Embedded Systems
Lecture 3: Models of Computation

Björn Franke
University of Edinburgh
Overview

• Introduction

• Dataflow Diagrams, Decision Tables,

• Finite State Machines (FSM)

• Synchronous/Asynchronous FSM

• Extensions to FSM for Embedded Specification
Motivation

• Why considering specs and models in detail?

• If something is wrong with the specification, then it will be difficult to get the design right, potentially wasting a lot of time.

• Typically, we work with models of the system under design (SUD)

• Most actual systems require more objects: Hierarchy (+ abstraction)
  
  • Behavioural hierarchy: states, processes, procedures
  
  • Structural hierarchy: processors, racks, printed circuit boards
Models of Computation

What does it mean, “to compute”? Models of computation define:

- Components and an execution model for computations for each component
- Communication model for exchange of information between components.
Requirements

- Presence of programming elements
- Executability (no algebraic specification)
- Support for the design of large systems (e.g. OO)
- Domain-specific support
- Readability
- Portability and flexibility
- Termination
- Support for non-standard I/O devices
- Non-functional properties
- Support for the design of dependable systems
- No obstacles for efficient implementation
- Adequate model of computation
Models of Computation

- Process Networks
- Threads
- Message Passing
- Synchronous/Reactive (SR)
- Concurrent State Machines (Statecharts and variants)
- Dataflow
- Rendezvous-based Models (CSP, CCS)
- Time-triggered Models
- Discrete-event Models
- Continuous-time with ODE solvers
Problems with Conventional Thread Model

• Even the core … notion of “computable” is at odds with the requirements of embedded software.

• In this notion, useful computation terminates, but termination is undecidable.

• In embedded software, termination is failure.

• However, to get predictable timing, subcomputations must terminate (and we must be able to decide whether or not they terminate)
Imperative and Declarative Models

• Imperative
  • Give algorithmic descriptions of behaviour which are directly executable.
  • Easy to produce examples and debug specifications
  • Allows fast prototyping & implementation of systems.
  • Examples: Data Flow Diagrams (DFDs), Statecharts, Tabular Languages

• Declarative
  • Specify properties that must be satisfied, not executable. Based on logic
  • Normally easier to state & prove properties, but more difficult for design.
  • Examples: traditional logics - predicate & temporal; real-time logic
Dataflow Diagrams (DFD)

Data Flow

Function

Input

Output

Storage

Temperature

Plus

Display

Airspace_Status

plane_id

plane_id

Position

Speed

Space_Status

a

b

c

z
Merits:
• Focuses on fundamental elements of application & data flow between them

Drawbacks:
• Scalability - DFDs for large applications can blow up; however, can be split into smaller, more detailed components
• Definitions ambiguous mainly because of informality - inputs arrive simultaneously? how are reads/writes handled?
• Absence of control - when to trigger a function? for conditional executions, is it correct to execute a function?
**Decision Tables**

- **$j^{th}$ rule reads:** if $Conditions_j$ then $Action_j$

- **$Column_j$ evaluates to True or False, depending on value of:**

$$((c_{1j} = Y \text{ and } C_1) \text{ or } (c_{1j} = N \text{ and not } (C_1)))$$

and ... and

$$((c_{nj} = Y \text{ and } C_n) \text{ or } (c_{nj} = N \text{ and not } (C_n)))$$

- **if** $a_{1j} = X$ **then do** $A_1$;

  $:\$

  if $a_{mj} = X$ **then do** $A_m$;
State Machines

• Different forms of state machines are in use for modelling & designing systems

• Standard Finite State Machine (FSM) comprises

  • a finite number of states

  • a next state function which maps states & events into states

  • FSM starts executing in its start state, moves from one state to another as per next state function, until it reaches halt state or exhausts input

• Two types of FSMs (both equivalent): Moore & Mealy

  • Moore FSM: Output = f(current state)

  • Mealy FSM: Output = f(current state, inputs)
Synchronous FSM

- There is a separate synchronising clock signal
- Current state & inputs examined only at active instant in clock cycle
  - Typically rising edge
- State changes only once in each clock cycle
- For Mealy machine, output is, typically, instantaneous function of inputs & current state
- Include start signal as input
Asynchronous FSM

• State responds immediately to input, so need some other way to identify each new input

• Model assumes that inputs do not change until machine settles into its new state

• Common to describe an FSM using a state diagram:
  • a labelled directed graph
  • nodes represent states
  • arcs represent transitions
FSM Example - Railway Crossing Gate
FSM Limitations & Solutions

• Limited descriptive power - e.g. can’t recognise balanced parentheses

• Pure FSMs cannot model applications which produce output - Mealy machines

• More powerful version of state machine allows guards, inputs, outputs & actions on transitions: g → i/a/o

  • g - guard (boolean expression, assertion or condition)
  • i - input (e.g. event)
  • a - sequence of actions
  • o - output

• If machine is in state U, guard g is true & input i occurs, then perform actions a, generate output o & enter state V
Extensions to FSMs for Embedded Specifications

• Need to be able to model concurrency & time

• Modelling concurrency:
  • allow several FSMs to run in parallel
  • describe communication & synchronisation between them
  • make use of shared/distributed memory model

• Modelling timing constraints:
  • specify transition firing times
  • clocks & timing events
  • Need to address problem of state explosion
Kahn process networks (KPN)

- Distributed Model of Computation
  - Group of deterministic sequential processes
  - Communicating through unbounded FIFO channels
- KPN exhibits deterministic behaviour
  - Does not depend on the various computation or communication delays
- Common model for describing signal processing systems
  - Infinite streams of data are incrementally transformed by processes executing in sequence or parallel
A Kahn process network of three processes without feedback communication. Edges A, B and C are communication channels. One of the processes is named process P.
MoC Overview Chart

<table>
<thead>
<tr>
<th>Communication/local computations</th>
<th>Shared memory</th>
<th>Message passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined components</td>
<td>Plain text, use cases</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Communicating finite state machines</td>
<td>StateCharts</td>
<td>SDL</td>
</tr>
<tr>
<td>Data flow</td>
<td>Scoreboarding + Tomasulo Algorithm (≠ Comp.Archict.)</td>
<td>Kahn networks, SDF</td>
</tr>
<tr>
<td>Petri nets</td>
<td></td>
<td>C/E nets, P/T nets, …</td>
</tr>
<tr>
<td>Discrete event (DE) model</td>
<td>VHDL*, Verilog*, SystemC*, …</td>
<td>Only experimental systems, e.g. distributed DE in Ptolemy</td>
</tr>
<tr>
<td>Von Neumann model</td>
<td>C, C++, Java</td>
<td>C, C++, Java with libraries CSP, ADA</td>
</tr>
</tbody>
</table>
Summary

- Introduction to MoC
- Dataflow Diagrams, Decision Tables
- Finite State Machines (Sync./Async.)
- Model of Computation Comparison
Preview

- Statecharts
- Coursework