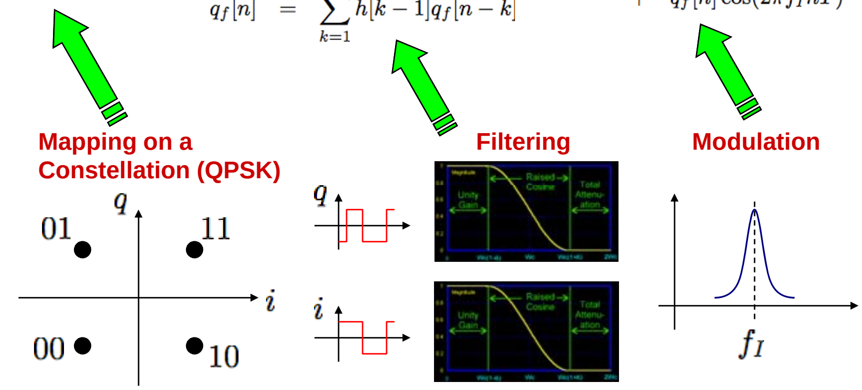
Composition of functions = overall base-band specification

$$x[n] = (Map_i(s) * h)[n]\sin(2\pi f_I nT) + (Map_q(s) * h)[n]\cos(2\pi f_I nT)$$

$$i[n] = Map_i(s[n])$$
$$q[n] = Map_q(s[n])$$

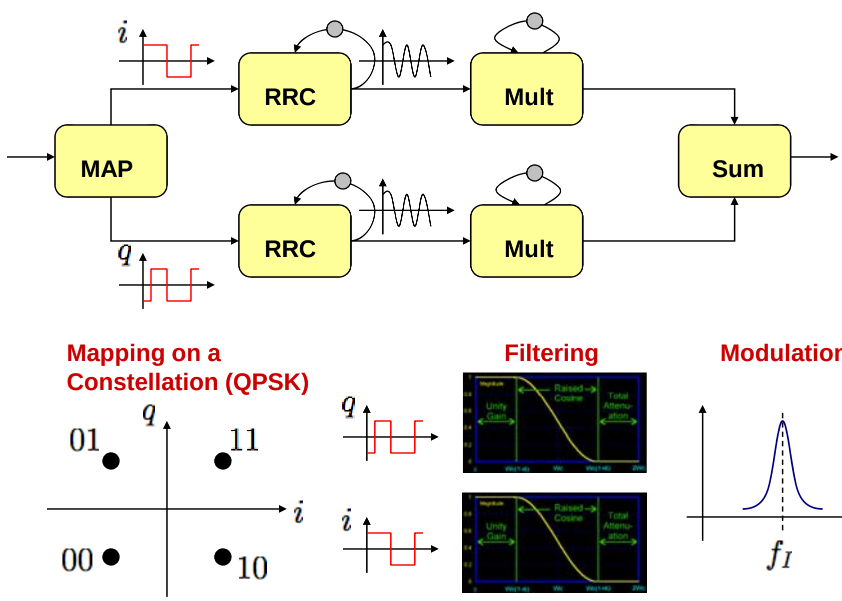
$$i_f[n] = \sum_{k=1}^{N} h[k-1]i_f[n-k]$$
$$q_f[n] = \sum_{k=1}^{N} h[k-1]q_f[n-k]$$

$$x[n] = i_f[n]\sin(2\pi f_I nT) + q_f[n]\cos(2\pi f_I nT)$$



EE249Fall06

Base-band Processing: Data Flow Model



EE249Fall06

Remarks

- Composition is achieved by input-output connection through communication channels (FIFOs)
- The operational semantics dictates the conditions that must be satisfied to execute a function (actor)
- Functions operating on streams of data rather than states evolving in response to traces of events (data vs. control)
- Convenient to mix denotational and operational specifications

29

EE249Fall07

Telecom/MM applications

- Heterogeneous specifications including
  - data processing
  - control functions
- Data processing, e.g. encryption, error correction…
  - computations done at regular (often short) intervals
  - efficiently specified and synthesized using DataFlow models
- Control functions (data-dependent and real-time)
  - say when and how data computation is done
  - efficiently specified and synthesized using FSM models
- Need a common model to perform global system analysis and optimization

30

EE249Fall07

Mixing the two models: 802.11b

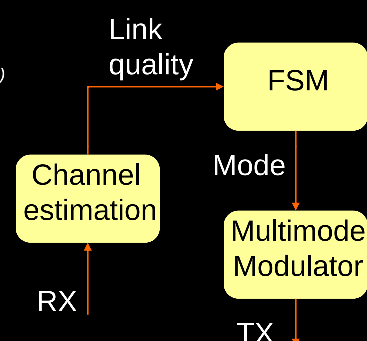
- State machine for control
  - Denotational: processes as sequence of events, sequential composition, choice etc.
  - Operational: state transition graphs
- Data Flow for signal processing
  - Functions
  - Data flow graphs
- And what happens when we put them together?

31

EE249Fall07

802.11b: Modes of operation

Data rate *(Mbit/s)*	Modulation	Coding rate	*Ndbps*	1472 byte transfer duration*(µs)*
6	BPSK	1/2	24	2012
9	BPSK	3/4	36	1344
12	QPSK	1/2	48	1008
18	QPSK	3/4	72	672
24	16-QAM	1/2	96	504
36	16-QAM	3/4	144	336
48	64-QAM	2/3	192	252
54	64-QAM	3/4	216	224



- Depending on the channel conditions, the modulation scheme changes
- It is natural to mix FSM and DF (like in figure)
- Note that now we have real-time constraints on this system (i.e. time to send 1472 bytes)

32

EE249Fall07

Reactive Real-time Systems

- Reactive Real-Time Systems
  - "React" to external environment
  - Maintain permanent interaction
  - Ideally never terminate
  - timing constraints (real-time)
- As opposed to
  - transformational systems
  - interactive systems

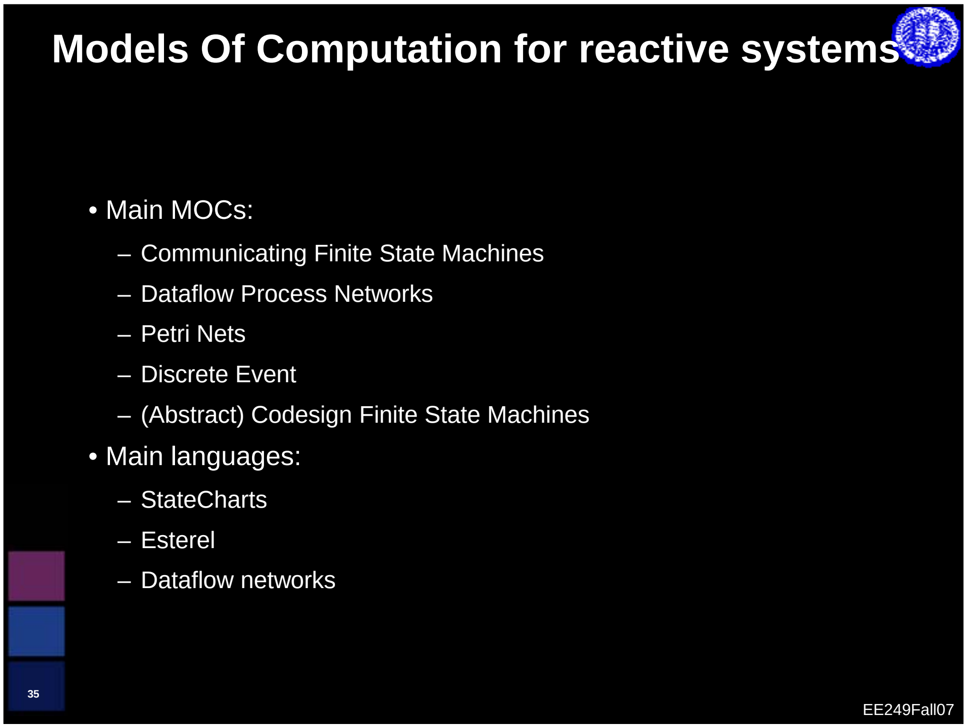
EE249Fall07

Models Of Computation for reactive systems

- We need to consider essential aspects of reactive systems:
  - time/synchronization
  - concurrency
  - heterogeneity
- Classify models based on:
  - how specify behavior
  - how specify communication
  - implementability
  - composability
  - availability of tools for validation and synthesis

34

EE249Fall07


Models Of Computation for reactive systems
• Main MOCs:
– Communicating Finite State Machines
– Dataflow Process Networks
– Petri Nets
– Discrete Event
– (Abstract) Codesign Finite State Machines
• Main languages:
– StateCharts
– Esterel
– Dataflow networks
35
EE249Fall07