Final exam Wednesday Aug 01, 09:00–11:00

- HS 001, 002, 003 in E1 3
- Seating arrangement will be posted at doors
- The exam will be **open book**. That is, you are allowed to use printouts of the lecture slides, books and any handwritten notes during the exam.
- Re-exam: 01.10.12 10:00-12:00, Günter Hotz lecture hall

REVIEW: Automated Formal Methods

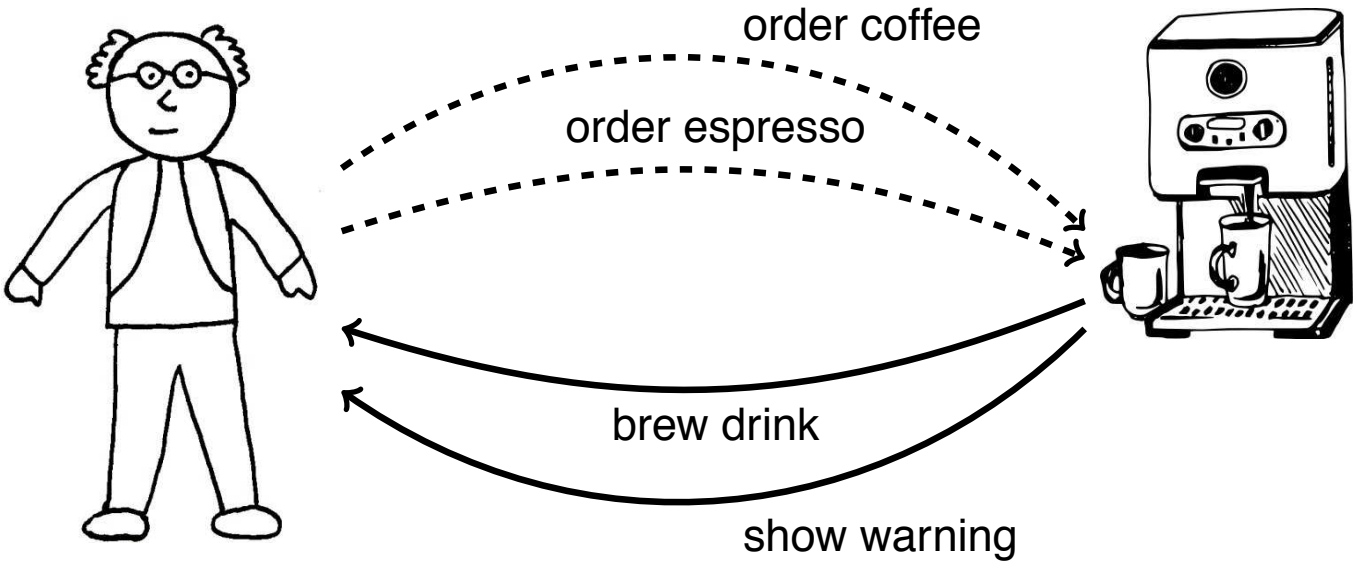
- Model Checking: automatically verify whether certain properties are guaranteed by the model; determine safe parameters
- Controller Synthesis: automatically construct control strategies that keep the system safe

Overview:

1. Intro: Analyzing FlexRay
2. Timed automata
3. Regions & zones
4. Model checking and controller synthesis
5. Hybrid automata

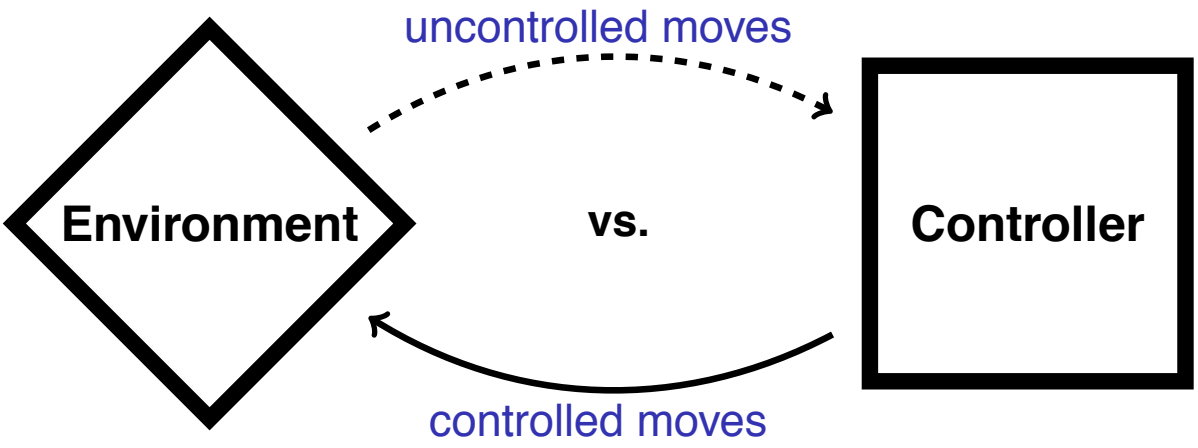
REVIEW: Controller Synthesis

We distinguish between external (uncontrolled) and internal (controlled) nondeterminism



REVIEW: Games

Game between two players



"wants to violate the spec."

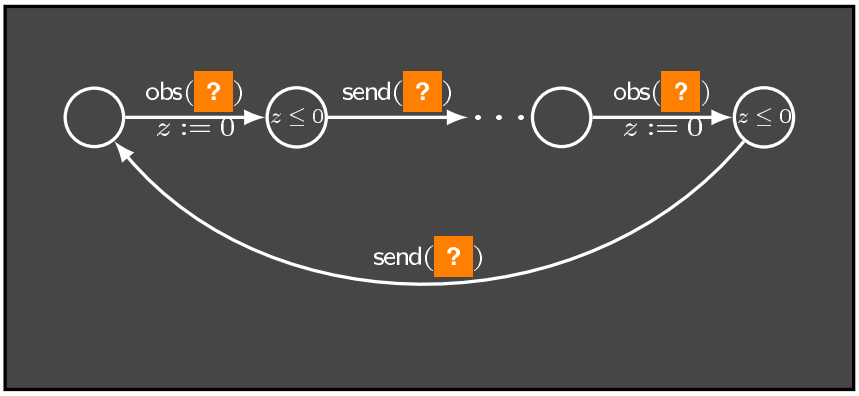
"wants to satisfy the spec."

Template-based synthesis

A **controller template** consists of a timed automaton, a finite set of Boolean parameters, and a total function Π defining which edges are enabled for a given parameter valuation.

Example templates

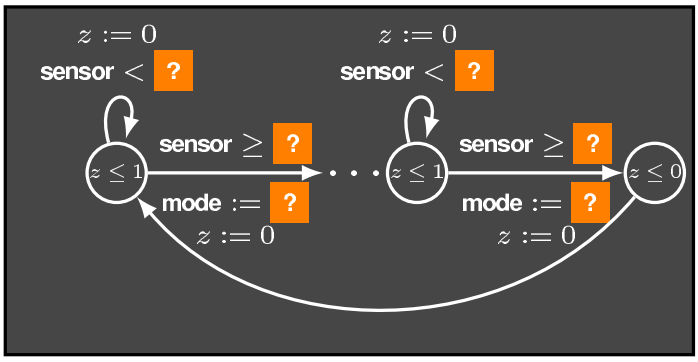
The *cyclic-executive* template:



$\#\text{phases} = n$

Example templates

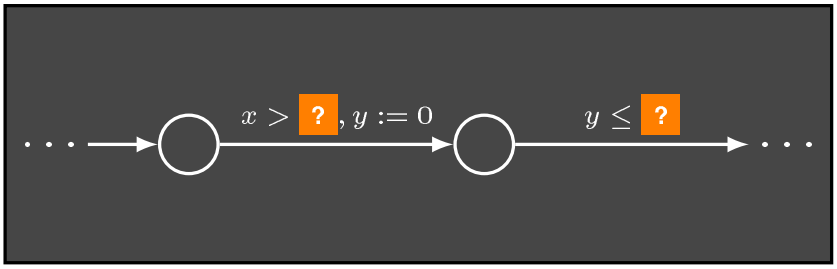
The *multi-phase program* template:



$\#\text{phases} = n$

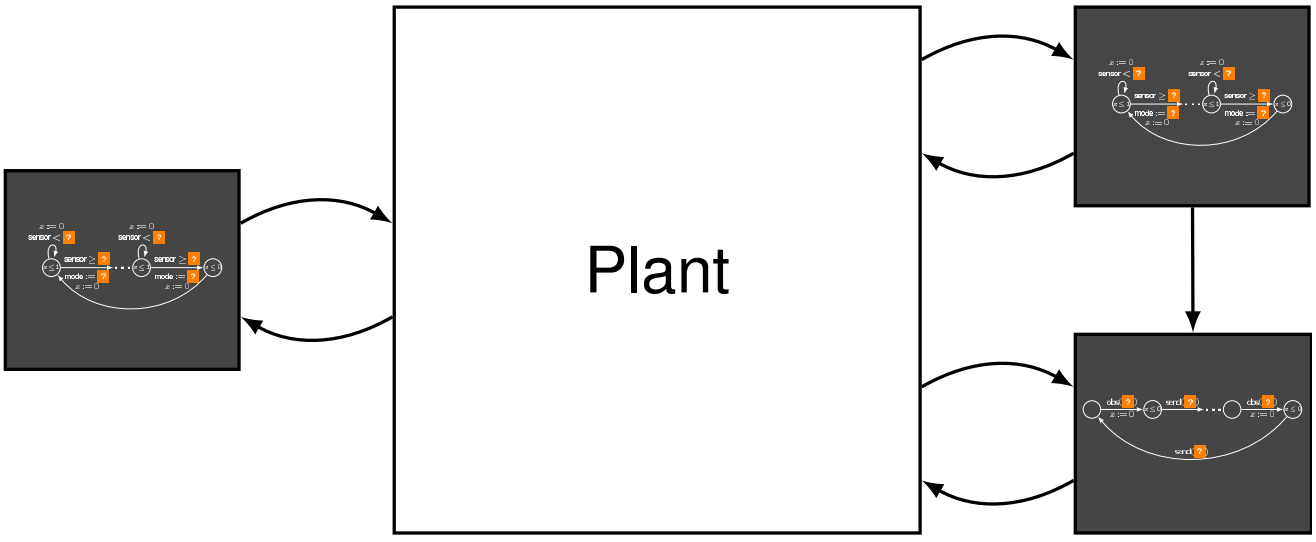
Example templates

Parameters in clock constraints (with finite range)

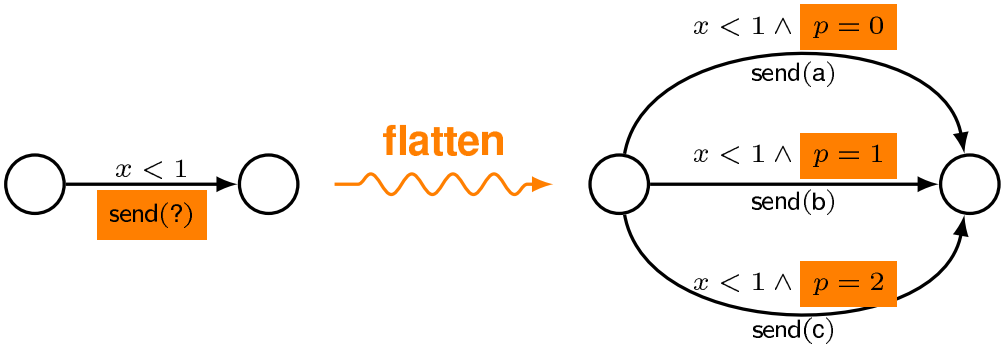


Example templates

Distributed controller:



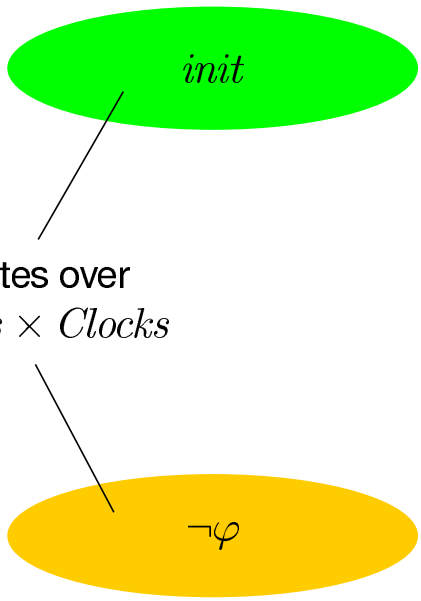
Parameters as variables



Three kinds of state variables:

$Locations$, $Clocks$, $Parameters$

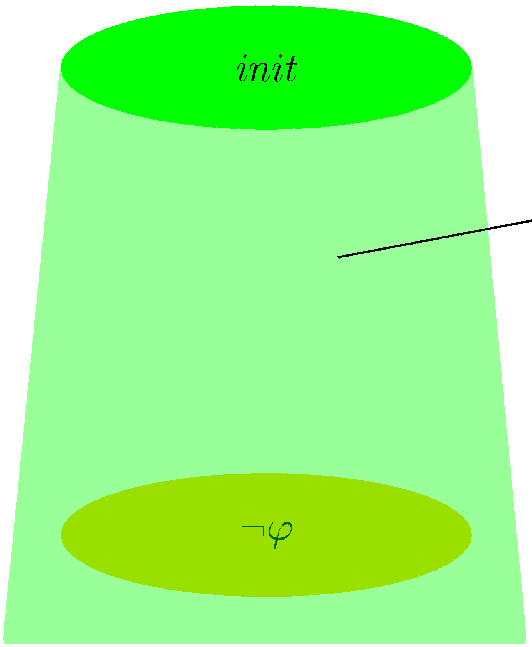
Parameter synthesis by model checking



Predicates over
$Locations \times Clocks$

State space over
$Locations \times Clocks \times Parameters$

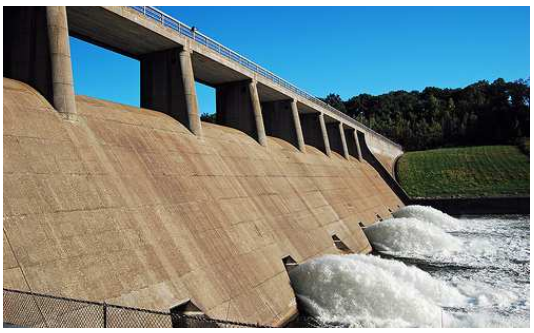
Parameter synthesis by model checking

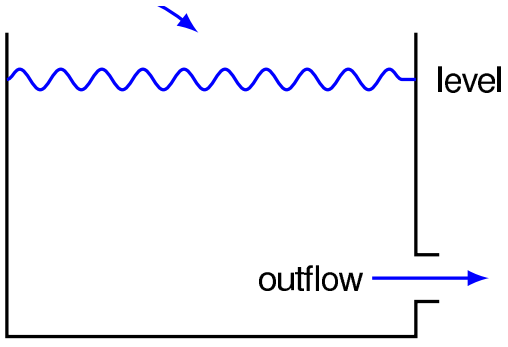


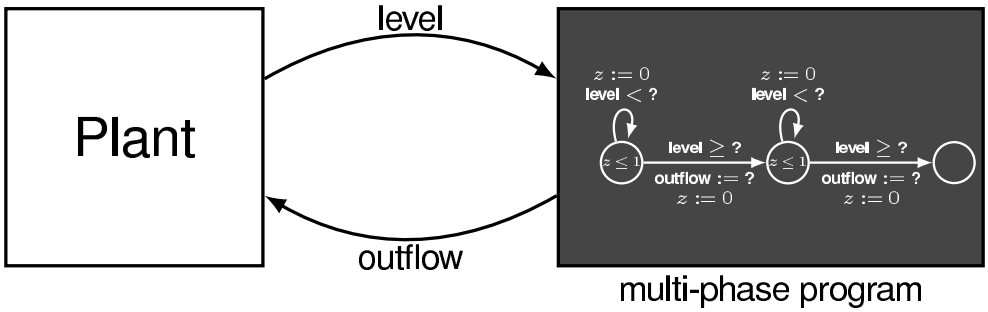
$P \equiv (F \Rightarrow \varphi)$

set of feasible parameter values: $\forall C\, \forall Loc\, .\, P$

Example: Outflow controller







Example: Outflow controller

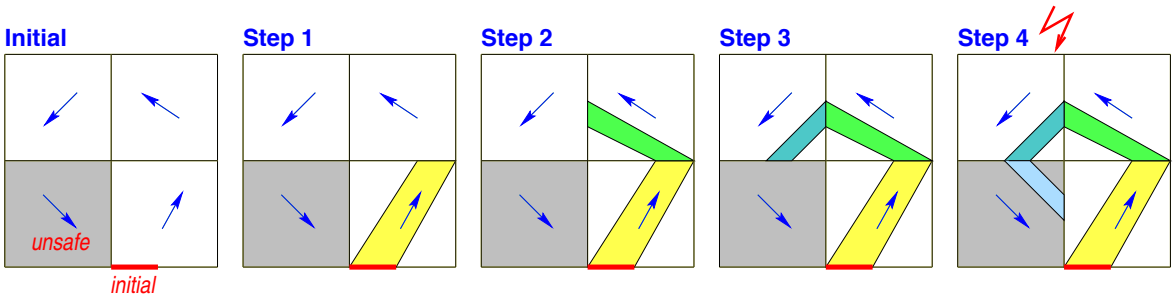
Benchmark	**Template-based SYNTHIA**				UPPAAL-TIGA			Standard SYNTHIA			
	Steps	Abs	Time	Mem	States	Time	Mem	Steps	Abs	Time	Mem
Dam 5	58	100	1	80	88592	2	65	230	149	4	80
Dam 25	268	380	13	87	3114648	307	443	1115	718	1182	91
Dam 50	530	730	87	105	13545848	5018	2355	TIMEOUT			
Dam 75	793	1080	329	111	TIMEOUT			TIMEOUT			
Dam 100	1055	1430	927	143	TIMEOUT			TIMEOUT			
Dam 125	1318	1780	1949	149	TIMEOUT			TIMEOUT			
Dam 150	1580	2130	3483	153	TIMEOUT			TIMEOUT			
Dam 175	1843	2480	5127	213	TIMEOUT			TIMEOUT			
Dam 200	TIMEOUT				TIMEOUT			TIMEOUT			

time in seconds / memory consumption in MB

Verifying Hybrid Automata

Towards model checking hybrid automata

- **Idea:** Iterate transition relation and continuous dynamics until an unsafe state is hit:



- **Result:** Terminates if HA is unsafe.
- **Requires:** Effective representations of transition relation, continuous dynamics, and initial, intermediate, and unsafe state sets s.t.
  1. Calculation of the state set reachable within $n \in \mathbb{N}$ steps is effective,
  2. Emptiness of intersection of unsafe state set with the state set reachable in n steps is decidable.

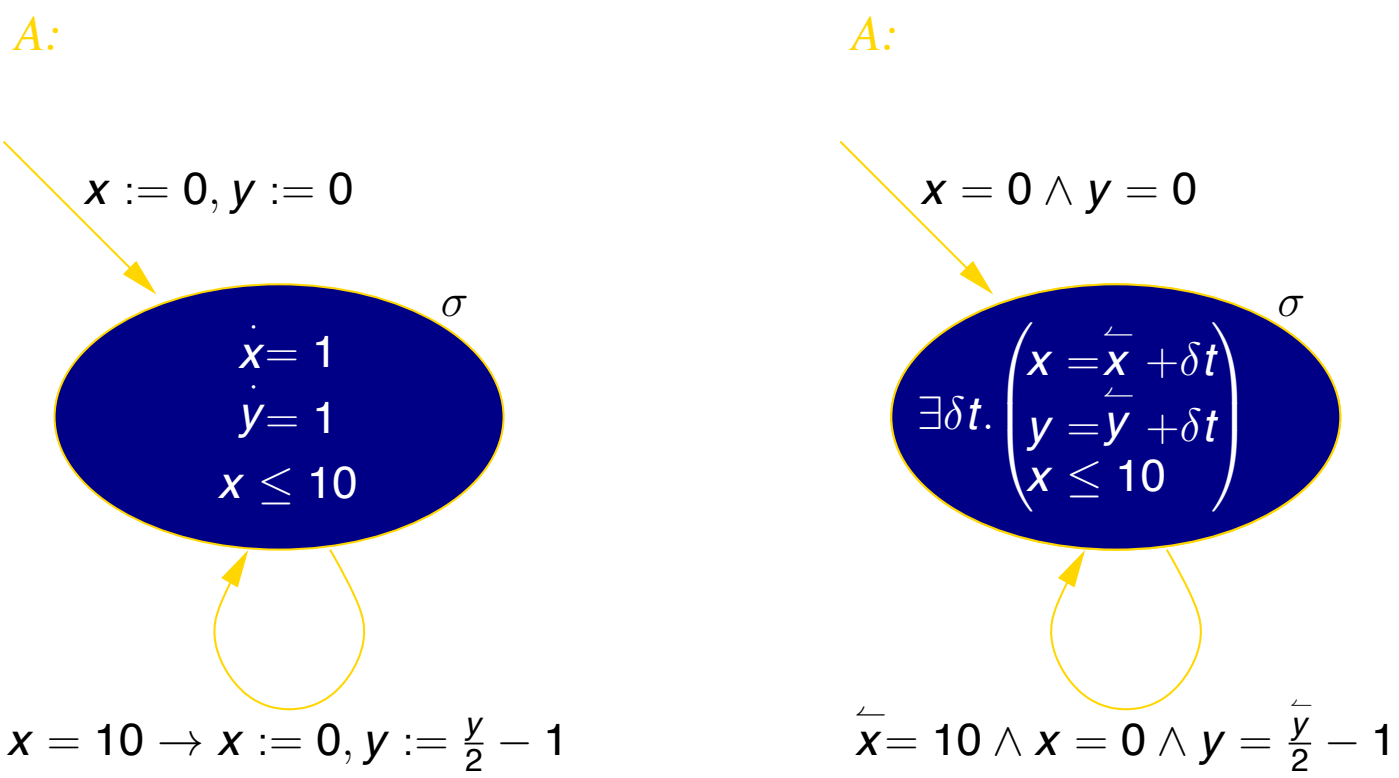
(implemented in e.g. HyTech [Henzinger, Ho, Wong-Toi, 1995–])

Hybrid automata with polyhedral constraints

We assume that the following predicates are given as polyhedral constraints:

- An initial state predicate $initial_\sigma \in \mathrm{FOL}(\mathbb{R}, =, +)$ defines the possible initial states in mode σ
- An activity predicate $act_\sigma \in \mathrm{FOL}(\mathbb{R}, =, +)$ defines the possible evolution of the continuous state while the system is in mode σ
- A transition predicate $trans_{\sigma \to \sigma'} \in \mathrm{FOL}(\mathbb{R}, =, +)$ defines guard and effect of transition from mode σ to mode σ'

From hybrid automata to logic



Reachability

Reachability of a final mode σ' from an initial mode σ and through an execution containing n transitions can be formalized through the inductively defined predicate $\phi^n_{\sigma \rightarrow \sigma'}$, where

$$\phi^0_{\sigma \rightarrow \sigma'} = \begin{cases} \text{false}\,, \text{ if } \sigma \neq \sigma' \,, \\ act_\sigma\,, \text{ if } \sigma = \sigma' \,, \end{cases}$$

$$\phi^{n+1}_{\sigma \rightarrow \sigma'} = \bigvee_{\tilde{\sigma} \in \Sigma} \exists\, \vec{x}_1, \vec{x}_2\,.\, \begin{pmatrix} \phi^n_{\sigma \rightarrow \tilde{\sigma}}[\vec{x}_1 / \vec{x}] \wedge \\ trans_{\tilde{\sigma} \rightarrow \sigma'}[\vec{x}_1, \vec{x}_2 / \overleftarrow{\vec{x}}, \vec{x}] \wedge \\ act_{\sigma'}[\vec{x}_2 / \overleftarrow{\vec{x}}] \end{pmatrix}$$

Safety of hybrid automata

$\Rightarrow$ An unsafe state is reachable within n steps iff

$$unsafe_n = \bigvee_{\sigma' \in \Sigma} Reach_{\sigma'}^{\leq n} \wedge \neg safe_{\sigma'}$$

is satisfiable, where

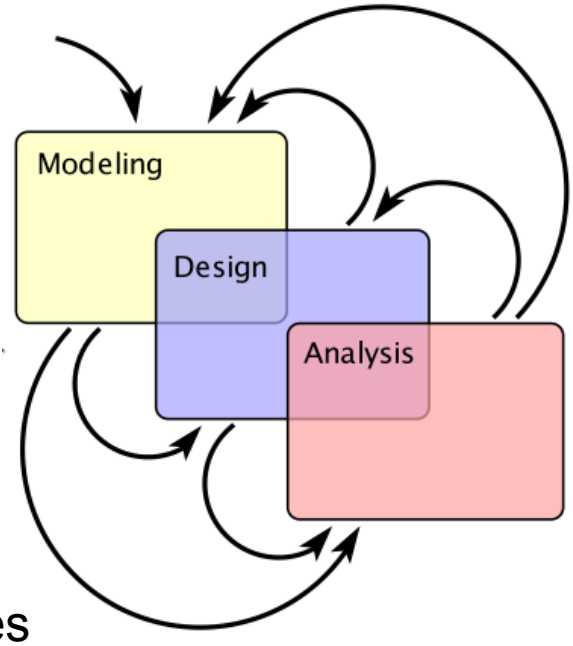
$$Reach_{\sigma'}^{\leq n} = \bigvee_{i \in \mathbb{N}_{\leq n}} \bigvee_{\sigma \in \Sigma} \phi^i_{\sigma \to \sigma'} \wedge initial_\sigma[\overleftarrow{\vec{x}} / \vec{x}]$$

characterizes the continuous states reachable in at most n steps within mode σ'.

- An unsafe state is reachable iff there is some $n \in \mathbb{N}$ for which $unsafe_n$ is satisfiable.
- The unsafe states are unreachable if (but not only if) $unsafe_n$ is unsatisfiable and $Reach_{\sigma'}^{\leq n+1} \Rightarrow Reach_{\sigma'}^{\leq n}$.

REVIEW

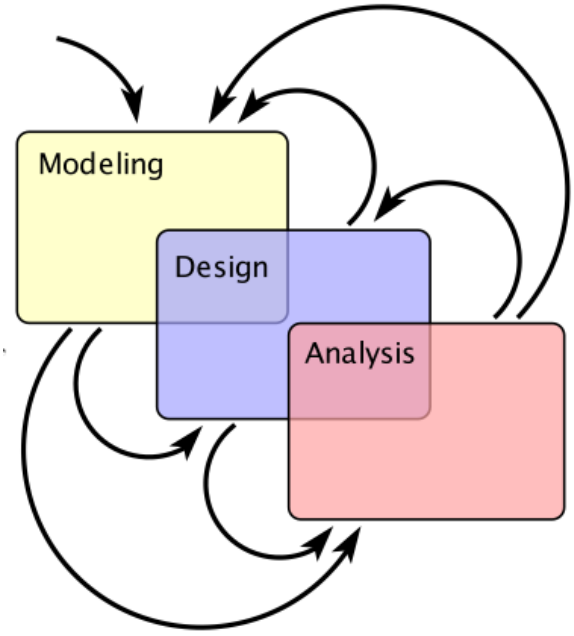
Modeling, Design, Analysis



Modeling is the process of gaining a deeper understanding of a system through imitation.
Models specify **what** a system does.

Design is the structured creation of artifacts. It specifies **how** a system does what it does. This includes optimization.

Analysis is the process of gaining a deeper understanding of a system through dissection.
It specifies **why** a system does what it does (or fails to do what a model says it should do).



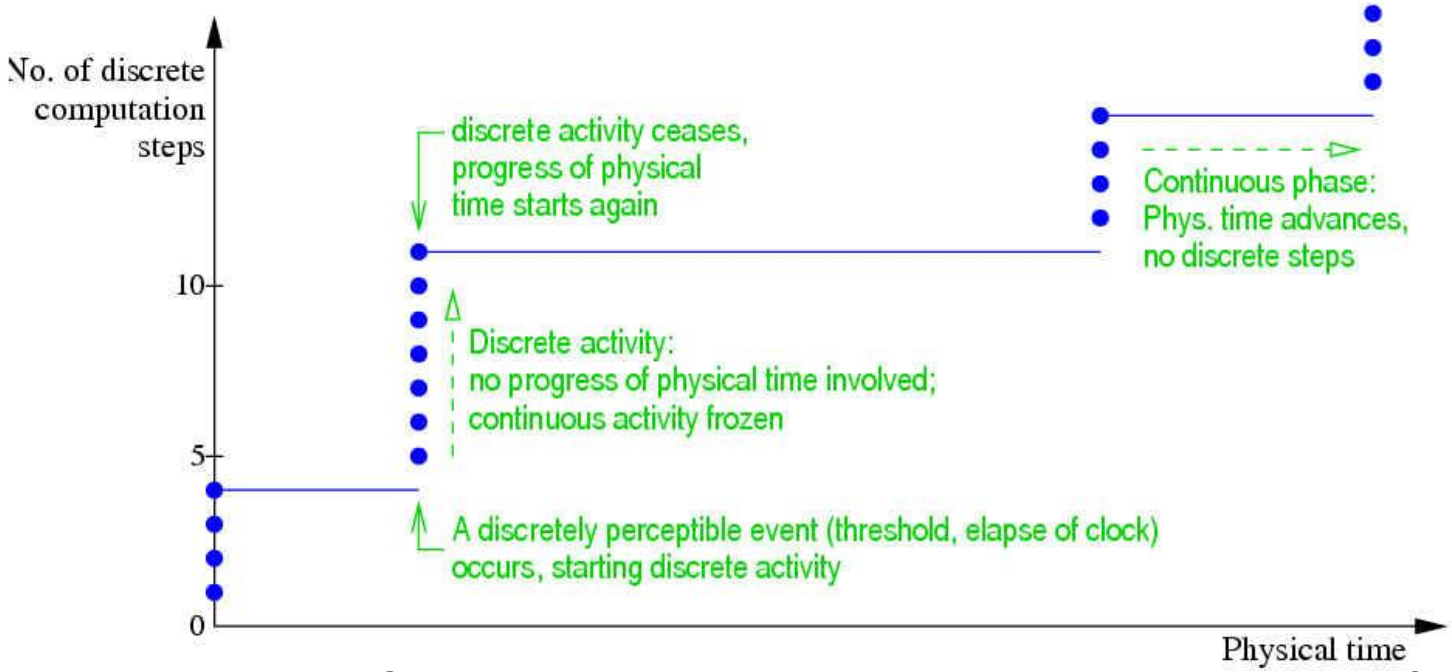
Modeling

Computational models

Communication/ local computations	Shared memory	Asynchronous message passing
Communicating finite state machines	Hybrid automata, statecharts, synchronous composition	
Data flow		Petri nets, Kahn process networks, SDF
Discrete event model	VHDL	

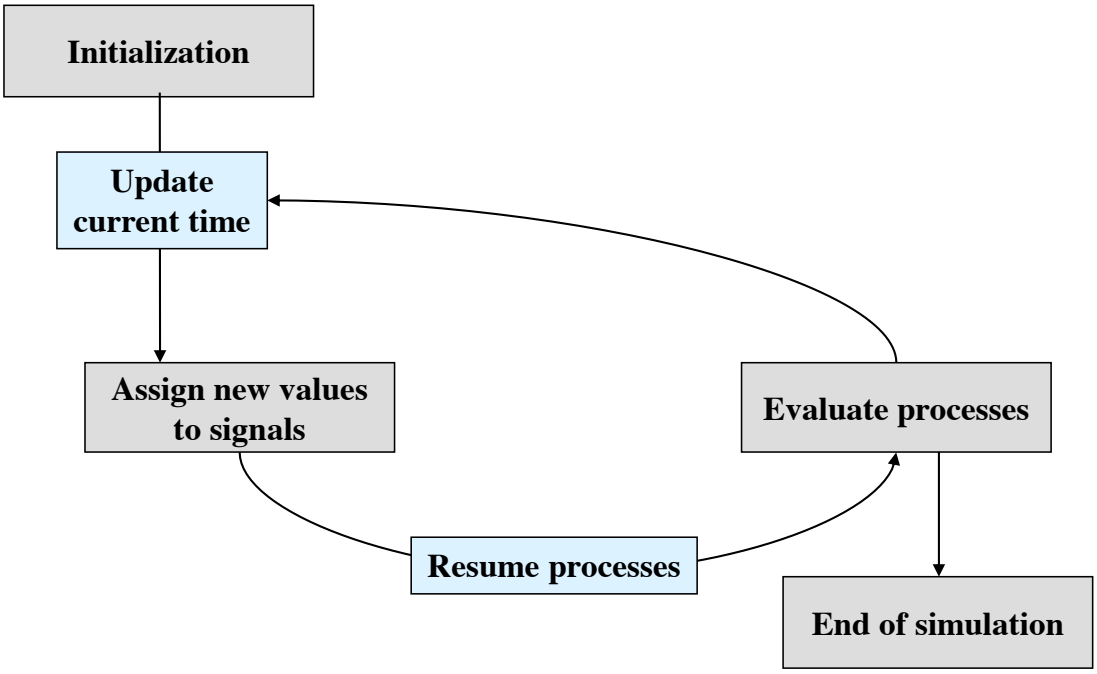
The super-step time model

- Two-dimensional time:



- Assumption: Computation time is neglegible compared to dynamics of the environment.

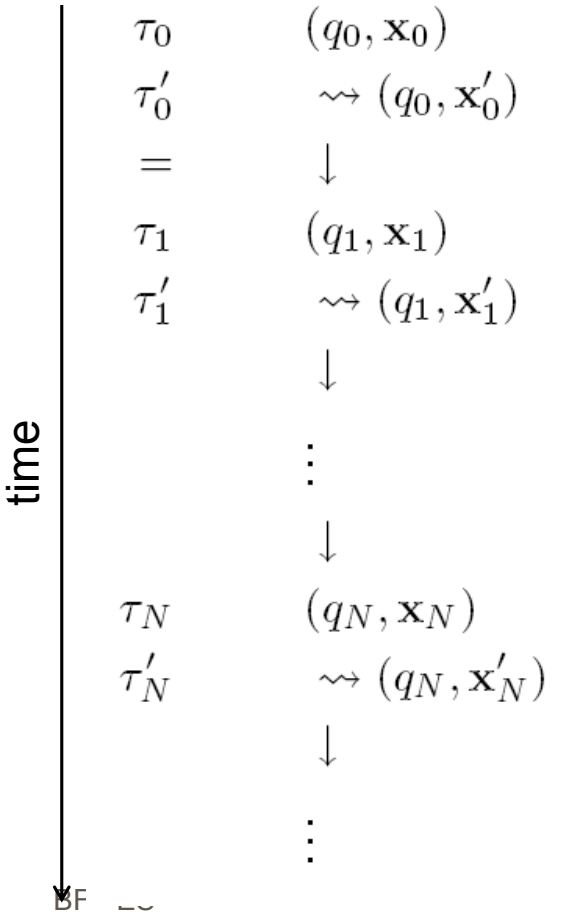
VHDL Semantics



Timed and Hybrid automata

- Motivation
  - The design of an embedded system must consider both the continuous evolution of the environment and the discrete computation of the controller
- Major points
  - Modeling with hybrid automata
  - Semantics (hybrid time sets, hybrid trajectories)
  - Zenoness
  - Automatic verification
  - Automatic controller synthesis

Zeno Behavior



An execution of a hybrid automaton with time set τ is **zeno** iff $\langle \tau \rangle = \infty$ but $|\tau| < \infty$.

Timed Automata with Nondeterministic Delays [Alur/Dill]

A *timed automaton* is a tuple

$$TA = (Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L) \quad \text{where:}$$

- Loc is a finite set of locations.
- $Loc_0 \subseteq Loc$ is a set of initial locations
- C is a finite set of clocks
- $L : Loc \rightarrow 2^{AP}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation, and
- $inv : Loc \rightarrow CC(C)$ is an invariant-assignment function

Clock Constraints

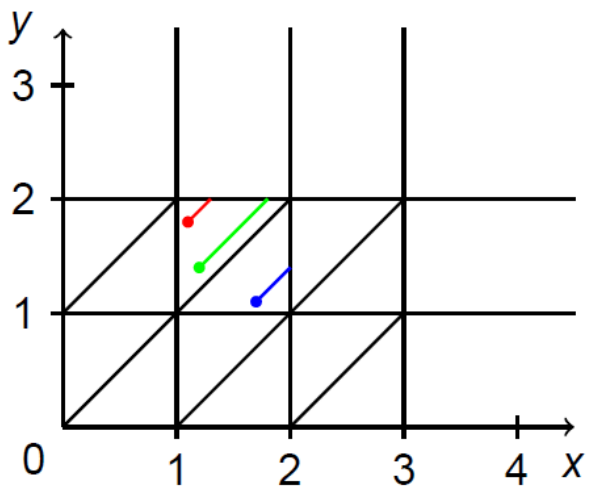
Clock constraints over set C of clocks are defined by:

$$g ::= \mathit{True} \mid x < c \mid x \leq c \mid \neg g \mid g \wedge g$$

- where $c \in \mathbb{N}$ and clocks $x, y \in C$
- rational constants would do; neither reals nor addition of clocks!
- let $CC(C)$ denote the set of clock constraints over C
- shorthands: $x \geq c$ denotes $\neg(x < c)$
  and $x \in [c_1, c_2)$ or $c_1 \leq x < c_2$ denotes $\neg(x < c_1) \wedge (x < c_2)$

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

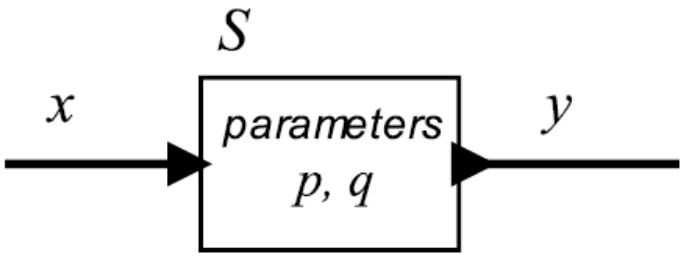


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing
3. maximal constants

Actor Model of Continuous-Time Systems

A *system* is a function that accepts an input *signal* and yields an output signal.



The domain and range of the system function are sets of signals, which themselves are functions.

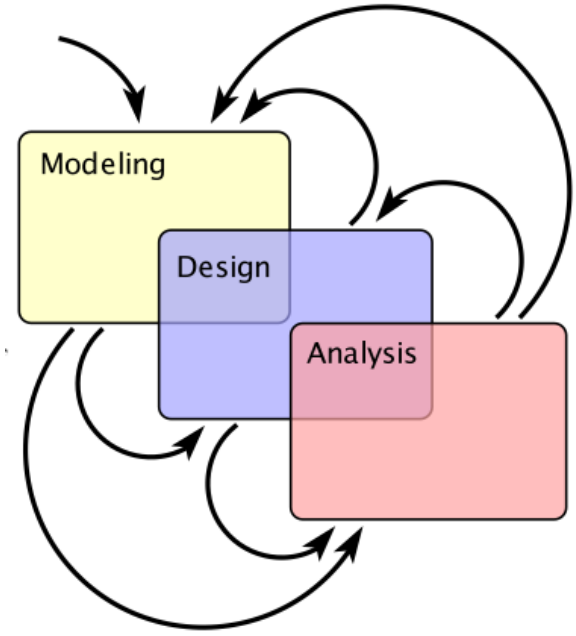
$$x: \mathbb{R} \to \mathbb{R}, \quad y: \mathbb{R} \to \mathbb{R}$$

$$S: X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

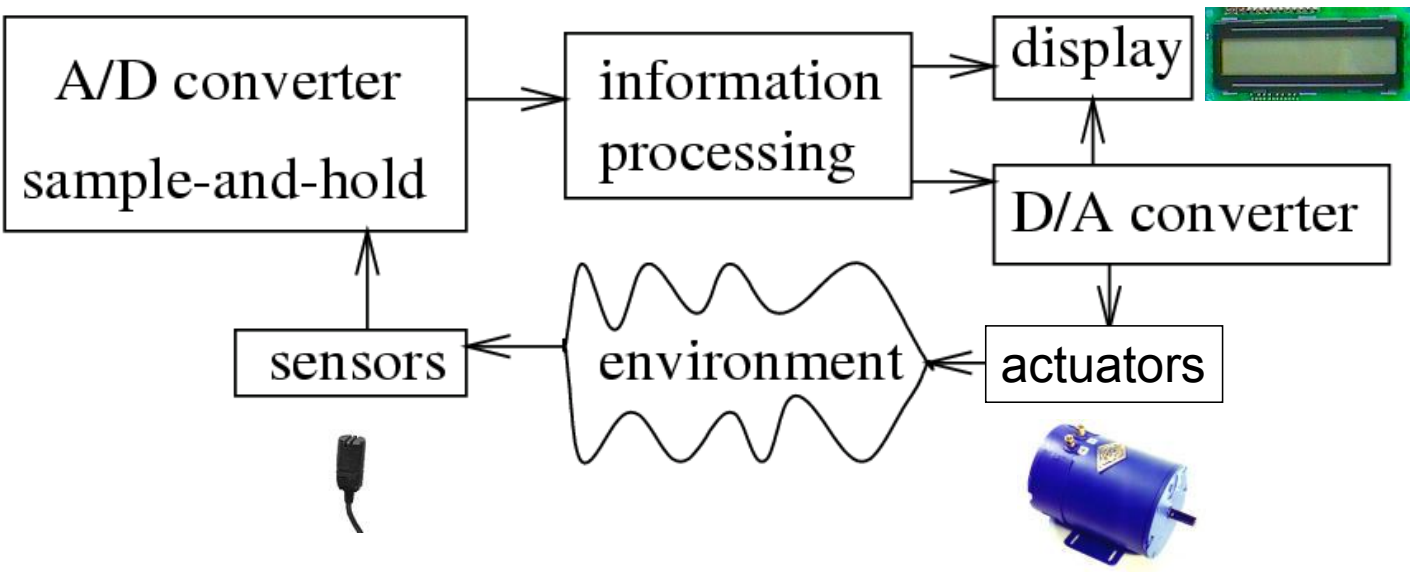
Parameters may affect the definition of the function *S*.

Design



Embedded System Hardware

- Embedded system hardware is frequently used in a loop (***„hardware in a loop“***):



Sensors, A/D + D/A converters

- Motivation
  - Embedded systems interact with physical environment
- Major points
  - Sample & hold circuits
  - A/D converters
  - D/A converters
  - Aliasing
  - Interfaces

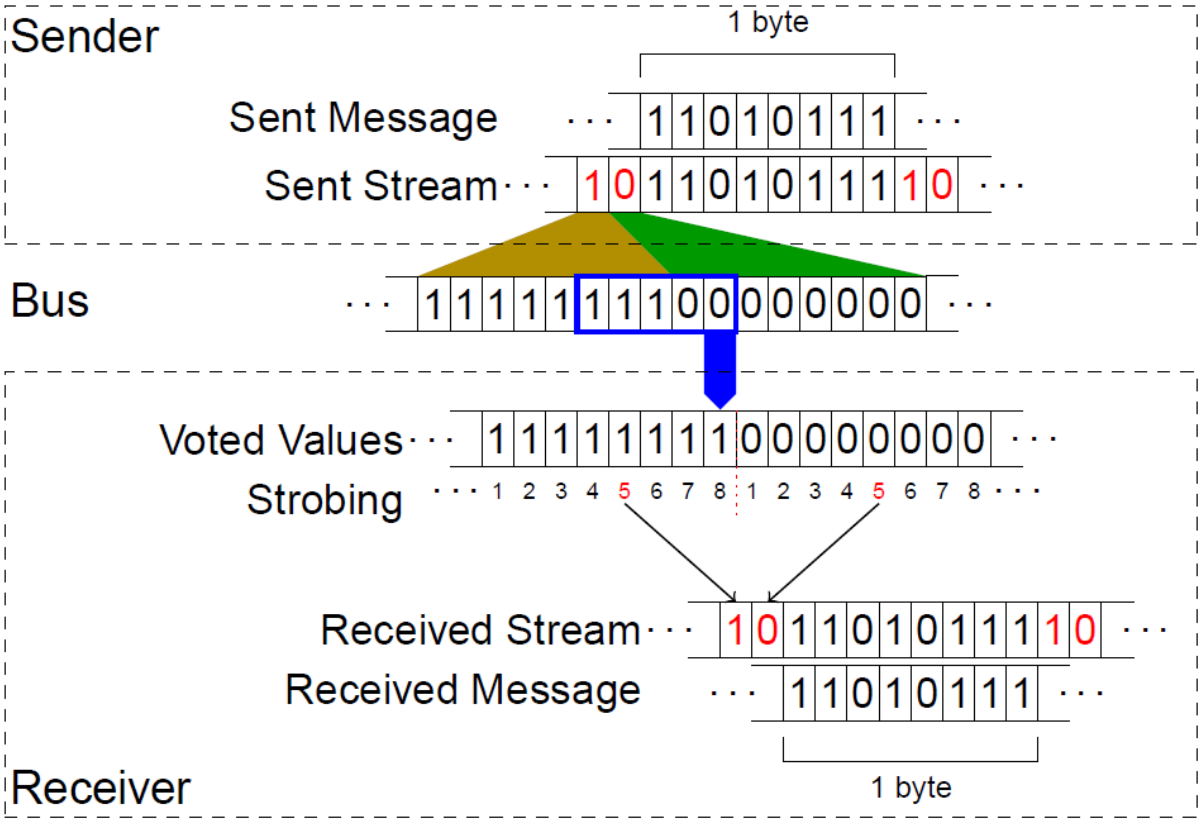
Information processing

- Motivation
  - Embedded systems must be efficient
  - Embedded processors need not be instruction set compatible with standard PCs
- Major points
  - Power/energy efficiency
  - Code size efficiency
  - Runtime efficiency
  - Reconfigurable logic, multimedia processors, scratch pad memory…

Real-time communication

- Motivation
  - Modular system development, support, and evolution
  - Network vs. wiring harness
- Major points
  - Electrical robustness
  - Priority-based arbitration
  - TDMA
  - CSMA
  - FlexRay

FlexRay



Scheduling

- Motivation
  - Key issue in implementing RT-systems
  - Different algorithms have different assumptions and cost
- Major points
  - Aperiodic scheduling
  - Periodic scheduling
  - Scheduling with resource constraints
  - Multiprocessor scheduling

EDF – Earliest Deadline First

- **EDF:** At every instant execute the task with the earliest absolute deadline among all the ready tasks.

- **Theorem (Horn '74):**
  Given a set of n independent task **with arbitrary arrival times**, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Aperiodic scheduling: Non-preemptive version

- **Theorem** (Jeffay et al. '91): EDF is an optimal **non-idle** scheduling algorithm also in a non-preemptive task model.

- Non-preemptive scheduling with **idle** schedules allowed is **NP-hard**

- Possible approaches:
  - Heuristics
  - Bratley's algorithm: Branch-and-bound

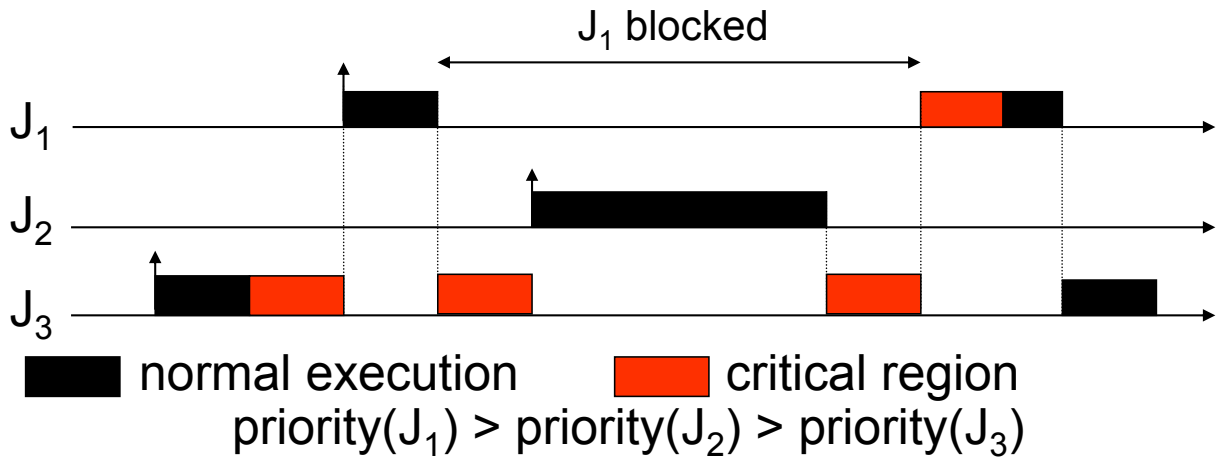
Periodic scheduling

- **Theorem:** A set of periodic tasks τ_1, ..., τ_n with $D_i = T_i$ is schedulable with EDF iff $U \leq 1$.

- **Theorem (Liu, Layland, 1973):**
  RM is optimal among all fixed-priority scheduling algorithms.

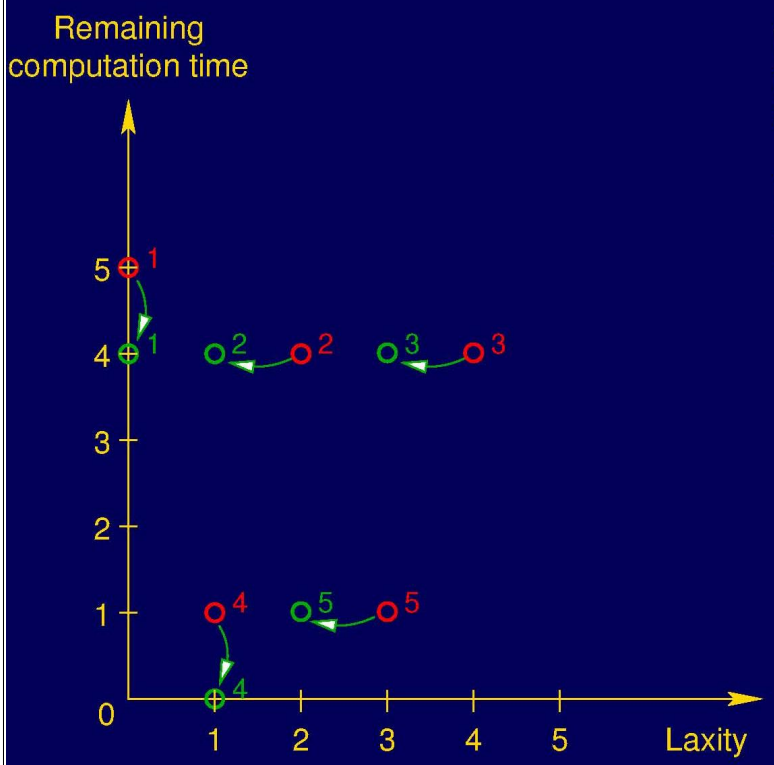
- Any set of n periodic tasks with a processor utilization factor ≤ $U_{lub} = n(2^{1/n} - 1)$ can be scheduled by RM.

The priority inversion problem



- Blocking time equal to length of critical section + computation time of J_2.
- Unbounded time of priority inversion, if J_3 is interrupted by tasks with priority between J_1 and J_3 during its critical region.

Multiprocessor scheduling



When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

LLF is optimal.

Periodic scheduling

1. Divide the time line into time slices such that each period of each process is divided into an integral number of time slices.
   Slice length $T = \mathrm{GCD}(T_1, \ldots, T_n)$.
2. Within each time slice, allocate processor time in proportion to the utilization $U_i = \frac{C_i}{T_i}$ originating from the various tasks.
   Processing time per slice $r_i = TU_i = T\frac{C_i}{T_i}$.
   Hence, each task runs $\frac{T_i}{T} r_i = \frac{T_i}{T} T \frac{C_i}{T_i} = C_i$ time units within its period.
3. Allocate r_i according to the following algorithm
   (a) Look for the first processor proc_j that has free capacity in its time slices.
   (b) Allocate that portion of r_i to proc_j that proc_j can accommodate.
   (c) If all of r_i has been allocated then proceed with the next task (goto step a).
   (d) Otherwise allocate the remainder of r_i to proc_{j+1}.
   proc_{j+1} has enough spare capacity as it has not previously been used and $r_i \leq T$ due to $U_i \leq 1$. Furthermore, due to $r_i \leq T$, we don't generate temporal overlap between the two partial runs of task i.

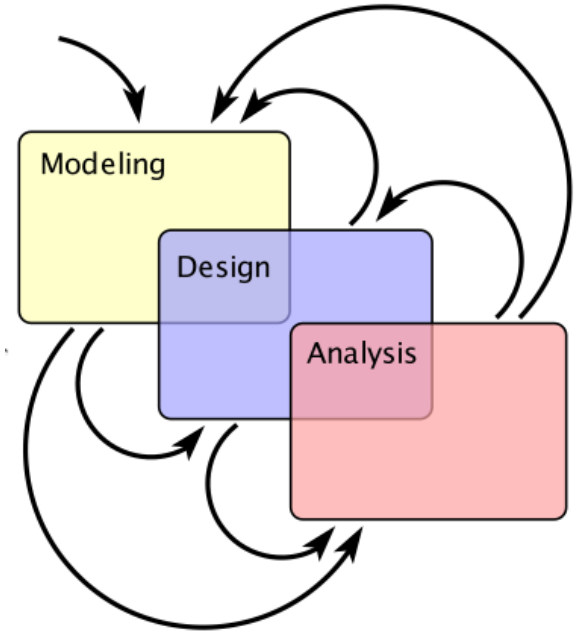
Partitioning

- Motivation
  - HW/SW codesign
  - Software (alone) may not have sufficent performance
  - Hardware (alone) may be too expensive
- Major points
  - Integer Linear Programming (ILP)
  - Hierarchical clustering
  - Kernighan-Lin algorithm
  - F-M heuristic

Fault tolerance: failure modes

- Fail-silent failures
  - subsystem either produces correct results
    or produces (recognizable) incorrect results
    or remains quiet
  - **can be masked as long as at least one system survives**
- Consistent failures
  - If subsystem produces incorrect results all recipients receive same (incorrect) result
  - **can be masked iff the failing systems form a minority**
- Byzantine failures
  - subsystem reports different results to different dependent systems
  - **can be masked iff strictly less than a third of the systems fail**

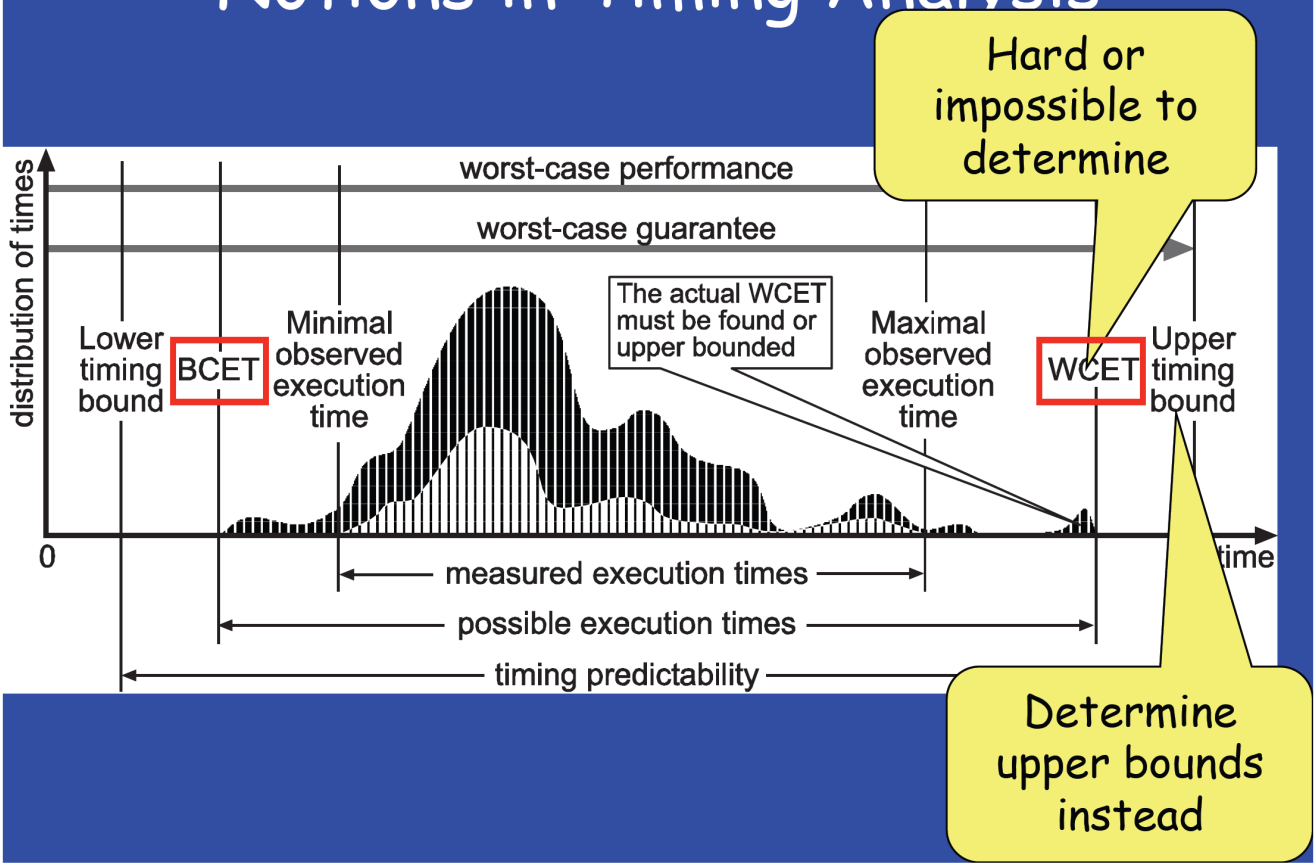
Analysis



Estimation and Verification

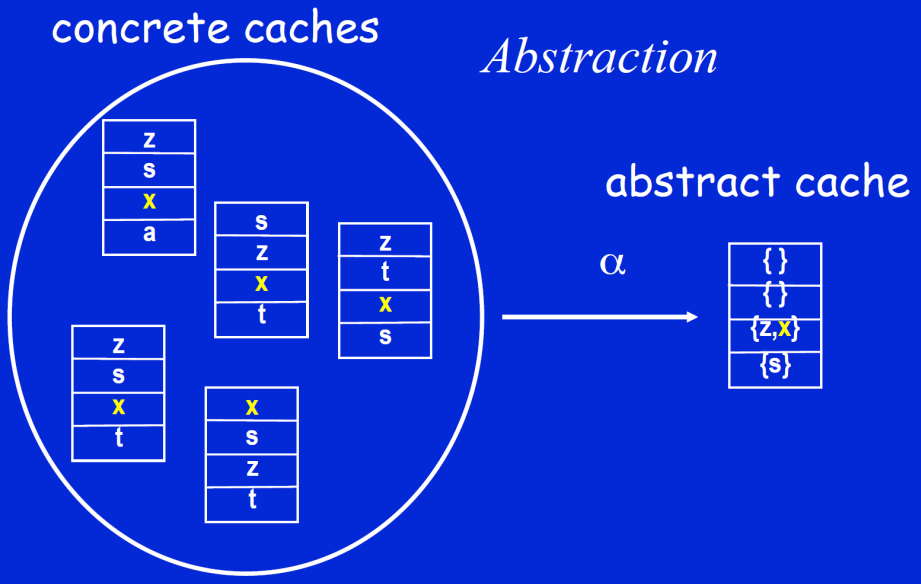
- Motivation
  - Design-space exploration
  - Real-time guarantees
  - Fault tolerance
  - Correctness
  - Safety
- Major points
  - WCET analysis based on abstract interpretation
  - Testing
  - Reliability analysis
  - Verification
  - Controller synthesis

Notions in Timing Analysis

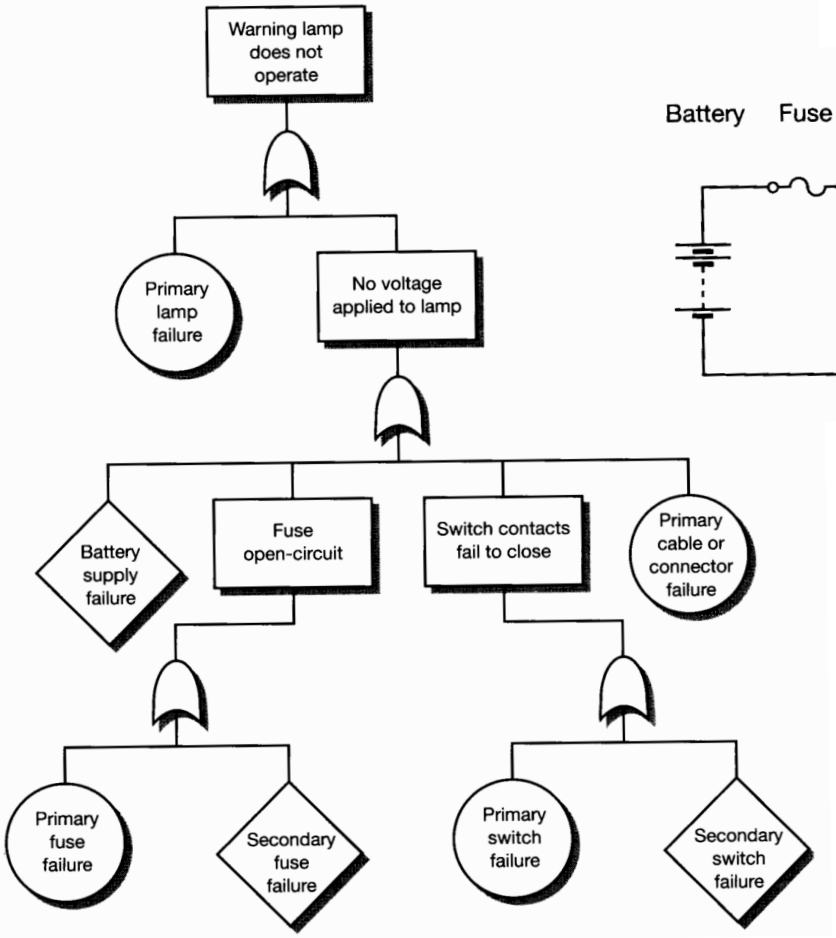


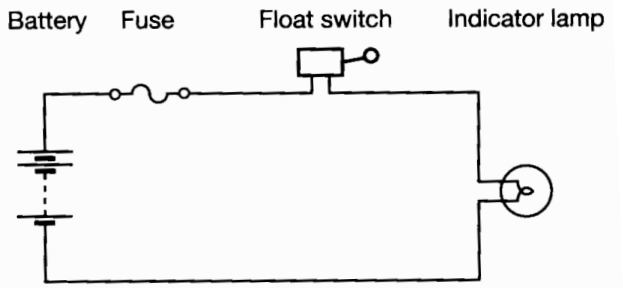
Abstract Domain: Must Cache

Representing sets of concrete caches by their description



Fault tree analysis



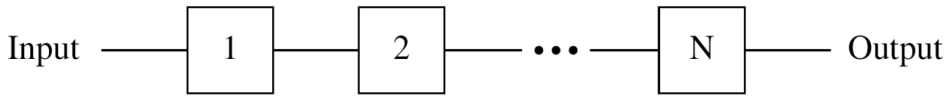


Neil Storey:
Safety-critical computer systems

Inductive computation of reliability

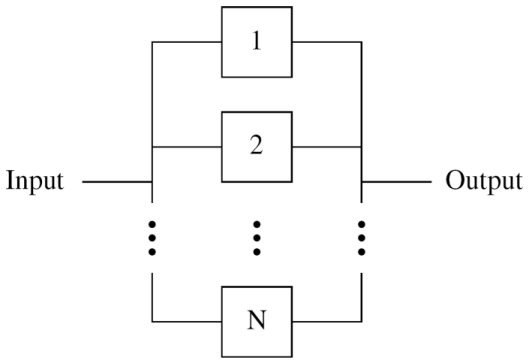
- **Assumption:** failures of the individual components are independent
- **Serial composition**

$$\prod_{i=1}^{N} R_i(t)$$

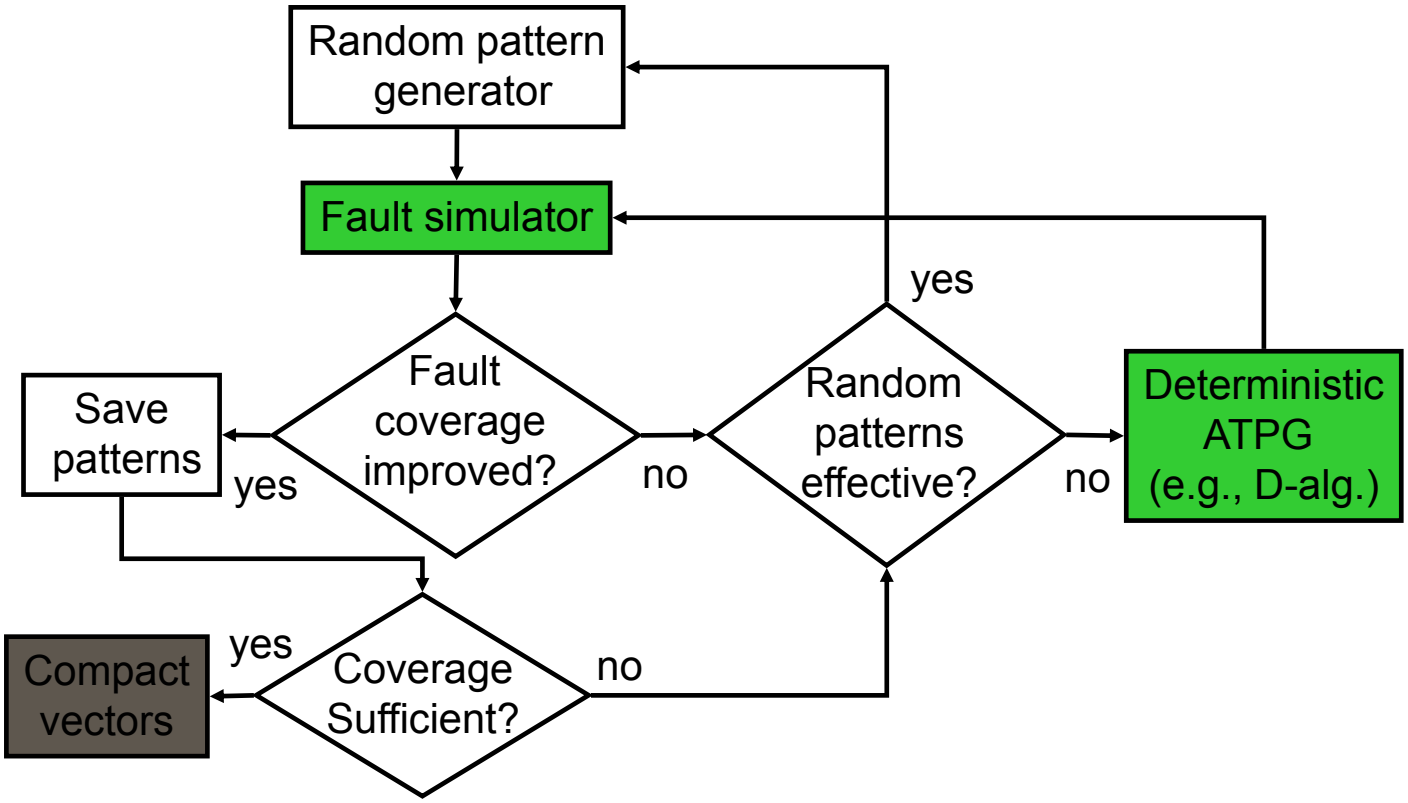


- **Parallel composition**

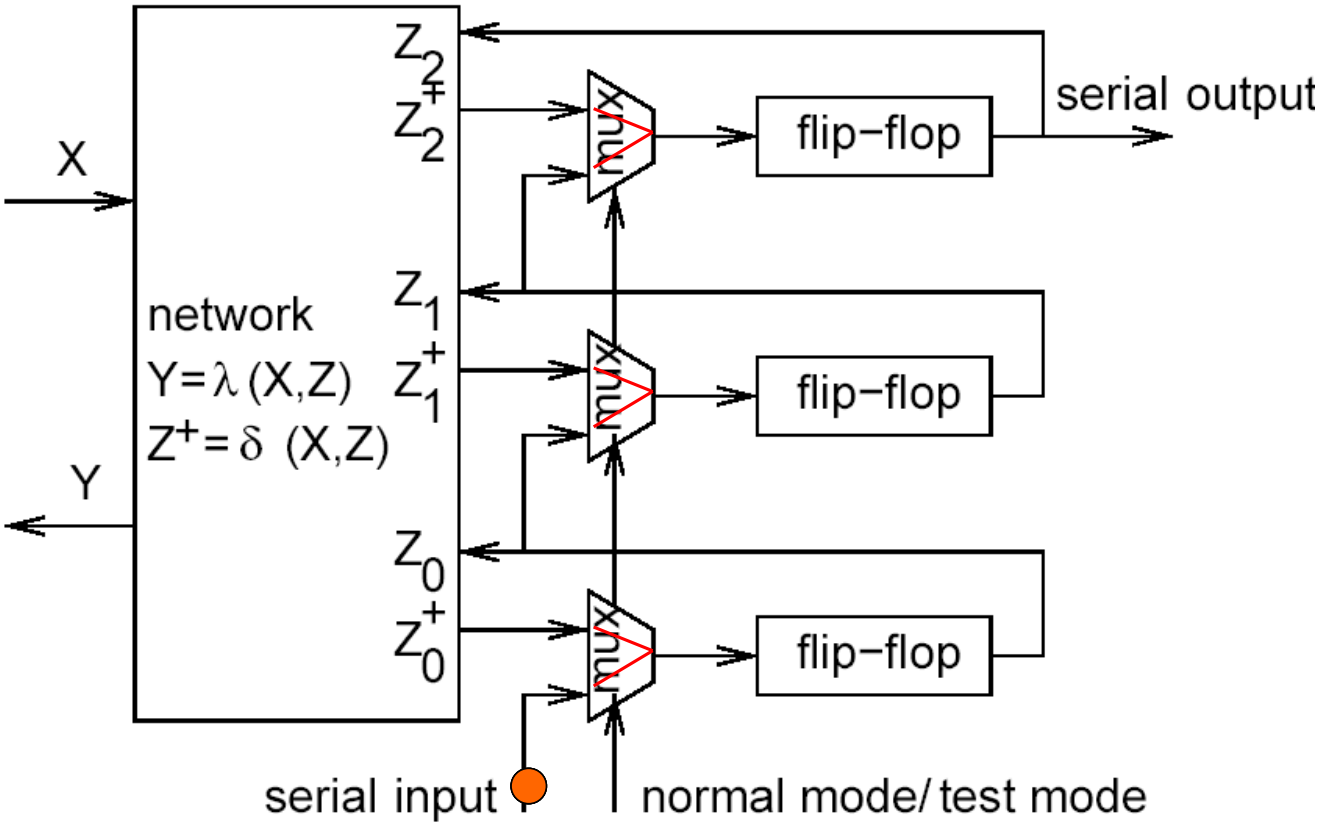
$$1 - \prod^{N} (1 - R_i(t))$$



An ATPG System



Scan design



Automated Formal Methods

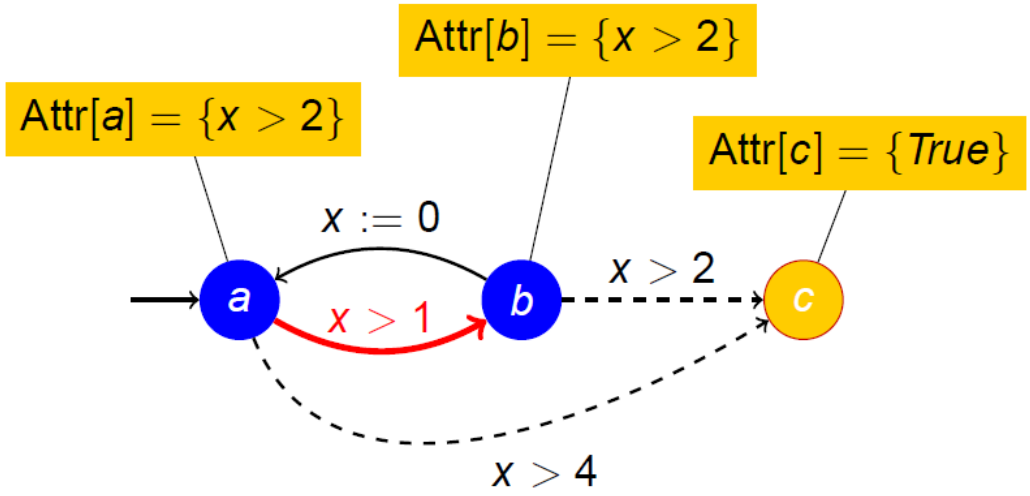
- Model Checking: automatically verify whether certain properties are guaranteed by the model; determine safe parameters
- Controller Synthesis: automatically construct control strategies that keep the system safe

Overview:

1. Intro: Analyzing FlexRay
2. Timed automata
3. Regions & zones
4. Model checking and controller synthesis
5. Hybrid automata

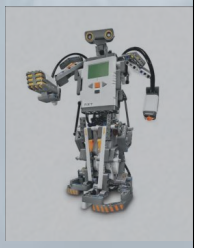
Zone-based Timed Game Solving

From where can $\dashrightarrow$ enforce a run to c ?



Development of Safety-Critical Embedded Systems

- Daniel Kästner, Florian Martin, Reinhard Wilhelm.
- Advanced course (6 ECTS): Fr 10-12, E1.3 / HS003. Exercises, 2h.
- Goal: Working with industry tools for developing safety-critical embedded systems and understanding their theoretical background.
- Contents: Functional safety, model-based code generation, synchronous programming, task scheduling, static program analysis for safety aspects (worst-case execution time, stack usage, runtime errors).
- Tools used:
  - SCADE: CASE tool for safety-critical embedded systems (avionics)
  - Symta/S: Task scheduling & schedulability analysis (automotive)
  - aiT WCET Analyzer / StackAnalyzer / Astrée: Static program analyzers (avionics & automotive)
- Practical project with Lego Mindstorms



Automata, Games, and Verification

- Bernd Finkbeiner, Hazem Torfah, Markus Rabe
- Advanced course (6 ECTS)
- Goal: logical and game-theoretic foundations of automatic verification and synthesis
- Contents:
  - Automata over infinite words and trees (omega-automata)
  - Infinite two-player games
  - Logical systems for the specification of nonterminating behavior
  - Transformation of automata according to logical operations

Seminar: Real-Time Systems & Synthesis

- Bernd Finkbeiner, Michael Gerke, Peter Faymonville
- Seminar (7 ECTS)
- Organizational meeting in October
- Preparatory meetings during lecture period
- Kolloquium (1-2 days) after exams in February