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
FlexRay

- communication protocol for distributed components in cars
- used in BMW X 5 and BMW’s 7 series for X-by-wire
- developed by: BMW, Bosch, Daimler, Freescale, General Motors, NXP Semiconductors, Volkswagen, et al.
FlexRay as the Future Drive-by-Wire Standard

⇒ Safety-critical!
Jitter and Glitch Correction

Sender

Rx

1

0

0

Tx

wire delay
and Rx
omitted

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

Receiver

voted value

1

0

0

1

1 2 3 4 5 6 7 8

Bernd Finkbeiner (UdS)  Embedded Systems  2011/02/03  6 / 48
Protocol Operation

Sender

Sent Message
\[ \cdots 11010111 \cdots \]

Sent Stream
\[ \cdots 101101011110 \cdots \]

Bus

\[ \cdots 111111111000000000 \cdots \]

Voted Values
\[ \cdots 111111111000000000 \cdots \]

Strobing
\[ \cdots 1234567812345678 \cdots \]

Received Stream
\[ \cdots 101101011110 \cdots \]

Received Message
\[ \cdots 11010111 \cdots \]

Receiver

1 byte

1 byte
Newest FlexRay Specification, Version 2.1, Revision A:

“[FlexRay] attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell [. . .]. There are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated.”
Michael Gerke’s Model of the Protocol

- protocol
- jitter (parameterized)
- glitches
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: ... ꝉ ꝉ ꝉ ꝉ ...
Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... ³ ³ ³ ³ ³ ³ ...
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: \[ \ldots \text{[Glitch]} \text{[Glitch]} \text{[Glitch]} \text{[Glitch]} \ldots \]
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: \ldots \# \# \# \# \ldots
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... \[\text{\cancel{\text{\color{red}z}} \ \text{\cancel{\text{\color{red}z}}} \ \text{\color{red}z} \ \text{\color{red}z}} \ldots\]
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: \[ \_ \_ \_ \_ ]
Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)

Note: one message $\approx 21,000$ samples
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

- 2 arbitrarily placed glitches in the complete message (at most 2)

E.g.: \ldots \ 

Note: one message \approx 21,000 samples
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)

Note: one message \( \approx 21.000 \) samples

The protocol does not tolerate:
2 arbitrarily placed glitches in every sequence of 82 consecutive samples (2 out of 82)
Parameter exploration using binary search: boundaries for variation of a single parameter

<table>
<thead>
<tr>
<th>glitch tolerance</th>
<th>delay variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 out of 4)</td>
<td>$1.435\text{ns} \rightarrow 7.6075\text{ns}$</td>
</tr>
<tr>
<td>(2 at most)</td>
<td>$1.435\text{ns} \rightarrow 7.6075\text{ns}$</td>
</tr>
<tr>
<td>(1 at most)</td>
<td>$1.435\text{ns} \rightarrow 12.020\text{ns}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>glitch tolerance</th>
<th>deviation of clock from standard rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 out of 4)</td>
<td>$0.15% \rightarrow 0.46%$</td>
</tr>
<tr>
<td>(2 at most)</td>
<td>$0.15% \rightarrow 0.46%$</td>
</tr>
<tr>
<td>(1 at most)</td>
<td>$0.15% \rightarrow 1.09%$</td>
</tr>
<tr>
<td>(no glitches)</td>
<td>$0.15% \rightarrow 1.74%$</td>
</tr>
</tbody>
</table>
Model Checking

Device

Device Descript.

architecture behaviour of processor is
process fetch
  if halt=0 then
    if mem_wait=0 then
      nextins <= dport
...
Safety requirement: Gate has to be closed whenever a train is in “In”.
Finite-State Automata

- Open
- Closing
- Empty
- Appr.
- In

Opening → Closing

Closed → Open

enter → ~enter

* = leave, enter, tick

~leave = enter, tick

~enter = leave, tick

bernd finkbeiner (udS)
Model Checking

Opening Empty

Closed Empty

Opening Appr.

Closed Appr.

Opening In

Closed In
A Discrete-Time Coffee Machine

![Coffee Machine Diagram]

- **idle**
  - Coffee ordered
  - Tea ordered
  - Coffee prepared
  - Tea prepared

- **coffee-ordered**
  - tick
  - tick

- **tea-ordered**
  - tick
  - tick

- **coffee-prepared**
  - tick
  - tick

- **tea-prepared**
  - tick
  - tick

Bernd Finkbeiner (UdS)
a graph with locations and edges

a location is labeled with the valid atomic propositions

taking an edge is instantaneous, i.e, consumes no time
equipped with real-valued clocks $x, y, z, \ldots$

clocks advance implicitly, all at the same speed

logical constraints on clocks can be used as guards of actions
Timed Automata

- clocks can be reset when taking an edge
- assumption: all clocks are zero when entering the initial location initially
Timed Automata

- guards indicate when an edge *may be taken*
- a location invariant specifies the *amount of time that may be spent in a location*
  - before a *location invariant* becomes invalid, an edge must be taken
A Real-Time Coffee Machine

Bernd Finkbeiner (UdS)  Embedded Systems  2011/02/03  21 / 48
**Clock constraints** over set $C$ of clocks are defined by:

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

- 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 $\neg (x < c_1) \land (x < c_2)$
A *timed automaton* is a tuple

$$TA = (\text{Loc}, \text{Act}, \text{C}, \leadsto, \text{Loc}_0, \text{inv}, \text{AP}, \text{L})$$

where:

- \( \text{Loc} \) is a finite set of locations.
- \( \text{Loc}_0 \subseteq \text{Loc} \) is a set of initial locations
- \( \text{C} \) is a finite set of clocks
- \( \text{L} : \text{Loc} \to 2^{\text{AP}} \) is a labeling function for the locations
- \( \leadsto \subseteq \text{Loc} \times \text{CC}(\text{C}) \times \text{Act} \times 2^{\text{C}} \times \text{Loc} \) is a transition relation, and
- \( \text{inv} : \text{Loc} \to \text{CC}(\text{C}) \) is an invariant-assignment function
Intuitive Interpretation

- Edge $\ell \xrightarrow{g:\alpha,C'} \ell'$ means:
  - action $\alpha$ is enabled once guard $g$ holds
  - when moving from location $\ell$ to $\ell'$, any clock in $C'$ will be reset to zero

- $inv(\ell)$ constrains the amount of time that may be spent in location $\ell$
  - the location $\ell$ must be left before the invariant $inv(\ell)$ becomes invalid
Guards vs. Location Invariants

The effect of a lowerbound guard:

\[ x \geq 2 \quad \text{reset}(x) \]

\[ \begin{array}{c}
\text{value of } x \\
\hline
2 \quad 4 \quad 6 \quad 8 \quad 10 \\
\end{array} \]

\[ \text{time} \]
The effect of a lowerbound and upperbound guard:
The effect of a guard and an invariant:
Arbitrary Clock Differences

\[ y \geq 2 \quad \text{reset}(y) \]
\[ x \geq 2 \quad \text{reset}(x) \]
Composing Timed Automata

Let \( TA_i = (Loc_i, Act_i, C_i, \leadsto_i, Loc_{0,i}, inv_i, AP, L_i) \) and \( H \) an action-set

\[
TA_1 \parallel_H TA_2 = (Loc, Act_1 \cup Act_2, C, \leadsto, Loc_0, inv, AP, L)
\]

where:

- \( Loc = Loc_1 \times Loc_2 \) and \( Loc_0 = Loc_{0,1} \times Loc_{0,2} \) and \( C = C_1 \cup C_2 \)
- \( inv(\langle \ell_1, \ell_2 \rangle) = inv_1(\ell_1) \land inv_2(\ell_2) \) and \( L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2) \)
- \( \leadsto \) is defined by the inference rules:

\[
\begin{align*}
\text{for } \alpha \in H & \quad \frac{\ell_1 g_{1,\alpha} D_1 \leadsto_1 \ell_1' \land \ell_2 g_{2,\alpha} D_2 \leadsto_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle g_{1,\alpha} D_1 \cup D_2 \leadsto \langle \ell_1', \ell_2' \rangle} \\
\text{for } \alpha \not\in H & \quad \frac{\ell_1 \leadsto_1 D \ell_1'}{\langle \ell_1, \ell_2 \rangle g_{\alpha, D} \leadsto \langle \ell_1', \ell_2 \rangle}
\end{align*}
\]
Example: Railroad Crossing
Example: Railroad Crossing

Gate
Example: Railroad Crossing

\((\text{Train} \{\text{approach, exit}\} \parallel \text{Controller}) \parallel \{\text{lower, raise}\} \text{Gate}\)
A *clock valuation* \( v \) for set \( C \) of clocks is a function \( v : C \rightarrow \mathbb{R}_{\geq 0} \)
- assigning to each clock \( x \in C \) its current value \( v(x) \)

Clock valuation \( v + d \) for \( d \in \mathbb{R}_{\geq 0} \) is defined by:
- \( (v+d)(x) = v(x) + d \) for all clocks \( x \in C \)

Clock valuation reset \( x \) in \( v \) for clock \( x \) is defined by:

\[
(reset \ x \ in \ v)(y) = \begin{cases} 
  v(y) & \text{if } y \neq x \\
  0 & \text{if } y = x.
\end{cases}
\]

- \( \text{reset } x \text{ in } (\text{reset } y \text{ in } v) \) is abbreviated by \( \text{reset } x, y \text{ in } v \)
Timed automaton semantics

For timed automaton $TA = (Loc, Act, C, \leadsto, Loc_0, inv, AP, L)$:
state graph $S(TA) = (Q, Q_0, E, L')$ over $AP$ where:

- $Q = Loc \times val(C)$, state $s = \langle \ell, v \rangle$ for location $\ell$ and clock valuation $v$
- $Q_0 = \{ \langle \ell_0, v_0 \rangle \mid \ell_0 \in Loc_0 \land v_0(x) = 0 \text{ for all } x \in C \}$
- $L'(\langle \ell, v \rangle) = L(\ell)$
- $E$ is the edge set defined on the next slide
Timed automaton semantics

The edge set $E$ consist of the following two types of transitions:

- **Discrete transition:** $\langle \ell, v \rangle \xrightarrow{\alpha} \langle \ell', v' \rangle$ if all following conditions hold:
  - there is an edge labeled $(g : \alpha, D)$ from location $\ell$ to $\ell'$ such that:
  - $g$ is satisfied by $v$, i.e., $v \models g$
  - $v' = v$ with all clocks in $D$ reset to 0, i.e., $v' = \text{reset } D$ in $v$
  - $v'$ fulfills the invariant of location $\ell'$, i.e., $v' \models \text{inv}(\ell')$

- **Delay transition:** $\langle \ell, v \rangle \xrightarrow{d} \langle \ell, v+d' \rangle$ for positive real $d$
  - if for any $0 \leq d' \leq d$ the invariant of $\ell$ holds for $v+d'$, i.e., $v+d' \models \text{inv}(\ell)$
Let for any $t < d$, for fixed $d \in \mathbb{R}_{>0}$, clock valuation $\eta + t \models inv(\ell)$

A possible execution fragment starting from the location $\ell$ is:

$\langle \ell, \eta \rangle \xrightarrow{d_1} \langle \ell, \eta + d_1 \rangle \xrightarrow{d_2} \langle \ell, \eta + d_1 + d_2 \rangle \xrightarrow{d_3} \langle \ell, \eta + d_1 + d_2 + d_3 \rangle \xrightarrow{d_4} \ldots$

where $d_i > 0$ and the infinite sequence $d_1 + d_2 + \ldots$ converges towards $d$

such path fragments are called time-convergent

$\Rightarrow$ time advances only up to a certain value

Time-convergent execution fragments are unrealistic and ignored
Example: light switch

The path

\[ \pi = \langle \text{off}, 0 \rangle \langle \text{off}, 1 \rangle \langle \text{on}, 0 \rangle \langle \text{on}, 1 \rangle \langle \text{off}, 1 \rangle \langle \text{off}, 2 \rangle \langle \text{on}, 0 \rangle \langle \text{on}, 1 \rangle \langle \text{off}, 1 \rangle \ldots \]

is \textit{time-divergent}. The path

\[ \pi' = \langle \text{off}, 0 \rangle \langle \text{off}, 1/2 \rangle \langle \text{off}, 3/4 \rangle \langle \text{off}, 7/8 \rangle \langle \text{off}, 15/16 \rangle \ldots \]

is \textit{time-convergent}. 
State $s \in S(TA)$ contains a *timelock* if there is a reachable state $s$ where there is no time-divergent path from $s$.

Timelocks are considered as *modeling flaws* that should be avoided.
Consider a timed automaton with clocks $x$ and $y$ having maximal constants 3 and 2, respectively.
Consider a timed automaton with clocks $x$ and $y$

having maximal constants 3 and 2, respectively.

**Equivalence relation** $\sim_R$
Consider a timed automaton with clocks $x$ and $y$ having maximal constants 3 and 2, respectively.

Equivalence relation $\sim_R$

constraints
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
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
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
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

$\implies$ finite index!
Finite Semantics: Region Automaton

A → $l_0$ → $l_1$

- $a, x > 0$
- $b, x < 1, x := 0$

$[A]_r$

$l_0$

- $x = 0$
- $0 < x < 1$
- $0 < x < 1$
- $1 < x$

$l_1$

- $x = 1$
- $x = 0$
- $x = 1$
- $1 < x$

$l_2$

- $x = 0$
- $0 < x < 1$
- $0 < x < 1$
- $1 < x$
Reachability is decidable

Theorem [Alur, 1994]:

\[ \exists \text{ path } (l, \tilde{t}) \rightarrow (l', \tilde{t}') \]

iff

\[ \exists \text{ path } (l, [\tilde{t}]_R) \rightarrow (l', [\tilde{t}']_R) \]

Symbolic data structures

- Clock Region = Finest integral unit
- Clock Zone = Convex union of clock regions
- Federation = (Non-convex) union of clock zones
Finite Semantics: Zone Graph

\[ A \rightarrow l_0 \xrightarrow{a, x > 0} l_1 \]
\[ b, \quad x < 1 \quad \{x\} \]
\[ l_2 \]

\[ \llbracket A \rrbracket_z \]

\[ l_0 \quad true \]
\[ l_1 \quad x > 0 \]
\[ l_2 \quad true \]
Controller Synthesis

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

Game between two players

Environment \quad \text{vs.} \quad \text{Controller}

uncontrolled moves

controlled moves

“wants to violate the spec.”  \quad \text{“wants to satisfy the spec.”}
Plants are modeled as timed game automata (TGA)

\[
\begin{align*}
\text{coffee} & \quad x := 0 \\
\text{espresso} & \quad x := 0 \\
\text{heat} & \quad x \leq 9 \\
\text{drink} & \quad \text{warm} \\
\text{heat} & \quad x \leq 15 \\
\text{warning} & 
\end{align*}
\]
Controller = subautomaton representing \textit{winning strategies}

- \textit{coffee} with state $x := 0$
- \textit{espresso} with state $x := 0$
- \textit{heat} with state $x = 5$
- \textit{drink} with state $x \leq 20$
- \textit{warm} with state $x = 10$
- \textit{warning} with state $x > 20$

Bernd Finkbeiner (UdS)
Reachability Games Played on Timed Automata

From where can $\longrightarrow$ enforce a run to $c$?
From where can $\rightarrow$ enforce a run to $c$?

$$x := 0$$

$$x > 1$$

$$x > 2$$

$$x > 4$$

$\text{Attr}[c] = \{ True \}$
From where can \( \rightarrow \rightarrow \) enforce a run to \( c \) ?

\[
\begin{align*}
\text{Attr}[a] &= \{ x > 4 \} \\
\text{Attr}[c] &= \{ \text{True} \}
\end{align*}
\]
Zone-based Timed Game Solving

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

Attr[$a$] = $\{x > 4\}$

$\xRightarrow{x := 0} a \xRightarrow{x > 1} b \xRightarrow{x > 2} c$

Attr[$b$] = $\{x > 2\}$

Attr[$c$] = $\{True\}$

$x > 4$
From where can $\rightarrow$ enforce a run to $c$?

$\text{Attr}[a] = \{ x > 2 \}$

$\text{Attr}[b] = \{ x > 2 \}$

$\text{Attr}[c] = \{ \text{True} \}$

$x := 0$  \quad x > 1$  \quad x > 2$

$x > 4$
Summary

- Timed automata
- Automatic verification
- Automatic controller synthesis