Formal Specification and Verification

Formal specification (3)

13.12.2016

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Exam

1) 20.02.2017–24.02.2017 (first week after end of lectures)
   27.02.2017–3.03.2017 (second week after end of lectures)
   (Attention: 27.02: Rosenmontag)
   6.03.2017–10.03.2017 (third week after end of lectures)

2) Before start of the lectures of the Sommersemester (18.04.2017)

Doodle poll soon.

Until now

Transition systems and program graphs

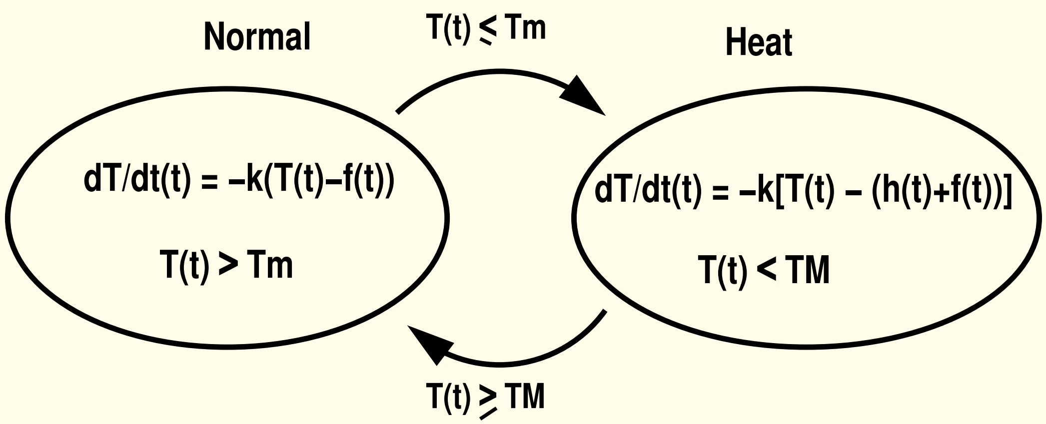
Generalizations of transition systems

- More detailed description of states: Abstract state machines
- Emphasis on processes and their interdependency: CSP

- Durations: Timed automata
- Continuous evolution + discrete control: Hybrid automata

Hybrid Automata

Hybrid Automata



f : R –> R evolution of external temperature

h : R –> R evolution of heater temperature

Hybrid Automata

Hybrid automaton (HA) $S = (X, Q, \text{flow}, \text{Inv}, \text{Init}, E, \text{jump})$ where:

(1) $X = \{x_1, \dots, x_n\}$ finite set of real valued variables
Q finite set of control modes

(2) $\{\text{flow}_q \mid q \in Q\}$ specify the continuous dynamics in each control mode (flow$_q$ predicate over $\{x_1, \dots, x_n\} \cup \{\dot{x}_1, \dots, \dot{x}_n\}$).

(3) $\{\text{Inv}_q \mid q \in Q\}$ mode invariants (predicates over X).

(4) $\{\text{Init}_q \mid q \in Q\}$ initial states for control modes (predicates over X).

(5) E: control switches (finite multiset with elements in $Q \times Q$).

(6) $\{\text{guard}_e \mid e \in E\}$ guards for control switches (predicates over X).

(7) Jump conditions $\{\text{jump}_e \mid e \in E\}$, (predicates over $X \cup X'$), where $X' = \{x'_1, \dots, x'_n\}$ is a copy of X consisting of "primed" variables.

Linear Hybrid Automata

Atomic linear predicate: linear inequality (e.g. $3x_1 - x_2 + 7x_5 \leq 4$).

Convex linear predicate: finite conjunction of linear inequalities.

A state assertion s for S: family $\{s(q) \mid q \in Q\}$, where $s(q)$ is a predicate over X (expressing constraints which hold in state s for mode q).

Definition [Henzinger 1997] A linear hybrid automaton (LHA) is a hybrid automaton which satisfies the following requirements:

(1) Linearity:

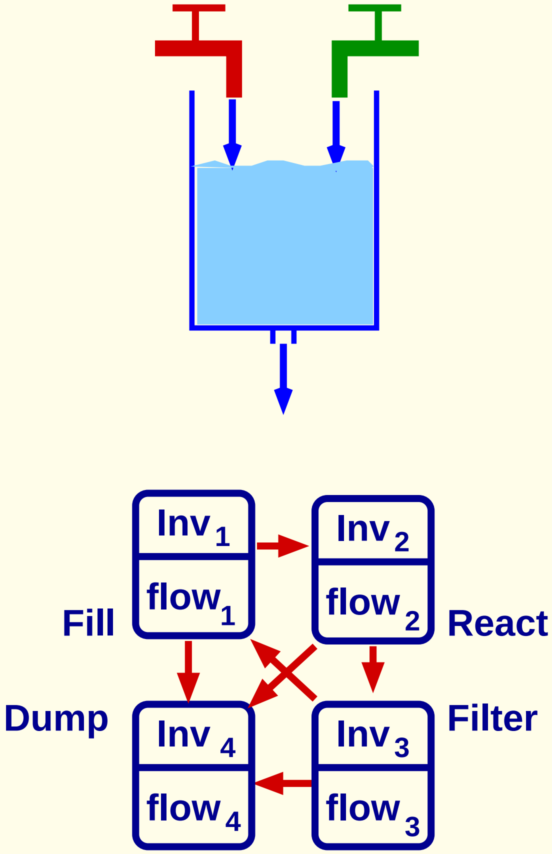
- For every $q \in Q$, flow_q, Inv_q, and Init_q are convex linear predicates.
- For every $e = (q, q') \in E$, jump_e and guard_e are convex linear predicates.

We assume that flow_q are conjunctions of *non-strict* inequalities.

(2) Flow independence:

For every $q \in Q$, flow_q is a predicate over $\dot{X}$ only.

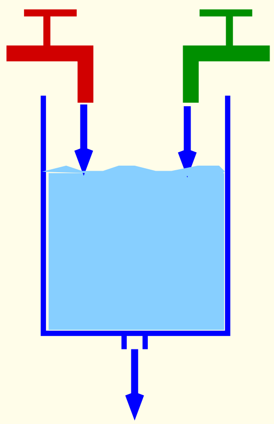
Example



Chemical plant

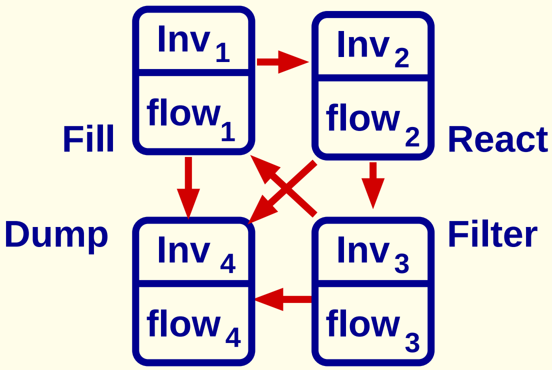
Two substances are mixed; they react;
the resulting product is filtered out;
then the procedure is repeated.

Example



Chemical plant

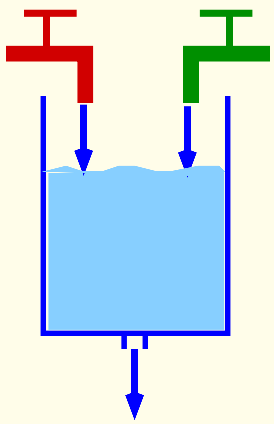
Two substances are mixed; they react;
the resulting product is filtered out;
then the procedure is repeated.

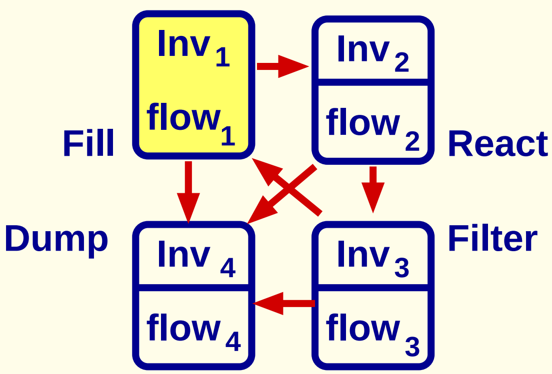


Check:

- No overflow
- Substances in the right proportion
- If substances in wrong proportion,
  tank can be drained in $\leq$ 200s.

Example





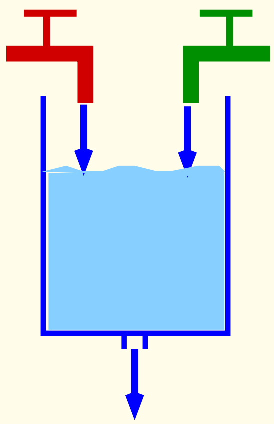
Mode 1: Fill Temperature is low, 1 and 2 do not react.

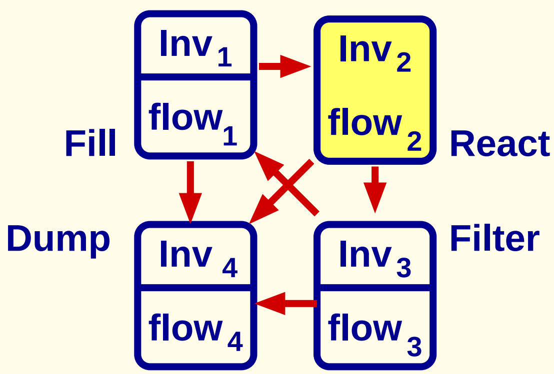
Substances 1 and 2 (possibly mixed with a small quantity of 3) are filled in the tank in equal quantities up to a margin of error.

$$\begin{array}{ll} \mathrm{Inv}_1 & x_1 + x_2 + x_3 \leq L_f \ \wedge \ \bigwedge_{i=1}^{3} x_i \geq 0 \ \wedge \\ & -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \ \wedge \ 0 \leq x_3 \leq \mathrm{min} \\ \mathrm{flow}_1 & \dot{x_1} \geq \mathrm{dmin} \wedge \dot{x_2} \geq \mathrm{dmin} \wedge \dot{x_3} = 0 \wedge -\delta_a \leq \dot{x_1} - \dot{x_2} \leq \delta_a \end{array}$$

If proportion not kept: system jumps into mode 4 (**Dump**);
If the total quantity of substances exceeds level L_f (tank filled) the system jumps into mode 2 (**React**).

Example



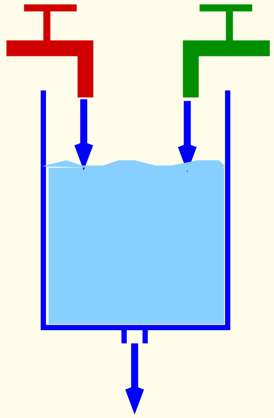


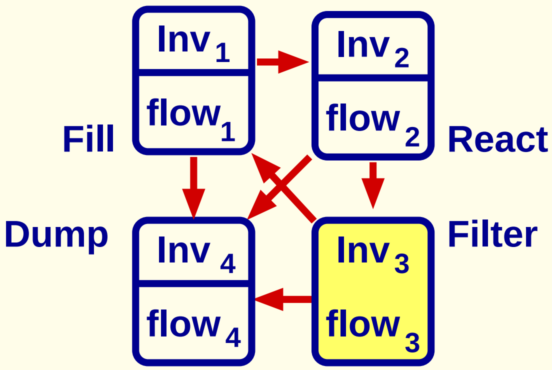
Mode 2: React Temparature is high. Substances 1 and 2 react. The reaction consumes equal quantities of substances 1 and 2 and produces substance 3.

$$
\begin{array}{ll}
\mathrm{Inv}_2 & L_f \leq x_1 + x_2 + x_3 \leq L_{\mathrm{overflow}} \;\wedge\; \bigwedge_{i=1}^{3} x_i \geq 0 \;\wedge \\
 & -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \;\wedge\; 0 \leq x_3 \leq \max \\
\mathrm{flow}_2 & \dot{x}_1 \leq -\mathrm{dmin} \wedge \dot{x}_2 \leq -\mathrm{dmin} \wedge \dot{x}_3 \geq \mathrm{dmin} \\
 & \wedge\; \dot{x}_1 = \dot{x}_2 \;\wedge\; \dot{x}_3 + \dot{x}_1 + \dot{x}_2 = 0
\end{array}
$$

If the proportion between substances 1 and 2 is not kept
the system jumps into mode 4 (**Dump**);
If the total quantity of substances 1 and 2 is below some minimal level min the system jumps into mode 3 (**Filter**).

Example



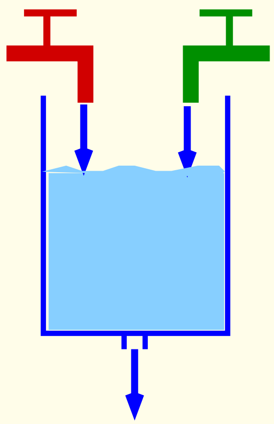


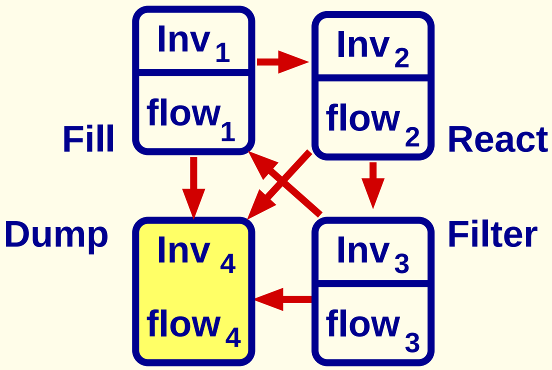
Mode 3: Filter Temperature is low. Substance 3 is filtered out.

$$\begin{array}{ll} \mathrm{Inv}_3 & x_1 + x_2 + x_3 \leq L_{\mathrm{overflow}} \quad \wedge \quad \bigwedge_{i=1}^{3} x_i \geq 0 \quad \wedge \\ & -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \quad \wedge \quad x_3 \geq \min \\ \mathrm{flow}_3 & \dot{x_1} = 0 \wedge \dot{x_2} = 0 \quad \wedge \quad \dot{x_3} \leq -\mathrm{dmin} \end{array}$$

If proportion not kept: system jumps into mode 4 (**Dump**); Otherwise, if the concentration of substance 3 is below some minimal level min the system jumps into mode 1 (**Fill**).

Example





Mode 4: Dump The content of the tank is emptied.

For simplicity we assume that this happens instantaneously:

$$\mathrm{Inv}_4 : \bigwedge_{i=1}^{3} x_i = 0 \text{ and } \mathrm{flow}_4 : \bigwedge_{i=1}^{3} \dot{x}_i = 0.$$

Remark

The material on ASMs is not required for the exam (only the general idea)

The definitions of timed automata and hybrid automata are required for the exam.

More complex specifications and specification languages

- Languages for describing various processes
- Languages based on Set theory (OZ, B)
- Languages for describing durations
- Complex languages

More complex specifications and specification languages

- Languages for describing various processes
- Languages based on Set theory (OZ, B)
- Languages for describing durations
- Complex languages

CSP

Communicating Sequential Processes, or CSP, is a language for describing processes and patterns of interaction between them.

It is supported by an elegant, mathematical theory, a set of proof tools, and an extensive literature.

CSP

Communicating Sequential Processes, or CSP, is a language for describing processes and patterns of interaction between them.

It is supported by an elegant, mathematical theory, a set of proof tools, and an extensive literature.

- Each process: transition system
- Operations on processes: sequential, parallel composition

efects on states

CSP

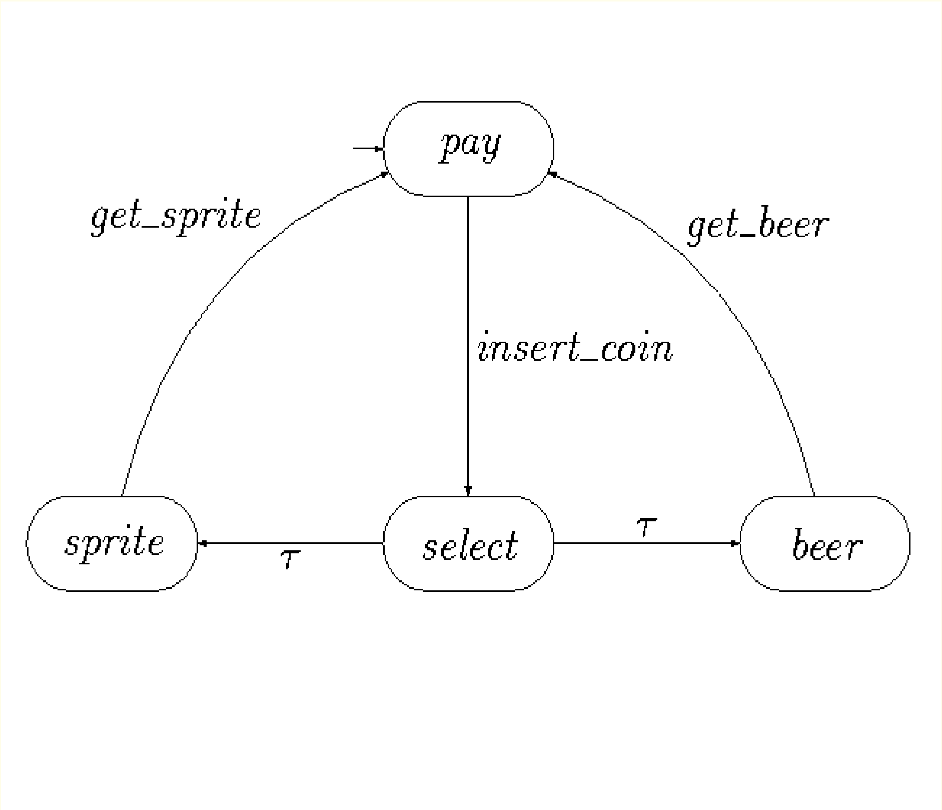
General idea:

Given:

- Set of event names
- Process: behaviour pattern of an object (insofar as it can be described in terms of the limited set of events selected as its alphabet)

CSP

Example:



Events: insert-coin, get-sprite, get-beer

CSP

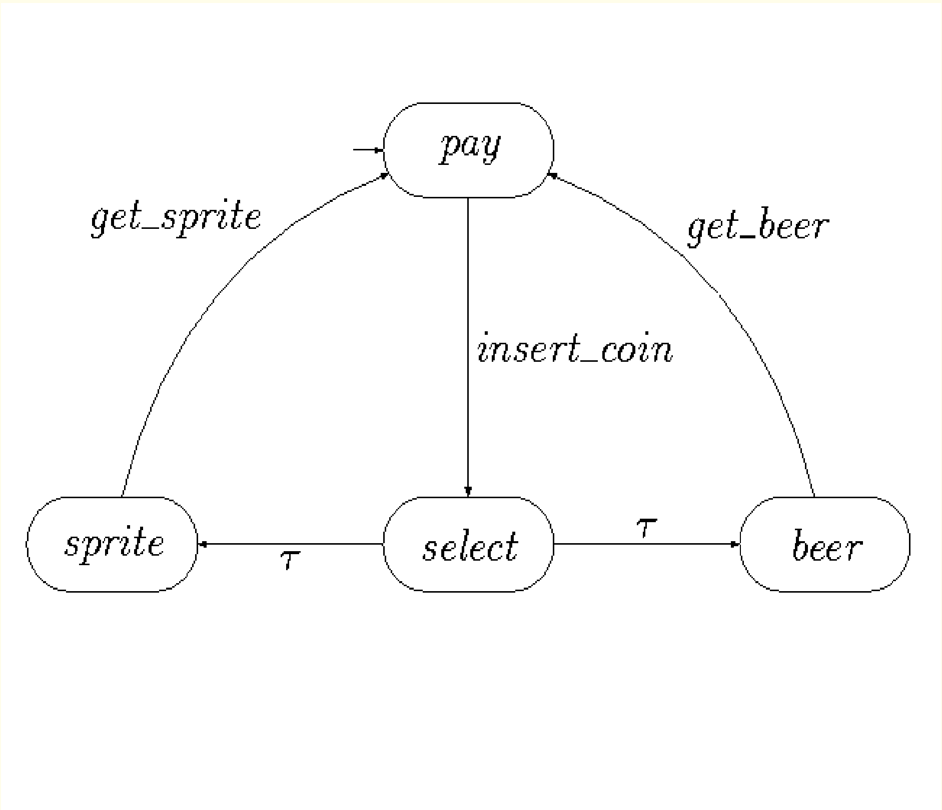
Prefix:

$P = a \rightarrow Q$ (*a* then *Q*)

where a is an event and Q a process

After event a, process P behaves like process Q

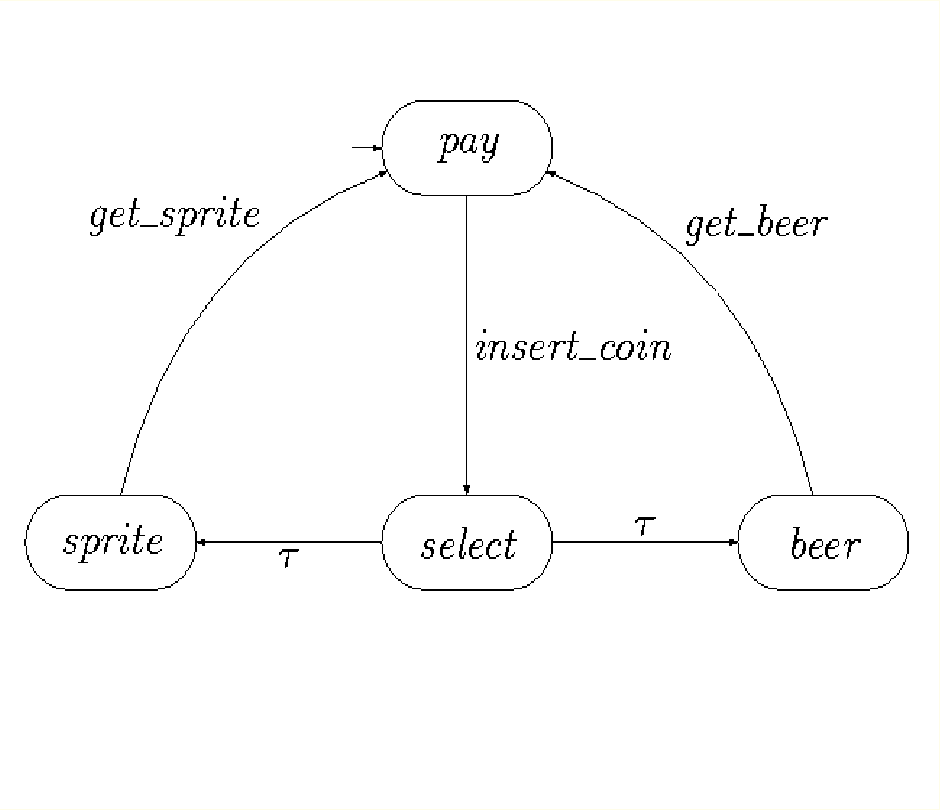
CSP: Example



A simple vending machine which consumes one coin before breaking

$$(\textit{insert-coin} \rightarrow \textit{STOP})$$

CSP: Example



A simple vending machine that successfully serves two customers before breaking

$$(\mathit{insert\text{-}coint} \rightarrow (\mathit{get\text{-}sprite} \rightarrow (\mathit{insert\text{-}coin} \rightarrow (\mathit{get\text{-}beer} \rightarrow \mathit{STOP}))))$$

CSP

Example: (recursive definitions)

Consider the simplest possible everlasting object, a clock which never does anything but tick (the act of winding is deliberately ignored)

$$Events(CLOCK) = \{tick\}$$

Consider next an object that behaves exactly like the clock, except that it first emits a single tick

$$(tick \rightarrow CLOCK)$$

The behaviour of this object is indistinguishable from that of the original clock. This reasoning leads to formulation of the equation

$$CLOCK = (tick \rightarrow CLOCK)$$

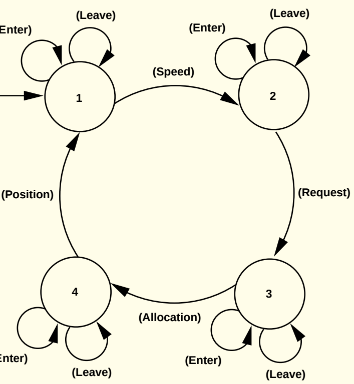
This can be regarded as an implicit definition of the behaviour of the clock.

Modular Specifications: CSP-OZ-DC (COD)

COD [Hoenicke,Olderog'02] allows us to specify in a modular way:

- the control flow of a system
  using Communicating Sequential Processes (CSP)
- the state space and its change
  using Object-Z (OZ)
- (dense) real-time constraints over durations of events
  using the Duration Calculus (DC)

Example: Controller for line track (RBC)



RBC

`method` $enter : [s1? : Segment;\ t0? : Train;\ t1? : Train;\ t2? : Train]$

`method` $leave : [ls? : Segment;\ lt? : Train]$

`local_chan` $alloc, req, updPos, updSpd$

`main`	$\stackrel{c}{=}$	$((enter \rightarrow \mathtt{main})$	$State2$	$\stackrel{c}{=}$	$((alloc \rightarrow State3)$
	$\Box$	$(leave \rightarrow \mathtt{main})$		$\Box$	$(enter \rightarrow State2)$
	$\Box$	$(updSpd \rightarrow State1))$		$\Box$	$(leave \rightarrow State2))$
$State1$	$\stackrel{c}{=}$	$((enter \rightarrow State1)$	$State3$	$\stackrel{c}{=}$	$((enter \rightarrow State3)$
	$\Box$	$(leave \rightarrow State1)$		$\Box$	$(leave \rightarrow State3)$
	$\Box$	$(req \rightarrow State2))$		$\Box$	$(updPos \rightarrow \mathtt{main}))$

SegmentData

$train : Segment \rightarrow Train$	[Train on segment]
$req : Segment \rightarrow \mathbb{Z}$	[Requested by train]
$alloc : Segment \rightarrow \mathbb{Z}$	[Allocated by train]

TrainData

$segm : Train \rightarrow Segment$	[Train segment]
$next : Train \rightarrow Train$	[Next train]
$spd : Train \rightarrow \mathbb{R}$	[Speed]
$pos : Train \rightarrow \mathbb{R}$	[Current position]
$prev : Train \rightarrow Train$	[Prev. train]

$sd : SegmentData$
$td : TrainData$

$\forall t : Train \Gamma tid(t) > 0$
$\forall t1, t2 : Train \mid t1 \neq t2 \Gamma tid(t1) \neq tid(t2)$
$\forall s : Segment \Gamma prevs(nexts(s)) = s$
$\forall s : Segment \Gamma nexts(prevs(s)) = s$
$\forall s : Segment \Gamma sid(s) > 0$
$\forall s : Segment \Gamma sid(nexts(s)) > sid(s)$
$\forall s1, s2 : Segment \mid s1 \neq s2 \Gamma sid(s1) \neq sid(s2)$
$\forall s : Segment \mid s \neq snil \Gamma length(s) > d + gmax \cdot \Delta t$
$\forall s : Segment \mid s \neq snil \Gamma 0 < lmax(s) \wedge lmax(s) \leq gmax$
$\forall s : Segment \Gamma lmax(s) \geq lmax(prevs(s)) - decmax \cdot \Delta t$
$\forall s1, s2 : Segment \Gamma tid(incoming(s1)) \neq tid(train(s2))$

`Init`

$\forall t : Train \Gamma train(segm(t)) = t$
$\forall t : Train \Gamma next(prev(t)) = t$
$\forall t : Train \Gamma prev(next(t)) = t$
$\forall t : Train \Gamma 0 \leq pos(t) \leq length(segm(t))$
$\forall t : Train \Gamma 0 \leq spd(t) \leq lmax(segm(t))$
$\forall t : Train \Gamma alloc(segm(t)) = tid(t)$
$\forall t : Train \Gamma alloc(nexts(segm(t))) = tid(t)$
$\quad \vee\ length(segm(t)) - bd(spd(t)) > pos(t)$
$\forall s : Segment \Gamma segm(train(s)) = s$

`effect_`*updSpd*

$\Delta(spd)$

$\forall t : Train \mid pos(t) < length(segm(t)) - d \wedge spd(t) - decmax \cdot \Delta t > 0$
$\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq lmax(segm(t))$
$\forall t : Train \mid pos(t) \geq length(segm(t)) - d \wedge alloc(nexts(segm(t))) = tid(t)$
$\quad \Gamma \max\{0, spd(t) - decmax \cdot \Delta t\} \leq spd'(t) \leq \min\{lmax(segm(t)), lmax(nexts(segm(t)))\}$
$\forall t : Train \mid pos(t) \geq length(segm(t)) - d \wedge \neg\ alloc(nexts(segm(t))) = tid(t)$
$\quad \Gamma spd'(t) = \max\{0, spd(t) - decmax \cdot \Delta t\}$

$\vdots$

CSP

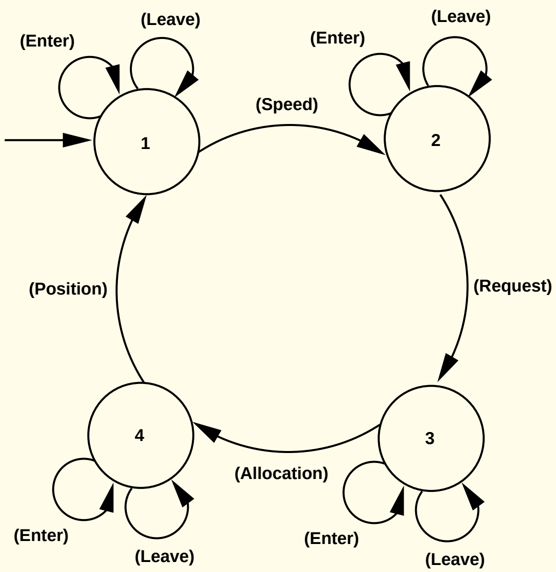
OZ

Example: Controller for line track (RBC)

CSP part: specifies the processes and their interdependency.

The RBC system passes repeatedly through four phases, modeled by events:

- updSpd (speed update)
- req (request update)
- alloc (allocation update)
- updPos (position update)



Between these events, trains may leave or enter the track (at specific segments), modeled by the events leave and enter.

Example: Controller for line track (RBC)

CSP part: specifies the processes and their interdependency.

The RBC system passes repeatedly through four phases, modeled by events with corresponding COD schemata:

CSP:

$\texttt{method}\ \mathit{enter} : [s1? : \mathit{Segment};\ t0? : \mathit{Train};\ t1? : \mathit{Train};\ t2? : \mathit{Train}]$

$\texttt{method}\ \mathit{leave} : [\mathit{ls}? : \mathit{Segment};\ \mathit{lt}? : \mathit{Train}]$

$\texttt{local\_chan}\ \mathit{alloc}, \mathit{req}, \mathit{updPos}, \mathit{updSpd}$

$$\begin{array}{llll}
\texttt{main} \stackrel{c}{=} ((\mathit{updSpd} \to \mathit{State1}) & \mathit{State1} \stackrel{c}{=} ((\mathit{req} \to \mathit{State2}) & \mathit{State2} \stackrel{c}{=} ((\mathit{alloc} \to \mathit{State3}) & \mathit{State3} \stackrel{c}{=} ((\mathit{updPos} \to \texttt{main}) \\
\quad \Box (\mathit{leave} \to \texttt{main}) & \quad \Box (\mathit{leave} \to \mathit{State1}) & \quad \Box (\mathit{leave} \to \mathit{State2}) & \quad \Box (\mathit{leave} \to \mathit{State3}) \\
\quad \Box (\mathit{enter} \to \texttt{main})) & \quad \Box (\mathit{enter} \to \mathit{State1})) & \quad \Box (\mathit{enter} \to \mathit{State2})) & \quad \Box (\mathit{enter} \to \mathit{State3}))
\end{array}$$

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the `Init` schema, update rules.

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the `Init` schema, update rules.

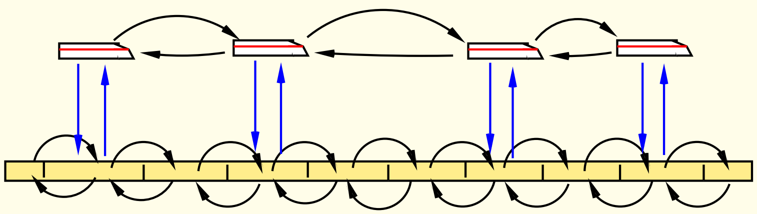
- 1. Data classes declare function symbols that can change their values during runs of the system

Data structures:

- 2-sorted pointers

train: trains

segm: segments



SegmentData	
$train : Segment \rightarrow Train$	
	[Train on segment]
$req : Segment \rightarrow \mathbb{Z}$	[Requested by train]
$alloc : Segment \rightarrow \mathbb{Z}$	
	[Allocated by train]

TrainData	
$segm : Train \rightarrow Segment$	
	[Train segment]
$next : Train \rightarrow Train$	[Next train]
$spd : Train \rightarrow \mathbb{R}$	[Speed]
$pos : Train \rightarrow \mathbb{R}$	[Current position]
$prev : Train \rightarrow Train$	[Prev. train]

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the `Init` schema, update rules.

- 1. Data classes declare function symbols that can change their values during runs of the system, and are used in the OZ part of the specification.

- 2. Axioms: define properties of the data structures and system parameters which do not change
  - $gmax : \mathbb{R}$ (the global maximum speed),
  - $decmax : \mathbb{R}$ (the maximum deceleration of trains),
  - $d : \mathbb{R}$ (a safety distance between trains),
  - Properties of the data structures used to model trains/segments

Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the `Init` schema, update rules.

- 3. Init schema. describes the initial state of the system.
  - trains - doubly-linked list; placed correctly on the track segments
  - all trains respect their speed limits.

- 4. Update rules specify updates of the state space executed when the corresponding event from the CSP part is performed.

  Example: Speed update

$$
\begin{array}{l}
\texttt{effect\_}\mathit{updSpd} \\
\Delta(\mathit{spd}) \\
\hline
\forall t : \mathit{Train} \mid \mathit{pos}(t) < \mathit{length}(\mathit{segm}(t)) - d \wedge \mathit{spd}(t) - \mathit{decmax} \cdot \Delta t > 0 \\
\quad \Gamma \max\{0, \mathit{spd}(t) - \mathit{decmax} \cdot \Delta t\} \leq \mathit{spd}'(t) \leq \mathit{lmax}(\mathit{segm}(t)) \\
\forall t : \mathit{Train} \mid \mathit{pos}(t) \geq \mathit{length}(\mathit{segm}(t)) - d \wedge \mathit{alloc}(\mathit{nexts}(\mathit{segm}(t))) = \mathit{tid}(t) \\
\quad \Gamma \max\{0, \mathit{spd}(t) - \mathit{decmax} \cdot \Delta t\} \leq \mathit{spd}'(t) \leq \min\{\mathit{lmax}(\mathit{segm}(t)), \mathit{lmax}(\mathit{nexts}(\mathit{segm}(t)))\} \\
\forall t : \mathit{Train} \mid \mathit{pos}(t) \geq \mathit{length}(\mathit{segm}(t)) - d \wedge \neg\, \mathit{alloc}(\mathit{nexts}(\mathit{segm}(t))) = \mathit{tid}(t) \\
\quad \Gamma \mathit{spd}'(t) = \max\{0, \mathit{spd}(t) - \mathit{decmax} \cdot \Delta t\}
\end{array}
$$

Formal specification

- Specification for program/system
- Specification for properties of program/system

Verification tasks:

Check that the specification of the program/system has the required properties.