COMPUTER SCIENCE

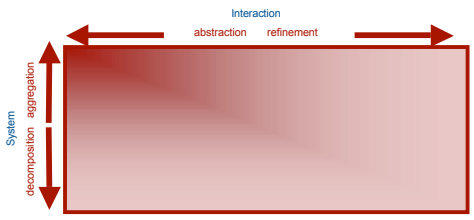
07 - Notation-State-Based

Rick Adrion

COMPUTER SCIENCE

What are we trying to represent?

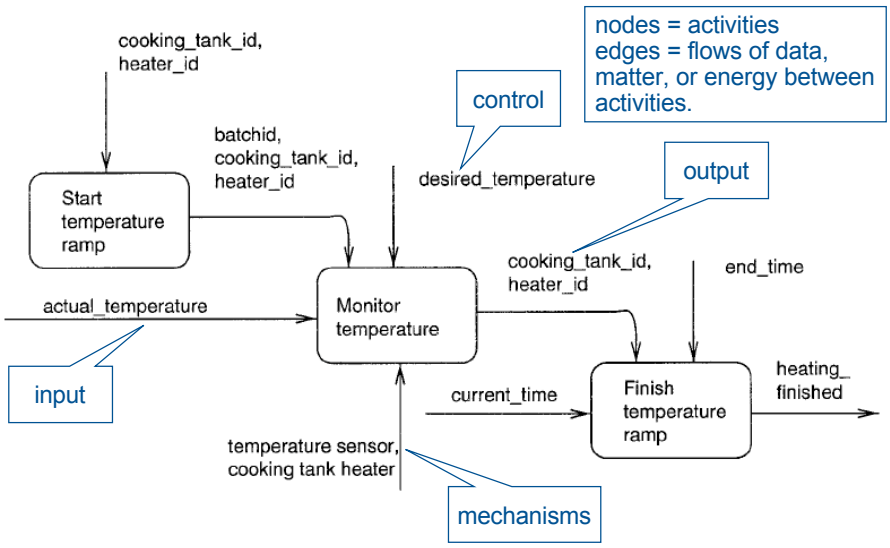
- What?
  - Decomposition
  - Communications
  - Functions
  - Behavior
- Where?
  - System level
  - Component level
- Why?
  - No one notation good for representing each
  - Need to make connections, make consistent, complete



communication specification techniques

- Context Diagrams
- SADT Activity Diagrams
- Statemate Activity Charts
- Object Communication Diagrams
- JSD System Network Diagrams
- SDL Block Diagrams
- Sequence Diagrams
- Collaboration Diagrams

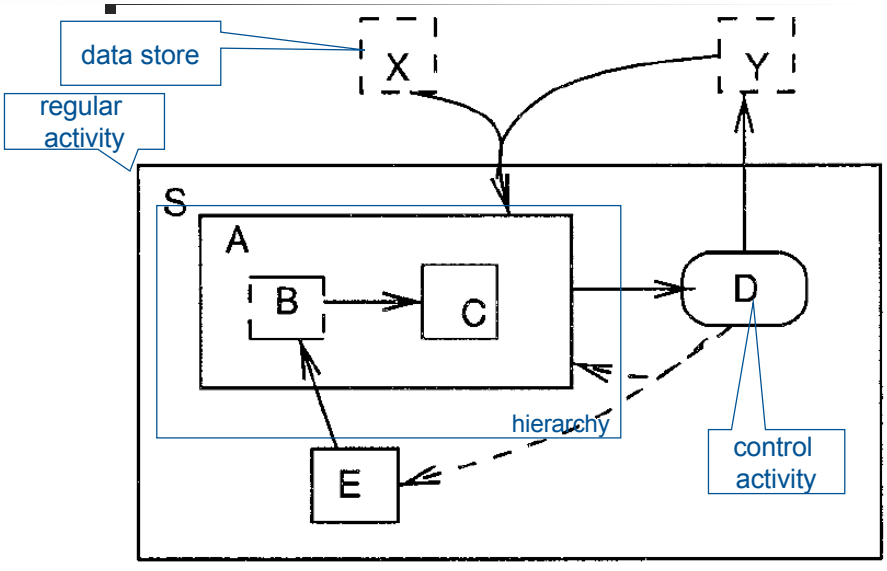
SADT Activity Diagrams



SADT Activity Diagrams

- well suited to the representation of functional decomposition
  - can be structured hierarchically
    - one activity box into a lower-level activity diagram
  - not a physical but a conceptual decomposition
- In contrast to DFDs
  - activity diagrams have only one kind of component
  - the interfaces between the activities distinguish input, output, control,and mechanism.

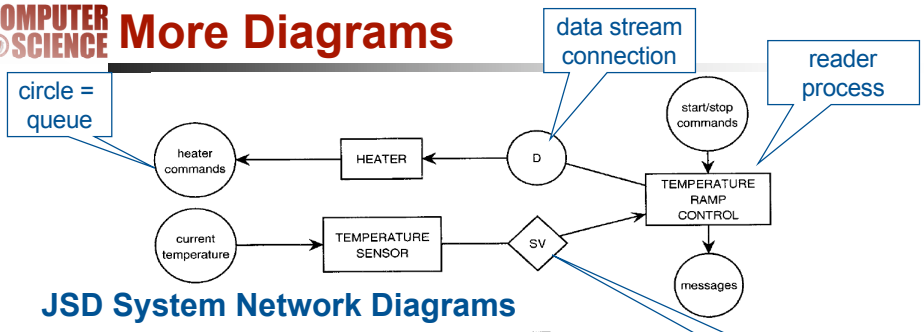
Statemate Activity Charts



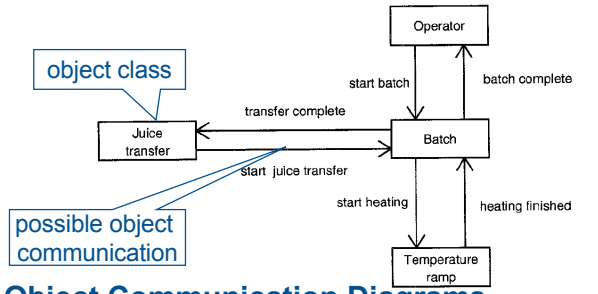
Statemate Activity Charts

- Control activities are always sub-activities of regular activities
  - a regular activity can have at most one control activity as an immediate component.
- Control activities are specified by extended finite state machines, called statecharts
  - are finite state machines extended with variables that can be updated.
  - Control activities can thus maintain a state in their variables, can perform data processing (testing and updating the variables), and can contain control.

More Diagrams



JSD System Network Diagrams



Object Communication Diagrams

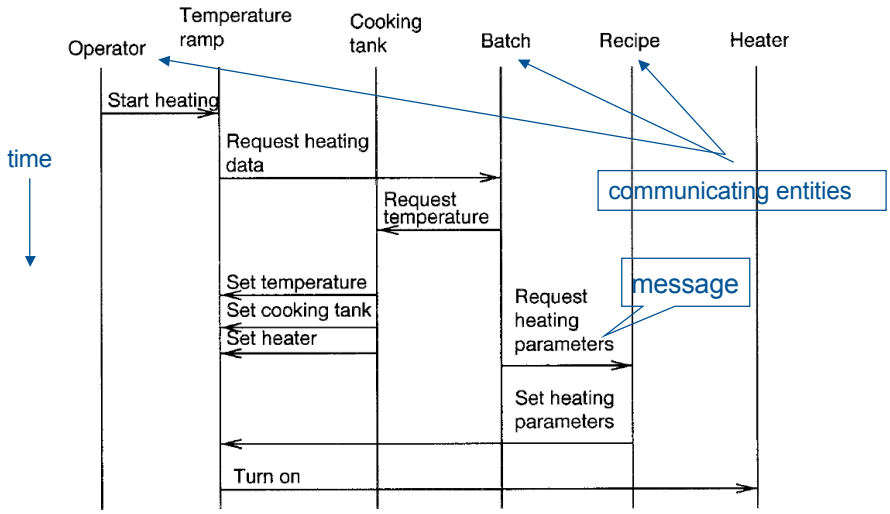
COMPUTER SCIENCE

More diagrammatic notations

- Object Communication Diagrams
  - used in the Shlaer–Mellor method to represent object communications
    - nodes represent object classes
    - edges represent possible object communications
    - temporal ordering of communications is not represented.
  - start batch shows a possible communication between an instance of Operator and an instance of Batch
- JSD System Network Diagrams
  - used in JSD to specify system functions
    - nodes represent processes
    - directed edges represent communications.
    - processes ⇒ **extended finite state machine**, specified by means of a process structure diagram
  - recognize only one kind of component, a process that maintains a state and has behavior over time
  - flat, not hierarchical

COMPUTER SCIENCE

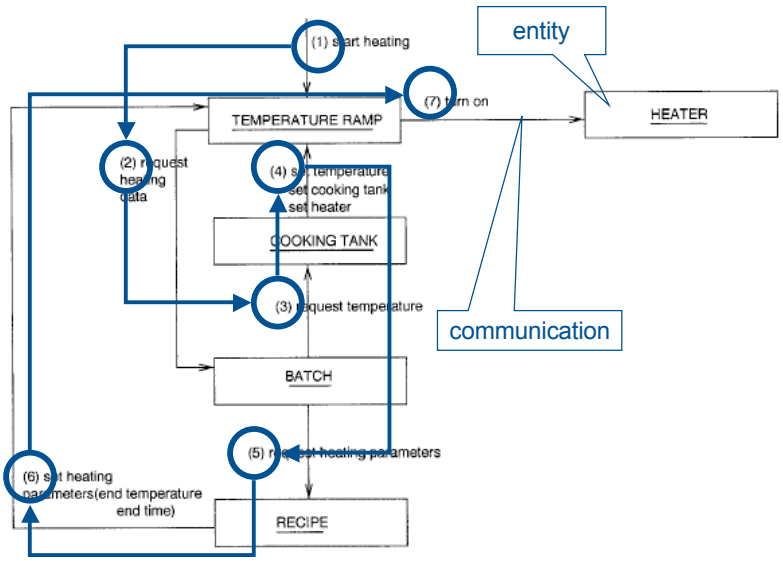
Sequence Diagrams



Sequence Diagrams

- represents a particular sequence of messages exchanged between a number of entities
- standardized as message sequence charts in telecom
- popular in object-oriented methods to represent communications between objects
- shows one particular communication sequence in one run of the system
  - show behavior as well as communication
- can be extended with conventions to represent timeouts, global conditions across different entities, delayed message reception, etc.

Collaboration Diagrams



Collaboration Diagrams

- popular in object-oriented methods to represent message exchanges between objects
- UML extended with annotations that represent dataflows between the communicating objects and various notations that represent the way in which the communications are implemented.
- differ from other notations
  - nodes represent objects, not activities (DFDs, activity diagrams, activity charts, and block diagrams)
  - nodes represent objects, not object classes (object communication diagrams
- as in sequence diagrams, represent the sequence of messages in **one particular scenario** whereas all other communication diagrams represent possible communications in **all possible scenarios**.

Representing communication

- two kinds of diagrams to represent communication
  - diagrams that show communication sequence (sequence diagrams and collaboration diagrams)
    - illustrate communications as well as behavior in one particular run of the system
  - diagrams that show a set of possible communications without indicating any sequence (all other diagrams). Diagrams
    - show communication only and do not refer to a particular run of the system.
- the kinds of things that can communicate (conceptual components)
  - .Finite state machines
    - control processes in DFDs are specified by finite state machines.
  - Extended finite state machines
    - contain control, local variables, and tests and updates of these variables.

function specification techniques

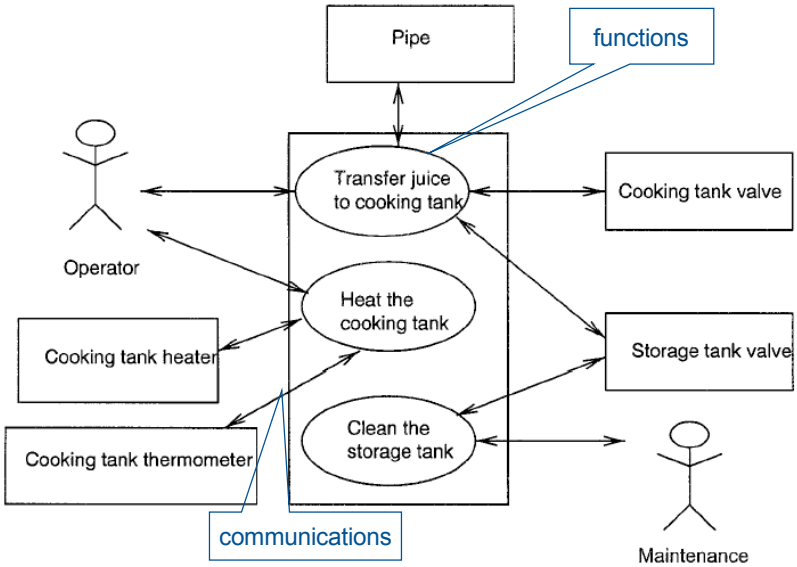
- used to specify the external functions of a system
- types
  - Function Refinement Trees
  - Event-Response Specification
  - Use Case Diagrams

Use Case diagrams

- represent external system functionality
- a use case
  - an interaction between the system and an external entity that has a use for that external entity
  - need not be atomic
  - can be described by
    - a narrative text
    - by a specification of pre- and post-conditions
    - a state transition diagram.

COMPUTER SCIENCE

Use Case diagrams



COMPUTER SCIENCE

Use Case schema

Use case: Heat Cooking Tank

Description: Heat a cooking tank to the temperature prescribed by the recipe of the juice to be mixed, and keep it at that temperature for the time prescribed by the recipe.

Reads: Cooking tank, Batch, Recipe.

Changes:

In: Operator: Batch ID.
Thermometer: Current temperature.

communications

Out: Thermometer: Temperature request.
Heater: switch on, switch off.
Operator: heating finished.

functions

Assumes: Juice is present in tank.

Results: Juice has been kept at desired temperature for the desired time.

Transactions:

- The system switched on the cooking tank heater.
- The system checks the cooking tank thermometer every 10 seconds. When the desired temperature is reached, the heater is switched off, when the temperature is too low, the heater is switched on again.
- When the end temperature is reached, the heater is switched off and a message is sent to the operator.

function specification techniques

- each technique draws attention to a different aspect of external system functionality
  - no need for formality
  - follow good heuristics
  - don't put more meaning in these diagrams than is intended
- function refinement tree
  - relates the overall mission of a system to its functions down to its atomic transactions.
  - shows why the system must have that function
- Use case diagrams
  - summarize the communications that take place during particular functions.
- context diagram
  - can be used to specify the data exchanges between the system and its environment.
  - draw one partial context diagram for each use case and one for each major system function

behavior specification techniques

- show how functions of a system or of its components are ordered in time
- Types
  - Process Graphs
  - JSD Process Structure Diagrams
  - Finite State Diagrams
  - Extended Finite State Diagrams
  - Mealy Machines
  - Moore Machines
  - Statecharts
  - SDL State Diagrams
  - Process Dependency Diagrams

COMPUTER SCIENCE

Finite State Machines (FSM's)

- FSM's describe behavior of a system:
  - The sequence of stages/steps/conditions that the system goes through
  - FSM shows how a system acts/reacts to inputs
  - Does this by showing progress through different states
- Hypothesis:
  - The universe in which the system being described must operate can be accurately modeled as always being in exactly one of a finite number of states (situations)
  - There are only a finite number of possible system inputs

COMPUTER SCIENCE

Finite State Machines (FSM)

- finite set of states S ={s_1, ... , s_n}
- finite set of inputs I = {i_1, ..., i_n}
- transition function δ: SxI $\Rightarrow$ S
  - can be a partial function
- represent as a graph
  - nodes $\Rightarrow$ states
  - edges $\Rightarrow$ inputs
  - graph $\Rightarrow$ transition function

Why Use FSM's?

- Primary appeal is visualization
- Intuitively: Can "watch" a stream of inputs "drive" the behavior of the system as a sequence of movements from state to state
- Kinds of questions FSM's seem adept at helping answer:
  - "What is a good way to think about the problem to be solved?"
  - "What is the solution approach?"
  - "How does this program work?"

Enhancements to FSM's

- Use of hierarchy
- Output annotations on edges
- Distinguished Initial and Terminal states
- Separate data definitions, local and global variables

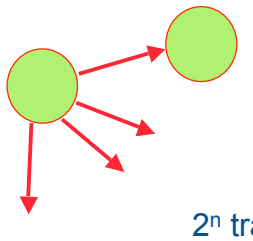
What is FSM good/not good for?

- Focus on specific issue: safety concern
  - Model unsafe state
  - Model state transitions
  - Can unsafe state be reached?
- Drawbacks
  - No sense of functionality
  - No sense of how functionality achieved
  - Difficult and generally impossible to reason about timing

FSM are limited

- Library example
- getbook: index X library ⇒ book
- $1{,}000{,}000^{+}$ books in a good library

state = books on shelf

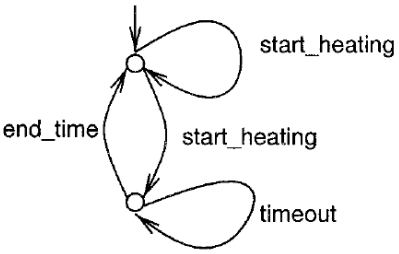


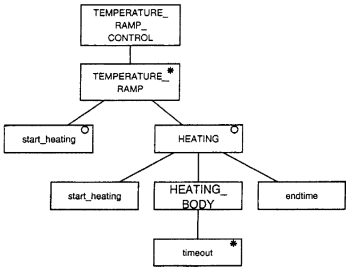
new state = libe - book

2^n transitions could be >> 10^6 states

Some FSM-based notations

- Process Graphs
  - nondeterministic
  - may have infinitely many nodes
  - from any node, infinitely many edges may depart
  - used as interpretation structures for formal specifications in process algebra and dynamic logic
- JSD Process Structure Diagrams.
  - used in Jackson Structured Programming (JSP)
    - represent the structure of files and of regular programs
  - used in JSD
    - represent the behavior of a system in a modular way
  - a visual way to represent a regular expression by means of a tree diagram





FSM

- Finite State Diagrams
  - contain only finitely many states and transitions
- Extended Finite State Diagrams.
  - number of states can be increased by introducing variables that may be tested and updated by the finite state machine
  - *global state* = *explicit state* (STD nodes) + *extended state* (variables)
  - local variables or external variables
    - local variable is declared together with the specification of the STD + scope rules for these variables (*usually the entire state machine specification*)
    - external variable is declared outside the specification of the STD but can be accessed by means of special operations that act as an interface between the specification and the variables
    - in dataflow models, data stores are external variables with respect to the control processes in the DFD
    - global state change requires communication between the state machine and the external data stores

Extended FSM

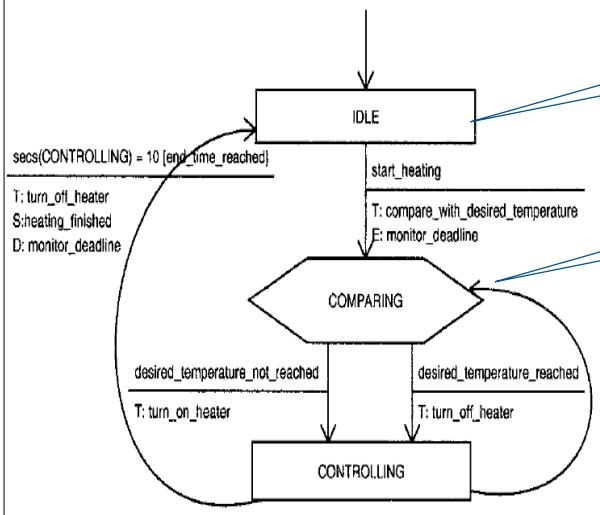
- state transition may change the values of variables
- a guard may be specified for each transition that says when the transition can occur.
  - weak interpretation,
    - the transition cannot occur if the guard is false
    - guard is in this case a necessary condition of the transition
  - strong interpretation,
    - the transition can occur if and only if the guard is true
    - guard is a necessary and sufficient condition of the transition
  - usually initially specify guards with the weak semantics
    - when all conditions for a transition are specified, interpret the conjunction of all weak guards as a strong guard
    - a guard could be the conjunction of all preconditions specified for a transition
- include tests in a state machine that are used to determine the next state
  - such tests can be used to resolve nondeterminism
  - a test determines which of a set of possible transitions will occur, thus a test consists of a guard for each of the possible transitions.

Mealy & Moore Machines

- Mealy machine
  - output actions are associated with transitions
- Moore Machines
  - outputs are associated with states

COMPUTER SCIENCE

Mealy FSM for the juice plant controller



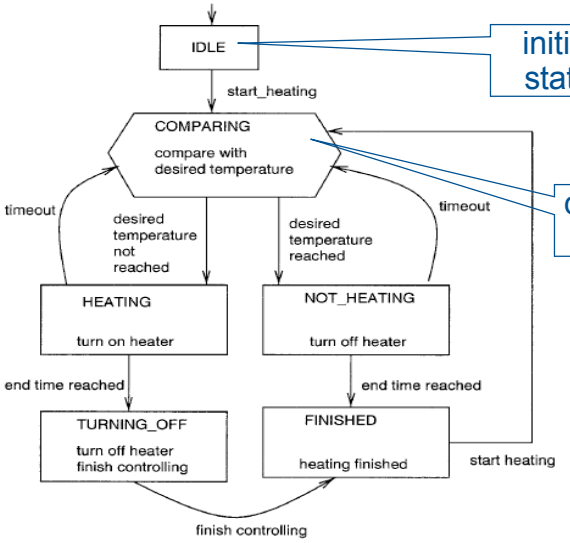
initial state

decision state

In YSM
- Create and set named clocks
- Each state has a clock that counts the time that has elapsed since the machine last entered that state

COMPUTER SCIENCE

Moore FSM for the juice plant controller

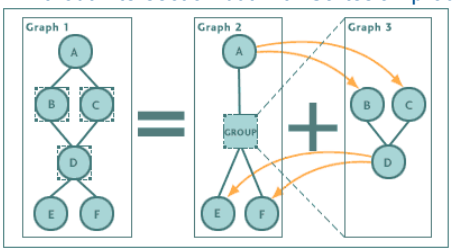


initial state

decision state

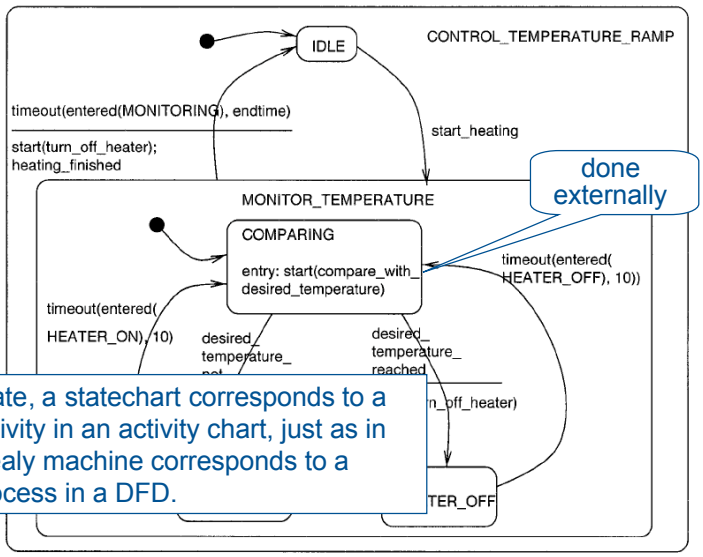
Statecharts

- higraph without intersection but with Cartesian products



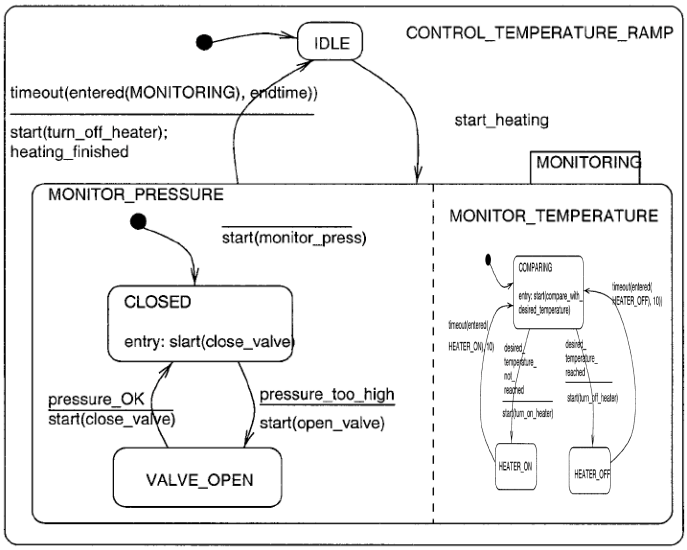
- node inclusion allows us to partition a state into substates
- Cartesian products allow us to specify parallelism
- actions can be specified
  - along transitions (Mealy)
  - upon entry of states (Moore), and
  - exit of states
- local variables represent the extended state.

Statecharts



In Statemate, a statechart corresponds to a control activity in an activity chart, just as in YSM a Mealy machine corresponds to a control process in a DFD.

Statechart with parallelism



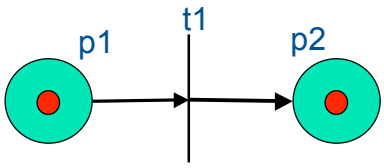
FSMs

- different notations are very similar
  - all represent states by labeled nodes and transitions by labeled directed (hyper)edges
  - all except process dependency diagrams have a formal semantics
    - many different possible semantics of statecharts and statechart-like notations
- time in state machines.
  - point or interval semantics of time
  - discrete, dense, or continuous model of time, and assume that transitions take time or are instantaneous
  - these choices leads to important differences in the behavior specified by a state diagram
  - semantics of time is carefully defined in SDL and in the Statemate semantics of statecharts.

COMPUTER SCIENCE

Petri Nets

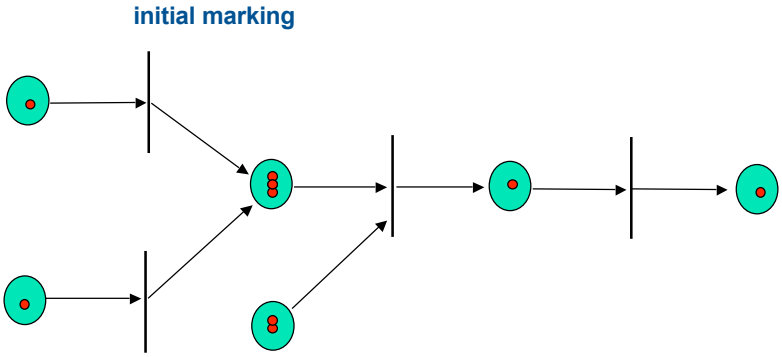
- Petri nets are "marked" graphs
  - two node types: places & transitions
  - tokens mark the nodes
  - transitions are enabled ("fire") if all connected places contain tokens



COMPUTER SCIENCE

Petri Nets

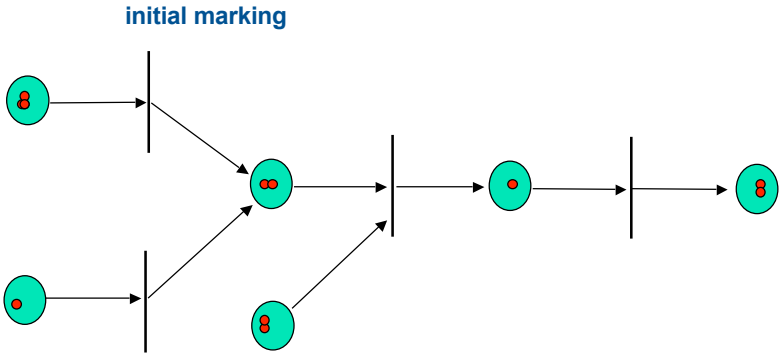
- Example - simultaneous firing



COMPUTER SCIENCE

Petri Nets

- Example - asynchronous, atomic firing



COMPUTER SCIENCE

Petri Nets: Informal Definition

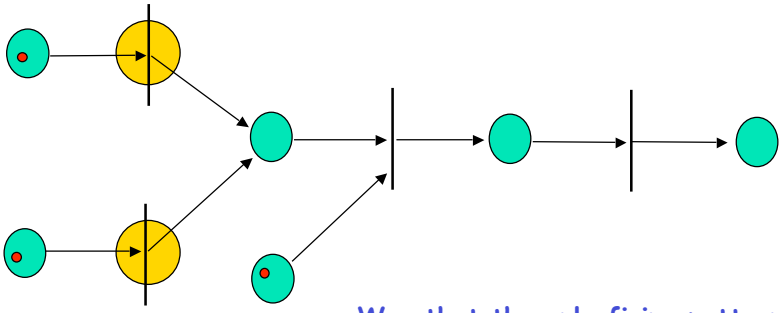
- Designed specifically for modeling systems with interacting concurrent components.
- Consists of a set of places and a set of transitions
  - Edges connect places and transitions.
  - Only transition → place and place → transition links are allowed.
- Each place can have a finite number of tokens.
- A transition is enabled if each of its input places has at least one token.
  - An enabled transition can fire: one token is taken from each input place and one token is put into each output place.

COMPUTER SCIENCE

Petri Nets

- Example - asynchronous, atomic firing

initial marking

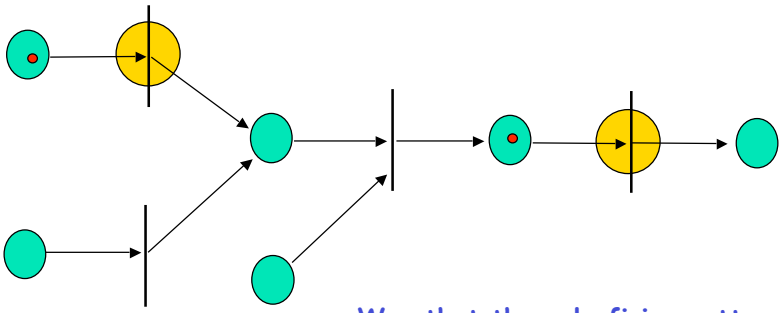


Was that the only firing pattern?

COMPUTER SCIENCE

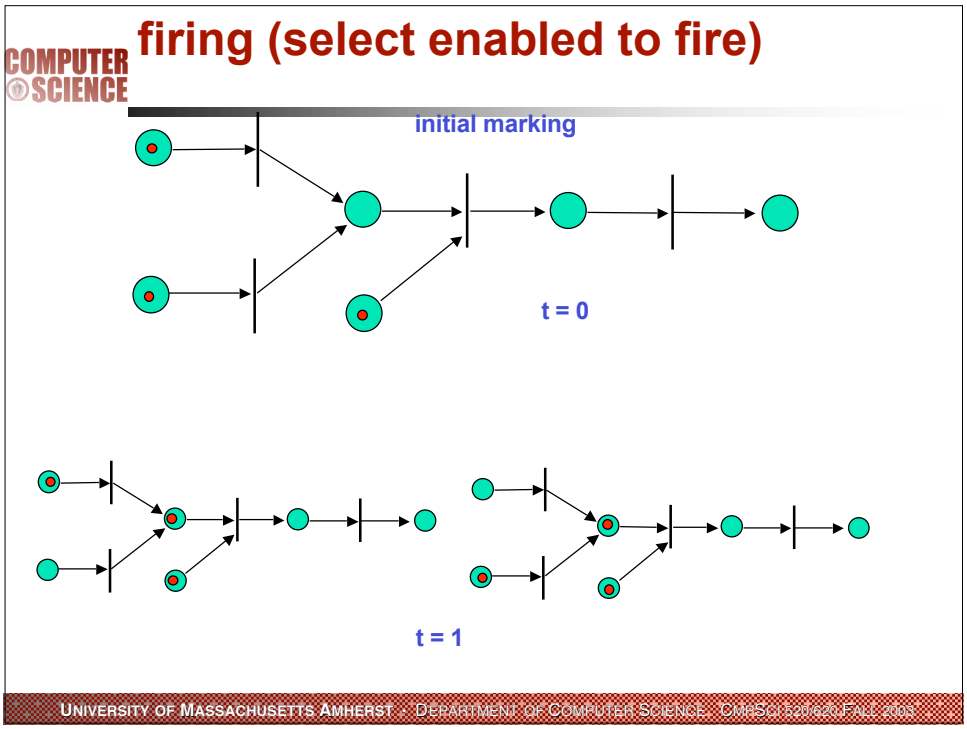
Petri Nets

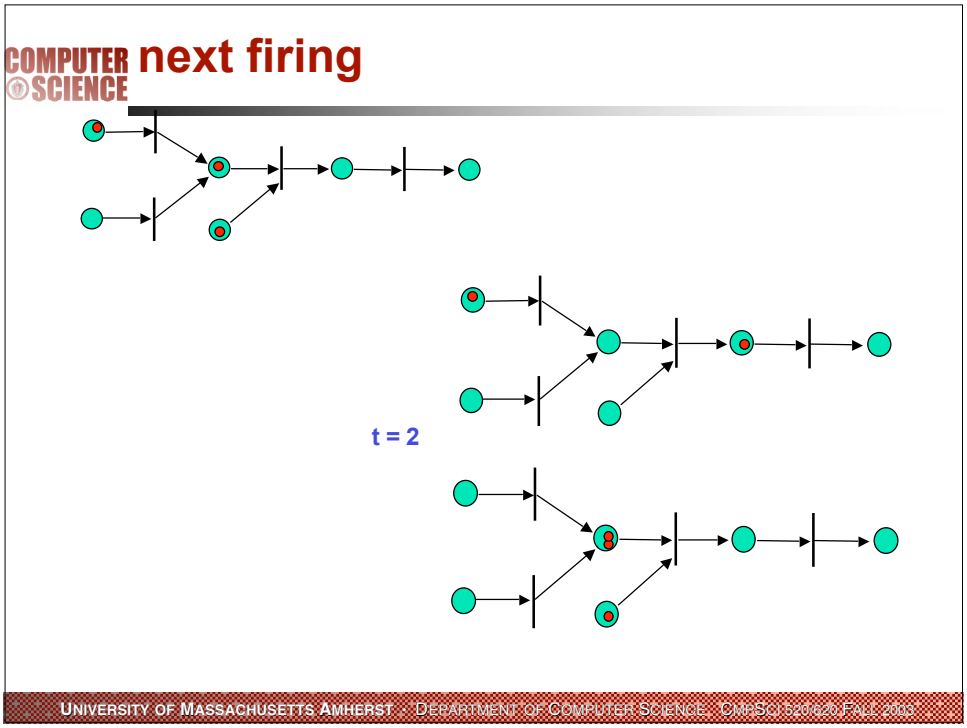
- Example - asynchronous, atomic firing

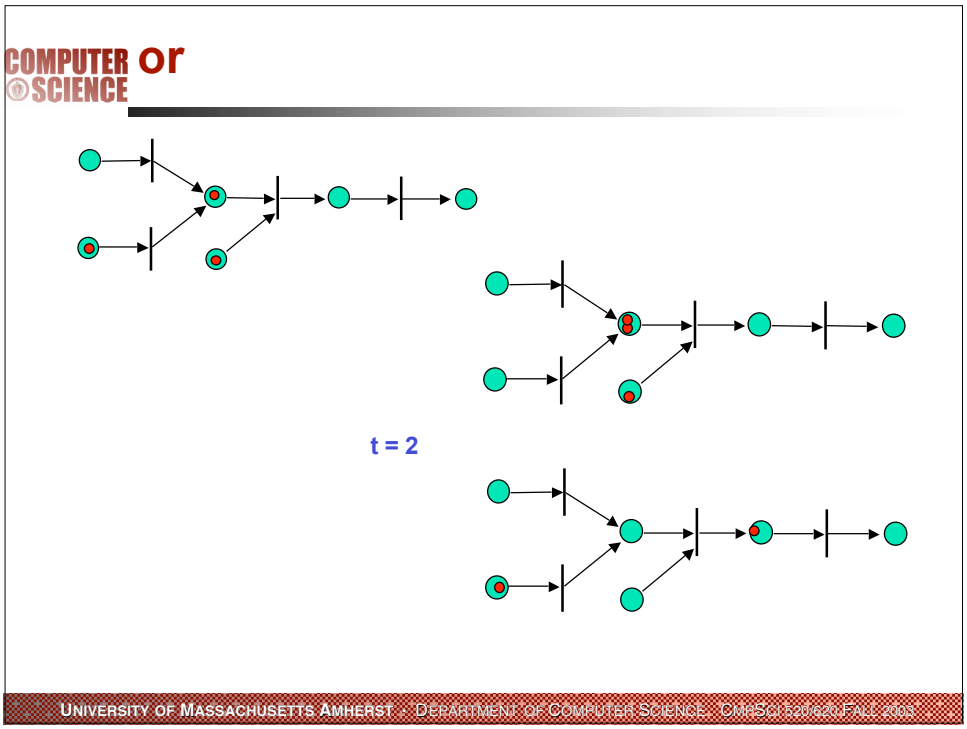


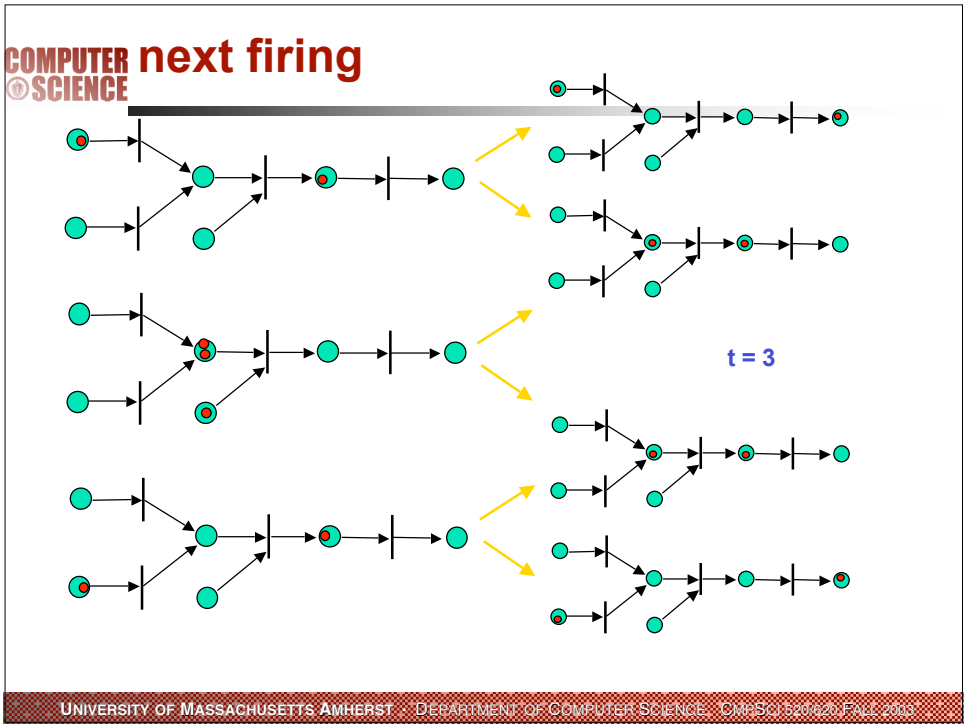
Was that the only firing pattern?

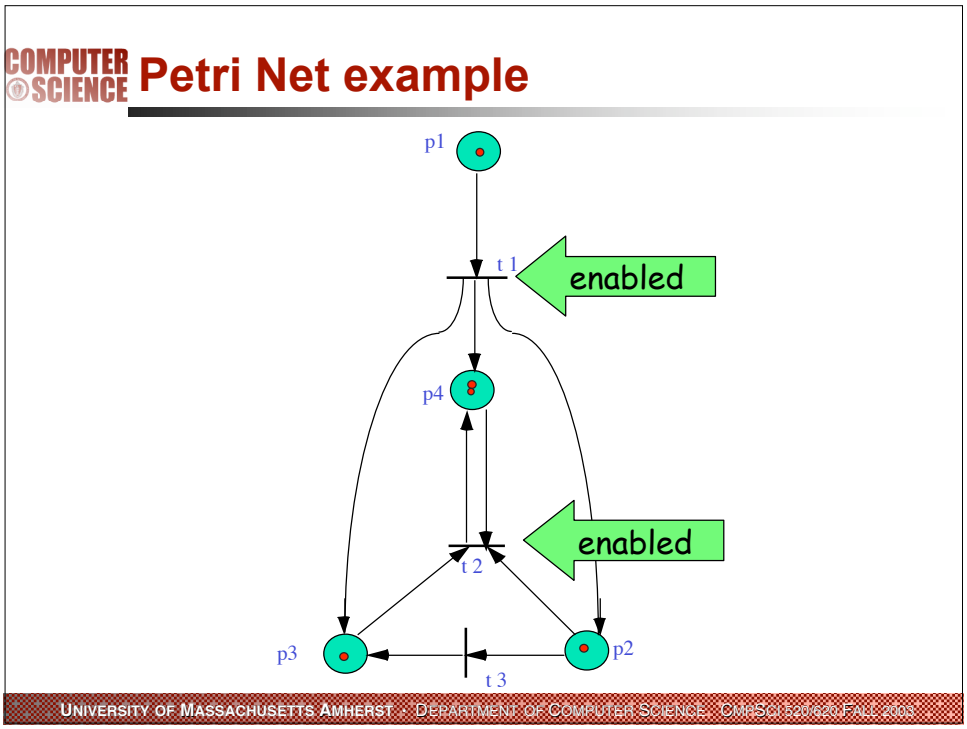
UNIVERSITY OF MASSACHUSETTS AMHERST · DEPARTMENT OF COMPUTER SCIENCE · CMPSCI 520/620 FALL 2003

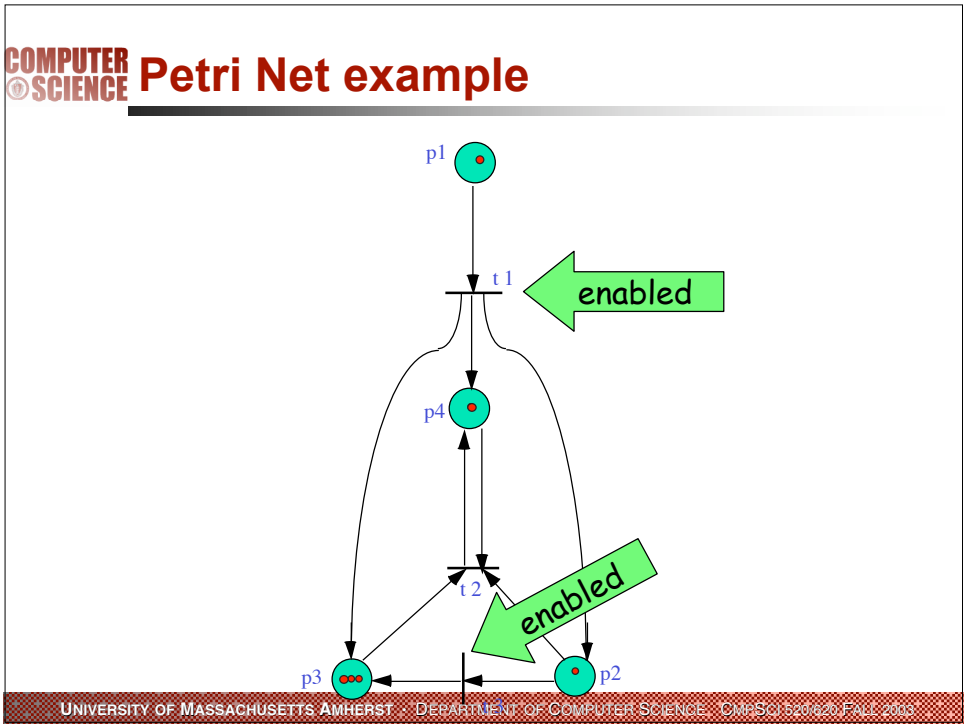






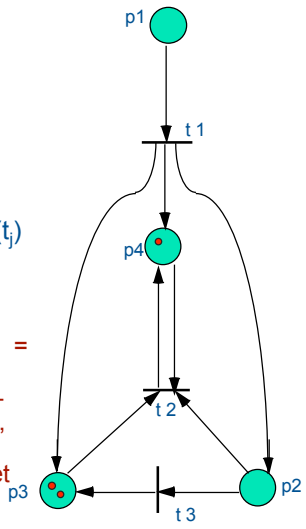






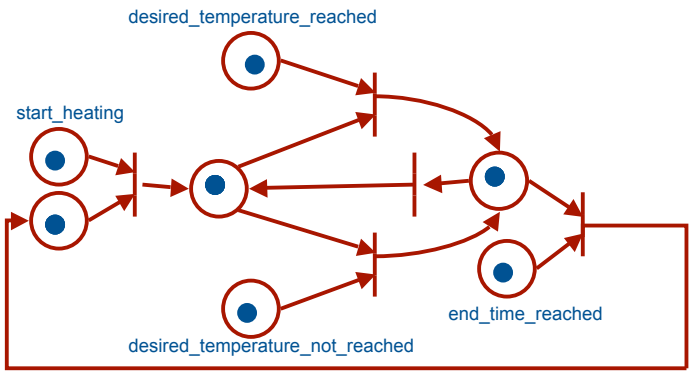
Petri Nets: Formal Definition

- A Petri Net is a four-tuple, C=(P,T,I,O)
- P = {p_1, p_2, ..., p_n}, n ≥ 0 is a finite set of places.
- T = {t_1, t_2, ..., t_m}, m ≥ 0 is a finite set of transitions.
  - I: T → P is the input function.
  - O: T → P is the output function.
- p_i is an input place of a transition t_j if $p_i \in I(t_j)$
- p_i is an output place of a transition t_j if $p_i \in O(t_j)$
- Petri Net markings
  - A marking m is a mapping P → N where N = 0, 1, 2, ...
  - The marking m can be represented as a n-vector m = (m_1, m_2, ...m_n), n = |P |, $m_i \in N$, $1 \le i \le n$
  - A marked Petri net M = (C, m) is a Petri net C and a marking m .



marking (0,0,2,1)

Petri Net for Heating Controller



COMPUTER SCIENCE

More on Petri nets

- if there exists a marking which is reachable from the initial marking where no transitions are enabled, such a transition is called a "deadlock"
- a PN with no possible deadlock is said to be live, called the "liveness property"
- in simplest PN, tokens are uninterpreted
  - in general, a selection policy can not be specified
  - have no "policy" for resolving conflicts, potential "starvation"
- many extensions:
  - Hierarchical Petri Nets
  - Colored tokens
  - "Or" transitions
  - Queues at places

COMPUTER SCIENCE

Petri Nets vs. Finite State Machines

- For any finite state machine, a Petri net can be built that models the finite state machine
  - Petri nets are as powerful as finite state machines
- Petri nets advantages:
  - net composition (in different forms) can be found easier than the cross-product of finite state machines
  - parallelism and nondeterminism are represented in a more understandable way
- Finite state machines advantage:
  - simpler graph structure for some applications (e.g. parsers)