Teaching Cyber-Physical Systems: A Programming Approach

Kerstin Bauer and Klaus Schneider

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern

WESE 2012

Outline

Cyber-physical vs. Hybrid Systems

- embedded systems
  are application-specific computer systems that are integrated in physical systems
- cyber-physical systems
  are systems whose physical parts are tightly integrated with its embedded systems and/or information systems

$\Longmapsto$ cyber-physical system behaviors often include both continuous and discrete behaviors $\Longmapsto$ hybrid systems

$\Longmapsto$ design of cyber-physical systems requires **modeling, simulation, and verification** of hybrid system behaviors

Cyber-physical vs. Hybrid Systems

- embedded systems
  are application-specific computer systems that are integrated in physical systems
- cyber-physical systems
  are systems whose physical parts are tightly integrated with its embedded systems and/or information systems

$\Longmapsto$ cyber-physical system behaviors often include both continuous and discrete behaviors $\Longmapsto$ hybrid systems

$\Longmapsto$ design of cyber-physical systems requires **modeling, simulation, and verification** of hybrid system behaviors

Hybrid Automata [Alur *et al.*1995]

- ...can be represented as finite state transition systems
- continuous behaviors
  - each state is labeled with a set of differential equations
  - these define the change of variable's values over time
- discrete behaviors
  - each transition is labeled by assignments to variables
  - without taking time, variables change on transitions
  - transitions can be taken if trigger conditions are valid
- HA were developed as formal models for verification
- HA do not scale with complex discrete behaviors
  $\Longmapsto$ state space explosion!

Hybrid Systems as Equation Systems?

- many tools describe hybrid systems as equations systems
  - each equation describes the behavior of one variable
  - equation systems can be grouped into modules
  - and modules can be connected by wires
- note: discrete programs can also be represented as equations:

```
while(σ) {
    x_1 = case (φ_1,1) do E_1,1 ... (φ_1,n_1) do E_1,n_1;
    ⋮
    x_m = case (φ_m,1) do E_m,1 ... (φ_m,n_m) do E_m,n_m;
}
```

- but: software engineers advocate the use of more sophisticated statements

Hybrid Systems as Equation Systems?

- many tools describe hybrid systems as equations systems
  - each equation describes the behavior of one variable
  - equation systems can be grouped into modules
  - and modules can be connected by wires
- note: discrete programs can also be represented as equations:

$$
\begin{array}{l}
\textbf{while}(\sigma)\ \{\\
\quad x_1 = \textbf{case}\ (\varphi_{1,1})\ \textbf{do}\ E_{1,1}\ \ldots\ (\varphi_{1,n_1})\ \textbf{do}\ E_{1,n_1};\\
\quad \vdots\\
\quad x_m = \textbf{case}\ (\varphi_{m,1})\ \textbf{do}\ E_{m,1}\ \ldots\ (\varphi_{m,n_m})\ \textbf{do}\ E_{m,n_m};\\
\}
\end{array}
$$

- but: software engineers advocate the use of more sophisticated statements

Languages for Hybrid Systems

- software engineers advocate the use of more sophisticated statements
  - to increase the readability of the programs
  - to allow the reuse of the programs
- many decades of research in programming languages developed
  - structured programming by statements like loops, ...
  - encapsulation and reuse by modules
  - generic/polymorphic data types
  - ...
- so why should we be satisfied with equation systems for hybrid systems?

Languages for Hybrid Systems

- only a few languages and tools exist [Carloni *et al.*2006]
- and these specialize on design phases:
  - modeling: Modelica, ...
  - simulation: MATLAB/Simulink, SystemC-AMS, ...
  - verification: PHAVer, HyTech, ...
- however, we need all of these together
- using many languages and tools in a single course?

Idea: Quartz Programming Language for Teaching

- our course on cyber-physical systems is based on our own language Quartz
- Quartz was derived from the synchronous language Esterel
- in the following part of the talk,
  - we consider the synchronous part of Quartz
  - its compilation to guarded actions,
  - its operational meaning
  - and then the extension to hybrid systems

Synchronous Languages

- embedded and cyber-physical systems require programs with a notion of time
- physical time is often too concrete for many reasons
- motivated by the success in HW design, synchronous languages introduced clocks in programs
- synchronous programming languages were developed around twenty years ago in France and Israel
- well-known synchronous languages are Esterel, Lustre, Signal, and also some variants of StateCharts
- our synchronous language: Quartz
  (it's a cousin of Esterel)

Micro/Macro Steps

- reactive systems
  - executions are divided into discrete reaction steps
  - within each reaction, new inputs are read
  - and current outputs as well as next internal state are computed
- synchronous languages distinguish between micro- and macro steps
  - micro steps (= atomic actions) are executed in zero time
  - macro steps require one unit of logical time
  - **pause** statement declares start/end of reaction step
  - macro step = code between two **pause** statements
- parallel macro steps require the same (logical) time

Micro/Macro Steps

- reactive systems
  - executions are divided into discrete reaction steps
  - within each reaction, new inputs are read
  - and current outputs as well as next internal state are computed
- synchronous languages distinguish between micro- and macro steps
  - micro steps (= atomic actions) are executed in zero time
  - macro steps require one unit of logical time
  - **pause** statement declares start/end of reaction step
  - macro step = code between two **pause** statements
- parallel macro steps require the same (logical) time

The Synchronous Model of Computation

```
{
  b = true;
  p : pause;
  if(a)
     b = true;
  r : pause;
}
 ||
{
  q : pause;
  if(!b)
     c = true;
  a = true;
  s : pause;
}
```

- first step: execute `b=true` and reach locations `p` and `q`
- second step: execute code between `p` and `r` together with code between `q` and `s`
- thus, `a = true; b = true`, but not `c = true`
- note the back-and forth communication of the two threads
- a dynamic schedule must be found that respects the data dependencies

Some Statements of Quartz

`nothing`	(empty statement)
ℓ : `pause`	(macro step)
$x = \tau, \texttt{next}(x) = \tau$	(assignments)
`if`(σ) S_1 `else` S_2	(conditional)
S_1 ; S_2	(sequence)
$S_1 \parallel S_2$	(parallel statement)
`do` S `while`(σ)	(loop)
`[weak] [immediate] abort S when`(σ)	(abortion)
`[weak] [immediate] suspend S when`(σ)	(suspension)
$\{\alpha\ x;\ S\}$	(local variables)

The Averest Tool

- compile program to set of guarded actions (γ, α)
- (γ, α) means: whenever γ holds, execute action α
- **actions are**
  - immediate assignments $x = \tau$
  - delayed assignments $\texttt{next}(x) = \tau$
- **causal execution order**
  - immediate assignments must be execute in data order
  - i.e., values must be written before read in the same step
  - simulators use value unknown $(\perp)$ and compute reaction as a fixpoint
  - code generators prefer static schedules

Example: ABRO (Program Code)

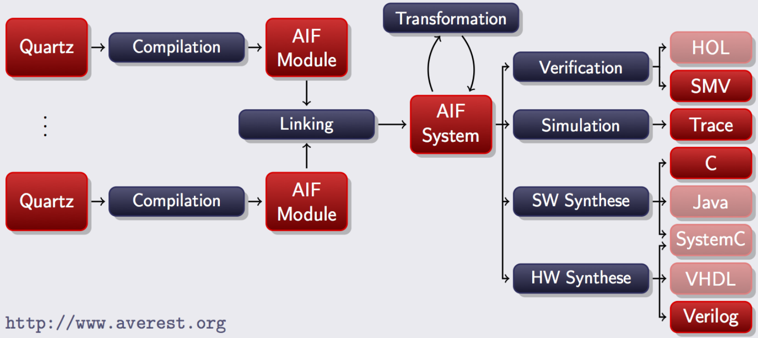
```
module ABRO(event ?a,?b,?r,!o) {
  loop
    abort {
      {wa: await(a); || wb: await(b);}
      emit(o);
      wr: await(r);
    } when(r);
}
```

Example: ABRO (Guarded Actions)

```
system ABRO:
interface:
   a,b,r: input event bool
   o: output event bool
locals:
   w0,wa,wb,wr: label bool
guarded actions:
   !r&wa&!a|r&(wr|wa|wb)|w0 => next(wa)=true
   !r&wb&!b|r&(wr|wa|wb)|w0 => next(wb)=true
   !r&(wr|a&wa&b&wb|!wa&b&wb|!wb&a&wa) => next(wr)=true
   !r&(a&wa&b&wb|!wa&b&wb|!wb&a&wa) => o=true
```

The Averest Tool

Averest Design Flow



http://www.averest.org

From Synchronous to Hybrid Systems

continuous transitions are implemented by flow statements

```
flow {
    x1 <- s1;
    ⋮
    xm <- sm;
    drv(y1) <- t1;
    ⋮
    drv(yn) <- tn;
} until(σ);
```

- a flow statement describes a part of a continuous transition
- it lists the differential equations
- and a condition σ that defines the end of the continuous transition
- several flow statements may run in parallel
- simply take the union of the differential equations, and terminate as soon as one release condition holds

From Synchronous to Hybrid Systems

- hybrid variables:
  - discrete assignments `x = E` and `next(x) = E`
  - continuous assignments `x <- E` and `drv(x) <- E`
- discrete vs. continuous values
  - reactions consist of a discrete and a continuous part
  - `cont(x)` denotes the value during the continuous part, while `x` denotes the discrete value
- release conditions
  - new guarded actions $(\gamma, \texttt{release}(\sigma))$
  - continuous transition terminates if σ holds

A Hybrid Reaction Step

- **discrete reaction**
  - read new inputs ($\Longmapsto$ partial variable environment)
  - evaluate immediate assignments in zero time

  $\Longmapsto$ complete variable environment for this reaction
- **continuous reaction**
  - evaluate continuous actions
  - takes physical time
  - terminates if one release condition becomes true

  $\Longmapsto$ updated variable environment
- **state transition**
  - evaluate delayed assignments `next(x)=E`

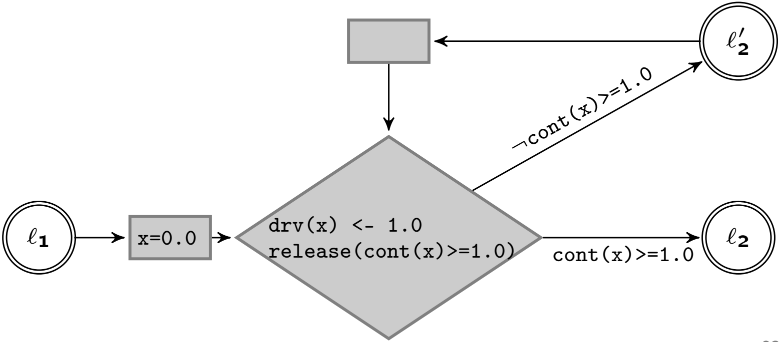
  $\Longmapsto$ new internal state

The Flow Statement

- statement

```
ℓ1: pause;
x = 0.0;
ℓ2,ℓ'2: flow{ drv(x) <— 1.0; } until(cont(x)>=1.0);
```

- meaning:



Our Course on Cyber-physical Systems

- course covers modeling, simulation, and verification
- we don't want to waste time by learning many languages
- instead, we focus on Quartz and Averest
- advantages for students: they
  - can use Quartz programs for modeling,
  - have access to internal system representations,
  - can work interactively with the system,
  - and can therefore write their own simulators and verification procedures

$\Longrightarrow$ goal: stimulate research interest of the students

Our Course on Cyber-physical Systems

- course covers modeling, simulation, and verification
- we don't want to waste time by learning many languages
- instead, we focus on Quartz and Averest
- **advantages for students: they**
    - can use Quartz programs for modeling,
    - have access to internal system representations,
    - can work interactively with the system,
    - and can therefore write their own simulators and verification procedures

$\Longmapsto$ **goal: stimulate research interest of the students**

Part 1: Modeling Hybrid Systems (4 Weeks)

- syntax and semantics of hybrid automata
- syntax and semantics of Quartz programs
- different aspects like non-zenoness and urgent transitions
- students learn to model hybrid systems using Quartz
- short overview on other tools like MATLAB/Simulink and Modelica is given
- running examples are already introduced here

Part 2: Simulation of Hybrid Systems (2 Weeks)

- brief introduction to numerical solution of differential equations
- methods to improve numeric accuracy (second order integration, etc.)
- students write and evaluate their own simple simulators
- we demonstrate main problems of simulators, i.e. discrete event detection/zero crossing

Part 3: Verification of Hybrid Systems (4 Weeks)

- short introduction to bisimulation and temporal logics
- region graph construction to reduce hybrid systems to finite state systems
- this will again be implemented based on guarded actions by the students
- students learn to specify and formally verify hybrid systems
- students develop and use appropriate abstractions for region graph constructions

Part 4: Real-Time Requirements (2 Weeks)

- many embedded systems have to react to input stimuli in time to fulfill their requirements
- students write a discrete controller and have to derive real-time bounds for it
- they should also prove that with the derived real-time bounds, the system will work correctly
- they shall also evaluate given faulty controllers with given worst-case reaction times
- WCET analysis is seen outside the scope of this course, but would be a nice complement

Conclusions

- course covers modeling, simulation, and verification
- we advocate: modeling systems by programs
- but we don't want to waste time by learning many languages
- instead, we focus on Quartz and Averest
- students can interactively work with Averest and can test their prototypes for simulation and verification

References and Further Reading I

[Alur *et al.*1995] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine.
The algorithmic analysis of hybrid systems.
Theoretical Computer Science (TCS), 138(1):3–34, 1995.

[Carloni *et al.*2006] L.P. Carloni, R. Passerone, A. Pinto, and A.L. Sangiovanni-Vincentelli.
Languages and tools for hybrid systems design.
Foundations and Trends in Electronic Design Automation, 1(1/2):1–193, 2006.