02917: Advanced Topics in Embedded Systems

Martin Fränzle
Andreas Eggers

Carl von Ossietzky Universität
Dpt. of Computing Science
Res. Group Hybrid Systems
Oldenburg, Germany
Multiple viewpoints

Requirements analysis

Programming

Aspects
"What?"

Algorithmics
"How?"

Tests & proofs
"Consistent?"

Validation / verification

Consistent?
Formal Methods

- Formal methods are mathematically-based techniques for the specification, development and verification of software and hardware systems. [R.W. Butler, 2001]

- Motivated by the expectation that appropriate mathematical analyses can contribute to the reliability and robustness of a design. [M. Holloway, 1997]

- Alternative to less exhaustive analyses:
Embedded computer systems

Cogito, ergo sum!

Estimates for number of embedded systems in current use exceed $10^{10}$.

[Rammig 2000, Motorola 2001]
Application domains

- **Consumer & household products:**
  - CD players, TV sets, handheld games, electronic pets, cameras, alarm clocks, remote controls, dishwashers, microwave ovens, ...

- **Office, telecommunications, etc.:**
  - Printers, network controllers, mobile phones, keyboards, CRTs and flatscreens, ...

- **Environmental control:**
  - λ control, programmable heating systems, exhaust control, ...

- **Traffic systems and traffic management:**
  - Cars (body, powertrain, suspension, brakes), signalling devices, balises, interlocks, autopilots, traffic information, ...

- **Medicine:**
  - Measurement devices (thermometers, RR’s, X-ray, sonographic imaging, EEG, ECG ...), treatment devices (perfusors, respirators, microwave radiation treatment, ...) 

- **Supplies:**
  - Power plants, distribution networks, ...
The roles, they are a changing...

Phase 1: Added value through add-on functionality:

- automatic climate control,
- adaptive power steering,
- keyless entry,
- navigation system.

Phase 2: Integration into/hooking onto vital safety components:

- anti-locking brake,
- electronic stability control,
- electric power steering,
- electronically variable transmission ratio of steering column.

Phase 3: Replacement of vital safety components:

- steer-by-wire,
- brake-by-wire,
- driver-less go (automatic parking, autonomous lane change, ...).
A little mishap...

Source of problem:
- Car geometry:
  - center of gravity
  - wheelbase
- Reduced component-count axles

Solution:
- Patched with embedded control (“ESP”)

How to validate the patch?
Around 1999, the car industry fought for a unification of European and American crash test scenarios because

“it is hardly possible to adjust the firing delay of the airbag to both the ECE and the NCAP frontal crash.”

ECE test: $56 \frac{km}{h}$,
NCAP test: $64 \frac{km}{h}$. 
A Suggestion: Formal Methods

The term refers to a broad set of notations and tools for

1. Mathematically rigorous documentation of requirements
   - less ambiguous than prose
   - amenable to formal analysis (check for consistency, check for adherence to well-formedness criteria,...)

2. Mathematically rigorous models of designs
   - rigorous semantics, removes ambiguity of design documentation in prose, albeit only wrt. the viewpoint focused at
   - early availability of abstract, yet concise model of the system under design
   - amenable to formal analysis (check for absence of design flaws, like deadlock, wrong interfaces,...; check for side-conditions of design steps,...)

3. Check of consistency between such
   - correctness of a design relative to requirements
   - replacability of a design by another one
State-Based Models
Open (contin. time & state) dynamical system

- **Time** is continuous: $\mathbb{R}_{\geq 0}$,

- **internal state** is a bunch of real-valued (or complex-valued) functions of time: $\vec{x}(.): Time \rightarrow \mathbb{R}^{n}$,

- **observable state** is a time-invariant function (usually projection) thereof,

- **environment influence** is a bunch of real-valued (or complex-valued) functions of time: $\vec{u}(.): Time \rightarrow \mathbb{R}^{m}$. 
Some state-based models

- **Finite automata**
  - discrete state space, finitely many states
  - evolution through discrete transitions

- **Differential equations**
  - continuous state space
  - continuous flows

- **Hybrid systems**
  - continuous and discrete state components
  - both jumps (≡ discrete transitions) and continuous flows
State-based models:

Finite Automata
Finite-state models

arise naturally in

• descriptions of **hardware components**,  
  • because these are computational devices with finite (though sometimes large...) memory,

• descriptions of **communication protocols**,  
  • because these engage into an alternation of finitely many different phases,

• descriptions of **embedded computer systems**,  
  • because these are computers with finite memory.
The coffee vending machine — architecture

Vending machine

Financial administr.

Brewer control

cout

coin
canc
req
caf
The coffee vending machine — dynamics

- **no pay**
- **paid**
- **idle**
- **wait**
- **brew**
An example run
The coffee vending machine

Unreachable states!

Free coffee!
Would you like to draw the automaton?
State-based models:

Differential equations
Open (contin. time & state) dynamical system

- **Time** is continuous: $\mathbb{R}_{\geq 0}$,
- **internal state** is a bunch of real-valued (or complex-valued) functions of time: $\vec{x}(.) : Time \rightarrow \mathbb{R}^n$,
- **observable state** is a time-invariant function (usually projection) thereof,
- **environment influence** is a bunch of real-valued (or complex-valued) functions of time: $\vec{u}(.) : Time \rightarrow \mathbb{R}^m$. 
Continuous modeling with DEs

1. Add further, derived state components: the derivatives \( \dot{x}(.) \), \( \ddot{x}(.) \), \ldots of the state components.

2. Formulate dynamics as equations between \( \dot{x}(.) \), \( \ddot{x}(.) \), \( u(.) \), \ldots

N.B. Higher-order derivatives \( x^{(n)} \), \( n > 1 \), can always be removed by

1. adding a fresh state variable \( y(.) \),
2. adding the equation \( y(t) = x^{(n)}(t) \),
3. replacing every occurrence of \( x^{(n+1)} \) by \( \dot{y} \).
The DE describes dynamics of the system by

- providing a state space $\mathbb{R}^n$,
- providing a (piecewise) continuous vector field $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$

constraining the possible evolutions through the equation

$$\frac{dx}{dt} = f(x)$$

The initial value $x_0 \in \mathbb{R}^n$ defines the start state of the dynamic evolution.

A solution in the sense of Carathéodory is a time-dependent signal $x : [0, a) \rightarrow \mathbb{R}^n$ such that

- $x$ is piecewise differentiable,
- $\forall t \in [0, a) \bullet x(t) = x_0 + \int_0^t f(x(s)) \, ds$.

Then $\frac{dx}{dt}(t) = f(x(t))$ for almost all $t \in [0, a)$. 
Differential equation with input

The DE describes dynamics of the system by

- providing a state space $\mathbb{R}^n$,
- providing an input space $\mathbb{R}^m$,
- providing a (piecewise) continuous vector field $f : \mathbb{R}^{n+m} \to \mathbb{R}^n$

constraining the possible evolutions through the equation

$$\frac{dx}{dt} = f(x, u)$$

The initial value $x_0 \in \mathbb{R}^n$ defines the start state of the dynamic evolution.

A solution wrt. a (piecewise) continuous input $u : [0, a) \to \mathbb{R}^m$ is a time-dependent signal $x : [0, a) \to \mathbb{R}^n$ such that

- $x$ is piecewise differentiable,
- $\forall t \in [0, a) \bullet x(t) = x_0 + \int_0^t f(x(s), u(s))ds$.

Then $\frac{dx}{dt}(t) = f(x(t), u(t))$ for almost all $t \in [0, a)$. 
Example: spring-mass system w. disturbance

- **Basic model:**
  \[ \ddot{y}(t) = \frac{F(t)}{m} \]
  \[ F(t) = k(l(t) - l_0) \]
  \[ l(t) = u(t) - y(t) \]

- **Replace higher-order derivatives:**
  Add \( \nu(t) = \dot{y}(t) \).
  
  Gives
  \[ \dot{y}(t) = \nu(t) \]
  \[ \dot{\nu}(t) = \frac{k}{m} (u(t) - y(t) - l_0) \]
State-based modeling:

Hybrid systems
Hybrid Systems

Plant Control

- Analog switch
- Continuous controllers
- Discrete supervisor

Disturbances ("noise")
Environmental influence
Control
Selection

Plant
Observable state
Part of observable state
Task selection

Setpoints
Active control law
- Hybrid Automata

\[ x = 20.0 \land y = 0.0 \]

- \[ \dot{x} = y \]
- \[ \dot{y} = -9.81 \]
- \[ x \geq 0 \]

- \[ x = 0.0 \land y \leq 0.0 \]
- \[ y' = -0.8 \cdot y \]

\[ x \]: vertical position of the ball

\[ y \]: velocity

\[ y > 0 \] ball is moving up

\[ y < 0 \] ball is moving down
State and Dimension Explosion

Number of continuous variables linear in number of cars
  - Positions, speeds, accelerations,
  - torque, slip, ...

Number of discrete states exponential in number of cars
  - Operational modes, control modes,
  - state of communication subsystem, ...

Symmetry reduction often impossible
  - Latency in ctrl. loop depends on number of cars due to communication subsystem.
  - Hidden channels due to coupled dynamics.

• Need a scalable approach
• Trying to achieve this through strictly symbolic methods.
Semantic Modeling of ES

“Embedded Systems are Predicates”

(©E. Hehner)
Symbolic transition system

Given a predicate language \( L \), a symbolic transition system \( STS \) over \( L \) comprises

- a set \( V \) of variable names belonging to \( L \), plus a sort-preserving renaming operation \( v' \) assigning to each variable in \( V \) a “copy” \( v' \notin V \),
- an initialization predicate \( I \in L \) with free \( (I) \subseteq V \),
- a (symbolic or predicative) transition relation \( T \in L \) with free \( (T) \subseteq V \cup V' \).

A run of symb. trans. sys. \( STS \) is a (finite or infinite) sequence 
\( r = \langle \sigma_1, \sigma_2, \sigma_3, \ldots \rangle \) of \( L \)-interpretations such that

**Initiation:** \( \sigma_1 \models I \) and

**Consecution:** \( \sigma_{i,i+1} \models T \) for each \( i < \text{len}(r) \), where

\[
\sigma_{i,i+1}(x) = \begin{cases} 
\sigma_i(x) & \text{iff } x \notin V' \\
\sigma_{i+1}(x) & \text{iff } x \in V'
\end{cases}
\]
Parallel composition

Assume $STS_1 = (V_1, I_1, T_1)$ and $STS_2 = (V_2, I_2, T_2)$

- control (i.e., constrain $v'$) a subset $Out_i \subset V_i$ of variables,
- leave the remaining variables unconstrained.

**Synchronous execution:** If $Out_1 \cap Out_2 = \emptyset$ then

$$STS_{\parallel} = (V_1 \cup V_2, I_1 \land I_2, T_1 \land T_2)$$

yields step-synchronous parallel execution.

**Asynchronous execution:**

$$STS_{\parallel} = (V_1 \cup V_2, I_1 \land I_2, (T_1 \land \bigwedge_{v \in O_2 \setminus O_1} v' = v) \lor (T_2 \land \bigwedge_{v \in O_1 \setminus O_2} v' = v))$$

yields (non-fair) interleaving.
Symbolic Representation: Principle

- symbolic representation of linear size
- provided ODEs are of appropriate kind.
Generalizing the concept: Simulink+Stateflow
‘Algebraic’ blocks

- time-invariant transfer function \( \text{output}(t) = f(\text{input}(t)) \)
- made 1st-order by making time implicit: \( \text{Flow} \equiv \text{output} = f(\text{input}) \)
- no constraints on initial value: \( \text{Init} \equiv \text{true} \)
- discontinuous jumps always admissible \( \text{Jump} \equiv \text{true} \)

All the formulae are elements of a suitably rich 1st-order logics over \( \mathbb{R} \).
Integrators

- integrates its input over time: $output(t) = init + \int_{0}^{t} input(u) \, du$.
- made semi-1st-order by using derivatives: $Flow \equiv \frac{doutput}{dt} = input$
- initial value is rest value: $Init \equiv output = init$,
- discontinuous jumps don’t affect output $Jump \equiv output = output$. 
Getting started with STS: NuSMV
Device

Device Descript.
architecture behaviour of processor is
process fetch
  if halt=0 then
    if mem_wait=0 then
      nextins <= dport
...
NuSMV

- NuSMV (A. Cimatti, E. Clarke, F. Giunchiglia, A. Morichetti, M. Roveri et al.) is an optimized reengineering of the symbolic model checker SMV (K. McMillan, 1993)
- It is dedicated to finite state systems, providing
  - Booleans, bounded integers, enumerations as data types.
- It has a structured, symbolic language for describing initial state sets and transitions:
  - either *declarative*
    \[
    \text{TRANS} \\
    \quad \text{next}(\text{output}) = !\text{input};
    \]
    where totality of the transition relation has to be guaranteed by the user, or
  - “*imperative*” (single assignment!) providing such guarantee
    \[
    \text{ASSIGN} \\
    \quad \text{next}(\text{output}) := !\text{input};
    \]
An inverter taking one step delay:

MODULE main
VAR
    input : boolean;
    output : boolean;
ASSIGN
    init(output) := 0;
    next(output) := !input;

Note that input is unconstrained!
An inverter taking arbitrary delay (and missing transient inputs):

```plaintext
MODULE main
VAR
    input : boolean;
    output : boolean;
ASSIGN
    init(output) := 0;
    next(output) := (!input) union output;
```

Note that `output` has a non-deterministic assignment!
The imperative syntax

MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;

MODULE main
VAR
  gate1 : inverter(gate3.output);
  gate2 : inverter(gate1.output);
  gate3 : inverter(gate2.output);

This is synchronous execution, while...
The imperative syntax

MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;

MODULE main
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);

this is asynchronous execution, and...
The imperative syntax

MODULE inverter(input) VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;
FAIRNESS
  running;

MODULE main VAR
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);

is *fair* asynchronous execution.
In order to ensure totlity of the transition relation, there is a

**single assignment rule:** For each variable, there is *at most one* assignment (which may contain case distinctions).

\[
\text{next(abr)} :=
\begin{cases}
\text{case forget} = 0 & : \text{ab} ; \\
\text{forget} = 1 & : \text{abr} ;
\end{cases}
\text{esac} ;
\]

Note that assignments to both \(v\) and \(\text{next}(v)\) do also constitute multiple assignments to \(v\)!

**non-circular dependencies rule:** Circular dependencies are to be broken by “delays”, i.e. a variable may only depend on older values of itself.
You’ll need the following types of CTL (computation tree logic formulae):

**φ holds invariably:** $\text{AG } \phi$
- On all computation paths, $\phi$ holds generally.

**φ leads to ψ:** $\text{AG } (\phi \rightarrow \text{AF } \psi)$
- On all computation paths, it holds generally that if $\phi$ holds then necessarily (≡ on all computation paths), $\psi$ holds eventually.

Tutorial available at
http://nusmv.irst.itc.it/NuSMV/tutorial/v24/tutorial.pdf
Course Schedule

(this week)
4 Slots a day, 2 in the morning, 2 in the afternoon.

Monday:

1. Introductory lecture
2. Exercise class: State-exploratory verification using NuSMV
4. Lecture: CTL and CTL model checking

Tuesday: public holiday

Wednesday:

1. Lecture: Checking Circuit Equivalence
2. Exercise class: dito
4. Lecture: Satisfiability solving of large propositional formulae
Thursday:

1. Lecture: Arithmetic satisfiability solving
2. Exercise class: Using arith. SAT solving for scheduling
4. Lecture: Symbolic reachability procedures

Friday:

1. Lecture: Hybrid state-space exploration I
2. Exercise class: Introduction to the projects, time to continue with previous exercises
3. Lecture: Hybrid state-space exploration II
4. Consultation time regarding projects