“Investigating a Finite–State Machine Notation for Discrete–Event Systems”

Nikolay Stoimenov

Outline

- David Knight’s notation – SAFES
  - SAFES – State Automata For Embedded Systems
- Background
  - Finite-State Machines FSM: output, hierarchy, communication
  - Languages with FSM: UML, SDL
- Syntax and Semantics of SAFES
- Implementation with Moses
- Demo
- Future work
David Knight's notation – SAFES

- David Knight has been developing hardware and software for small embedded systems for more than 20 years
  - Developed a notation for specifying software which is easily translated to code
  - Based on software components which encapsulate some behaviour
  - Software components are specified as finite-state machines
  - Can be connected together in a system in the same way as hardware components
SAFES Component as Text

dynamic
machine mName;
initial
    initialisationActions;
    next s1;
state s1:
    entryActions;
    when event1(int b)
        doThis;
        next s2;
    when event2();
        doThat;
        next s1;
endwhen;
state s2:
    etc...
endmachine;
endcomponent;

Formalising SAFES

- Need to formalise David's notation in order to be able to:
  - Build a tool for drawing diagrams
  - Build a code generator
  - Build a simulator
  - Build a tool for performing verification of components
Finite-State Machines FSM

- Idea developed in the 1940s by Pitts and McCulloch for modelling nerve-nets
- Further developed by Moore and Mealy for modelling hardware
- Today widely used in lexical analysers, text editors, models describing system behaviour and hardware
FSM with Output

- Two types: Mealy and Moore
- Moore machines associate output with the states
- Example of a counter mod 3 for binary integers:

![FSM Diagram](image1.png)

Background

FSM with Output 2

- Mealy Machines associate output with transitions
- More expressive?
- Actually equivalent to Moore Machines

![FSM Diagram](image2.png)

Background
FSM with Output 3

- Disadvantages of the two models
  - In software, there is usually need to have output associated with both transitions and states
  - A combination of Mealy and Moore type model
  - Similar to Entry/Exit Actions in UML Statecharts, also called “first wishes/last wishes”

Statecharts

- Statecharts (hierarchical FSM) were proposed by David Harel in the 1980s
- They were addressing problems with the specification and design of large reactive systems
- Any state in such a Statechart can be another Statechart
- Level-by-level system description
Communicating FSM

- Communicating FSM were developed initially for modelling communication protocols
- Each process (e.g. user, server) is represented by an FSM and all of them are connected with FIFO channels
- The channels are usually unbounded queues
Communicating FSM 2

Unified Modeling Language

- Unified Modeling Language UML is a set of notations developed in the early 1990s to model complex mission-critical systems using object-oriented techniques
  - It uses the idea of Statecharts to model the dynamic behaviour of systems
  - Even though specification for UML Statecharts exist, it fails to interpret complex Statechart models
Specification and Description Language

- Specification and Description Language SDL was developed in the 1970s by ITU (CCITT)
  - It is used for modelling large telecommunication systems
  - It is based on the idea of Communicating FSM
  - Very rich, constantly growing language. It took 10 years for semantics to be formalised

David Knight's Notation – SAFES

- Characteristics:
  - Combines Mealy and Moore models
  - Components are reactive and communicate with events
  - Events can carry data
  - Components can be connected only if the events they can input/output agree in type
  - Input/output events can be grouped into interfaces and a component can implement several interfaces
  - Communication channels have buffers of size 1
SAFES Syntax

- An individual FSM is a tuple: \((Q, T, q_0, I, O, V)\)
  - \(Q\) – a finite set of states
  - \(T\) – a finite set of transitions
  - \(q_0\) – an initial state
  - \(I\) – a finite set of input ports
  - \(O\) – a finite set of output ports
  - \(V\) – a finite set of variables
SAFES Syntax 3

- A system of communicating FSMs is a tuple: (F, C)
  - F – a finite set of individual FSMs
  - C – a finite set of connectors
  - A connector is a finite set of links
  - Each link connects one input port to one output port in the system

SAFES Syntax 4

[Diagram showing connections between components and connectors]
**SAFES Semantics**

- Semantics of an FSM are specified by specifying an Extended State ES for the machine and transitions which can change the ES.
- The ES is a tuple: \((q_c, V, I)\)
  - \(q_c\) – current state node in the FSM
  - \(V\) – the set of variables of the FSM
  - \(I\) – the set of input ports of the FSM
- There is only one transition which can change the ES: receiving an input event.
- The semantics of a system of FSMs are specified as a Kahn Process Network.

**MOSES**

- Specify syntax and formal semantics for the notation
  - Model it in an environment which supports domain specific visual formalisms e.g. The Moses Tool Suite
  - The specification of syntax in Moses is divided into two parts:
    - definition of the visual elements (e.g. colours, shapes, line types) and their attributes
    - specification of well-formedness of a picture in the language
MOSES 2

- Specifying visual language semantics is not a trivial problem
  - Moses allows an interpreter to be specified
  - The interpreter is specified as an Abstract State Machine ASM, a very general language for specifying operational semantics
  - ASMs are very simple yet very expressive having simple formal semantics that are amenable to formal analysis

MOSES 3

- Model checking is the process of verifying that an algorithm satisfies certain properties
- Basic model checking for SAFES has been implemented in Moses for checking safety properties only
- A safety property expresses that, under certain conditions, an event never occurs
Demo

Identified issues

- Concurrency and scheduling – SAFES have been used only on single CPU machines so far
- Data structures – component which is a data structure and can be used concurrently by other components
- Procedures – synchronous communication
- Size 1 buffers – loss of events if two events arrive at the same time
Future Work

- Notation
- Moses

Questions and Answers

- Thank You