Automatic Verification of Real-Time Systems with Rich Data

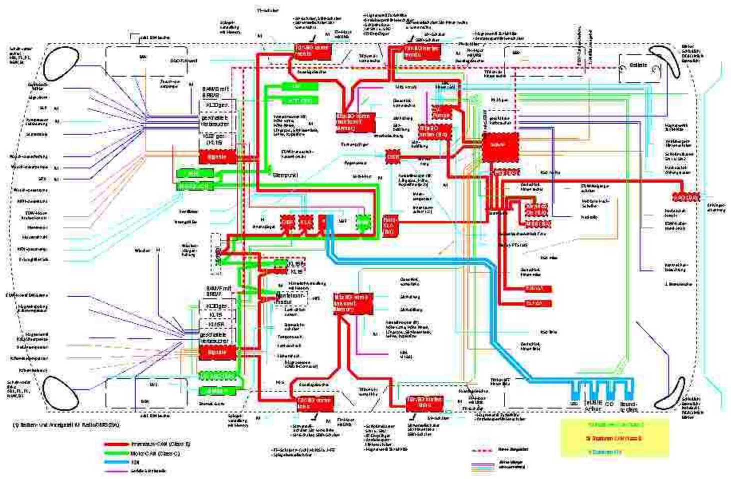
Ernst-Rüdiger Olderog

CARL VON OSSIETZKY universität OLDENBURG

Motivation

Embedded system =

system where computer is invisible part of it
to control its function

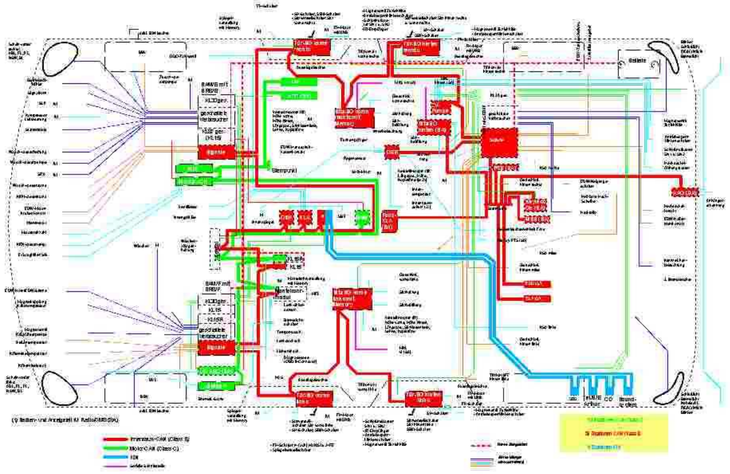


ECUs on board of a cars: Mercedes S class (1998)

Motivation

Embedded system =

system where computer is invisible part of it
to control its function



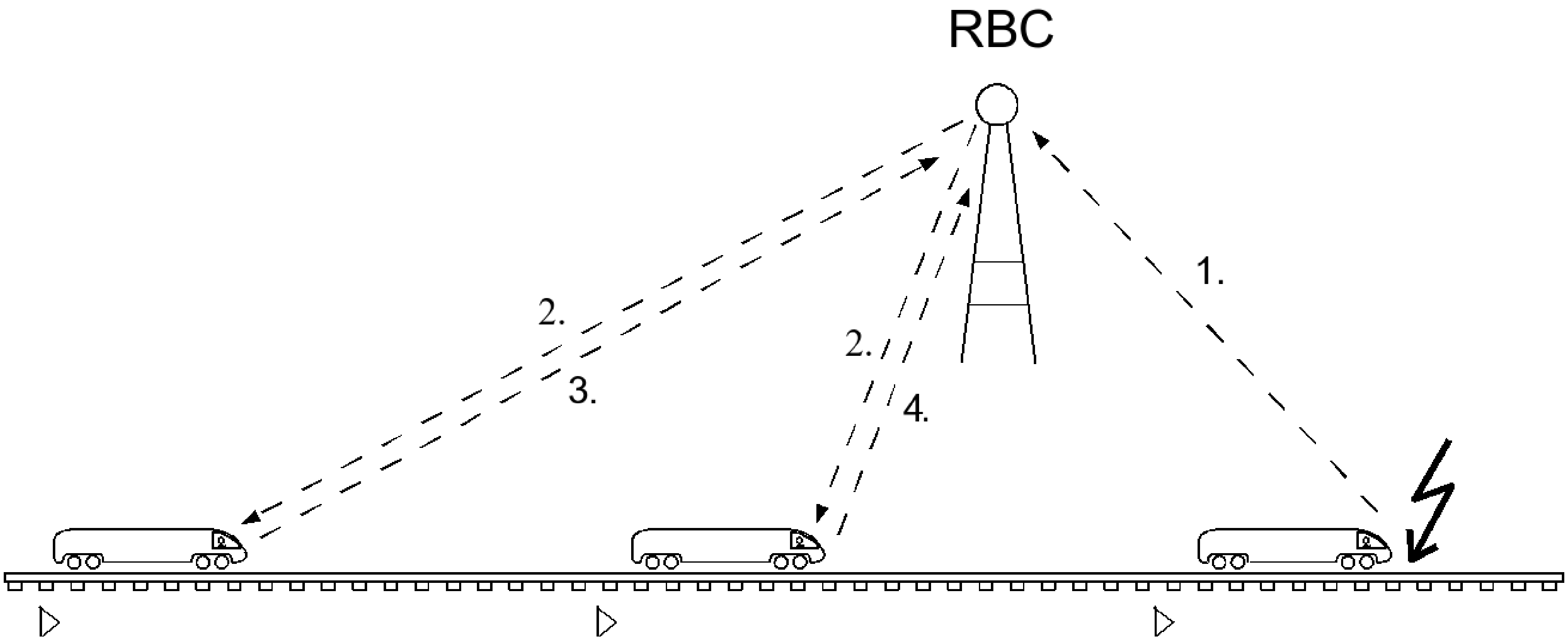
ECUs on board of a cars: Mercedes S class (1998)

Safety-critical applications :

malfunction of computer is costly and dangerous

Trains

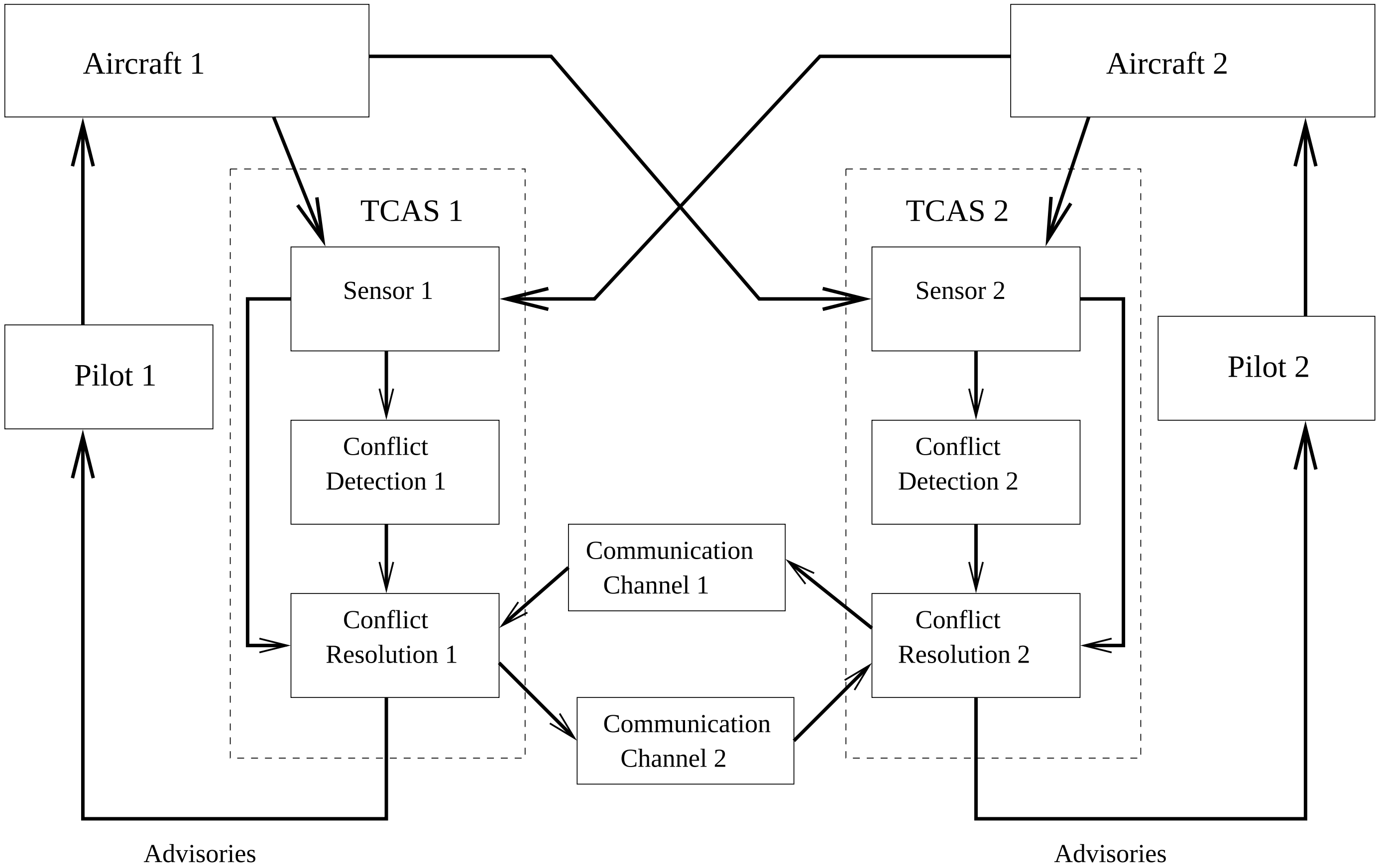
ETCS (European Train Control System) Level 3:



Safety Property: Collision Freedom

Planes

TCAS (Traffic Alert and Collision Avoidance System):

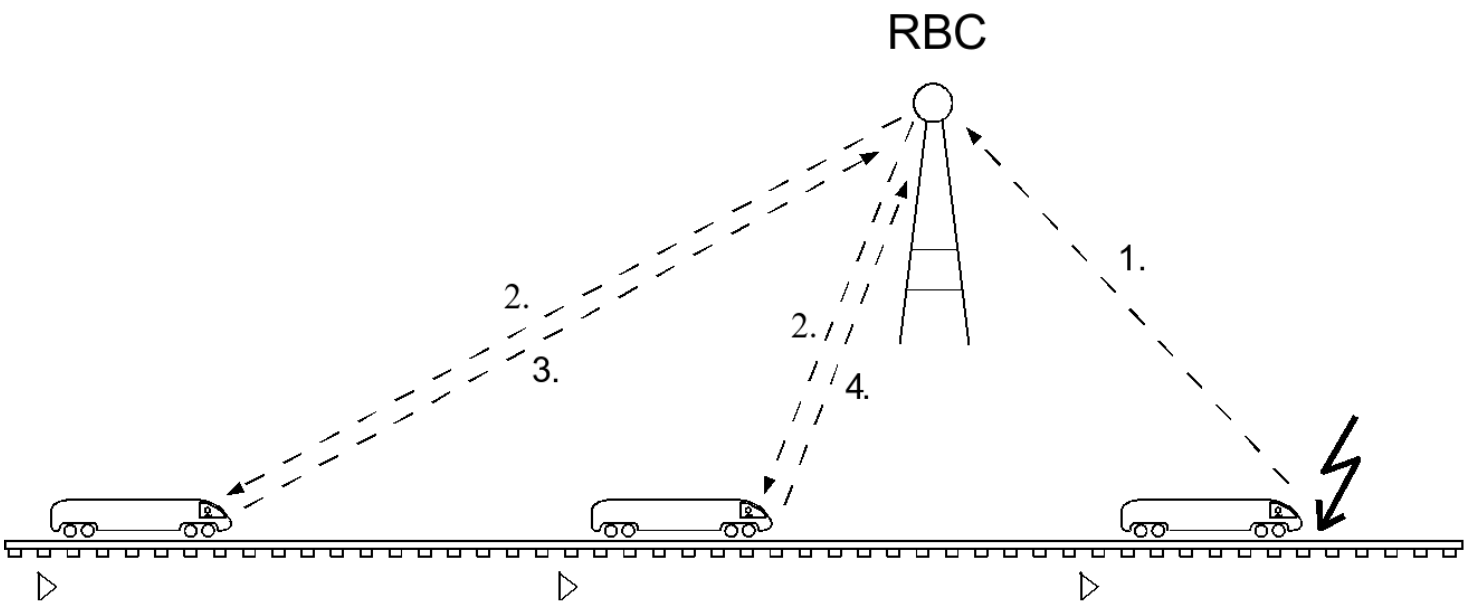


case of two aircrafts

Real-Time Systems

... are reactive systems where certain inputs
require the corresponding outputs
within given time bounds.

Example: European Train Control System (ETCS)



Safety Property: Collision Freedom

AVACS Project Group R

... advances the

automatic verification and analysis of real-time systems

in three complementary projects R1–R3:

- R1: Beyond Timed Automata

  high-level specifications: real-time and complex infinite data

- R2: Timing Analysis, Scheduling, and Distribution of Real-Time Tasks

  implementation level: complex target architectures

- R3: Heuristic Search and Abstract Model Checking for Real-Time Systems

  highly concurrent systems: many clocks and many components

R1: Beyond Timed Automata

E.-R. Olderog,
B. Finkbeiner, M. Fränzle, A. Podelski, V. Sofronie-Stokkermans

... investigates Real-Time Systems with Rich Data:

- System specification language: CSP-OZ-DC

integrates	processes	(Comm. Sequ. Processes)
	data	(Object-Z)
	time	(Duration Calculus)

- Real-time requirements: DC

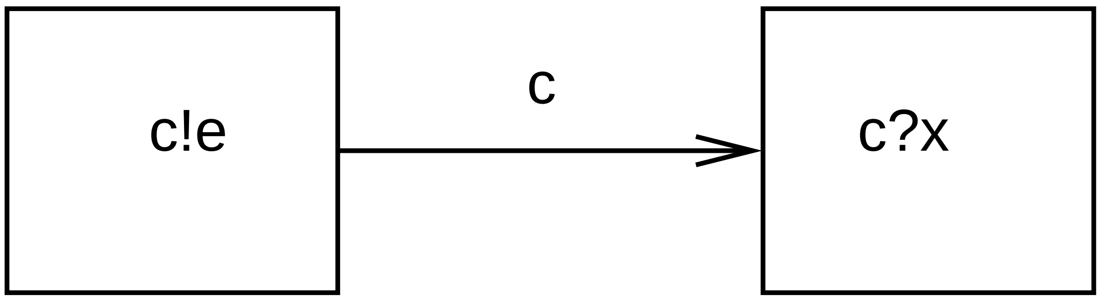
- Problem: Does specification satisfy requirement ?

Specification of Processes

CSP Communicating Sequential Processes

since 1978: Hoare, Brookes, Roscoe

- synchronous communication via channels:



- parallel composition and hiding
- mathematical theory

Specification of Data

Z since 1980: Abrial, Sufrin, Spivey

- state spaces and transformations
- mathematical tool kit
- schema calculus

S	
declarations	
predicate	$x' > x + 1$

OZ Object-Z

since 1995: Duke, Rose, Smith

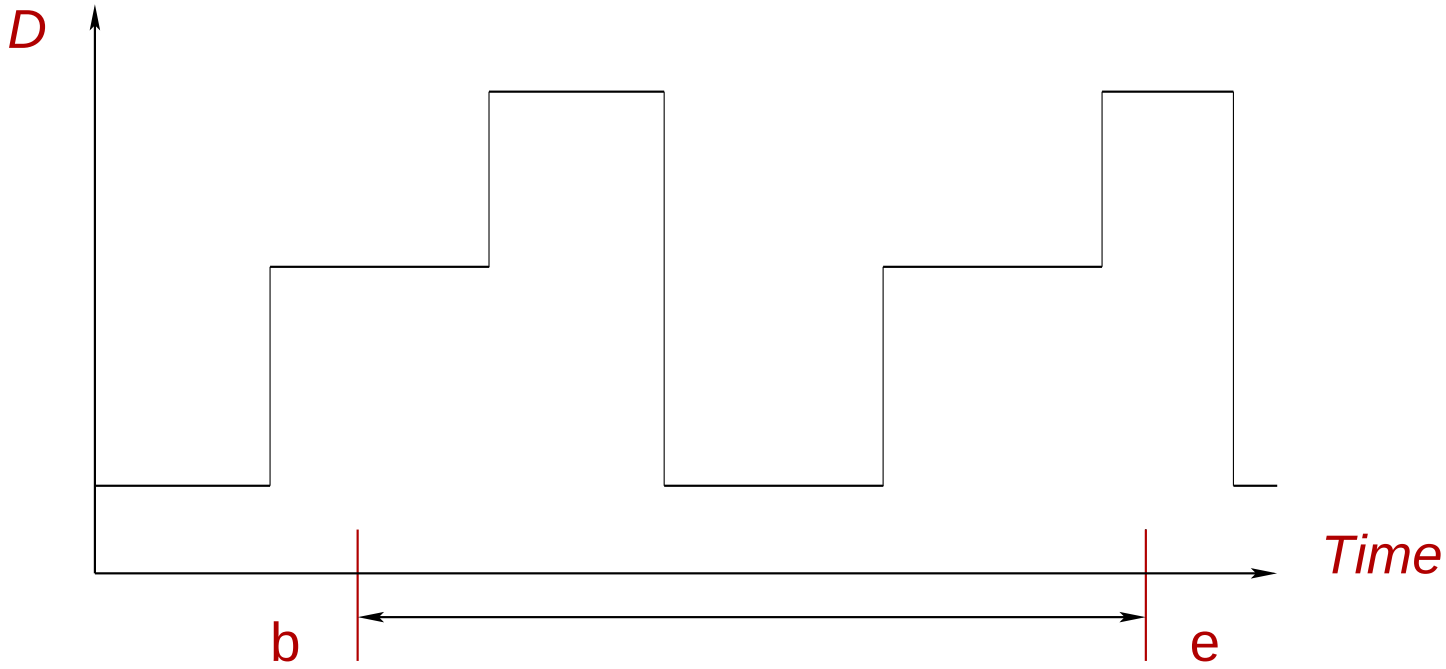
- class concept
- inheritance

Specification of Time

DC Duration Calculus

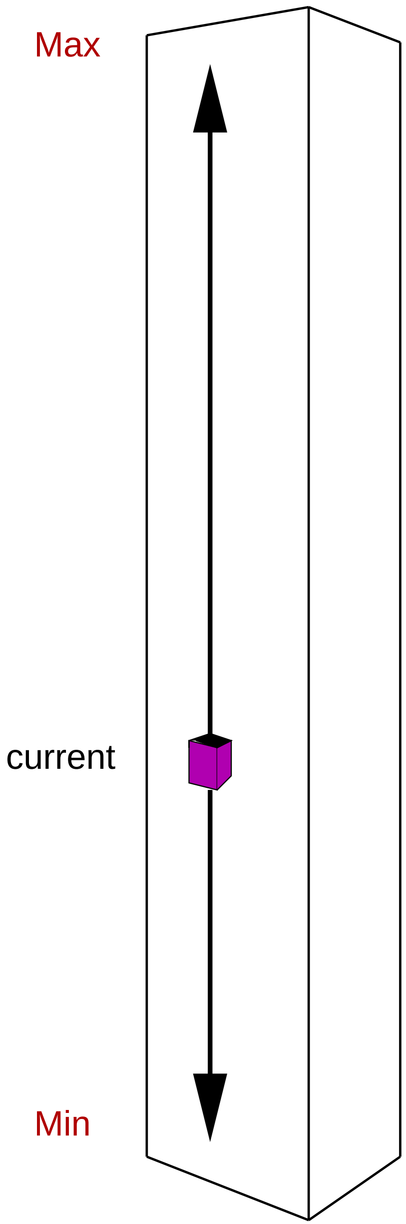
since 1991: Zhou, Hoare, Ravn, Hansen

- real-time logic and calculus
  for properties of $obs : Time \rightarrow D$



- interval-based properties: e.g. durations

Parameterized Elevator



Hoenicke & Maier (2005)

- Elevator specification:

  parameters Max, Min: integers

  real-time requirements: e.g.
  at least 3 sec between two floors

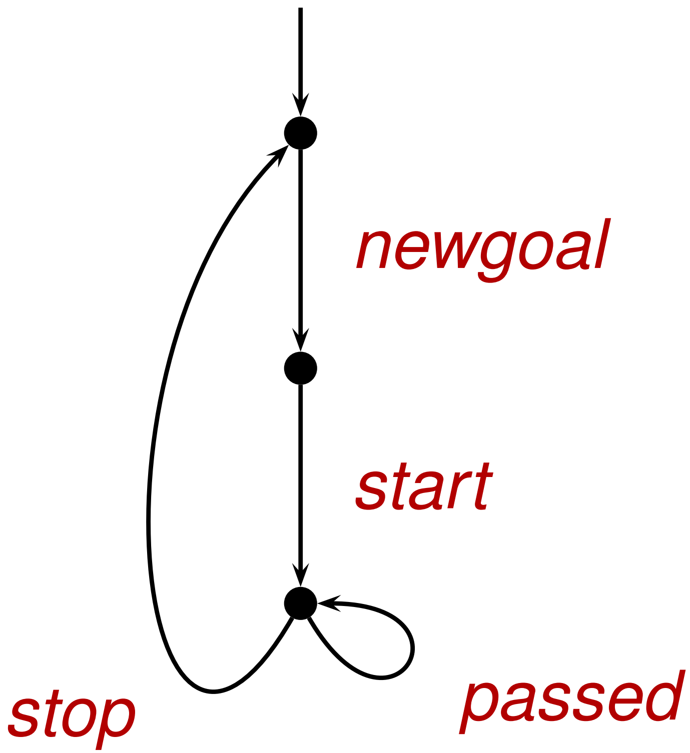
  time domain: reals

- Safety requirement:

  $Min \leq current \leq Max$

Specification: CSP-OZ-DC

Hoenicke & Olderog (since 2002)



Interface:

$\texttt{chan} \quad \mathit{start}, \mathit{passed}, \mathit{stop}, \mathit{newgoal}$

CSP specifies order of events:

$$\texttt{main} \overset{c}{=} \mathit{newgoal} \to \mathit{start} \to \mathit{Drive}$$

$$\mathit{Drive} \overset{c}{=} (\mathit{passed} \to \mathit{Drive}) \\ \quad \Box\ (\mathit{stop} \to \texttt{main})$$

Specification: CSP-OZ-DC

Object-Z specifies state space ...

$$\begin{array}{|l} Min, Max : \mathbb{Z} \\ \hline Min < Max \end{array}$$

$$\begin{array}{|l} current : \mathbb{Z} \\ goal : \mathbb{Z} \\ dir : \{-1, 0, 1\} \\ \hline \end{array}$$

[state space]

Init

$$\begin{array}{|l} goal = current = Min \\ dir = 0 \\ \hline \end{array}$$

Specification: CSP-OZ-DC

... and operations:

$\mathtt{com\_}newgoal$

$\Delta(goal)$

$Min \leq goal' \leq Max$ [nondeterminism]

$goal' \neq current$

$\mathtt{com\_}start$

$\Delta(dir)$

$goal > current \;\Rightarrow\; dir' = 1$

$goal < current \;\Rightarrow\; dir' = -1$

Specification: CSP-OZ-DC

... operations, cont'd:

$\texttt{com}\_passed$

$\Delta(current)$

$current' = current + dir$

$\texttt{com}\_stop$

$\Delta()$

$goal = current$ [precondition]

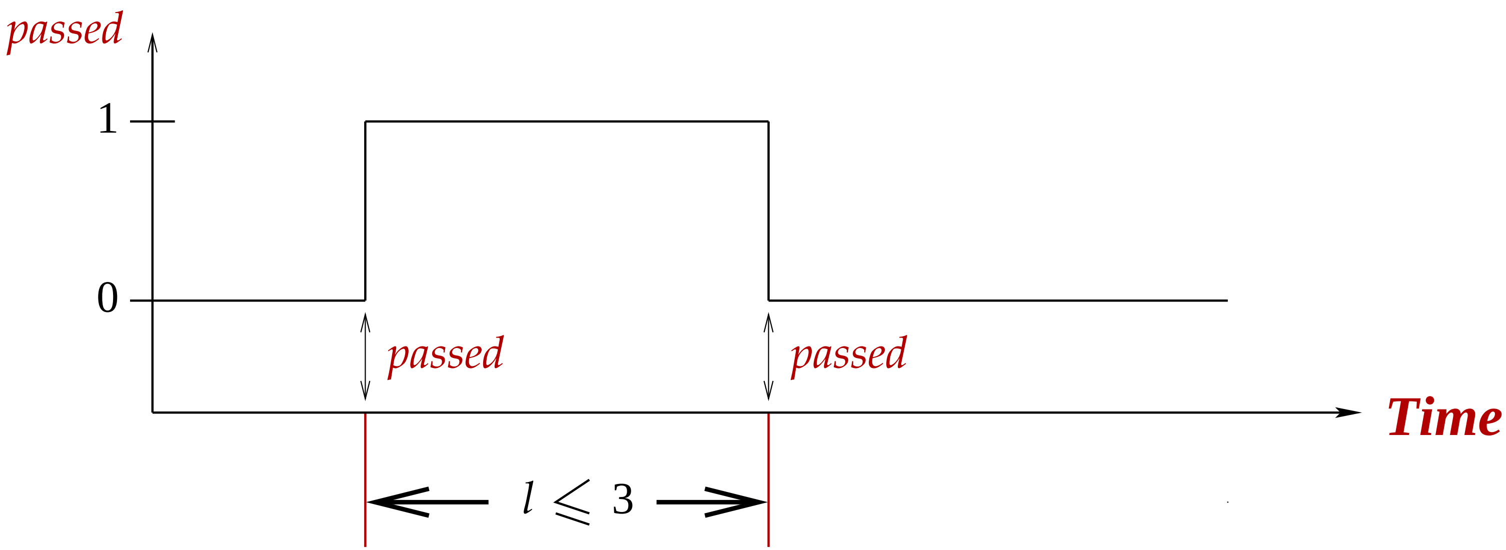
Specification: CSP-OZ-DC

Duration Calculus restricts timing of states and events:

- More than 3 seconds between two *passed* events:

$$\neg \Diamond (\updownarrow \mathit{passed} \,;\, \ell \leq 3 \,;\, \updownarrow \mathit{passed})$$

counterexample trace:

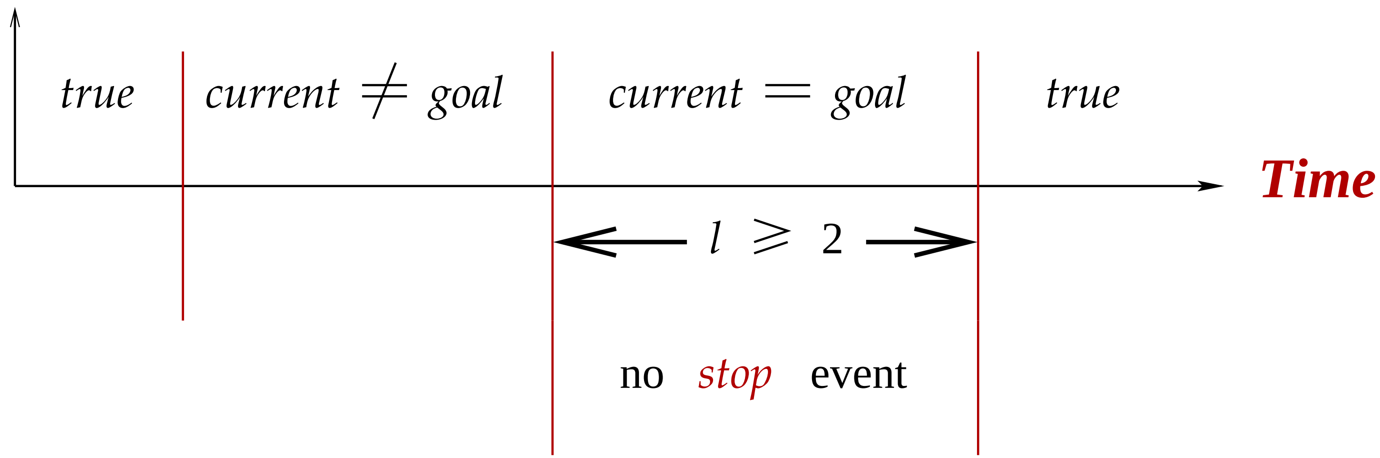


Specification: CSP-OZ-DC

- Event *stop* within 2 sec after reaching *goal*:

$$\neg \Diamond (\lceil current \neq goal \rceil \,;\, (\lceil current = goal \rceil \wedge \ell \geq 2 \wedge \boxminus stop))$$

counterexample trace:



Class Elevator

CSP

OZ

DC

Elevator

$\texttt{chan}\ start, passed, stop, newgoal$

$$\texttt{main} \overset{c}{=} newgoal \rightarrow start \rightarrow Drive$$
$$Drive \overset{c}{=} (passed \rightarrow Drive) \,\square\, (stop \rightarrow \texttt{main})$$

$Min, Max : \mathbb{Z}$

$Min < Max$

$current, goal : \mathbb{Z}$

$dir : \{-1, 0, 1\}$

$\texttt{Init}$

$goal = current = Min$

$dir = 0$

$\texttt{com\_}newgoal$

$\Delta(goal)$

$Min \leq goal' \leq Max$

$goal' \neq current$

$\texttt{com\_}start$

$\Delta(dir)$

$goal > current \ \Rightarrow\ dir' = 1$

$goal < current \ \Rightarrow\ dir' = -1$

$\texttt{com\_}passed$

$\Delta(current)$

$current' = current + dir$

$\texttt{com\_}stop$

$\Delta()$

$goal = current$

$\neg \Diamond (\updownarrow passed \,;\, \ell \leq 3 \,;\, \updownarrow passed)$

$\neg \Diamond (\lceil current \neq goal \rceil \,;\, (\lceil current = goal \rceil \wedge \ell \geq 2 \wedge \boxminus stop))$

Semantics of CSP-OZ-DC

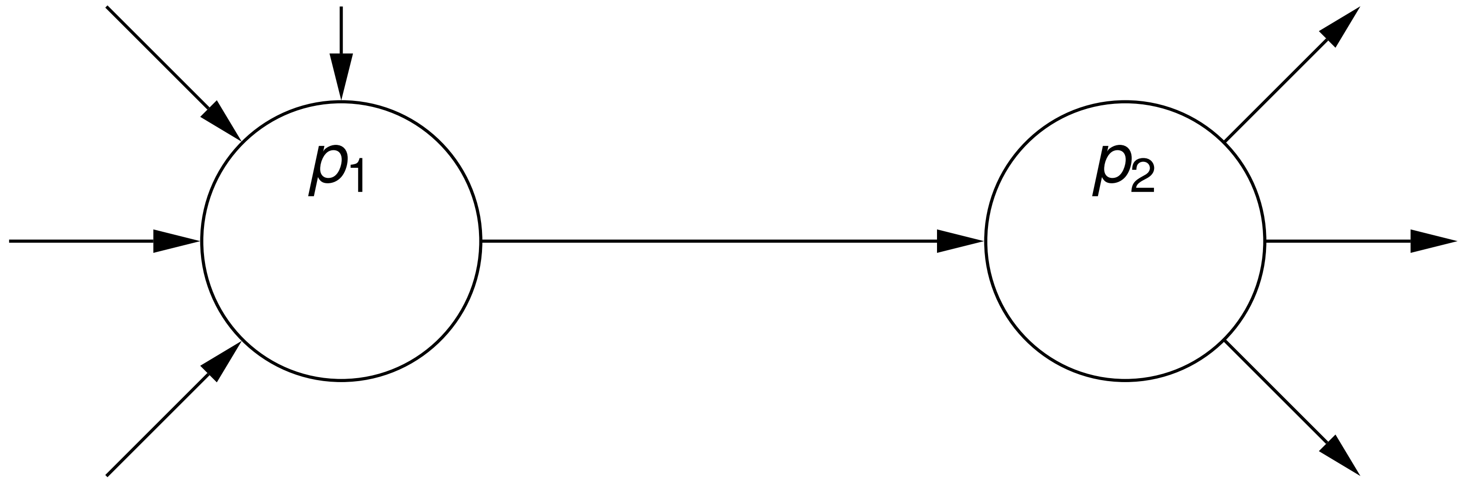
by translation into Phase-Event-Automata (PEA),
a variant of Timed Automata due to Hoenicke (2006)

This semantics is compositional:

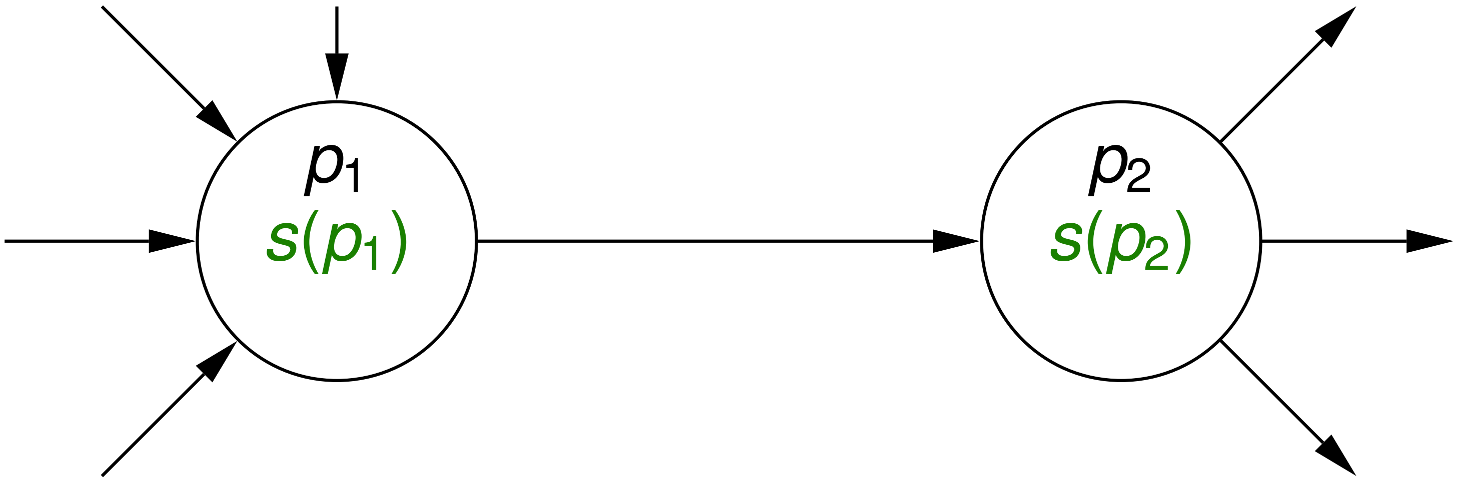
$$\mathcal{A}(COD) = \mathcal{A}(CSP) \parallel \mathcal{A}(OZ) \parallel \mathcal{A}(DC)$$

where $\parallel$ synchronises on both phases and events.

Phase-Event-Automata

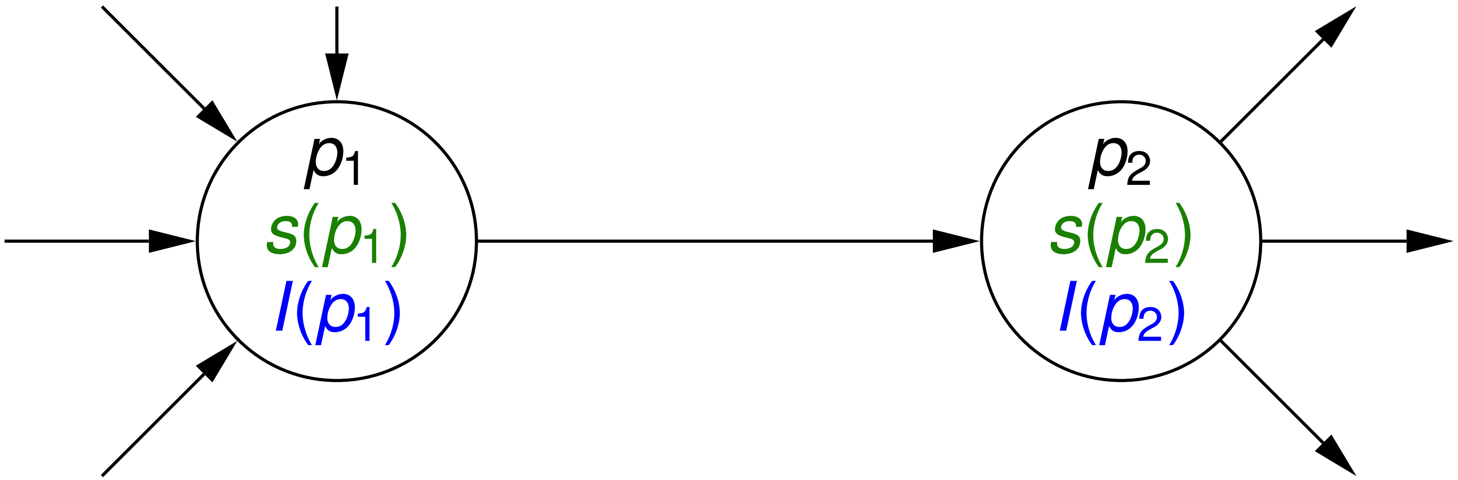


Phase-Event-Automata



$s(p_i)$ state invariant

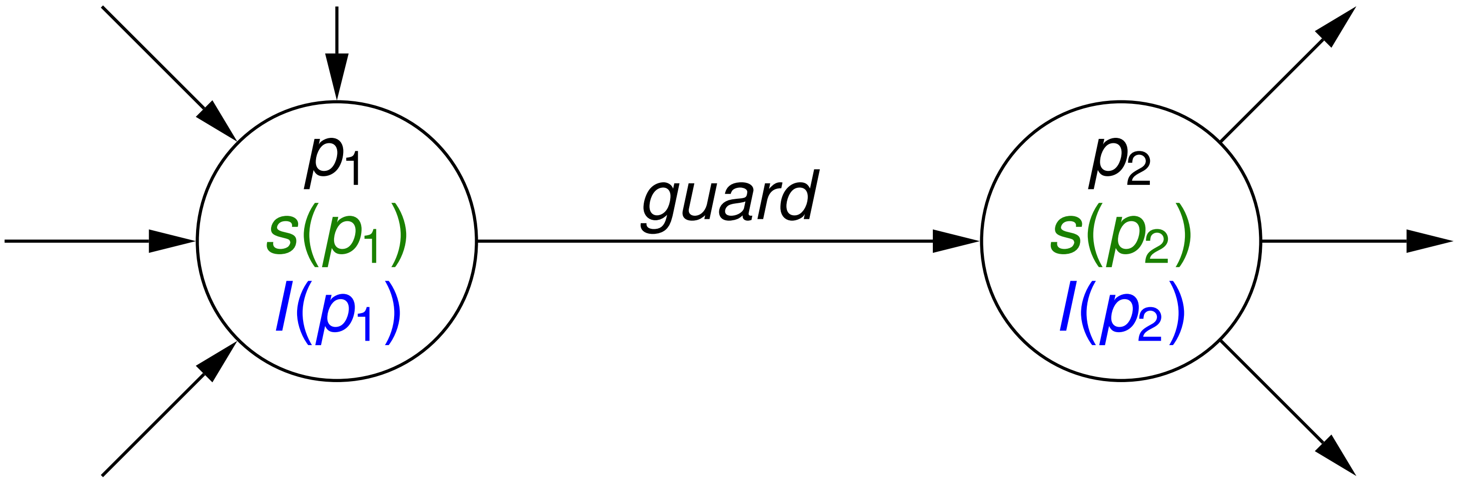
Phase-Event-Automata



$s(p_i)$ state invariant

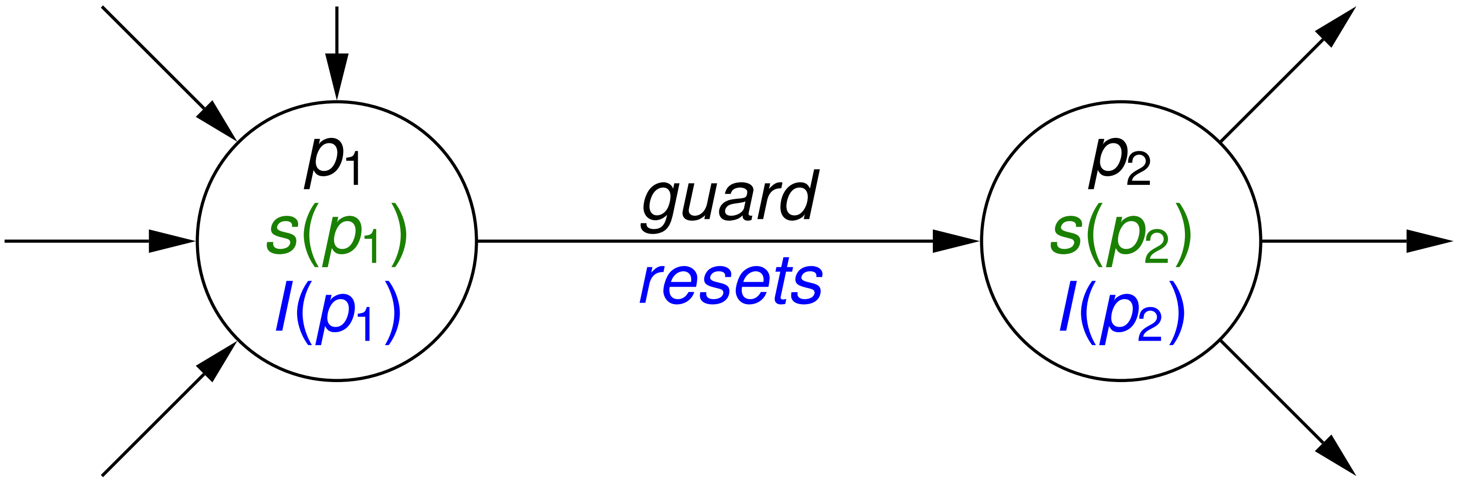
$I(p_i)$ clock invariant

Phase-Event-Automata



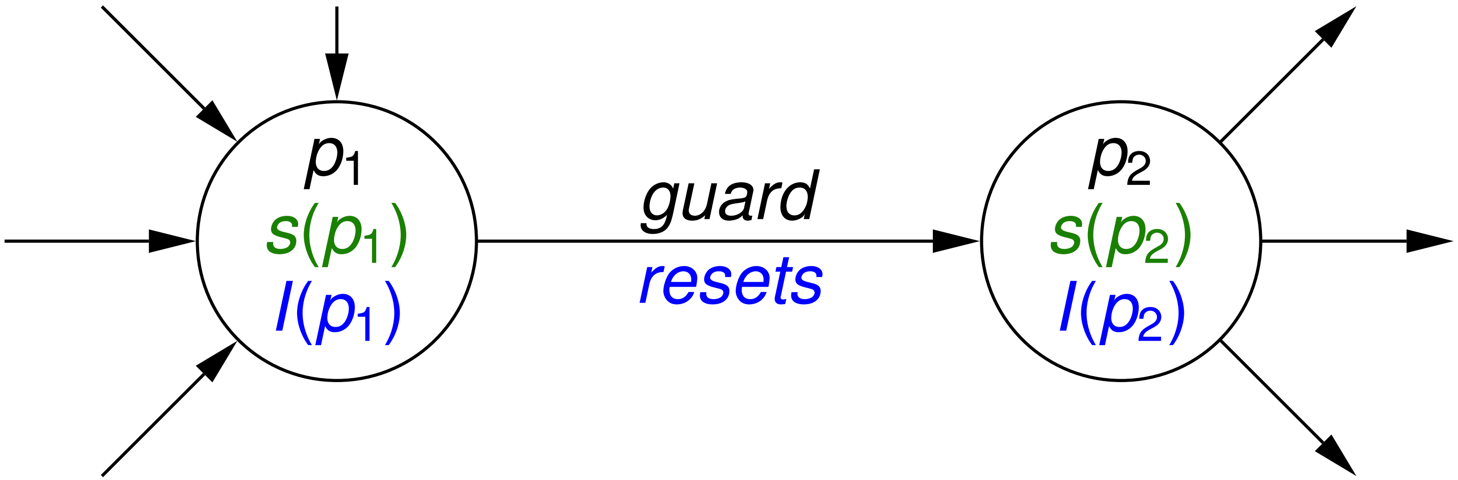
$s(p_i)$	state invariant
$I(p_i)$	clock invariant
guard	conditions over events, state space and time

Phase-Event-Automata



$s(p_i)$	state invariant
$I(p_i)$	clock invariant
guard	conditions over events, state space and time
resets	reset of clocks

Phase-Event-Automata



$s(p_i)$	state invariant
$I(p_i)$	clock invariant
guard	conditions over events, state space and time
resets	reset of clocks

Parallel Composition: $\mathcal{A}_1 \parallel \mathcal{A}_2$

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

each one describing an interval, where

- p_i is a phase,

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

each one describing an interval, where

- p_i is a phase,
- β_i is a valuation of the variables,

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

each one describing an interval, where

- p_i is a phase,
- β_i is a valuation of the variables,
- γ_i is a valuation of the clocks at the beginning of the interval,

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

each one describing an interval, where

- p_i is a phase,
- β_i is a valuation of the variables,
- γ_i is a valuation of the clocks at the beginning of the interval,
- Y_i is a set of events occurring at the beginning of the interval,

PEA Represent Sets of Runs

A run is a sequence of configurations

$$\rho = \langle \ldots, (p_i, \beta_i, \gamma_i, Y_i, t_i), \ldots \rangle$$

each one describing an interval, where

- p_i is a phase,
- β_i is a valuation of the variables,
- γ_i is a valuation of the clocks at the beginning of the interval,
- Y_i is a set of events occurring at the beginning of the interval,
- t_i is a duration of the interval.

Semantic Property of PEA

Compositionality Lemma

$$\rho \in \mathit{Runs}(\,\mathcal{A}_1 \parallel \mathcal{A}_2\,)$$

iff $\rho \downarrow \mathcal{A}_1 \in \mathit{Runs}(\mathcal{A}_1)$ and $\rho \downarrow \mathcal{A}_2 \in \mathit{Runs}(\mathcal{A}_2)$

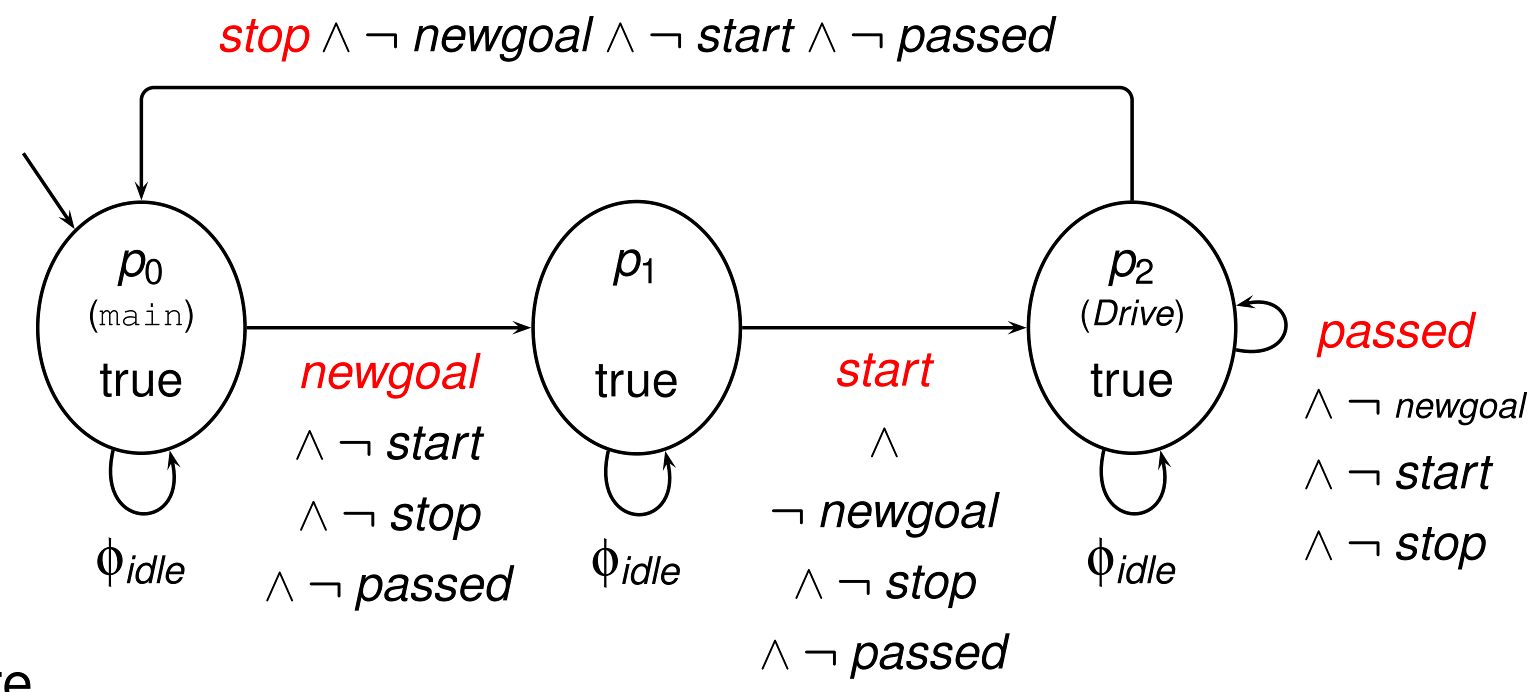
This lemma is at the core of a modular verification method for parallel compositions of PEA:

if a small set of parallel PEA satisfies a safety property, also a larger set of parallel PEA will satisfy it.

Translation of CSP

$$\mathtt{main} \stackrel{c}{=} \mathit{newgoal} \rightarrow \mathit{start} \rightarrow \mathit{Drive}$$

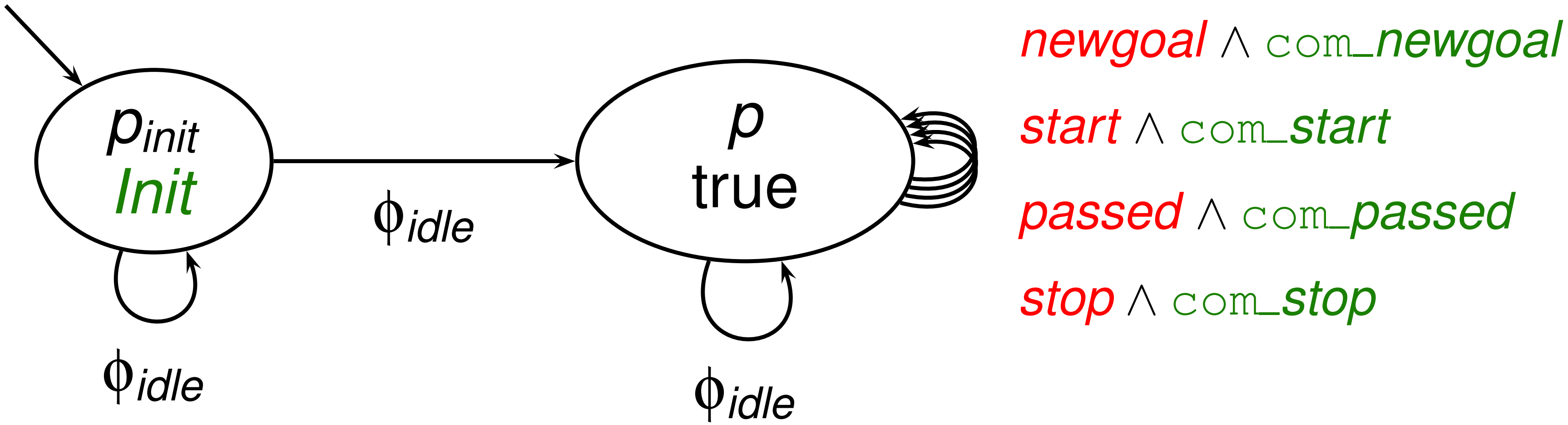
$$\mathit{Drive} \stackrel{c}{=} (\mathit{passed} \rightarrow \mathit{Drive}) \,\Box\, (\mathit{stop} \rightarrow \mathtt{main})$$



where

$$\phi_{idle} := \neg\, \mathit{newgoal} \wedge \neg\, \mathit{start} \wedge \neg\, \mathit{passed} \wedge \neg\, \mathit{stop}$$

Translation of OZ



where

$$\phi_{idle} := \neg\, newgoal \wedge \neg\, start \wedge \neg\, passed \wedge \neg\, stop$$
$$\wedge\, current = current' \wedge goal = goal' \wedge dir = dir'$$

Translation of DC

Full DC cannot be translated into PEA: e.g.

$$\neg \Diamond(\updownarrow ev \,;\, \ell = 1 \,;\, \updownarrow ev),$$

which means

$$\neg (\mathit{true} \,;\, \updownarrow ev \,;\, \ell = 1 \,;\, \updownarrow ev \;;\; \mathit{true})$$

would need infinitely many clocks.

Translation of DC

However, we can translate a useful subset: counterexample formulae.

Example 1:

$$\neg \Diamond(\updownarrow \mathit{passed} \,;\, \ell \leq 3 \,;\, \updownarrow \mathit{passed}) :$$

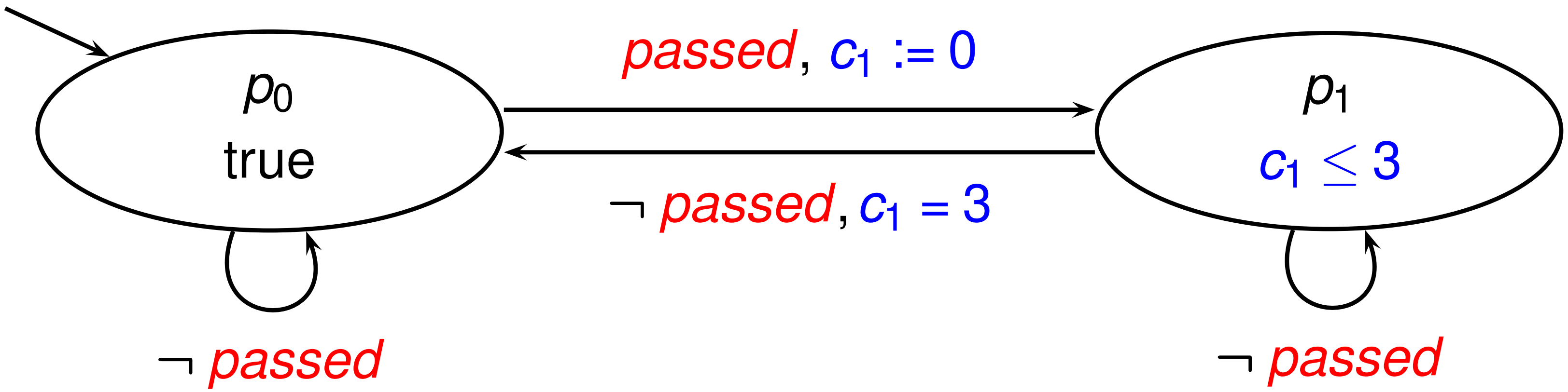
Translation of DC

However, we can translate a useful subset: counterexample formulae.

Example 1:

$$\neg \Diamond(\, \updownarrow passed \,;\, \ell \leq 3 \,;\, \updownarrow passed \,) :$$

Phase-Event-Automaton:



Translation of DC

Example 2:

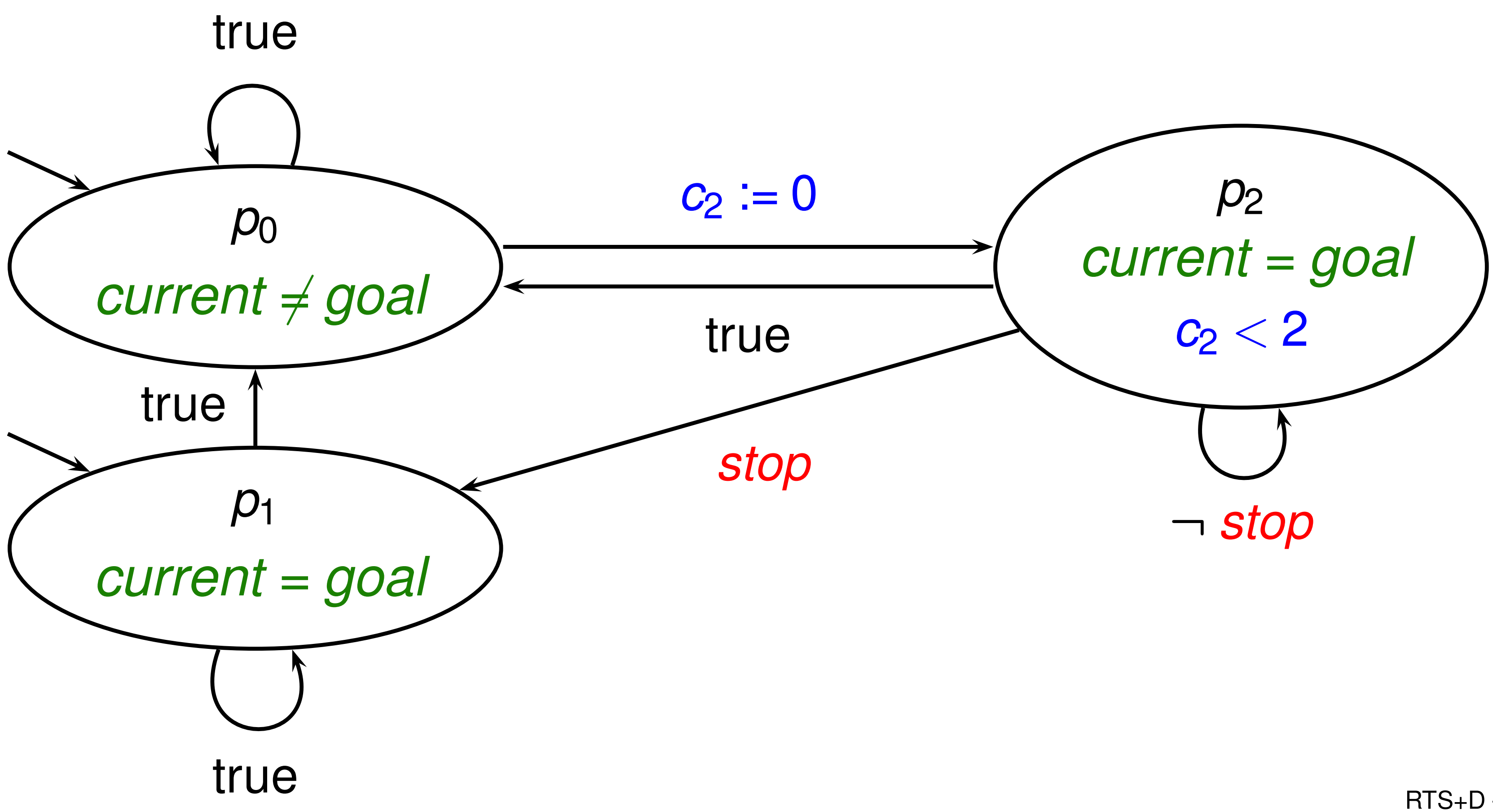
$$\neg \Diamond(\lceil \mathit{current} \neq \mathit{goal} \rceil ; (\lceil \mathit{current} = \mathit{goal} \rceil \wedge \ell \geq 2 \wedge \boxminus \mathit{stop}))$$

Translation of DC

Example 2:

$$\neg \Diamond(\lceil current \neq goal \rceil \,;\, (\lceil current = goal \rceil \wedge \ell \geq 2 \wedge \boxminus\, stop))$$

Phase-Event-Automaton:



Automatic Verification

Automata-theoretic approach to verification:

$$
\begin{array}{lccc}
 & \mathit{COD} & \text{satisfies} & \mathit{DC}\ ? \\
 & \downarrow & & \downarrow \\
\text{PEA:} & \mathcal{A}(\mathit{CSP}) \parallel \mathcal{A}(\mathit{OZ}) \parallel \mathcal{A}(\mathit{DC}) & \parallel & \mathcal{A}_{\mathit{test}}(\neg \mathit{DC})
\end{array}
$$

Is bad state of $\mathcal{A}_{\mathit{test}}(\neg \mathit{DC})$ not reachable ?

Automatic Verification

Automata-theoretic approach to verification:

$$COD \quad \text{satisfies} \quad DC\ ?$$

$$\downarrow \qquad\qquad\qquad \downarrow$$

PEA: $\mathcal{A}(CSP) \parallel \mathcal{A}(OZ) \parallel \mathcal{A}(DC) \quad \parallel \quad \mathcal{A}_{test}(\neg DC)$

Is bad state of $\mathcal{A}_{test}(\neg DC)$ not reachable ?

$$\downarrow$$

TCS: $\mathcal{T}(...)$

Transition Constraint System

Model checking using ARMC or SLAB or H-PILoT on TCS

Transition Constraint Systems

specify states and transitions by formulas (constraints):

- transition constraints relate pre- and post-state
- no notion of events, no notion of real-time

However, events and clocks can be encoded.

- events: changes of Boolean variables::

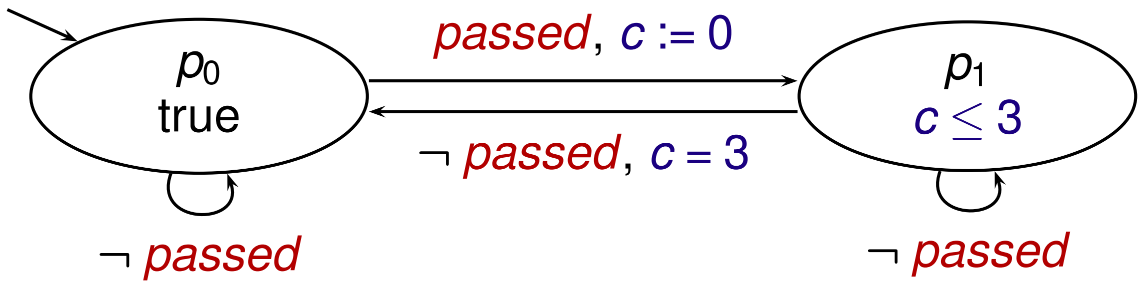
$$stop' \neq stop$$

- clocks: real-valued variables á la Lamport:

$$c' = c + \mathrm{len} \wedge \mathrm{len} > 0$$

Translation of PEA into TCS

Phase-Event-Automaton:



Transition Constraint System:

$$
\begin{aligned}
Tr \quad \Leftrightarrow \quad & ph = 0 \land \neg passed \land c' = c + \mathrm{len} \land ph' = 0 \\
\lor\ & ph = 0 \land passed \land c' = \mathrm{len} \land c' \leq 3 \land ph' = 1 \\
\lor\ & ph = 1 \land \neg passed \land c' = c + \mathrm{len} \land c' \leq 3 \land ph' = 1 \\
\lor\ & ph = 1 \land \neg passed \land c = 3 \land c' = c + \mathrm{len} \land ph' = 0
\end{aligned}
$$

Model Checker ARMC

Podelski & Rybalchenko (since 2002)

Abstraction Refinement Model Checker

Characteristics:

- ARMC checks for reachability,
- employs the CEGAR method:
  counterexample-guided abstraction refinement,
- uses Craig interpolation for predicate discovery,,
- evaluates implications in a decidable fragment of first-order logic: linear arithmetic over reals,
- extended with uninterpreted function symbols.
- is implemented in SICStus Prolog.

Experimental Results

Hoenicke & Maier (2005):

- The formula $\mathit{Min} \leq \mathit{current} \leq \mathit{Max}$ was checked. ARMC proved validity in 2 minutes.

Experimental Results

Hoenicke & Maier (2005):

- The formula $\mathit{Min} \leq \mathit{current} \leq \mathit{Max}$ was checked.
  ARMC proved validity in 2 minutes.
- Valid for all possible choices of *Min* and *Max*.

Experimental Results

Hoenicke & Maier (2005):

- The formula $Min \leq current \leq Max$ was checked.

  ARMC proved validity in 2 minutes.

- Valid for all possible choices of *Min* and *Max*.

- Property depends on real-time:

  If one DC formula is omitted
  ARMC found counterexample in 20 seconds.

Model Checker SLAB

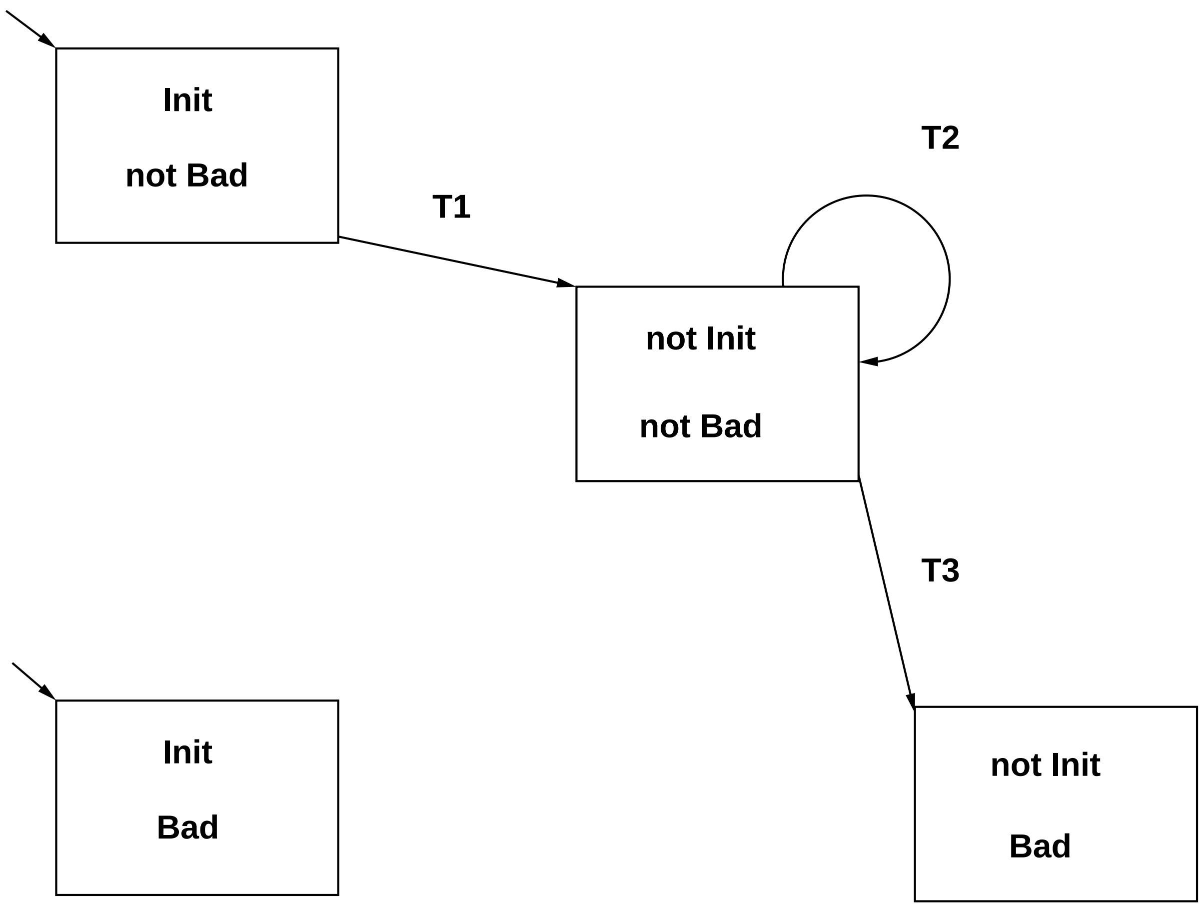
Brückner, Dräger, Finkbeiner & Wehrheim (2008)
Dräger, Kupriyanov, Finkbeiner & Wehrheim (2010)

Slicing Abstraction Model Checker

Characteristics:

- SLAB checks for realizability of abstract error paths
- abstracts both states and transitions,
- uses slicing of abstractions and local refinement,
- employs Craig interpolation for predicate discovery,
- checks satisfiability in a decidable fragment of first-order logic: linear arithmetic over reals,

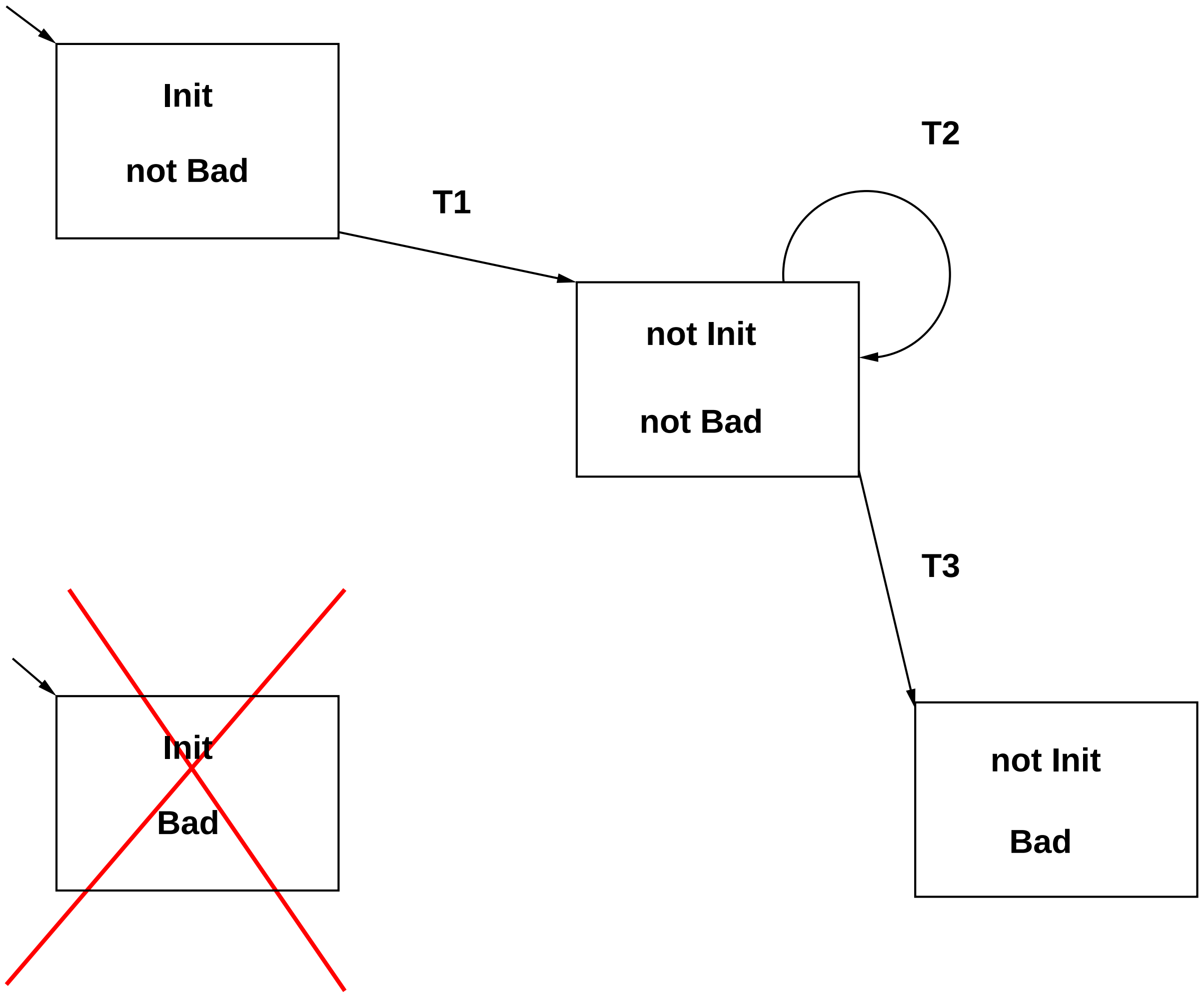
Abstract Error Paths



Does abstract error path correspond to a concrete one ?

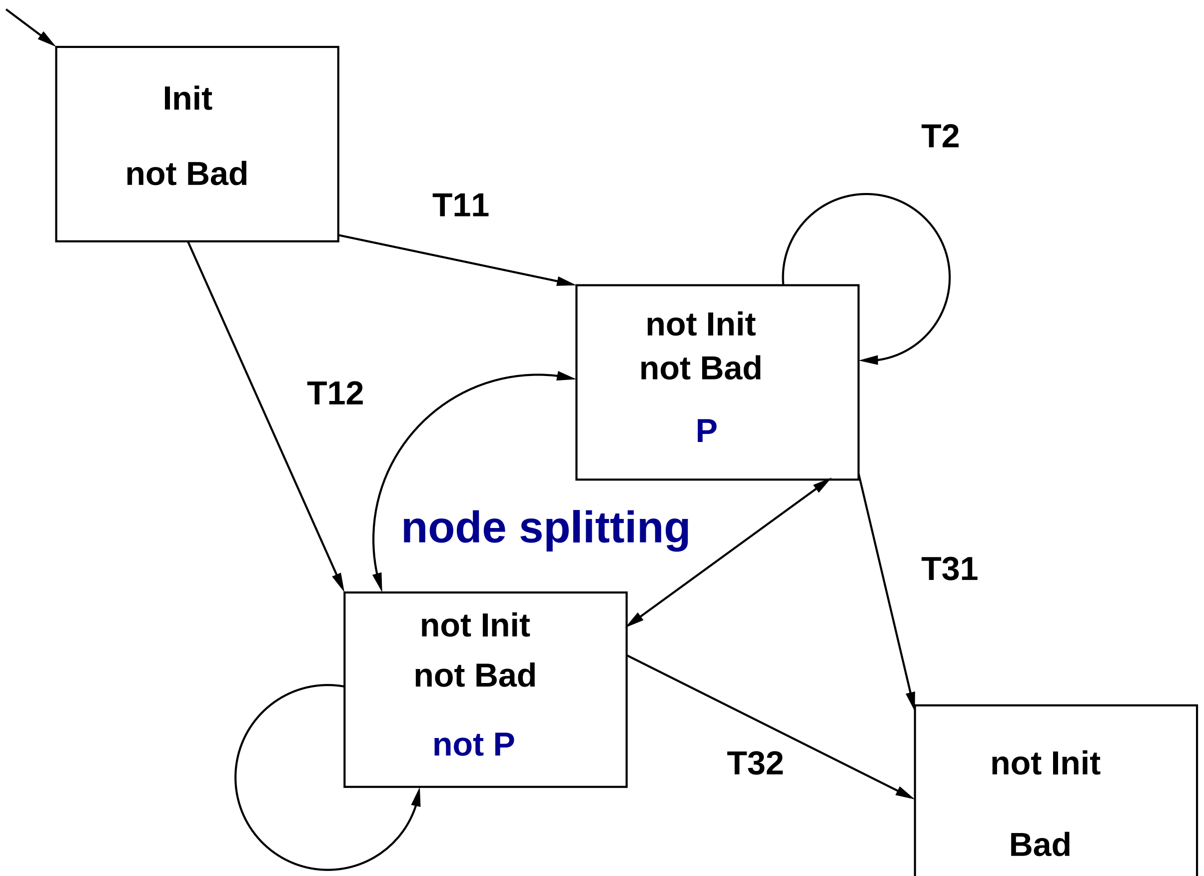
Slicing Abstactions

E.g. node elimination if $\quad$ Init $\wedge$ Bad $\Rightarrow$ *false* :

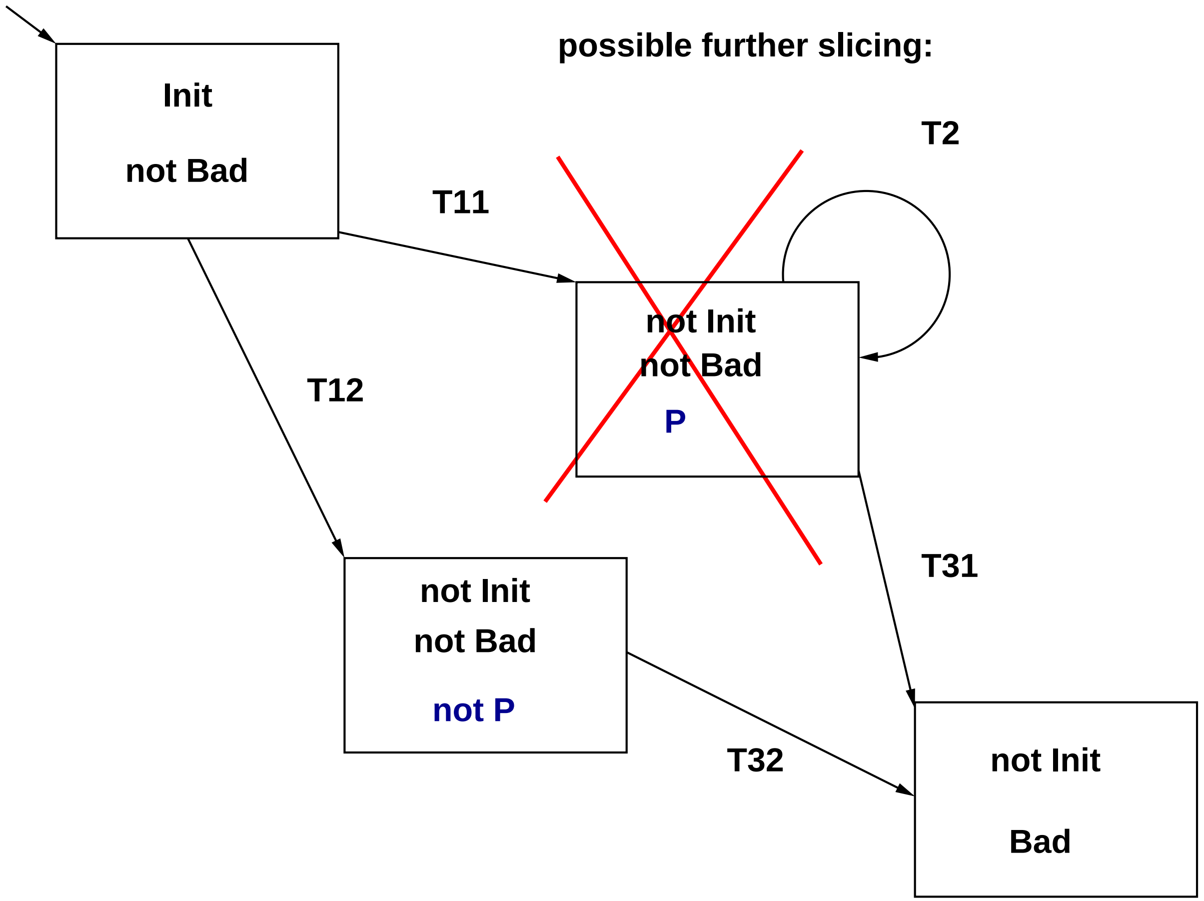


Local Refinement

If error path does *not* correspond to a concrete one
Craig interpolation is used to discover a
predicate P for node splitting.



Termination

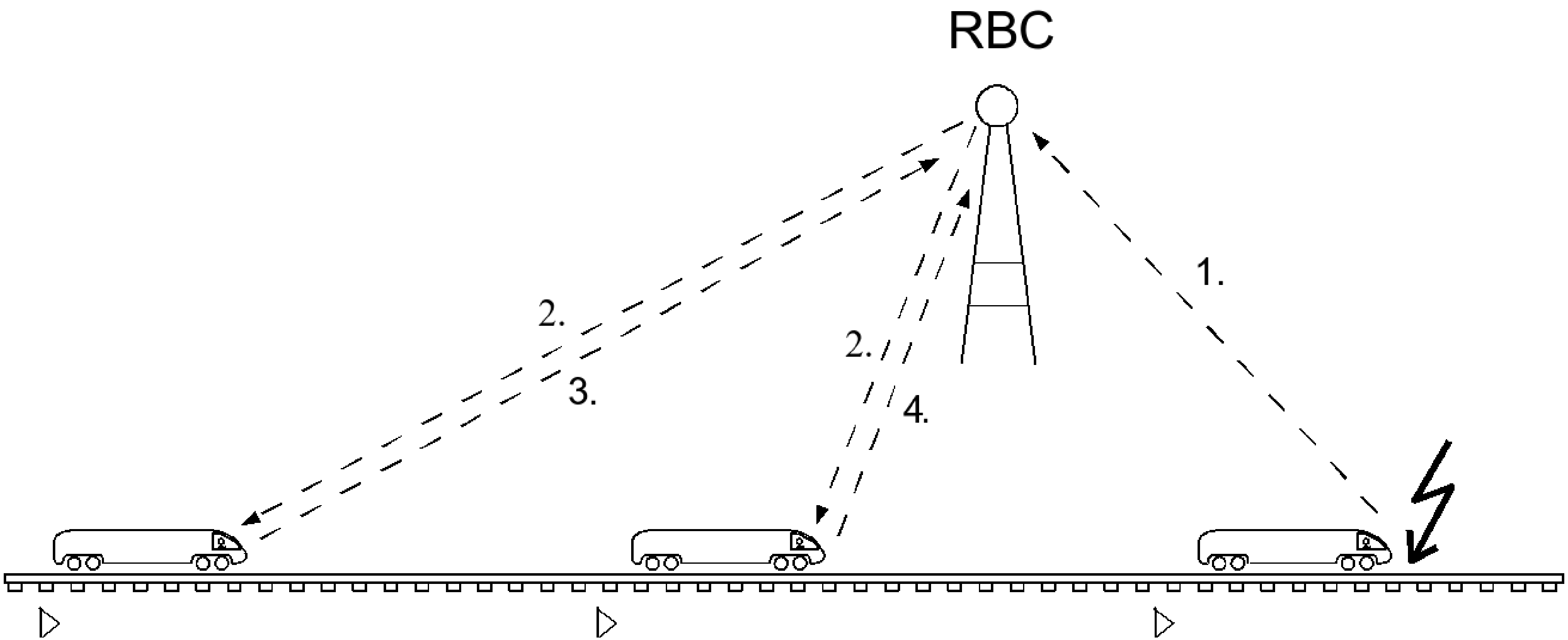


Checking *terminates* if

(1) the error path is realizable (system erroneous) or

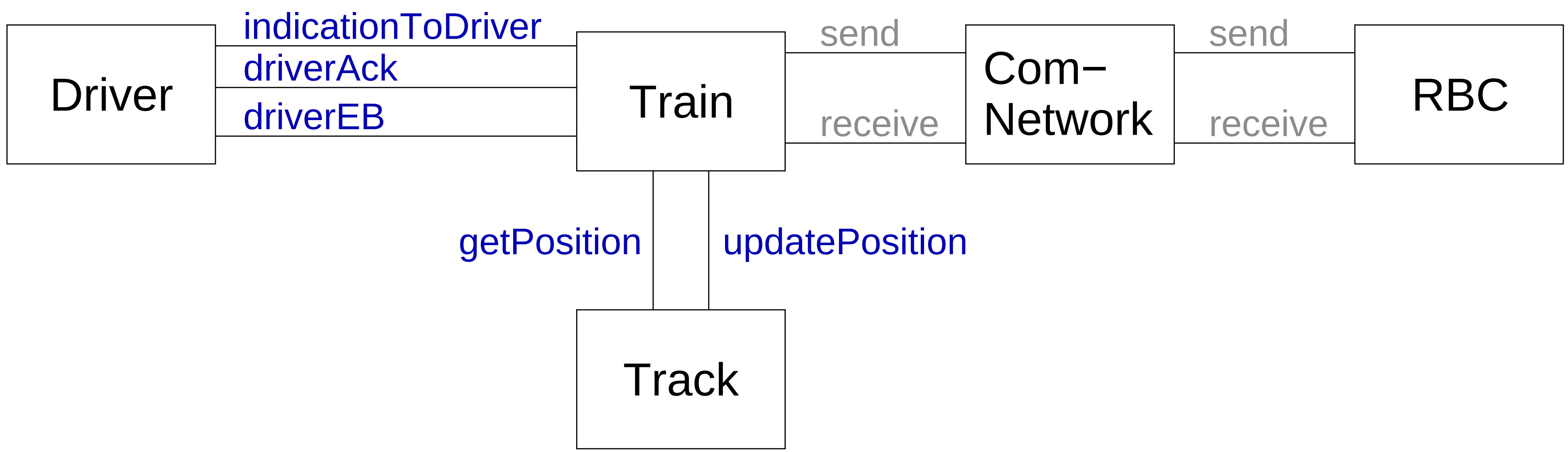
(2) the slice becomes empty (system correct).

ETCS: Emergency Messages



Components of Case Study

Specification in CSP-OZ-DC   J. Faber & Meyer (2006)



Infinite data types: $\mathit{Position} = \mathbb{R}, \quad \mathit{Speed} = \mathbb{R}_{\geq 0}$

Parameters: *Length*, *TargetSpd*, ...

Specification: CSP-OZ-DC

Hoenicke & Olderog (since 2002)

Interface:

$$\texttt{chan}\ \mathit{updPos} : [\mathit{id} : \{\mathit{ID}\},\ \mathit{pos}! : \mathit{Position}]$$

$$\texttt{chan}\ \mathit{compSBI} : [\mathit{loa}?,\ \mathit{sbi}! : \mathit{Position}]$$

CSP specifies sequencing of events:

$$\texttt{main} \overset{c}{=} \mathit{Running} \mid\mid\mid \mathit{HandleEM}$$

$$\mathit{Running} \overset{c}{=} \mathit{updPos}.\mathit{ID}?\mathit{pos} \rightarrow \mathit{getLOA}.\mathit{ID}?\mathit{loa} \rightarrow \mathit{compSBI}!\mathit{loa}?\mathit{sbi} \rightarrow$$

$$\textbf{if}\ \ \mathit{sbi} \leq \mathit{pos}\ \ \textbf{then}\ \ \ldots\ \textbf{else}\ \ \ldots$$

Specification: CSP-OZ-DC

Object-Z specifies state space ...

$$\begin{array}{|l}
sbi : Position \\
curPos : Position \\
curSpd : Speed \\
\ldots
\end{array}$$

... and operations:

$$\begin{array}{|l}
\multicolumn{1}{l}{\texttt{com_}compSBI} \\
\hline
\Delta(sbi) \\
loa?, sbi! : Position \\
\hline
sbi' = loa? - TargetSpdDist - StopDist - MaxDist \\
sbi! = sbi'
\end{array}$$

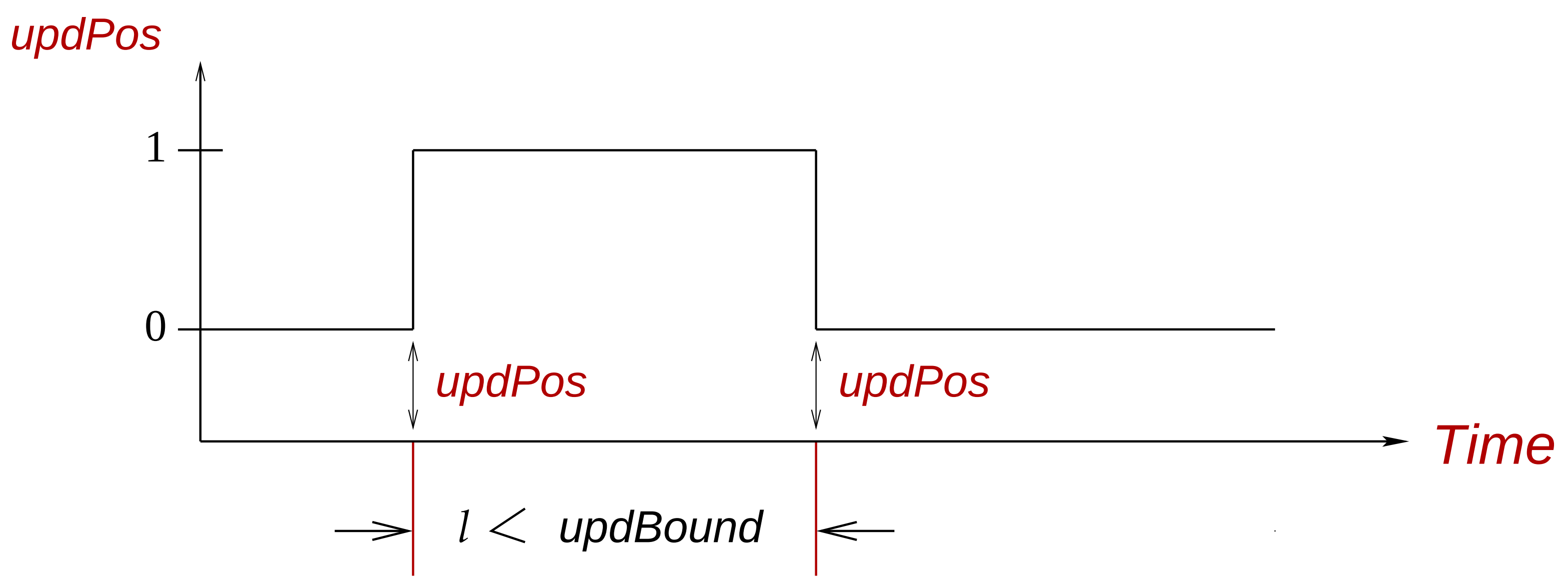
Specification: CSP-OZ-DC

Duration Calculus restricts timing of states and events:

- At least *updBound* seconds between two *updPos* events:

$$\neg\Diamond(\updownarrow \mathit{updPos}\,;\ \ell < \mathit{updBound}\,;\ \updownarrow \mathit{updPos})$$

counterexample trace:



Specification: CSP-OZ-DC

$RearTrain(ID : TrainID;\ \ StartPos, StartSBI : Position)$

$\mathtt{chan}\ updPos : [id : \{ID\},\ pos! : Position]$

$\mathtt{chan}\ compSBI : [loa?,\ sbi! : Position]$

$\ldots$

CSP

$$\mathtt{main} \overset{c}{=} Running \;|||\; HandleEM$$

$$Running \overset{c}{=} updPos.ID?pos \rightarrow getLOA.ID?loa \rightarrow compSBI!loa?sbi \rightarrow$$
$$\quad \textbf{if}\ \ sbi \leq pos\ \ \textbf{then}\ \ldots\ \textbf{else}\ \ldots$$

$\ldots$

OZ

$sbi : Position$

$curPos : Position$

$curSpd : Speed$

$\ldots$

$\mathtt{com}\_compSBI$

$\Delta(sbi)$

$loa?, sbi! : Position$

$sbi' = loa? - TargetSpdDist - StopDist - MaxDist$

$sbi! = sbi'$

$\ldots$

DC

$\neg \Diamond (\updownarrow updPos\,;\ \ell < updBound\,;\ \updownarrow updPos)$

$\ldots$

Properties Checked

Meyer, Faber, Hoenicke & Rybalchenko (2008)

Two trains:

- RT requirements automatically verified with ARMC.

  Example:

  $\neg \Diamond(\updownarrow \mathit{receive.EmAlert}\ ;\ \boxminus \mathit{applyEB} \wedge \boxminus \mathit{driverAck} \wedge \mathit{reactTime} < \ell)$

  where $\mathit{reactTime} = 8$ sec.

  Experimental results 2008:

  4,900 locations, 99,000 transitions, 47 variables

  ARMC: 216 minutes

Properties Checked

Meyer, Faber, Hoenicke & Rybalchenko (2008)

Two trains:

- RT requirements automatically verified with ARMC.

  Example:

  $\neg \Diamond(\updownarrow \mathit{receive.EmAlert} \,;\, \boxminus \mathit{applyEB} \wedge \boxminus \mathit{driverAck} \wedge \mathit{reactTime} < \ell)$

  where $\mathit{reactTime} = 8$ sec.

  Experimental results 2008:

  4,900 locations, 99,000 transitions, 47 variables
  ARMC: 216 minutes

- Collision freedom:

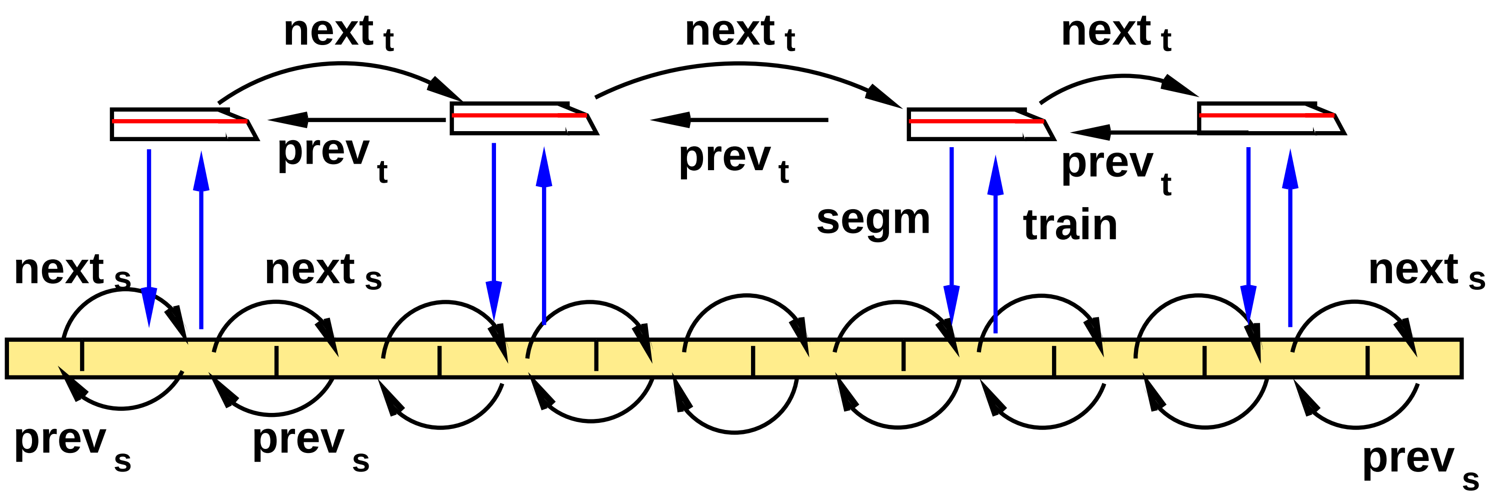
  2008: with manual decomposition into RT requirements

ETCS: More Properties Checked

Application: ETCS with arbitrary no. of trains / segments:

Faber, Jacobs & Sofronie-Stokkermans (2010)

simplified CSP-OZ-DC model, but with
2-sorted pointer data structure:



Verified: invariant property of collision freedom.

Data Verification with H-PILoT

Ihlemann, S. Jacobs & Sofronie-Stokkermans (2009)

Hierarchical Proving by Instantiation in Local Theory extensions

Characteristics:

- Tool H-PILoT supports local theory extensions $\mathcal{T}_0 \subseteq \mathcal{T}_1$.
- Satisfiability of (quantified) formulae in extension $\mathcal{T}_1$
  is reduced to
  satisfiability of ground formulae in the base theory $\mathcal{T}_0$.
- Standard SMT solvers check
  satisfiability of ground formulae in the base theory $\mathcal{T}_0$.
- Hierarchical reasoning / interpolation / QE
  for new classes of theories of data types, e.g.,
  - recursive functions,
  - many-sorted pointer structures.

Syspect Tool

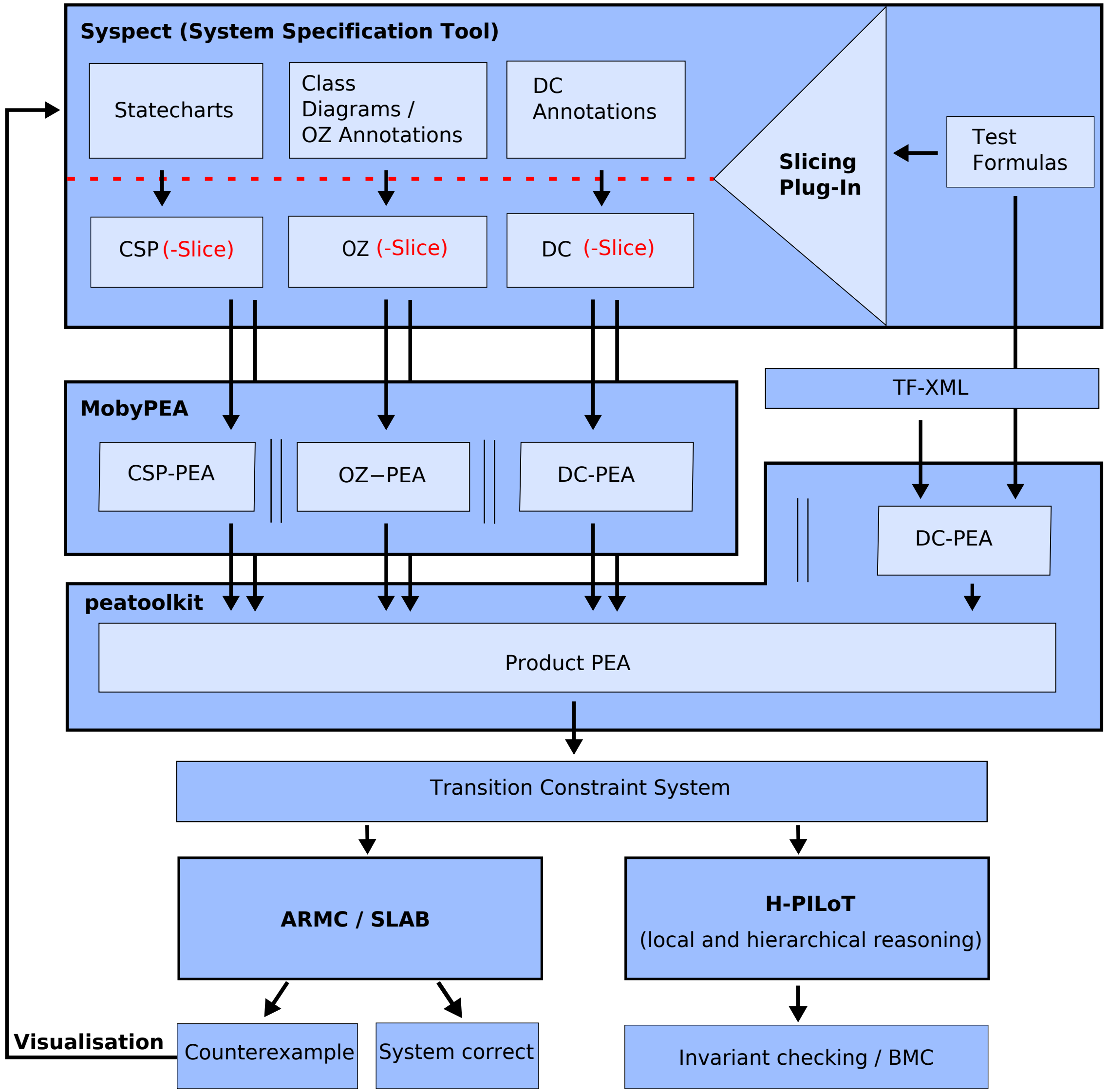
for modelling, specificying and verifying RTS systems with rich data.

Students' work continued in AVACS project "Beyond Timed Automata".

Faber, Linker, Olderog, Quesel (2011)

level	language	purpose
modelling	UML profile	model M of a real-time system R with rich data
	$\downarrow$	
specification	CSP-OZ-DC	specification S of R as the formal semantics of M
	$\downarrow$	
verification	PEA	operational semantics O of S
	$\downarrow$	
	TCS	representation of O as input for verification engines like ARMC, SLAB, or H-PILoT

Tool Chain for Syspect Verification



Further Developments

- Explicit Durations
- Parallel Composition

Translatable DC Classes

(1) Full DC cannot be translated into PEA.

(2) Counterexample formulae:
powerset construction takes care of overlapping timed phases and yields deterministic PEA

PhD thesis Hoenicke (2006)

(3) Explicit Durations:
translation (2) extended by stop watches

In discrete time setting: Availability Automata

Hoenicke, Meyer & Olderog (2010)

Explicit Durations

... can express timed availability requirements:

$$\int(speed \geq target) \geq 0.9 \cdot \ell$$

“For at least 90 % of the time interval, the train meets its target speed.”

... correspond to integrators (stop watches), a source of undecidability separating TA and LHA.

New translations to automata for reachability analysis:

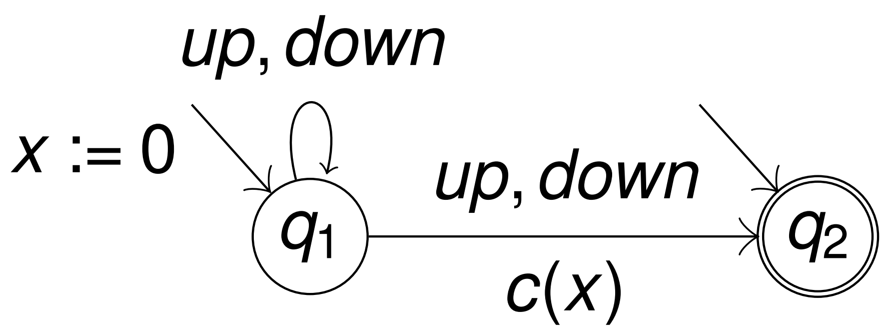
- Multi-Priced Time Automata — continuous time
- Availability Automata (new) — discrete time

Availablity Automata

Availabilities in discrete setting (words).

regular availability expressions (rea) $\mapsto$ availability automata (aa)

Example: $((up + down)^*.\checkmark)_{\{up\} \geq \frac{1}{3}}$



availability counter x with test $c(x) = (\{up\} \geq \frac{1}{3})$

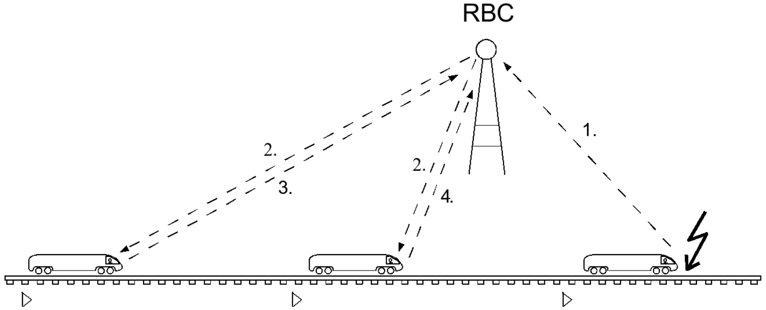
Kleene theorem. A language is recognized by a flat rae *if and only if* it is accepted by a simple aa.

Powerset construction. Every simple aa can be *determinized* and *complemented* by inverting final states.

Avoiding Product Construction

Large COD specifications yield (too) many parallel PEA.

ETCS Emergency Messages:
collision freedom for two trains



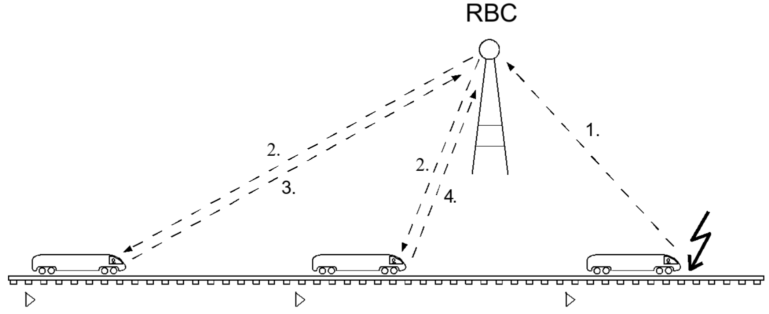
Full COD specifications yields
18 parallel PEAs.

Avoiding Product Construction

Large COD specifications yield (too) many parallel PEA.

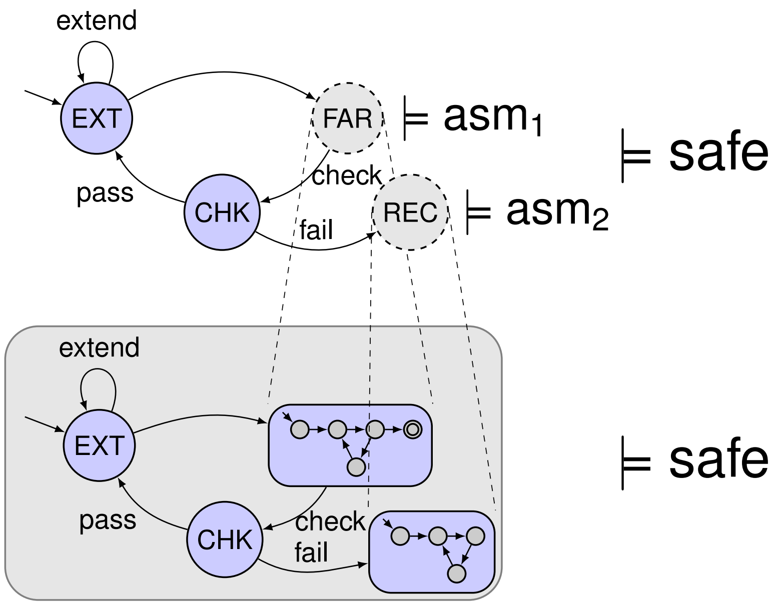
ETCS Emergency Messages:
collision freedom for two trains

Full COD specifications yields
18 parallel PEAs.



Formal decomposition of COD specification into a Verification Architecture and its instantiation:

PhD thesis Faber (2011)



Structural Transformations

For real-time systems with data
(modelled by Extended Timed Automata), we

- isolate conditions (like independence of transitions or memorylessness of locations),
- which enable property-preserving transformations that replace parallel by sequential composition and eliminate loops.
- This results in systems that allow for an easier conceptual and automatic analysis.

More details: see lecture by Mani Swaminathan.

Conclusion

Semantic methods + automatic verification techniques

Ref: Specification Language

1. J. Hoenicke & E.-R. Olderog.
   CSP-OZ-DC: A combination of specification techniques for processes, data and time.
   Nordic Journal of Computing, 9(4), 2003.
2. J. Hoenicke.
   Combination of Processes, Data, and Time.
   PhD Thesis, Univ. Oldenburg, 2006.
3. J. Hoenicke, R. Meyer & E.-R. Olderog.
   Kleene, Rabin, and Scott are available.
   In *Proc. CONCUR*, LNCS 6269, 2010.
4. M. Möller, E.-R. Olderog, H. Rasch & H. Wehrheim.
   Integrating a formal method into a software engineering process with UML and Java.
   Formal Aspects of Computing 20, 2008.

Ref: Verification Engines

1. A. Podelski & A. Rybalchenko.
   ARMC: The logical choice for software model checking with abstraction refinement.
   Proc. Practical Aspects of Declarative Languages (PADL), LNCS 4281, 2007.
2. K. Dräger, A. Kupriyanov, B. Finkbeiner & H. Wehrheim.
   SLAB: A certifying model checker for infinite state concurrent systems.
   In *Proc. TACAS*, LNCS 6015, 2010.
3. C. Ihlemann & V. Sofronie-Stokkermans.
   System description: H-PILoT.
   In Proc. *CADE 2009*, LNCS, 2009.
4. J. Faber, S. Linker, E.-R. Olderog & J.-D. Quesel.
   Syspect – Modelling, Specifying, and Verifying Real-Time Systeme with Rich Data.
   Int. J. Software Informatics 5 (1–2), 2011.

Ref: Automatic Verification

1. J. Hoenicke & P. Maier.
   Model-checking of specifications integrating processes, data and time.
   In *Proc. Formal Methods* (FM), LNCS 3583, 2005.
2. J. Faber & R. Meyer.
   Model checking data-dependent real-time properties of the European Train Control System.
   In *Proc. Formal Methods in Computer Aided Design* (FMCAD), IEEE, 2006.
3. R. Meyer, J. Faber, J. Hoenicke & A. Rybalchenko.
   Model checking Duration Calculus: A practical approach.
   Formal Aspects of Computing 20, 2008.
4. J. Faber, C. Ihlemann, S. Jacobs & V. Sofronie-Stokkermans.
   Automatic verification of parametric specifications with complex topologies.
   In *Proc. Integrated Formal Methods* (IFM), LNCS 6396, 2010.

Ref: Reduction

1. I. Brückner.
   Slicing concurrent real-time system specifications for verification.
   In *Proc. Integrated Formal Methods* (IFM), 2007.

2. J. Faber.
   Verification Architectures: Compositional reasoning for real-time systems.
   In *Proc. Integrated Formal Methods* (IFM), LNCS 6396, 2010.

3. E.-R. Olderog & M. Swaminathan.
   Structural transformations for data-enriched real-time systems.
   Formal Aspects of Computing 27, 2015.