Quantitative Verification and Synthesis, of Embedded Systems

Kim G. Larsen
CISS – Aalborg University
DENMARK
Embedded Systems

**Plant**
Continuous

![Diagram of plant](image1)

**Controller**
Program
Discrete

sensors

actuators

**Eg.**
Realtime Protocols
Pump Control
Air Bags
Robots
Cruise Control
ABS
CD Players
Production Lines

**Quantities:**
- timing
- energy
- memory
- bandwidth
- uncertainties

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [2]
Verification

Plant
- Continuous

Controller Program
- Discrete

Model of tasks (automatic?)

Model of environment (user-supplied / non-determinism)

UPPAAL Model

SAT $\phi$ ??

Kim Larsen [3]
Synthesis

Plant
Continuous

Controller Program
Discrete

Model of environment (user-supplied)

Partial UPPAAL Model

Synthesis of tasks (automatic)

SAT $\phi$ !!

Kim Larsen [4]

ARTIST Summer School, Autrans, France
September 10, 2009
Tasks:
- Computation times
- Deadlines
- Dependencies
- Arrival patterns
- Uncertainties

Scheduling Principles (OS)
- EDF, FPS, RMS, DVS, ..

Resources
- Execution platform
- PE, Memory
- Networks
- Drivers
- Uncertainties
Approach – Timed Automata

- **Performance Evaluation**
  - Estimate resources (e.g. energy) required by given SP.

- **Scheduling & Synthesis**
  - Synthesize (optimal) SP ensuring given objective.
  - Scheduling: SP controls everything (including execution time, availability of resources).
  - Synthesis: scheduling under uncertainties (e.g. execution time, availability of resources).

**TALK:**
- What can we do?
- What can we do efficiently?
- What would we like to do?
- What can not be done?

Kim Larsen [6]

ARTIST Summer School, Autrans, France
September 10, 2009
Overview

- Scheduling
  - Timed Automata
- Optimal Scheduling
  - Priced Timed Automata
- Schedulability Analysis
  - Stop-watch Automata
  - Safety Criticial JAVA (FPGA)
- Synthesis
  - Timed Game Automata
Timed Automata
や、「リアルタイムシステムの安全性」は開発者だけでなく、高齢者に電子化されつつある現代社会の
もっとも高い関心事のひとつです。しかし、リアルタイムシステムの設計は複雑で、同時に動作するようなシ
ステムの安全性を確保するのは難しく、通常の方法（テスト）では非常にコストがかかります。
このツールは、「モデル検査」といった新しい技術によって、そのような安全性・安全なリアルタイム・システム
の開発を支援するビジュアルな統合モデル検証ツールです。
Timed Automata

[Alur & Dill’89]

Semantics:

(Idle, x=0)
d(2.5) → (Idle, x=2.5)
use? → (InUse, x=0)
d(5) → (InUse, x=5)
done! → (Idle, x=5)
Semantics:

\[(\text{Idle}, \text{Init}, B=0, x=0)\]

- \(d(3.1415) \rightarrow (\text{Idle}, \text{Init}, B=0, x=3.1415)\)
- \(\text{use} \rightarrow (\text{InUse}, \text{Using}, B=6, x=0)\)
- \(d(6) \rightarrow (\text{InUse}, \text{Using}, B=6, x=6)\)
- \(\text{done} \rightarrow (\text{Idle}, \text{Done}, B=6, x=6)\)
Compute:

\[(D \times (C \times (A + B)) + ((A + B) + (C \times D))\]

using 2 processors

P1 (fast)

P2 (slow)

13 pico-sec !!
Compute:
\[(D \times (C \times (A + B)) + ((A + B) + (C \times D)))\]
using 2 processors

P1 (fast)
P2 (slow)

12 pico-sec
OPTIMAL !!
Task Graph Scheduling

M = \{M_1, M_2\}
Task Graph Scheduling

E<> (Task1.End and ... and Task7.End)

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [15]
## Experimental Results

<table>
<thead>
<tr>
<th>name</th>
<th>#tasks</th>
<th>#chains</th>
<th># machines</th>
<th>optimal</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>437</td>
<td>125</td>
<td>4</td>
<td>1178</td>
<td>1182</td>
</tr>
<tr>
<td>000</td>
<td>452</td>
<td>43</td>
<td>20</td>
<td>537</td>
<td>537</td>
</tr>
<tr>
<td>018</td>
<td>730</td>
<td>175</td>
<td>10</td>
<td>700</td>
<td>704</td>
</tr>
<tr>
<td>074</td>
<td>1007</td>
<td>66</td>
<td>12</td>
<td>891</td>
<td>894</td>
</tr>
<tr>
<td>021</td>
<td>1145</td>
<td>88</td>
<td>20</td>
<td>605</td>
<td>612</td>
</tr>
<tr>
<td>228</td>
<td>1187</td>
<td>293</td>
<td>8</td>
<td>1570</td>
<td>1574</td>
</tr>
<tr>
<td>071</td>
<td>1193</td>
<td>124</td>
<td>20</td>
<td>629</td>
<td>634</td>
</tr>
<tr>
<td>271</td>
<td>1348</td>
<td>127</td>
<td>12</td>
<td>1163</td>
<td>1164</td>
</tr>
<tr>
<td>237</td>
<td>1566</td>
<td>152</td>
<td>12</td>
<td>1340</td>
<td>1342</td>
</tr>
<tr>
<td>231</td>
<td>1664</td>
<td>101</td>
<td>16</td>
<td>t.o.</td>
<td>1137</td>
</tr>
<tr>
<td>235</td>
<td>1782</td>
<td>218</td>
<td>16</td>
<td>t.o.</td>
<td>1150</td>
</tr>
<tr>
<td>233</td>
<td>1980</td>
<td>207</td>
<td>19</td>
<td>1118</td>
<td>1121</td>
</tr>
<tr>
<td>294</td>
<td>2014</td>
<td>141</td>
<td>17</td>
<td>1257</td>
<td>1261</td>
</tr>
<tr>
<td>295</td>
<td>2168</td>
<td>965</td>
<td>18</td>
<td>1318</td>
<td>1322</td>
</tr>
<tr>
<td>292</td>
<td>2333</td>
<td>318</td>
<td>3</td>
<td>8009</td>
<td>8009</td>
</tr>
<tr>
<td>298</td>
<td>2399</td>
<td>303</td>
<td>10</td>
<td>2471</td>
<td>2473</td>
</tr>
</tbody>
</table>

**Symbolic A**\(^*\)  
Branch-\&-Bound  
60 sec

Abdeddaïm, Kerbaa, Maler
Zones – From infinite to finite

State
(n, x=3.2, y=2.5 )

Symbolic state (set)
(n, 1≤x≤4, 1≤y≤3)

Zone:
conjunction of
x-y<=n, x<=>n

Kim Larsen [17]
Zones – Operations

(n, 2 ≤ x ≤ 4 ∧ 1 ≤ y ≤ 3 ∧ y - x ≤ 0)

(n, 2 ≤ x ∧ 1 ≤ y ∧ -3 ≤ y - x ≤ 0)

(n, 2 ≤ x ∧ 1 ≤ y ≤ 3 ∧ y - x ≤ 0)

Delay

(n, x = 0 ∧ 1 ≤ y ≤ 3)

(n, 2 ≤ x ≤ 4 ∧ 1 ≤ y)

Reset

Extrapolation

Convex Hull

Delay (stopwatch)

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [18]
Datastructures for Zones

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form
  
  \[
  \begin{array}{c|c|c|c|c|c}
  x_0 & x_1 & x_2 & x_3 \\
  \hline
  1 & 2 & 4 & -2 \\
  3 & 4 & 2 & -2 \\
  \end{array}
  \]

  [RTSS97]

- Clock Difference Diagrams
  
  [CAV99]

- PW List
  
  [SPIN03]
Datastructures for Zones

- Difference Bounded Matrices (DBMs)
- Minimal Constraint Form

**UPPAAL DBM Library**

- Elegant RUBY bindings for easy implementations

- Clock Difference Diagrams

- PW List

**UPPAAL**

*Alexandre David*

*Kim Larsen* [20] ARTIST Summer School, Autrans, France September 10, 2009
Case Studies: Controllers

- Gearbox Controller [TACAS’98]
- Bang & Olufsen Power Controller [RTPS’99, FTRTFT’2k]
- SIDMAR Steel Production Plant [RTCSA’99, DSVV’2k]
- Real-Time RCX Control-Programs [ECRTS’2k]
- Terma, Verification of Memory Management for Radar (2001)
- Scheduling Lacquer Production (2005)
- Memory Arbiter Synthesis and Verification for a Radar Memory Interface Card [NJC’05]

- Adapting the UPPAAL Model of a Distributed Lift System, 2007
- Analyzing a χ model of a turntable system using Spin, CADP and Uppaal, 2006
- Designing, Modelling and Verifying a Container Terminal System Using UPPAAL, 2008
- Model-based system analysis using Chi and Uppaal: An industrial case study, 2008
Case Studies: Protocols

- Philips Audio Protocol [HS’95, CAV’95, RTSS’95, CAV’96]
- Bounded Retransmission Protocol [TACAS’97]
- Bang & Olufsen Audio/Video Protocol [RTSS’97]
- TDMA Protocol [PRFTS’97]
- Lip-Synchronization Protocol [FMICS’97]
- ATM ABR Protocol [CAV’99]
- ABB Fieldbus Protocol [ECRTS’2k]
- Distributed Agreement Protocol [Formats05]
- Leader Election for Mobile Ad Hoc Networks [Charme05]

- Analysis of a protocol for dynamic configuration of IPv4 link local addresses using Uppaal, 2006
- Formalizing SHIM6, a Proposed Internet Standard in UPPAAL, 2007
- Verifying the distributed real-time network protocol RTnet using Uppaal, 2007
- Analysis of the Zeroconf protocol using UPPAAL, 2009
Using UPPAAL as Back-end

- Vooduu: verification of object-oriented designs using Uppaal, 2004
- Formalising the ARTS MPSOC Model in UPPAAL, 2007
- Timed automata translator for Uppaal to PVS
- Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT, 2008
- Verification of COMDES-II Systems Using UPPAAL with Model Transformation, 2008
Priced Timed Automata
Task Graph Scheduling – Revisited

Compute:

\[(D \times (C \times (A + B)) + ((A + B) + (C \times D)))\]

using 2 processors

P1 (fast)

P2 (slow)

ENERGY:

\[
\begin{array}{c|c|c}
\text{component} & \text{energy (W)} & \text{time (ps)} \\
\hline
+ & 5ps & 2ps \\
* & 3ps & 5ps \\
* & 7ps & 7ps \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{state} & \text{energy (W)} \\
\hline
\text{idle} & 10W \\
\text{in use} & 90W \\
\text{idle} & 20W \\
\text{in use} & 30W \\
\end{array}
\]

Energy: 1.39 nano-joule !!
Compute:

\[(D \times (C \times (A + B)) + ((A + B) + (C \times D))\]

using 2 processors

P1 (fast)

P2 (slow)

ENERGY:

P1

\[
\begin{array}{c}
1 & 3 & 4 \\
\end{array}
\]

P2

\[
\begin{array}{c}
2 & 5 & 6 \\
\end{array}
\]

Energy: 1.32 nano-joule

OPTIMAL !!
Optimal Task Graph Scheduling

- Energy-rates: $C : M \rightarrow N$
- Compute schedule with minimum completion-cost??

ARTIST Summer School, Autrans, France
September 10, 2009
Kim Larsen [27]
Priced Timed Automata

Timed Automata + **COST** variable

$_{1}$

$c' = 4$

$0 \leq y \leq 4$

$x := 0$

$c += 1$

$_{2}$

$c' = 2$

$x \leq 2$

$3 \leq y$

$c += 4$

$_{3}$

$y \leq 4$

$y \leq 4$

behavior

Kim Larsen [28]

ARTIST Summer School, Autrans, France
September 10, 2009
Priced Timed Automata

Timed Automata + **COST** variable

Timed Automata + **COST** variable

\[ l_1 \quad \text{c'} = 4 \quad 0 \leq y \leq 4 \quad x := 0 \quad c' = 1 \quad c+ = 4 \]

\[ l_2 \quad \text{c'} = 2 \quad x \leq 2 \quad 3 \leq y \quad y \leq 4 \quad x := 0 \quad \]

\[ l_3 \quad \text{c'} = 2 \quad x \leq 2 \quad 3 \leq y \quad y \leq 4 \]

**TRACES**

\[ (l_1, x=y=0) \xrightarrow{\varepsilon(3)} (l_1, x=y=3) \xrightarrow{1} (l_2, x=0, y=3) \xrightarrow{4} (l_3, _, _) \]

\[ \sum c = 17 \]
Priced Timed Automata

Timed Automata + \textbf{COST} variable

\begin{align*}
(l_1, x=y=0) &\xrightarrow{\varepsilon(3)} (l_1, x=y=3) \xrightarrow{1} (l_2, x=0, y=3) \xrightarrow{4} (l_3, -, -) \\
(l_1, x=y=0) &\xrightarrow{\varepsilon(2.5)} (l_1, x=y=2.5) \xrightarrow{1} (l_2, x=0, y=2.5) \xrightarrow{\varepsilon(.5)} (l_2, x=0.5, y=3) \xrightarrow{4} (l_3, -, -) \\
(l_1, x=y=0) &\xrightarrow{1} (l_2, x=0, y=0) \xrightarrow{\varepsilon(3)} (l_2, x=3, y=3) \xrightarrow{0} (l_2, x=0, y=3) \xrightarrow{4} (l_3, -, -)
\end{align*}

\(\sum c=17 \quad \sum c=16 \quad \sum c=11\)
Cost-Optimality Reachability

Behrmann, Fehnker, et al (HSCC’01)
Alur, Torre, Pappas (HSCC’01)

\[ \text{Cost of step } n \]

\[ c_1 + c_2 + \ldots + c_n \]

\[ \sigma \]

Efficient Implementation:
CAV’01 and TACAS’04

Competitive with and Complementary to MILP

Optimal Schedule \( \sigma^* \):
\[ \text{val}(\sigma^*) = \inf_{\sigma} \text{val}(\sigma) \]

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [31]
Optimal Infinite Scheduling

Maximize throughput:
i.e. maximize Reward / Time in the long run!
Optimal Infinite Scheduling

Minimize Energy Consumption:
i.e. minimize Cost / Time in the long run
Maximize throughput:

i.e. maximize Reward / Cost in the long run
Mean Pay–Off Optimality

Bouyer, Brinksma, Larsen: HSCC04, FMSD07

Value of path $\sigma$: $\text{val}(\sigma) = \lim_{n \to \infty} \frac{c_n}{r_n}$

Optimal Schedule $\sigma^*$: $\text{val}(\sigma^*) = \inf_{\sigma} \text{val}(\sigma)$
Discount Optimality  \( \lambda < 1 \): discounting factor

Larsen, Fahrenberg: INFINITY’08

Cost of time \( t_n \)

Time of step \( n \)

Value of path \( \sigma \):  \( \text{val}(\sigma) = \int_{t=0}^{t=\infty} c(t) \lambda^t dt \)

Optimal Schedule \( \sigma^* \):  \( \text{val}(\sigma^*) = \inf_{\sigma} \text{val}(\sigma) \)
Maximize throughput while respecting: $0 \leq E \leq \text{MAX}$
Energy-Bounded Infinite Runs


Cost of time $t_n$

Time of step $n$

Energy

Time

$\sigma$

$\neg$ BAD

Several open problems

Kim Larsen [38]

ARTIST Summer School, Autrans, France
September 10, 2009
Multiple Objective Scheduling

The Pareto Frontier for Reachability in Multi Priced Timed Automata is computable

[Larsen&Rasmussen: FoSSaCS05]
Schedulability Analysis
Task Scheduling

utilization of CPU

- $P(i)$, $[E(i), L(i)]$, .. : period or earliest/latest arrival or .. for $T_i$
- $C(i)$: execution time for $T_i$
- $D(i)$: deadline for $T_i$

Scheduler

- $T_1$ ready done
- $T_2$ stop run
- $T_n$ is running
- $\{T_4, T_1, T_3\}$ ready ordered according to some given priority: (e.g. Fixed Priority, Earliest Deadline,..)

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [41]
Modeling Task

Scheduler

- ready
- done
- stop
- run

States:
- Idle
- Ready
- Running
- Error

Transitions:
- t<=L[id]
- t>=E[id]
- ready[id]!
- t=0
- done[id]!
- ax==C[id]
- run[id]?
- ax=0
- t>D[id]
- ax<=C[id]

ARTIST Summer School, Autrans, France
September 10, 2009
Kim Larsen [42]
Modeling Scheduler

Scheduler

T_1

ready

done

T_2

stop

run

T_n

len > 0

run[front()]!

len == 0

ready[e]?

enqueue(e)

e : id_t

done[e]?

dequeue()

Free

Occ

e : id_t

ready[e]?

enqueue(e)
Modeling Queue

In UPPAAL 4.0
User Defined Function

// Put an element at the end of the queue
void enqueue(id_t element)
{
    int tmp=0;
    list[len++] = element;
    if (len>0)
    {
        int i=len-1;
        while (i>1 && P[list[i]]>P[list[i-1]])
        {
            tmp = list[i-1];
            list[i-1] = list[i];
            list[i] = tmp;
            i--;
        }
    }
}

// Remove the front element of the queue
void dequeue()
{
    ......
}
Schedulability = Safety Property

\[ A \neg (\text{Task0.Error or Task1.Error or ...}) \]

May be extended with preemption
Preemption – Stopwatches!

Scheduler

Task

Defeating undecidability 😊
Handling realistic applications?

Smart phone:

[Application from Marcus Schmitz, TU Linkoping]
Smart phone

- Tasks: 114
- Deadlines: [0.02: 0.5] sec
- Execution: [52 : 266.687] cycles
- Platform:
  - 6 processors, 25 MHz
  - 1 bus

Verified in 1.5 hours!

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [48]
Safety Critical Java
Schedulability Analysis

With Bent Thomsen
 Petur Olesen
  Thomas Bøghholm
A Safety Critical System
Hardware

- JOP (Java Optimized Processor)
- Native execution of Java Bytecode
- Bytecode implemented in Microcode
- Avoid unpredictable data-cache
- Time predictable
- Developed new method and stack cache
- Implemented in FPGA
Java Optimizing Processor

FPGA

Martin Schöberl
University of Tech., Vienna

ARTIST Summer School, Autrans, France
September 10, 2009
Kim Larsen [52]
JOP Block Diagram

3-4 different FPGAs
6 different boards

FPGA

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [53]
public static void main(String[] args) {
    new SporadicPushMotor(
        new SporadicParameters(4, 4000, 60), 0);
    new SporadicPushMotor(
        new SporadicParameters(2, 4000, 60), 1);

    PeriodicMotorSpooler motorSpooler =
    new PeriodicMotorSpooler(
        new PeriodicParameters(4000));

    new PeriodicReadSensor(
        new PeriodicParameters(2000), motorSpooler);

    RealtimeSystem.start();
}
protected boolean run()
    
    if i<5 {
        i = i + 4;
    } else {
        i = i * 4;
    }
    
    return true;

Method: run ()Z
  0: aload_0
  1: getfield
  4: ifeq -> 20
  7: aload_0
  8: dup
  9: getfield
  12: iconst_1
  13: iadd
  14: putfield
  17: goto -> 30
  20: aload_0
  21: dup
  22: getfield
  25: iconst_1
  26: isub
  27: putfield
  30: iconst_1
  31: ireturn
Byte code – Timed Automata

```java
protected boolean run()
    if i<5 {
        i = i + 4;
    } else {
        i = i * 4;
    }
    return true;
}
```

Timing = WCET from microcode
public static void main(String[] args) {
    new SporadicPushMotor(
        new SporadicParameters(4, 4000, 60), 0);
    new SporadicPushMotor(
        new SporadicParameters(2, 4000, 60), 1);
    PeriodicMotorSpooler motorSpooler =
        new PeriodicMotorSpooler(
            new PeriodicParameters(4000));
    new PeriodicReadSensor(
        new PeriodicParameters(2000), motorSpooler);
    RealtimeSystem.start();
}

private void handleBrick() {
    Sensors.synchronizedReadSensors();
    int input = (Sensors.getBufferedSensor(0) + Sensors
        .getBufferedSensor(1)) >> 1;
    if (input > lastRead) {
        lastRead = input;
    } else if ((lastRead - input) >= TRESHOLD) {
        awaitingBrick = false;
        if (lastRead > BRICK_DETECTED) {
            brickFound(lastRead);
        }
    }
}

brickFound(int lastRead) {
    // Function body
}

awaitingBrick = true;

if (awaitingBrick) {
    if (input > lastRead) {
        lastRead = input;
    } else if ((lastRead - input) >= TRESHOLD) {
        awaitingBrick = false;
        if (lastRead > BRICK_DETECTED) {
            brickFound(lastRead);
        }
    }
}
SARTS – to Timed Automata

Detection of Deadline Violation
Integrated SARTS w ECLIPSE
Visualize WCET in ECLIPSE

18 methods + 4 tasks = 76 components
METAMOC

Modular Execution Time Analysis using Model Checking

With Rene R Hansen
Andreas Dalsgaard
Mads Olesen
Martin Toft
WCET: Worst Case Execution Time

- Isolated, non-blocking code
- Annotation with loop counts.
- Cache used for speeding up access of memory blocks.
- Full associatively.
- LRU replacement strategy.

```javascript
var A;
var B;
var C;
var D;
var E;

while (E < 11) 10 {
  output B;
  output C;
  output A;
  output D;
  output C
}
```
WCET: Worst Case Execution Time

In general: hard or impossible to predict

JOP: No data caching
No pipelining
No branch prediction

Sound Upper Bound
Tool Chain

- Annotated Executable
  - disassemble (objdump, Dissy)
  - ARM assembly
  - value analysis (WALi)

- Pipeline (UPPAAL model)
  - ARM-to-UPPAAL
  - Control Flow Graph (UPPAAL model)

- Main Memory (UPPAAL model)
  - combine
  - Complete model (UPPAAL model)

- Cache specs.
  - generate (cache-gen)
  - Caches (UPPAAL model)

- Model check (UPPAAL)
  - WCET
Disassembler – Dissy
Model Creation – pyuppaal

- Library written in Python for manipulating UPPAAL models
- Import/Export UPPAAL models
- Automatic layout of models
- Used to generate most of our models
- Open Source
Experiments

- Conducted on the concrete implementation for the ARM920T processor
- Examine three qualities:
  - Size and complexity of processes
  - How much sharper WCETs are found by taking caching into account
  - Resource usage (time and memory)
- No evaluation of the pipeline
- No reference WCETs available
- Benchmark programs from the WCET Analysis Project by Mälardalen Real-Time Research Center
  - Wide selection of computation tasks
  - Used to benchmark WCET analysis methods
Experiments

- The most interesting findings:
  - Taking the **instruction cache** into account yields WCETs that are up to 97% sharper (78% on average at -02)
  - Taking the **data cache** into account yields WCETs that are up to 68% sharper (31% on average at -02)
  - Almost all results are obtained within five minutes

- Some programs fail due to
  - State space explosion (9)
  - Write to program counter (2)
  - Floating point operations
  - Value analysis problems
Future Work

- Still work in progress
- UPPAAL provides flexible framework for modularization
- Analysis times acceptable.

- Integration of abstract caches
- Better value analysis
- Integration of WCET and Schedulability Analysis
More Information

http://sarts.boegholm.dk
http://metamoc.martintoft.dk
When you are in

( Task1.End Task2.End Task3._id2 Task4.End Task5._id0 Task6._id0 Task7._id0 M1._id4 M2._id7 )
f1=1 f2=1 f3=0 f4=1 f5=0 f6=0 f7=0 f0=1 B1=0 B2=6
(4<=x2 && time==18 && x2<=8),

Take transition

Task6._id0->Task6._id1 { a == 1 && b == 1, use1!, B1 := D1 }
M1._id4->M1._id5 { 1, use1?, x1 := 0 }

When you are in

( Task1.End Task2.End Task3.End Task4._id0 Task5._id0 Task6._id1 Task7._id0 M1._id5 M2._id6 )
f1=1 f2=1 f3=1 f4=1 f5=0 f6=0 f7=0 f0=1 B1=3 B2=10
(18<=time && x1<=6 && time<=22 && time-x1<=18),

Take transition

Task5._id0->Task5._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

( Task1.End Task2.End Task3.End Task4._id0 Task5._id0 Task6._id1 Task7._id0 M1._id5 M2._id6 )
f1=1 f2=1 f3=1 f4=0 f5=0 f6=0 f7=0 f0=1 B1=3 B2=2
(x1-time==-10 && time==10),

Take transition

Task4._id0->Task4._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

( Task1.End Task2.End Task3.End Task4.End Task5._id1 Task6._id0 Task7._id0 M1._id5 M2._id6 )
f1=1 f2=1 f3=1 f4=1 f5=0 f6=0 f7=0 f0=1 B1=8 B2=8
(x1<=3 && x1-time==-18) || (20<=time && x1-time<=12 && time<=21 && time-x1<18),

Take transition

Task6._id0->Task6._id2 { a == 1 && b == 1, use2!, B2 := D2 }
M2._id6->M2._id7 { 1, use2?, x2 := 0 }

When you are in

...
Optimal Scheduling under Uncertainty

Decidable with 1 clock
Acyclic
Bounded length
Strong non-zeno cost-behaviour

Undecidable with 3 clocks or more

Open problem with 2 clocks

[BLMR06] [LTMM02] [ABM04] [BCFL04] [BBR05, BBM06]

Kim Larsen [72]
\( \phi \) : Never two trains at the crossing at the same time
Synthesis (ex Train Gate)

Controller

\(\phi\): Never two trains at the crossing at the same time

Environment
Synthesis

Two Player Game

Find strategy for controllable actions st behaviour satisfies $\phi$

$\phi$: Never two trains at the crossing at the same time
Timed Games

Reachability

Memoryless strategy:
\[ F : Q \rightarrow E_c \cup \lambda \]

Winning Run:
\[ \text{States}(\rho) \cap G \neq \emptyset \]

Winning Strategy:
\[ \text{Runs}(F) \subseteq \text{WinRuns} \]
UPPAAL Tiga

Synthesis of winning strategies for TIMED GAMES

Efficient on-the-fly generation of winning strategies for safety & liveness objectives

CONCUR05, CAV07, FORMATS07

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [77]
Synthesis of Hydraulic Controller

Quasimodo
(Cassez, Jessen Larsen, Rainier, Raskin - HSCC09)

- Tool Chain
  - Synthesis: UPPAAL TIGA
  - Verification: PHAVer
  - Performance: SIMULINK

- 40% improvement of existing solutions.

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [78]
Oil Pump Control Problem

- **R1**: stay within safe interval \([ 5, 25 \] \)

- **R2**: minimize average/overall oil volume

\[
\int_{t=0}^{t=T} v(t) \, dt / T
\]
Tool Chain

Synthesis TIGA

Performance Evaluation
SIMULINK

Guaranteed
Correctness
Robustness

with
40% Improvement

ARTIST Summer School, Autrans, France
September 10, 2009

Kim Larsen [80]
Conclusion & Open Problems

- **Priced Timed Games**
  - Reachability & Model Checking
  - Energy-Bounded Prb.
  - Safety
- **Probabilistic Priced Timed Automata**
- **Timed Automata**
  - Fully Symbolic (CDD)
  - Static Analysis & Slicing of C-code
- **Timed Games**
  - Partial Observability
  - Alternating
  - CEGAR
- **Priced Timed Automata**
  - Optimal Infinite

**Thanks for your attention!**

www.uppaal.com

---

Kim Larsen [81]