Verification, Performance Analysis and Controller Synthesis of Real Time Systems using UPPAAL

Kim Guldstrand Larsen
UPPAAL Branches

- Real Time Verification
  - CLASSIC

- Real Time Scheduling & Performance Evaluation
  - CORA

- Real Time Controller Synthesis
  - TIGA

- Real Time Testing
  - TRON

UCb

Modeling Formalism Theory
Alg.& Datastr. Applications
Open Problems DEMO’s
Real Time Systems

Verification, Performance Analysis, and Controller Synthesis

using UPPAAL

Kim Guldstrand Larsen
CISS, Aalborg University, DENMARK

International Summer School
Marktoberdorf
August 5-17, 2008
BRICS Machine
Basic Research in Computer Science, 1993-2006

5+7+10 M Euro

100

100

Aalborg

Aarhus

Tools

Other relevant projects
ARTIST, AMETIST
Tools and BRICS

Applications

visualSTATE

UPPAAL

SPIN

Algorithmic

PVS

HOL

TLP

ALF

Logic

• Temporal Logic
• Modal Logic
• MSOL

• (Timed) Automata Theory
• Graph Theory
• BDDs
• Polyhedra Manipulation

Semantics

• Concurrency Theory
• Abstract Interpretation
• Compositionality
• Models for real-time & hybrid systems
CISS  
Center for Embedded Software Systems

- National Competence Center 2002 sponsored by:
  - Ministry of Tech. & Res.
  - North Jutland
  - Aalborg City
  - Aalborg University

- 40 projects
- 20 CISS employees
- 20 industrial PhDs

UCb
European Network of Excellence

32 partners

Joseph Sifakis
Co-winner of Turing Award 2007
ARTIST Director
Why Verification and Testing

- 30-40% of production time is currently spend on elaborate, ad-hoc testing:
  - Errors expensive and difficult to fix!
  - The potential of existing/improved testing methods and tools is enormous!
  - Time-to-market may be shortened considerable by verification and performance analyses of early designs!
Why Verification and Testing

- **IMPORTANCE for EMBEDDED SYSTEMS**
  - Often safety critical
  - Often economical critical
  - Hard to patch

- **CHALLENGES for EMBEDDED SYSTEMS**
  - Correctness of embedded systems depend crucially on use of
    - **resources**
    - e.g. real-time, memory, bandwidth, energy.
  - Need for
    - *quantitative models*
Collaborators

@UPPsala
- Wang Yi
- Paul Pettersson
- John Håkansson
- Anders Hessel
- Pavel Krcal
- Leonid Mokrushin
- Shi Xiaochun

@AALborg
- Kim G Larsen
- Gerd Behrman
- Alexandre David
- Jacob I. Rasmussen
- Brian Nielsen
- Arne Skou
- Marius Mikucionis
- Thomas Chatain

@Elsewhere
- Emmanuel Fleury, Didier Lime, Johan Bengtsson, Fredrik Larsson, Kåre J Kristoffersen, Tobias Amnell, Thomas Hune, Oliver Möller, Elena Fersman, Carsten Weise, David Griffioen, Ansgar Fehnker, Frits Vandraager, Theo Ruys, Pedro D’Argenio, J-P Katoen, Jan Tretmans, Judi Romijn, Ed Brinksma, Martijn Hendriks, Klaus Havelund, Franck Cassez, Magnus Lindahl, Francois Laroussinie, Patricia Bouyer, Augusto Burgueno, H. Bowmann, D. Latella, M. Massink, G. Faconti, Kristina Lundqvist, Lars Asplund, Justin Pearson...
Real Time Systems

Plant
Continuous

Controller Program
Discrete

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

Real Time System
A system where correctness not only depends on the logical order of events but also on their **timing**!!
Real Time Model Checking

Plant
Continuous

Controller Program
Discrete

Model of tasks (automatic?)

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

UPPAAL Model

sensors

actuators

SAT ??
Real Time Control Synthesis

**Plant**
*Continuous*

**Controller Program**
*Discrete*

**Model**
of environment (user-supplied)

Partial UPPAAL Model

**Synthesis**
of tasks (automatic)
Timed Automata
Alur & Dill 1989
**Timed Automata**

- **Location**: Off → Light → Bright
- **Reset**: x:=0
- **Guard**: conjunctions of \( x \sim n \) \( \Rightarrow 2\{<,\cdot,=,\cdot,>\} \)
- **Press?**
  - \( x>3 \)
  - \( x::0 \)
  - \( x:3 \)
  - \( x::3 \)

**Synchronizing action**
Timed Automata semantics

State

\((location, x=v, y=u)\) where \(v, u\) are in \(\mathbb{R}\)

Transitions

\((n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)\)

Discrete Trans

\((n, x=2.4, y=3.1415) \xrightarrow{e(1.1)} (n, x=3.5, y=4.2415)\)

Delay Trans

\(x \leq 5 \& y > 3\)

\(x := 0\)
Timed Automata

Transitions:

- delay 4.32 \rightarrow ( \text{Off}, x=0 )
- press? \rightarrow ( \text{Light}, x=0 )
- delay 2.51 \rightarrow ( \text{Light}, x=2.51 )
- press? \rightarrow ( \text{Bright}, x=2.51 )
Intelligent Light Control

Using Invariants

Off

Light $x \cdot 100$

Bright $x \cdot 100$

$x := 0$

$x > 3$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$

$x := 0$
Timed Automata

Invariants

Transitions

\( ( n, x=2.4, y=3.1415 ) \xrightarrow{e(3.2)} \)

\( ( n, x=2.4, y=3.1415 ) \xrightarrow{e(1.1)} ( n, x=3.5, y=4.2415 ) \)

Invariants ensure progress!!
Intelligent Light Control

Invariants

Transitions:

- (Off, x=0) → (Off, x=0)
- delay 4.32 → (Off, x=4.32)
- press? → (Light, x=0)
- delay 4.51 → (Light, x=4.51)
- press? → (Light, x=0)
- delay 100 → (Light, x=100)
- $\tau$ → (Off, x=0)

Note:

(Off, x=0) delay 103 →
Constraints

Definition
Let $X$ be a set of clock variables. The set $\mathcal{B}(X)$ of clock constraints $\phi$ is given by the grammar:

$$\phi ::= x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \land \phi_2$$

where $c \in \mathbb{N}$ (or $\mathbb{Q}$).
Clock Valuations and Notation

Definition
The set of clock valuations, $\mathbb{R}^C$ is the set of functions $C \rightarrow \mathbb{R}_{\geq 0}$ ranged over by $u, v, w, \ldots$.

Notation
Let $u \in \mathbb{R}^C$, $r \subseteq C$, $d \in \mathbb{R}_{\geq 0}$, and $g \in \mathcal{B}(X)$ then:

- $u + d \in \mathbb{R}^C$ is defined by $(u + d)(x) = u(x) + d$ for any clock $x$.

- $u[r] \in \mathbb{R}^C$ is defined by $u[r](x) = 0$ when $x \in r$ and $u[r](x) = u(x)$ for $x \not\in r$.

- $u \models g$ denotes that $g$ is satisfied by $u$. 
Timed Automata

Definition
A timed automaton $A$ over clocks $C$ and actions $Act$ is a tuple $(L, l_0, E, I)$, where:

- $L$ is a finite set of locations
- $l_0 \in L$ is the initial location
- $E \subseteq L \times B(X) \times Act \times P(C) \times L$ is the set of edges
- $I : L \rightarrow B(X)$ assigns to each location an invariant
Semantics

Definition
The semantics of a timed automaton $A$ is a labelled transition system with state space $L \times \mathbb{R}^C$ with initial state $(l_0, u_0)$* and with the following transitions:

- $(l, u) \xrightarrow{\epsilon(d)} (l, u + d)$ iff $u \in I(l)$ and $u + d \in I(l)$,
- $(l, u) \xrightarrow{a} (l', u')$ iff there exists $(l, g, a, r, l') \in E$ such that
  - $u \models g$,
  - $u' = u[r]$, and
  - $u' \in I(l')$

*$u_0(x) = 0$ for all $x \in C$
Example

With two clocks

Reachable?
Example

*With two clocks*

\[(L_0, x=0, y=0)\]
Example
With two clocks

\begin{align*}
\text{Reachable?} & \\
\text{(L}_0, x=0, y=0) & \rightarrow \epsilon(1.4) \\
\text{(L}_0, x=1.4, y=1.4)
\end{align*}
Example
With two clocks

Reachable?

\[(L_0, x=0, y=0) \rightarrow (L_1, x=1.4, y=1.4) \rightarrow a (L_0, x=1.4, y=0)\]
Example

With two clocks

\[(L_0, x=0, y=0) \Rightarrow \epsilon(1.4)\]
\[(L_0, x=1.4, y=1.4) \Rightarrow a\]
\[(L_0, x=1.4, y=0) \Rightarrow \epsilon(1.6)\]
\[(L_0, x=3.0, y=1.6) \Rightarrow a\]
\[(L_0, x=3.0, y=0)\]
Networks Light Controller & User

Transitions:

\[
\begin{align*}
( \text{Off, Rest, } x=0, y=0 ) & \rightarrow ( \text{Off, Rest, } x=20, y=20 ) \\
\text{delay } 20 & \rightarrow ( \text{Light, Busy, } x=0, y=0 ) \\
\text{press?!} & \rightarrow ( \text{Light, Busy, } x=2, y=2 ) \\
\text{delay } 2 & \rightarrow ( \text{Light, Busy, } x=0, y=0 ) \\
\text{press?!} & \rightarrow ( \text{Bright, Rest, } x=0, y=0 )
\end{align*}
\]
Networks of Timed Automata (a’la CCS)

Two-way synchronization on complementary actions.

Closed Systems!

Example transitions

\((l_1, m_1, \ldots, x=2, y=3.5, \ldots) \xrightarrow{\tau} (l_2, m_2, \ldots, x=0, y=3.5, \ldots)\)
Network Semantics

\[ T_1 \parallel_x T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \parallel_x s_0^2) \quad \text{where} \]

\[
\begin{align*}
S_1 & \xrightarrow{\mu} S_1' \\
S_1 \parallel_x S_2 & \xrightarrow{\mu} S_1 \parallel_x S_2 \\
S_2 & \xrightarrow{\mu} S_2'
\end{align*}
\]

\[
\begin{align*}
S_1 & \xrightarrow{a!} S_1' \\
S_1 \parallel_x S_2 & \xrightarrow{\tau} S_1 \parallel_x S_2' \\
S_2 & \xrightarrow{a?} S_2'
\end{align*}
\]

\[
\begin{align*}
S_1 & \xrightarrow{e(d)} S_1' \\
S_1 \parallel_x S_2 & \xrightarrow{e(d)} S_1 \parallel_x S_2'
\end{align*}
\]
Network Semantics
(URGENT synchronization)

\[ T_1 \parallel x T_2 = (S_1 \times S_2, \rightarrow, s_0^1 \parallel x s_0^2) \]

where

\[ S_1 \xrightarrow{\mu} S_1' \]
\[ S_1 \parallel x S_2 \xrightarrow{\mu} S_1 \parallel x S_2 \]
\[ S_2 \xrightarrow{\mu} S_2' \]
\[ S_1 \parallel x S_2 \xrightarrow{\mu} S_1 \parallel x S_2' \]

\[ S_1 \xrightarrow{a!} S_1' \]
\[ S_1 \parallel x S_2 \xrightarrow{\tau} S_1 \parallel x S_2' \]
\[ S_2 \xrightarrow{a?} S_2' \]
\[ S_1 \parallel x S_2 \xrightarrow{\tau} S_1 \parallel x S_2' \]

\[ \forall d' < d, \forall u \in UAct: \]
\[ \neg (s_1 e(d') u? \land s_2 e(d') u!) \]
Light Control Interface
Light Control Interface

- press? d release?  → touch! 0.5·d·1
- press? 1  → starthold!
- press? d release?  → endhold!  d >1

-- Interface --

- press?
- release?

-- Control Program --

- touch!
- starthold!
- endhold!

- press? 0.2 release? ...
- press? 0.7 release? ...
- press? 1.0 2.4 release? ...

- Ø
- touch!
- starthold!
- endhold!

UCb
Light Control Interface

Control Program

User

UCb
Light Control Network

UCb
Validation

Light Controller

Interface

Switch

Dim

User
Timed Automata
Modeling
and
Decidability

Kim Guldstrand Larsen
Overview

- BRICK Sorting
- Reachability Checking
  - Region Construction
- Bisimulation Checking
- Model Checking
- Trace Inclusion Checking
Brick Sorting
LEGO Mindstorms/RCX

- **Sensors:** temperature, light, rotation, pressure.
- **Actuators:** motors, lamps,
- **Virtual machine:**
  - 10 tasks, 4 timers, 16 integers.
- **Several Programming Languages:**
  - NotQuiteC, Mindstorm, Robotics, legOS, etc.
A Real Real Timed System

The Plant
Conveyor Belt & Bricks

Controller Program
LEGO MINDSTORM

What is the CONTROL program doing?
First UPPAAL model
Sorting of Lego Boxes

Boxes

Controller
MAIN PUSH

Conveyer Belt

Piston
remove eject

99
81
90
18
9

black red

Red
Black

9

Ken Tindell
NQC programs

```c
int active;
int DELAY;
int LIGHT_LEVEL;

task MAIN{
  DELAY=75;
  LIGHT_LEVEL=35;
  active=0;
  Sensor(IN_1, IN_LIGHT);
  Fwd(OUT_A,1);
  Display(1);

  start PUSH;

  while(true){
    wait(IN_1<=LIGHT_LEVEL);
    ClearTimer(1);
    active=1;
    PlaySound(1);

    wait(IN_1>LIGHT_LEVEL);
  }
}

task PUSH{
  while(true){
    wait(Timer(1)>DELAY && active==1);
    active=0;
    Rev(OUT_C,1);
    Sleep(8);
    Fwd(OUT_C,1);
    Sleep(12);
    Off(OUT_C);
  }
}
```
A Black Brick & The Guard
GLOBAL DECLARATIONS:
const int ctime = 75;

int[0,1] active;
clock x, time;

chan eject, ok;
chan blk, red, remove, go;
The Production Cell in LEGO

Course at DTU, Copenhagen

Rasmus Crüger Lund
Simon Tune Riemanni
From RCX to UPPAAL – and back

- Model includes Round-Robin Scheduler.
- Compilation of RCX tasks into TA models.
- Presented at ECRTS 2000 in Stockholm.

- From UPPAAL to RCX: Martijn Hendriks.
UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

License

The UPPAAL tool is free for non-profit applications. For information about commercial licenses, please email sales(at)uppaal(dot)com.

To find out more about UPPAAL, read this short introduction. Further information may be found at this web site in the pages About, Documentation, Download, and Examples.

Mailing Lists

UPPAAL has an open discussion forum group at Yahoo!Groups intended for users of the tool. To join or post to the forum, please refer to the information at the discussion forum page. Bugs should be reported using the bug tracking system. To email the development team directly, please use uppaal(at)list(dot)it(dotted)u(a)d(dotted)se.
Decidability
The Region Construction
Reachability?

Reachable from initial state (L0, x=0, y=0)?

OBSTACLE:
Uncountably infinite state space

locations \times R^C

clock-valuations
Constraints

Definition
Let X be a set of clock variables. The set \( B(X) \) of clock constraints \( \phi \) is given by the grammar:

\[
\phi ::= x \leq c \mid c \leq x \mid x < c \mid c < x \mid \phi_1 \land \phi_2
\]

where \( c \in \mathbb{N} \) (or \( \mathbb{Q} \)).
Clock Valuations and Notation

Definition
The set of clock valuations, \( \mathbb{R}^C \) is the set of functions \( C \rightarrow \mathbb{R}_{\geq 0} \) ranged over by \( u, v, w, \ldots \).

Notation
Let \( u \in \mathbb{R}^C \), \( r \subseteq C \), \( d \in \mathbb{R}_{\geq 0} \), and \( g \in \mathcal{B}(X) \) then:

- \( u + d \in \mathbb{R}^C \) is defined by \( (u + d)(x) = u(x) + d \) for any clock \( x \)

- \( u[r] \in \mathbb{R}^C \) is defined by \( u[r](x) = 0 \) when \( x \in r \) and
  \( u[r](x) = u(x) \) for \( x \not\in r \).

- \( u \models g \) denotes that \( g \) is satisfied by \( u \).
Timed Automata

Definition

A timed automaton $A$ over clocks $C$ and actions $Act$ is a tuple $(L, l_0, E, I)$, where:

- $L$ is a finite set of locations
- $l_0 \in L$ is the initial location
- $E \subseteq L \times B(X) \times Act \times P(C) \times L$ is the set of edges
- $I : L \rightarrow B(X)$ assigns to each location an invariant
Semantics

Definition
The semantics of a timed automaton $A$ is a labelled transition system with state space $L \times \mathbb{R}^C$ with initial state $(l_0, u_0)^*$ and with the following transitions:

- $(l, u) \xrightarrow{\epsilon(d)} (l, u + d)$ iff $u \in I(l)$ and $u + d \in I(l)$,
- $(l, u) \xrightarrow{a} (l', u')$ iff there exists $(l, g, a, r, l') \in E$ such that
  - $u \models g$,
  - $u' = u[r]$, and
  - $u' \in I(l')$

*$u_0(x) = 0$ for all $x \in C$
Derived Relations and Reachability

\[(l, u) \xrightarrow{\delta} (l', u') \quad \text{iff} \quad \exists d > 0. \ (l, u) \xrightarrow{\epsilon(d)} (l', u').\]

\[(l, u) \xrightarrow{\alpha} (l', u') \quad \text{iff} \quad \exists a \in \text{Act}. \ (l, u) \xrightarrow{\alpha} (l', u').\]

\[(l, u) \xrightarrow{\sim} (l', u') \quad \text{iff} \quad (l, u)(\xrightarrow{\delta} \cup \xrightarrow{\alpha})^* (l', u').\]

**Definition**

The set of reachable locations, \( \text{Reach}(A) \), of a timed automaton \( A \) is defined as:

\[ l \in \text{Reach}(A) \equiv^\Delta \exists u. (l_0, u_0) \xrightarrow{\sim} (l, u) \]
Time Abstracted Bisimulation

Definition
Let $G \subseteq L$ be a set of goal locations. An equivalence relation $R$ on $L \times \mathbb{R}^C$ is a TAB wrt $G$ if whenever $(l, u)R(n, v)$ the following holds:

1. $l \in G$ iff $n \in G$,
2. whenever $(l, u) \xrightarrow{\delta} (l', u')$ then $(n, v) \xrightarrow{\delta} (n', v')$ with $(l', u')R(n', v')$
3. whenever $(l, u) \xrightarrow{a} (l', u')$ then $(n, v) \xrightarrow{a} (n', v')$ with $(l', u')R(n', v')$
Stable Quotient

Definition
Let $R$ be a TAB wrt $G$. The induced quotient has classes of $R$, $\pi \in (L \times R^C/R)$, as states. For classes $\pi, \pi'$ the transitions are

- $\pi \xrightarrow{\delta} \pi'$ iff $(l, u) \xrightarrow{\delta} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.
- $\pi \xrightarrow{a} \pi'$ iff $(l, u) \xrightarrow{a} (l', u')$ for some $(l, u) \in \pi$, $(l', u') \in \pi'$.

Theorem
Let $R$ be TAB wrt $G$. Then, a location from $G$ is reachable iff there exists an equivalence class $\pi$ of $R$ such that $\pi$ is reachable in the quotient and $\pi$ contains a state whose location is in $G$. 

UCb
Stable Quotient

Partitioning

UCb
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning

UCb
Stable Quotient

Partitioning

UCb
Stable Quotient

Partitioning

Reachable?
Stable Quotient

Partitioning

0 \rightarrow \varepsilon \rightarrow 1 \rightarrow a \rightarrow 2 \rightarrow \varepsilon \rightarrow 3 \rightarrow a \rightarrow 4 \rightarrow \varepsilon \rightarrow 5 \rightarrow c \rightarrow 6
Region Equivalence

For each clock $x$ let $c_x$ be the largest integer with which $x$ is compared in any guard or invariant of $A$. $u$ and $u'$ are region equivalent, $u \equiv u'$ iff the following holds:

1. For all $x \in C$, either $\lfloor u(x) \rfloor = \lfloor u'(x) \rfloor$ or $u(x), u'(x) > c_x$;

2. For all $x, y \in C$ with $u(x) \leq c_x$ and $u(y) \leq c_y$,
   
   $fr(u(x)) \leq fr(u(y))$ iff $fr(u'(x)) \leq fr(u'(y))$;

3. For all $x \in C$ with $u(x) \leq c_x$,

   $fr(u(x)) = 0$ iff $fr(u'(x)) = 0$. 

UCb
Regions

Finite Partitioning of State Space

For each clock $x$ let $c_x$ be the largest integer with which $x$ is compared in any guard or invariant of $A$. $u$ and $u'$ are region equivalent, $u \equiv u'$ iff the following holds:

1. For all $x \in C$, either $\lfloor u(x) \rfloor = \lfloor u'(x) \rfloor$ or $u(x), u'(x) > c_x$;

2. For all $x, y \in C$ with $u(x) \leq c_x$ and $u(y) \leq c_y$, $fr(u(x)) \leq fr(u(y))$ iff $fr(u'(x)) \leq fr(u'(y))$;

3. For all $x \in C$ with $u(x) \leq c_x$, $fr(u(x)) = 0$ iff $fr(u'(x)) = 0$.

An equivalence class (i.e. a region) in fact there is only a finite number of regions!!
Logical Characterization of Regions

Each region may be represented by specifying

1. for every clock $x$ a constraint from

   \[ \{ x = c \mid c = 0, 1, \ldots, c_x \} \cup \{ c - 1 < x < c \mid c = 1, \ldots, c_x \} \cup \{ x > c_x \} \]

2. for every pair of clocks $x, y$ such that $c - 1 < x < c$ and $d - 1 < y < d$
   appears in 1., whether $fr(x)$ is $<$, $=$ or $>$ than $fr(y)$.

**Theorem**

The number of regions is $n! \cdot 2^n \cdot \prod_{x \in C}(2c_x + 2)$. 
Stability of Regions

Lemma
1. If $u \simeq u'$ then $\forall d. \exists d'. u + d \simeq u' + d'$;
2. If $u \simeq u'$ then* $\forall g \in B(X). u \models g \iff u' \models g$;
3. If $u \simeq u'$ then $\forall r \subseteq C. u[r] \simeq u'[r]$.

Theorem
Let $(l, u) \equiv (l', u')$ iff $l = l'$ and $u \simeq u'$. Then $\equiv$ is a TAB with respect to any set of goal locations $G$.

*Here $\equiv$ and $g$ should agree on the maximal constants.
Regions

Successor Operation (wrt delay)

An equivalence class (i.e. a *region*)

Successor regions, $\text{Succ}(r)$
Regions

Reset Operation

An equivalence class (i.e. a region) $r$
An Example Region Graph

(a) $x \geq 2$

(b)

A

\[
\begin{align*}
& l \\
& x = 0
\end{align*}
\]

B

\[
\begin{align*}
& l \\
& 0 < x < 1
\end{align*}
\]

C

\[
\begin{align*}
& l \\
& x = 1
\end{align*}
\]

D

\[
\begin{align*}
& l \\
& 1 < x < 2
\end{align*}
\]

E

\[
\begin{align*}
& l \\
& x = 2
\end{align*}
\]

F

\[
\begin{align*}
& l \\
& x > 2
\end{align*}
\]
Modified light switch

\[
x \geq 1
\]
\[
\begin{cases}
x, y
\end{cases}
\]

\[
y = 3
\]
\[
\begin{cases}
x
\end{cases}
\]

\[
x \geq 2
\]
\[
\begin{cases}
x
\end{cases}
\]

\[
\text{inv}(\text{off}) = \text{true}
\]
\[
\text{inv}(\text{on}) = y \leq 3
\]
reachable part of region graph
Decidability & Complexity

Theorem
The transition relations $\delta$ and $\alpha$ between regions may be computed effectively using the triplet or the logical characterization of regions.

Theorem
Let $A$ be a timed automaton with location $l$. It is decidable whether $l \in \text{Reach}(A)$.

Theorem
Let $A$ be a timed automaton with location $l$. Deciding $l \in \text{Reach}(A)$ is PSPACE-complete.
Fundamental Results

- Reachability ☺

- Model-checking ☺
  - TCTL, $L_{nu}$, $T_{mu}$, ...

- Bisimulation, Simulation
  - Timed ☺; Untimed ☺

- Trace-inclusion
  - Timed ☹; Untimed ☺
Timed Bimulation

Wang’91, Cerans’92
Timed Bisimulation

R is a timed bisimulation if whenever sRt then the following holds:

i) \( (s \xrightarrow{a} s') \Rightarrow (\exists t'. t \xrightarrow{a} t' \land s'Rt') \)

ii) \( (t \xrightarrow{a} t') \Rightarrow (\exists s'. s \xrightarrow{a} s' \land s'Rt') \)

for all \( a \in \text{Act} \cup \text{Del} \).

\[ \text{Del} = \{d : d \in R_{\geq 0}\} \]

We write \( s \approx t \) whenever sRt for some timed bisimulation R.
Timed Simulation

R is a timed simulation if whenever sRt then the following holds:

i) \( (s\xrightarrow{a}s') \Rightarrow (\exists t'. t\xrightarrow{a}t' \land s'Rt') \)

for all \( a \in \text{Act} \cup \text{Del} \).

\[ \text{Del} = \{ d : d \in R_{\geq 0} \} \]

We write \( s \prec t \) iff \( sRt \) for some timed simulation \( R \).
Examples
Towards Timed Bisimulation Algorithm

Towards Timed Bisimulation Algorithm

independent “product-construction”
Towards Timed Bisimulation Algorithm

Definition

B is a timed product-bisimulation iff whenever \( s \in B \) then the following holds:

i) if \( s \xrightarrow{d} s' \) then \( s' \in B \)

ii) if \( s \xrightarrow{a_1} s' \) then \( s' \xrightarrow{a_2} s'' \) s.t. \( s'' \in B \)

iii) if \( s \xrightarrow{a_2} s' \) then \( s' \xrightarrow{a_1} s'' \) s.t. \( s'' \in B \)

We write \( TB(s) \) whenever \( s \in B \) for some timed product-bisimulation.

Theorem

\( TB(s) \iff \left( s_1 \approx s_2 \right) \)
Timed Bisimulation Algorithm =
Checking for TB-ness using Regions
Timed Trace Inclusion
Undecidability
Timed Trace Languages

- **Timed trace**
  \[(t_1,a_1),(t_2,a_2), \ldots, (t_k,a_k)\]
  where \(a_i\) is an action and \(t_i \in \mathbb{R}\), with \(t_i\)’s non-decreasing.

\[
L(A) = \{ (t_1,a), (t_2,a) : t_1 \cdot 1 \not\in t_1 \cdot t_2 \cdot t_1 + 1 \} 
\]

\[
L(X) = \{ (t_1,a), (t_2,a) : t_1 \cdot t_2 \cdot 2 \}
\]
Timed Trace Languages

- Timed trace
  \[(t_1,a_1),(t_2,a_2),..,(t_k,a_k)\]
  where \(a_i\) is an action and \(t_i \in \mathbb{R}\), with \(t_i\)'s strictly increasing.

- PROPOSITIONs
  - Given a timed automaton \(A\) it is **UNDECIDABLE** whether the set of timed traces of \(A\) is the **UNIVERSAL** set.
  - Given two timed automata \(A\) and \(B\) it is **UNDECIDABLE** whether the set of timed traces of \(A\) is **INCLUDED** in the set of timed traces of \(B\).
Two-counter Machine

- \( M = ( \{ b_0, b_1, \ldots, b_k \}, C, D ) \)

where \( b_i \)'s are instructions
C and D are counters ranging over \( \mathbb{N} \).
Initially both C and D are 0.

- Instructions (\( i < k \)):
  - Increment \( b_j \): \( C := C + 1 \); goto \( b_l \)
  - Decrement \( b_j \): if \( C \neq 0 \)
    - then \( C := C - 1 \); goto \( b_l \)
    - else goto \( b_m \)

- \( b_k \) represents termination
Two-counter Machine

- M = ( \{ b_0, b_1, \ldots, b_k \} , C, D )

- Configuration of M:
  ( b_i , c , d)
  where c and d are values of C and D.

- Computation of M is a “valid” sequence of configurations starting with (b_0, 0, 0) and ending with (b_k, _, _).

- **Proposition**
  Deciding whether a two-counter machine has a (halting) computation is UNDECIDABLE.
Timed Trace Language for Two-counter Machine

Let M be a two-counter machine. We define L(M) to be the set of timed traces over

\[ \Sigma = \{b_0, \ldots, b_k, c, d\} \]

such that whenever

\[(b_{i0}, c_0, d_0)(b_{i1}, c_1, d_1) \ldots (b_{in}, c_n, d_n)\]

is a computation of M then the timed trace \( s \) is in \( L(M) \) where:

- \( \text{Untime}(s) = b_{i0}c^{c_0}d^{d_0}b_{i1}c^{c_1}d^{d_1} \ldots b_{in}c^{c_n}d^{d_n} \)
- \( \text{Time}(b_j) = j \)
- “proper matching of \( c \)’s and \( d \)’s”
Proper Matching

- Clearly M has a (halting) computation iff \( L(M) \neq \emptyset \)
- One can show that \( L(M)^C \) can be captured by a Timed Automaton (a union of several small ones).
Example Automata

Violation of $b_i : D := D + 1 ; \text{goto } b_j$

1) Jumping to another instruction than $b_j$:

2) Decreasing or breaking encoding of C
The UPPAAL Verification Engine

Kim Guldstrand Larsen
Overview

- Train Crossing
  - Full Modeling & Specification Formalism
  - Schedulability Analysis

- UPPAAL Verification Engine
  - Symbolic On-the-fly Exploration
  - Zones & DBMs
  - CDDs

- Verification Options
  - Over- / Under Approximations
  - Storage Strategies
Timed Automata in UPPAAL
Train Crossing
UPPAAL

**Graphical Design Tool**
- timed automata
- clocks
- communication
- datatypes & functions
- cost variables
- uncontr. behaviour

**Verifier**
- exhaustive & automatic checking of requirements
- diagnostic traces
- optimal scheduling
- controller synthesis

**Graphical Simulator**
- visualization and recording
- inexpensive fault detect.
- MSCs

**Tool Environment**
for
modeling, simulation, verification, optimization & synthesis of real-time systems
Train Crossing

Stopable Area
[10,20]
[7,15]

list
enqueue()
dequeue()
front()

UCb

Gate

Crossing

River
Train Crossing

Stopable Area

[10,20]

[7,15]

appr

stop

go

Communication via channels!

id-"parameter"

enqueue()
decqueue()
front()
Queries: Specification Language
Logical Specifications

- **Validation Properties**
  - Possibly: \( E<>P \)

- **Safety Properties**
  - Invariant: \( A[]P \)
  - Pos. Inv.: \( E[]P \)

- **Liveness Properties**
  - Eventually: \( A<>P \)
  - Leadsto: \( P \rightarrow Q \)

- **Bounded Liveness**
  - Leads to within: \( P \rightarrow_t Q \)

The expressions \( P \) and \( Q \) must be type safe, side effect free, and evaluate to a boolean.

Only references to integer variables, constants, clocks, and locations are allowed (and arrays of these).
Logical Specifications

- Validation Properties
  - Possibly: $E<>P$

- Safety Properties
  - Invariant: $A[P]$
  - Pos. Inv.: $E[P]$

- Liveness Properties
  - Eventually: $A<>P$
  - Leadsto: $P \rightarrow Q$

- Bounded Liveness
  - Leads to within: $P \rightarrow_t Q$
Logical Specifications

- Validation Properties
  - Possibly: \( E<> P \)

- Safety Properties
  - Invariant: \( A[] P \)
  - Pos. Inv.: \( E[] P \)

- Liveness Properties
  - Eventually: \( A<> P \)
  - Leadsto: \( P \rightarrow Q \)

- Bounded Liveness
  - Leads to within: \( P \rightarrow.t Q \)
Logical Specifications

- **Validation Properties**
  - Possibly: $E<> P$

- **Safety Properties**
  - Invariant: $A[] P$
  - Pos. Inv.: $E[] P$

- **Liveness Properties**
  - Eventually: $A<> P$
  - Leadsto: $P \rightarrow Q$

- **Bounded Liveness**
  - Leads to within: $P \rightarrow^* Q$
Logical Specifications

- Validation Properties
  - Possibly: $E<> P$

- Safety Properties
  - Invariant: $A[\ ] P$
  - Pos. Inv.: $E[\ ] P$

- Liveness Properties
  - Eventually: $A<> P$
  - Leadsto: $P \rightarrow Q$

- Bounded Liveness
  - Leads to within: $P \rightarrow .t Q$
Train Crossing

Communication via channels!

Stopable Area

[10,20]

[7,15]

[3,5]

appr stop
go

list

enqueue() dequeue() front()

id-”parameter"

Crossing

River

Gate

UCb
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]
- Experimental Batch Plant (2000)
- RCX Production Cell (2000)
- 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]
Case Studies: Protocols

- Philips Audio Protocol [HS’95, CAV’95, RTSS’95, CAV’96]
- Collision-Avoidance Protocol [SPIN’95]
- Bounded Retransmission Protocol [TACAS’97]
- Bang & Olufsen Audio/Video Protocol [RTSS’97]
- TDMA Protocol [PRFTS’97]
- Lip-Synchronization Protocol [FMICS’97]
- Multimedia Streams [DSVIS’98]
- ATM ABR Protocol [CAV’99]
- Leader Election for Mobile Ad Hoc Networks [Charme05]
- ABB Fieldbus Protocol [ECRTS’2k]
- Distributed Agreement Protocol [Formats05]
Zones & DBMs
Regions
Finite Partitioning of State Space

Time Abstracted Bisimulation
Equivalence classes (i.e. a region) in fact there is only a finite number of regions!!

Theorem
The number of regions is $n! \cdot 2^n \cdot \prod_{x \in \mathcal{O}} (2c_x + 2)$.

$0 < x < 1 \land 0 < y < 1 \land y - x > 0$
Zones
*From infinite to finite*

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

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

Zone:
conjunction of
\(x-y \leq n, x \Leftrightarrow n\)
Symbolic Transitions

Thus \((n, 1 \leq x \leq 4, 1 \leq y \leq 3) = a \Rightarrow (m, 3 < x, y = 0)\)
Symbolic Exploration

\[ y := 0 \]
\[ y \leq 2 \]
\[ x := 0 \]
\[ x \leq 2 \]
\[ y \leq 2, x \geq 4 \]

Reachable?
Symbolic Exploration

y := 0
y <= 2
x := 0
x <= 2
y <= 2, x >= 4

L0 L1

Reachable?

Delay

UCb
Symbolic Exploration

UCb
Symbolic Exploration

y:=0
y<=2
y<=2, x>=4

x:=0
x<=2

Left
Symbolic Exploration

y:=0

y<=2

x:=0

x<=2

y<=2, x>=4

L0

L1

Reachable?

Delayed
Symbolic Exploration

y:=0

y<=2

x:=0

x<=2

y<=2, x>=4

Reachable?

Left

UCb
Symbolic Exploration

Reachable?

UCb
Symbolic Exploration

y:=0, x:=0
y<=2, x<=2
y<=2, x>=4

L0
L1
Reachable?

Delay

UCb
Symbolic Exploration

Reachable?

UCb
Forward Reachability

**Init** -> **Final**?

**INITIAL**
- **Passed** := $\emptyset$
- **Waiting** := $\{(n_0, Z_0)\}$

**REPEAT**

**UNTIL**
- **Waiting** = $\emptyset$

return false
Forward Reachability

INITIAL
Passed := Ø;
Waiting := {(n₀,Z₀)}

REPEAT

pick (n,Z) in Waiting

if (n,Z) = Final
return true

for all (n,Z) → (n',Z'):

if for some (n',Z'') Z' ⊆ Z''
continue

else
add (n',Z') to Waiting
move (n,Z) to Passed

UNTIL Waiting = Ø
return false
Forward Reachability

Init -> Final?

INITIAL
Passed := Ø;
Waiting := \{(n_0,Z_0)\}

REPEAT
pick \((n,Z)\) in Waiting
if \((n,Z) = \text{Final}\) return true
for all \((n,Z) \rightarrow (n',Z')\)
if for some \((n',Z'')\) \(Z' \subseteq Z''\) continue
else add \((n',Z')\) to Waiting
move \((n,Z)\) to Passed
UNTIL Waiting = Ø
return false

Init

Waiting

Final?

Passed

PW
Forward Reachability

**INITIAL**

\[ \text{Passed} := \emptyset; \]
\[ \text{Waiting} := \{(n_0, Z_0)\} \]

**REPEAT**

pick \((n, Z)\) in Waiting

if \((n, Z) = \text{Final}\) return true

for all \((n, Z) \rightarrow (n', Z'):\)

if for some \((n', Z'')\) \(Z' \subseteq Z''\) continue

**UNTIL** Waiting = \(\emptyset\)

return false
Forward Reachability

**INITIAL**
- \( \text{Passed} := \emptyset; \)
- \( \text{Waiting} := \{(n_0, Z_0)\} \)

**REPEAT**
- pick \((n, Z)\) in \(\text{Waiting}\)
- if \((n, Z) = \text{Final}\) return true
- for all \((n, Z) \rightarrow (n', Z'):\)
  - if for some \((n', Z'')\) \(Z' \subseteq Z''\) continue
  - else add \((n', Z')\) to \(\text{Waiting}\)

**UNTIL** \(\text{Waiting} = \emptyset\)
- return false
Forward Reachability

\[ \text{Init} \rightarrow \text{Final} \]

\[
\text{INITIAL} \quad \text{Passed} := \emptyset; \\
\text{Waiting} := \{(n_0,Z_0)\}
\]

\[
\text{REPEAT} \\
\text{pick } (n,Z) \text{ in Waiting} \\
\text{if } (n,Z) = \text{Final return true} \\
\text{for all } (n,Z) \rightarrow (n',Z'):\ \\
\text{if for some } (n',Z'') \text{ Z' } \subseteq \text{ Z'' continue} \\
\text{else add } (n',Z') \text{ to Waiting} \\
\text{move } (n,Z) \text{ to Passed}
\]

\[
\text{UNTIL} \quad \text{Waiting} = \emptyset \\
\text{return false}
\]
Forward Reachability

**INITIAL**

\[ \text{Passed} := \emptyset; \]
\[ \text{Waiting} := \{(n_0, Z_0)\} \]

**REPEAT**

pick \((n, Z)\) in \text{Waiting}

**if** \((n, Z) = \text{Final}\) **return** true

**for all** \((n, Z) \rightarrow (n', Z'):\)

**if** for some \((n', Z'')\) \(Z' \subseteq Z''\) **continue**

**else** add \((n', Z')\) to \text{Waiting}

move \((n, Z)\) to \text{Passed}

**UNTIL** \text{Waiting} = \emptyset

return false
Canonical Datastructures for Zones

Difference Bounded Matrices

Bellman 1958, Dill 1989

Inclusion

D1

\[
\begin{align*}
x &\leq 1 \\
y - x &\leq 2 \\
z - y &\leq 2 \\
z &\leq 9
\end{align*}
\]

Graph

\[
\begin{array}{c}
0 \\
1 \\
2 \\
9 \\
3 \\
\end{array}
\]

D2

\[
\begin{align*}
x &\leq 2 \\
y - x &\leq 3 \\
y &\leq 3 \\
z - y &\leq 3 \\
z &\leq 7
\end{align*}
\]

Graph

\[
\begin{array}{c}
0 \\
2 \\
3 \\
7 \\
3 \\
\end{array}
\]

UCb
Canonical Datastructures for Zones

**Difference Bounded Matrices**

### Inclusion

**D1**

\[
\begin{align*}
x & \leq 1 \\
y-x & \leq 2 \\
z-y & \leq 2 \\
z & \leq 9
\end{align*}
\]

**Graph**

\[
\begin{align*}
0 & \rightarrow x \\
& \rightarrow 1 \\
& \rightarrow y \\
& \rightarrow 2 \\
& \rightarrow 3
\end{align*}
\]

**Shortest Path Closure**

\[
\begin{align*}
0 & \rightarrow 3 \\
& \rightarrow 4 \\
& \rightarrow 2
\end{align*}
\]

**D2**

\[
\begin{align*}
x & \leq 2 \\
y-x & \leq 3 \\
y & \leq 3 \\
z-y & \leq 3 \\
z & \leq 7
\end{align*}
\]

**Graph**

\[
\begin{align*}
0 & \rightarrow x \\
& \rightarrow 2 \\
& \rightarrow 3 \\
& \rightarrow 4
\end{align*}
\]

**Shortest Path Closure**

\[
\begin{align*}
0 & \rightarrow 3 \\
& \rightarrow 6 \\
& \rightarrow 2
\end{align*}
\]
Canonical Datastructures for Zones

Difference Bounded Matrices

Emptiness

\[ \begin{align*}
D & \quad \begin{cases}
    x \leq 1 \\
y \geq 5 \\
y - x \leq 3
\end{cases} \\
\text{Graph} & \quad 0 \xrightarrow{-5} 3 \xrightarrow{1} x
\end{align*} \]

Compact

UCb

Negative Cycle
iff
empty solution set
Canonical Datastructures for Zones

Difference Bounded Matrices

Future

\( \begin{align*}
1 & \leq x \leq 4 \\
1 & \leq y \leq 3
\end{align*} \)

Future D

\( \begin{align*}
1 & \leq x, 1 \leq y \\
-2 & \leq x-y \leq 3
\end{align*} \)

Shortest Path Closure

Remove upper bounds on clocks

Canonical Datastructures for Zones
Canonical Datastructures for Zones

Difference Bounded Matrices

Reset

$1 \leq x, \ 1 \leq y$

$-2 \leq x - y \leq 3$

$y = 0, \ 1 \leq x$

Remove all bounds involving $y$ and set $y$ to 0
Canonical Datastructures for Zones

Difference Bounded Matrices

\[ x_1 - x_2 \leq 4 \]
\[ x_2 - x_1 \leq 10 \]
\[ x_3 - x_1 \leq 2 \]
\[ x_2 - x_3 \leq 2 \]
\[ x_0 - x_1 \leq 3 \]
\[ x_3 - x_0 \leq 5 \]

Shortest Path Closure \(O(n^3)\)
Canonical Datastructures for Zones

Minimal Constraint Form

Shortest Path Closure $O(n^3)$

Shortest Path Reduction $O(n^3)$

Space worst $O(n^2)$ practice $O(n)$

$x1-x2 <= 4$
$x2-x1 <= 10$
$x3-x1 <= 2$
$x2-x3 <= 2$
$x0-x1 <= 3$
$x3-x0 <= 5$
SPACE PERFORMANCE

- Minimal Constraint
- Global Reduction
- Combination

- Audio
- Audio w Col
- B&O
- Box Sorter
- M. Plant
- Fischer 2
- Fischer 3
- Fischer 4
- Fischer 5
- Train Crossing
TIME PERFORMANCE

- Minimal Constraint
- Global Reduction
- Combination

Bars represent different categories such as Audio, Audio w Col, B&O, Box Sorter, M. Plant, Fischer 2, Fischer 3, Fischer 4, Fischer 5, and Train Crossing.
Clock Difference Diagrams
Earlier Termination

\[ \text{Init} \rightarrow \text{Final} \]

\[
\begin{align*}
\text{INIT} := \emptyset; \\
\text{Waiting} := \{(n, Z)\} \\
\text{REPEAT} \\
\text{pick (n, Z) in Waiting} \\
\text{if (n, Z) = Final} \\
\text{return true} \\
\text{for all (n, Z) \rightarrow (n', Z')} \\
\text{if for some (n'', Z'')} Z' \subseteq Z''} \\
\text{continue} \\
\text{else} \\
\text{add (n', Z') to Waiting} \\
\text{move (n, Z) to Passed} \\
\text{UNTIL Waiting = \emptyset} \\
\text{return false}
\end{align*}
\]
Earlier Termination

Init -> Final?

\[
\begin{align*}
\text{INIT} &:= \emptyset; \\
\text{Waiting} &:= \{(n_0, Z_0)\} \\
\text{REPEAT} &
\begin{align*}
\text{pick} (n, Z) &\text{ in Waiting} \\
\text{if} (n, Z) &= \text{Final} \\
\text{return} \text{true} \\
\text{for all} (n, Z) &\rightarrow (n', Z'):
\begin{align*}
\text{if} &\text{ for some } (n'', Z'') \\
Z' &\subseteq Z'' \\
\text{continue} \\
\text{else} &\text{ add } (n', Z') \text{ to Waiting} \\
\text{move} (n, Z) &\text{ to Passed}
\end{align*}
\end{align*}
\text{UNTIL} \text{Waiting} = \emptyset \\
\text{return} \text{false}
\end{align*}
\]

\[Z' \subseteq Z''\]

\[Z' \subseteq \left[ Z_i \right]\]
Clock Difference Diagrams
= Binary Decision Diagrams + Difference Bounded Matrices

CDD-representations

- Nodes labeled with differences
- Maximal sharing of substructures (also across different CDDs)
- Maximal intervals
- Linear-time algorithms for set-theoretic operations.
Related & Future Work

- **DDD:** Andersen et al.
- **NDD:** Asarin, Bozga, Kerbrat, Maler, Pnueli, Rasse.
- **IDD:** Strehl, Thiele.

- No efficient algorithm for FUTURE and RESET operation on CDD.
- No canonical form.

- An efficient, fully symbolic engine for TA is still missing!!
Additional “secrets”

- Sharing among symbolic states
  - location vector / discrete values / zones
- Distributed implementation of UPPAAL
- Symmetry Reduction
- Sweep Line Method
- Guiding wrt Heuristic Value
  - User-supplied / Auto-generated
- Slicing wrt “C” Code
Open Problems

- Fully symbolic exploration of TA (both discrete and continuous part) ?
- Canonical form for CDD’s ?
- Partial Order Reduction ?
- Compositional Backwards Reachability ?
- Bounded Model Checking for TA ?
- Exploitation of multi-core processors ?
- ...

UCb
Datastructures for Zones

- Difference Bounded

**UPPAAL DBM Library**
The library used to manipulate DBMs in UPPAAL.

Welcome!

PDFs [dill89, rokid93, lowfct95, bengt02] are efficient data structures to represent clock constrains in timed automata [ad90]. They are used in UPPAAL [joy97, bde04] as the core data structure to represent time. The library features all the common operations such as update (delay, or future), down (past), general updates, different extrapolation functions, etc., on DBMs and federations. The library includes subtractions. The API is in C and C++. The C++ part uses active clocks and hides memory management.

References


**Elegant RUBY bindings for easy implementations**

[SPIN03]
Optimal & Real Time Scheduling using Model Checking Technology
Overview

- Timed Automata and Scheduling
- Priced Timed Automata
- Optimal Scheduling
- Optimal Scheduling wrt Multiple Objectives
- Optimal Infinite Scheduling
April 2002 – June 2005 IST-2001-35304

- Academic partners:
  - Nijmegen
  - Aalborg
  - Dortmund
  - Grenoble
  - Marseille
  - Twente
  - Weizmann

- Industrial Partners:
  - Axxom
  - Bosch
  - Cybernetix
  - Terma
AMETIST

advanced methods for timed systems

OBJECTIVES

- powerful, unifying mathematical modelling
- efficient computerized problem-solving tools
- distributed real-time systems
- time-dependent behaviour and dynamic resource allocation

TIMED AUTOMATA
**CPS: Informal description**

- CPS obtains and makes available for other systems information about environment of a car. This information may be used for:
  - Parking assistance
  - Etc
  - Based on Short Range Radar (SRR) technology

- The CPS considered in this case study:
  - One sensor group only (currently 2 sensors)
  - Only the front sensors and corresponding controllers
  - Application: pre-crash detection, parking assistance, stop & go

**Case Studies**

- Cybernetix:
  - Advanced Noise Reduction Techniques
  - Costal Surveillance

- Terma:
  - Memory Interface

- Bosch:
  - Car Periphery Sensing
  - Lacquer Production

- AXXOM:
  - Radar Video Processing Subsystem
  - Personalisation

**Benchmarks**

**Product flow of a Product**

- Storage
- Filling Stations
- Laboratory
- Mixing Vessel
- Dispersion
- Spinler

**CPS obtains and makes available for other systems information about environment of a car. This information may be used for:**

- Parking assistance
- Etc
- Based on Short Range Radar (SRR) technology
Real Time Scheduling

- Only 1 “Pass”
- Cheat is possible (drive close to car with “Pass”)

CAN THEY MAKE IT TO SAFE WITHIN 70 MINUTES ???

UNSAFE

Crossing Times

- 5
- 10
- 20
- 25

SAFE

UCb
Real Time Scheduling

Solve Scheduling Problem using UPPAAL

SAFE

UNSAFE

UCb
Resources & Tasks

Resource

Idle

use?

InUse

x:=0

x:=B

done!

Synchronization

Task

Init

use!

Using

done?

Shared variable

B:=6

Done
Task Graph Scheduling
Optimal Static Task Scheduling

- Task $P = \{P_1, \ldots, P_m\}$
- Machines $M = \{M_1, \ldots, M_n\}$
- Duration $\Delta : (P \in M) ! N_1$
- $< : \text{less on } P \text{ (grad.)}$

- A task can be executed only if all predecessors have completed

- Each machine can process at most one task at a time

- Task cannot be preempted.

Compute schedule with minimum completion-time!
Task Graph Scheduling
Optimal Static Task Scheduling

- Task \( P = \{P_1, \ldots, P_m\} \)
- Machines \( M = \{M_1, \ldots, M_n\} \)

\[ (P \leq M) \]

\[ \text{E<> (Task1.End and ... and Task7.End)} \]
## 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>

Abdeddaïm, Kerbaa, Maler
Optimality
Priced Timed Automata

with Paul Pettersson, Thomas Hune, Judi Romijn, Ansgar Fehnker, Ed Brinksma, Frits Vaandrager, Patricia Bouyer, Franck Cassez, Henning Dierks Emmanuel Fleury, Jacob Rasmussen,..
EXAMPLE: Optimal rescue plan for cars with different subscription rates for city driving!

OPTIMAL PLAN HAS ACCUMULATED COST = 195 and TOTAL TIME = 65!
# Experiments

<table>
<thead>
<tr>
<th>COST-rates</th>
<th>SCHEDULE</th>
<th>COST</th>
<th>TIME</th>
<th>#Expl</th>
<th>#Pop’d</th>
</tr>
</thead>
<tbody>
<tr>
<td>G C B D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min Time</td>
<td>CG&gt; G&lt; BD&gt; C&lt; CG&gt;</td>
<td>60</td>
<td>1762</td>
<td>1538</td>
<td>2638</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>CG&gt; G&lt; BG&gt; G&lt; GD&gt;</td>
<td>55</td>
<td>65</td>
<td>252</td>
<td>378</td>
</tr>
<tr>
<td>9 2 3 10</td>
<td>GD&gt; G&lt; CG&gt; G&lt; BG&gt;</td>
<td>195</td>
<td>65</td>
<td>149</td>
<td>233</td>
</tr>
<tr>
<td>1 2 3 4</td>
<td>CG&gt; G&lt; BD&gt; C&lt; CG&gt;</td>
<td>140</td>
<td>60</td>
<td>232</td>
<td>350</td>
</tr>
<tr>
<td>1 2 3 10</td>
<td>CD&gt; C&lt; CB&gt; C&lt; CG&gt;</td>
<td>170</td>
<td>65</td>
<td>263</td>
<td>408</td>
</tr>
<tr>
<td>1 20 30 40</td>
<td>BD&gt; B&lt; CB&gt; C&lt; CG&gt;</td>
<td>975</td>
<td>85</td>
<td>time&lt;85</td>
<td>-</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td></td>
<td>0</td>
<td>-</td>
<td>406</td>
<td>447</td>
</tr>
</tbody>
</table>
Priced Timed Automata
Timed Automata + \textbf{COST} variable

Behrmann, Fehnker, et all (HSCC’01)
Alur, Torre, Pappas (HSCC’01)
Priced Timed Automata
Timed Automata + **COST** variable

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

TRACES

\[(I_1, x=y=0) \xrightarrow{\varepsilon(3)} (I_1, x=y=3) \xrightarrow{1} (I_2, x=0, y=3) \xrightarrow{4} (I_3, \_\_\_\_\_)\]

\[\sum c = 17\]
Priced Timed Automata
Timed Automata + **COST** variable

**Problem:**
Find the minimum cost of reaching location \( l_3 \)

Efficient Implementation:
CAV’01 and TACAS’04

\[
\begin{align*}
(l_1, x=y=0) &\xrightarrow{1} (l_2, x=0, y=0) \\
(l_2, x=0, y=0) &\xrightarrow{\epsilon(3)} (l_2, x=3, y=3) \\
(l_2, x=3, y=3) &\xrightarrow{0} (l_2, x=0, y=3) \\
(l_2, x=0, y=3) &\xrightarrow{4} (l_3, _, _) \\
\sum c &= 16 \\
\sum c &= 11
\end{align*}
\]
Optimal Task Graph Scheduling

Power-Optimality

- **Energy-rates:**
  - $C : M ! N$
- Compute schedule with minimum completion-cost!
Aircraft Landing Problem

Planes have to keep separation distance to avoid turbulences caused by preceding planes.

- $E$: earliest landing time
- $T$: target time
- $L$: latest time
- $e$: cost rate for being early
- $l$: cost rate for being late
- $d$: fixed cost for being late

The diagram illustrates the cost function as a function of time $t$:

- The cost for being early is given by $e^*(T-t)$.
- The cost for being late is given by $d + l^*(t-T)$.

The cost function increases linearly for times later than the target time $T$. The cost for being late increases with the delay from the target time $T$.
Modeling ALP with PTA

Planes have to keep separation distance to avoid turbulences caused by preceding planes.

Runway handles 2 types of planes:

- **Earliest landing time**: 129
- **Target landing time**: 153
- **Latest landing time**: 559
- **Cost rate for early**: 10
- **Cost rate for late**: 20
## Aircraft Landing

Source of examples: Baesley et al’2000

<table>
<thead>
<tr>
<th>problem instance</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of planes</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>44</td>
</tr>
<tr>
<td>number of types</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.36</th>
<th>0.70</th>
<th>1.71</th>
<th>14786.19</th>
<th>12461.47</th>
<th>0.68</th>
</tr>
</thead>
<tbody>
<tr>
<td>cputime (secs)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Results Summary

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal value</td>
<td>700</td>
<td>1480</td>
<td>820</td>
<td>2520</td>
</tr>
<tr>
<td>explored states</td>
<td>481</td>
<td>2149</td>
<td>920</td>
<td>5693</td>
</tr>
<tr>
<td>cputime (secs)</td>
<td>4.19</td>
<td>25.30</td>
<td>11.05</td>
<td>87.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>90</th>
<th>210</th>
<th>60</th>
<th>640</th>
<th>650</th>
<th>554</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>explored states</td>
<td>1218</td>
<td>1797</td>
<td>669</td>
<td>28821</td>
<td>47993</td>
<td>9035</td>
<td>92</td>
</tr>
<tr>
<td>cputime (secs)</td>
<td>17.87</td>
<td>39.92</td>
<td>11.02</td>
<td>755.84</td>
<td>1085.08</td>
<td>123.72</td>
<td>1.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>130</th>
<th>170</th>
<th>0</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>explored states</td>
<td>24</td>
<td>46</td>
<td>84</td>
<td>207715</td>
<td>189602</td>
<td>62</td>
<td>N/A</td>
</tr>
<tr>
<td>cputime (secs)</td>
<td>0.36</td>
<td>0.70</td>
<td>1.71</td>
<td>14786.19</td>
<td>12461.47</td>
<td>0.68</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N/A</th>
<th>N/A</th>
<th>N/A</th>
<th>65</th>
<th>64</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>cputime (secs)</td>
<td></td>
<td></td>
<td></td>
<td>1.97</td>
<td>1.53</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Symbolic “A*”
Zones

Definition
A zone $Z$ over a set of clocks $C$ is a finite conjunction of simple constraints of the forms:

$$x \geq l \quad x \leq u \quad x - y \geq l' \quad x - y \leq u'$$

where $x, y \in C$, $l, u \in \mathbb{N}$ and $l', u' \in \mathbb{Z}$.

For $u \in \mathbb{R}^C$ and $Z$ a zone we write $u \models Z$ if $u$ satisfies all constraints of $Z$.

Operations

Reset: $\{x\}Z = \{u[0/x] \mid u \models Z\}$

Delay: $Z^\uparrow = \{u + d \mid u \models Z\}$

Offset: $\Delta_Z \models Z$ such that $\forall u \models Z. \forall x \in C. \Delta_Z(x) \leq u(x)$. 

UCb
Priced Zone

Definition
A priced zone $P$ is a tuple $(Z,c,r)$, where:

- $Z$ is a zone
- $c \in \mathbb{N}$ describes the cost of $\Delta Z$
- $r : C \rightarrow \mathbb{Z}$ gives a rate for any clock $x \in C$.

We write $u \models P$ whenever $u \models Z$. For $u \models P$ we define $\text{Cost}(u, P)$ as follows:

\[
\text{Cost}(u, P) = c + \sum_{x \in C} r(x) \cdot (u(x) - \Delta_Z(x))
\]

Cost($x, y$) = $2y - x + 2$
Branch & Bound Algorithm

Cost := ∞
Passed := ∅
Waiting := { (l₀, Z₀) }
while Waiting ≠ ∅ do
    select (l, Z) from Waiting
    if l = lₙ and minCost(Z) < Cost then
        Cost := minCost(Z)
    if minCost(Z) + Rem(l, Z) ≥ Cost then break
    if for all (l', Z') in Passed: Z' ≠ Z then
        add (l, Z) to Passed
        add all (l', Z') with (l, Z) → (l', Z') to Waiting
    return Cost
Optimization
with Multi Objectives

with Jacob I. Rasmussen
EXAMPLE: Conditional

UNSAFE

Golf

Citroen

My CAR!

process Torch

free2

take?

free1

L = -L + 1

release?

L = 0

take !

y ≥ 5

release!

L = 1

take !

y ≥ 5

ready

take !

y = 0

safe

over

ready

u over

u ready

Minimizes Cost_{MYCAR}

subject to

Cost_{Citroen} · 60

Cost_{BMW} · 90

Cost_{Datsun} · 10

\min Cost_{MYCAR} = 270

time = 70

Optimal rescue plan for cars with different subscription rates for city driving!
Multiple Objective Scheduling

\[ P_1 \rightarrow P_2, P_3, P_4 \]

\[ P_2 \rightarrow P_5, P_7 \]

\[ P_3 \rightarrow P_4 \]

\[ P_4 \rightarrow P_5, P_7 \]

\[ P_5 \rightarrow P_7 \]

Costs:
- \( \text{cost}_1 \leq 4 \)
- \( \text{cost}_2 \leq 3 \)

UCb
Multiple Objective Scheduling

The Pareto Frontier for Reachability in Multi Priced Timed Automata is computable
[Illum, Larsen FoSSaCS05]
Optimal Infinite Scheduling

with Ed Brinksma
Patricia Bouyer
Arne Skou
Ulrich Fahrenberg
EXAMPLE: Optimal WORK plan for cars with different subscription rates for city driving!

maximal 100 min. at each location
Workplan I

Value of workplan:

\[
\frac{(4 \times 300)}{90} = 13.33
\]
Workplan II

Value of workplan: $\frac{560}{100} = 5.6$
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
Optimal Infinite Scheduling

Maximize throughput: i.e. maximize Reward / Cost in the long run
Optimal Infinite Scheduling I
Limit Ratio (Pay-off)

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

\[ \text{value of path } \sigma : \text{val}(\sigma) = \lim_{n \rightarrow 1} \frac{\text{ac}_n}{t_n} \]

\[ \text{THEOREM: } \sigma^* \text{ is computable} \]

Bouyer, Brinksma, Larsen HSCC’04, FMSD’08
Optimal Infinite Scheduling I

Discounting

\[ \lambda < 1 : \text{discounting factor} \]

\[ \text{Value of path } \sigma_0 : \quad \text{val}(\sigma_0) = \text{Optimal Schedule } \sigma^* : \quad \text{val}(\sigma^*) = \text{inf}_\sigma \text{ val}(\sigma) \]

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

Vol \(\geq\) Min and Vol \(\leq\) Max

\[ \text{Accumulated Time} \]

\[ \text{THEOREM: } \sigma^* \text{ is computable} \]

\[ \text{Fahrenberg, Larsen: INFINITY'08} \]

Path \(\sigma: \quad \text{val}(\sigma) = \sum_{i=1}^{\infty} c_i \cdot \lambda^{t_i} \]

Optimal Schedule \(\sigma^*: \quad \text{val}(\sigma^*) = \text{inf}_\sigma \text{ val}(\sigma) \]
Application

Dynamic Voltage Scaling
Future Work & Challenges

😊 Optimal Constrained Infinite Strategies for Multi-Priced TA

😊 / 😞 Model Checking wrt PTCTL

[ Raskin et al, FORMATS’04 ]
[ Bouyer, Larsen, Markey FoSSACS07 ]

😞 Efficient implementations of Optimal Infinite Scheduling

😊 Negative Cost Rates → Cost Bounded Infinite Runs

[ Bouyer, Fahrenberg, Larsen, Markey, Srba FORMATS08 ]
Further Information

Welcome to the UPPAAL home page. UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

The tool is developed in collaboration between the Design and Analysis of Real-Time Systems group at Uppsala University, Sweden and Basic Research in Computer Science at Aalborg University, Denmark.

Download: UPPAAL 3.2.11 (released September 6, 2002) is available from the download page. More information about the 3.2 version can be found in the press release. In addition to the 3.2 version, a snapshot of version 3.3.23 is also available from the download page.

License: The UPPAAL tool is free for non-profit applications but we have a license agreement that all users must fill in before downloading and using the tool. To find out more about UPPAAL, read this short introduction. Further information may be found at this web site in the pages About, Documentation, Download and Examples.

Mailing lists: UPPAAL has an open mailing list intended for users of the tool. To join the list, email uppaal-subscribe@yahoogroups.com, to post use uppaal@yahoogroups.com. For more information see the group page at Yahoo Groups. To email the development team directly, please use uppaal@cs.uu.se for bug reports and contributions, uulf@cs.uu.se otherwise.

www.uppaal.com
Real Time Controller Synthesis

with

Gerd Behrmann, Patricia Bouyer,
Franck Cassez, Agnes Counard, Alexandre David
Emmanuel Fleury, Didier Lime, Nicolas Markey,
Jean-Francois Raskin
Welcome!

UPPAAL TIGA (Fig. 1) is an extension of UPPAAL [BBL05] and it implements the first efficient on-the-fly algorithm for solving games based on timed game automata with respect to reachability and safety properties. Though timed games for long have been known to be decidable there has until now been a lack of efficient and truly on-the-fly algorithms for their analysis.

The algorithm we propose [CDFL05] is a symbolic extension of the on-the-fly algorithm suggested by Liu & Smolka [LS98] for linear-time model-checking of finite-state systems. Being on-the-fly, the symbolic algorithm may terminate long before having explored the entire state-space. Also the individual steps of the algorithm are carried out efficiently by the use of so-called zones as the underlying data structure. Our tool implements various optimizations of the basic symbolic algorithm, as well as methods for obtaining time-optimal winning strategies.

Latest News

Versions 0.10 and 0.11 released.
7 July 2007

Versions 0.10 and 0.11 are released today. Version 0.11 contains a new concrete simulator that allows the user to play strategies from the GUI. Both versions fix the following bugs: maximal constants in the formula are now taken into account, the command line simulator is new and works better, delay when no clock was used, better user feedback, end-of-game detection fixed, other bugs involving delays in the strategy, precision problems in the simulator, and leak in the DBM library. These new versions have also the following new features: options to control the type of strategy output, better control on the search ordering (forward and backward), cooperative strategies, and time optimal strategies. The manual has been updated to
Model Checking

Plant
Continuous

Controller Program
Discrete

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

Model of tasks (automatic?)

UPPAAL Model

UCb
Synthesis

Plant
Continuous

Controller Program
Discrete

Model of environment (user-supplied)

Partial UPPAAL Model

Synthesis of tasks/scheduler (automatic)

SAT $\varphi$ !!

UCb
Model Checking

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

ϕ: Never two trains at the crossing at the same time
Synthesis

Two Player Game

Find strategy for controllable actions st behaviour satisfies \( \varphi \):

Never two trains at the crossing at the same time
Untimed and Timed Games

Reachability / Safety Games

Uncontrollable
Controllable
Untimed Games

Memoryless Strategy:
\[ F : Q \rightarrow E_c \]
Untimed Games

Memoryless Strategy:
\[ F : Q \rightarrow E_c \]

Winning Run \( \rho \): 
\[ \text{States}(\rho) \not\subseteq G \neq \emptyset \]

Winning Strategy:
\[ \text{Runs}(F) \mu \text{WinRuns} \]
Untimed Games

Memoryless Strategy:
\[ F : Q \rightarrow E_c \]

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

Winning Strategy:
\[ \text{Runs}(F) \mu \text{WinRuns} \]
Untimed Games

Backwards Fixed-Point Computation

\[
c\text{Pred}(X) = \{ q \in Q \mid \exists q' \in X. q \xrightarrow{c} q' \}
\]
\[
u\text{Pred}(X) = \{ q \in Q \mid \exists q' \in X. q \xrightarrow{u} q' \}
\]
\[
\pi(X) = c\text{Pred}(X) \setminus u\text{Pred}(X^c)
\]

Theorem:
The set of winning states is obtained as the least fixpoint of the function:
\[
X \mapsto \pi(X) \ [ \text{Goal}]
\]
Untimed Games

Backwards Fixed-Point Computation

\[ c_{\text{Pred}}(X) = \{ q \in Q | \exists q' \in X. q \xrightarrow{c} q' \} \]
\[ u_{\text{Pred}}(X) = \{ q \in Q | \exists q' \in X. q \xrightarrow{u} q' \} \]

\[ \pi(X) = c_{\text{Pred}}(X) \setminus u_{\text{Pred}}(X^c) \]

Theorem:
The set of winning states is obtained as the least fixpoint of the function:
\[ X \mapsto \pi(X) \] [Goal]
Untimed Games

Backwards Fixed-Point Computation

The set of winning states is obtained as the least fixpoint of the function:

\[ X \mapsto \pi(X) \] [ Goal

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

\[ X \mapsto \pi(X) \] [ Goal

\[
\pi(X) = c_{\text{Pred}}(X) \setminus u_{\text{Pred}}(X^c)
\]

\[
c_{\text{Pred}}(X) = \{ q \in Q \mid \exists q' X. q \xrightarrow{c} q' \}
\]

\[
u_{\text{Pred}}(X) = \{ q \in Q \mid \exists q' X. q \xrightarrow{u} q' \}
\]
Untimed Games

Backwards Fixed-Point Computation

\[ c_{\text{Pred}}(X) = \{ q \in Q \mid \exists q' : X. q \xrightarrow{c} q' \} \]
\[ u_{\text{Pred}}(X) = \{ q \in Q \mid \exists q' : X. q \xrightarrow{u} q' \} \]

\[ \pi(X) = c_{\text{Pred}}(X) \setminus u_{\text{Pred}}(X^c) \]

**Theorem:**
The set of winning states is obtained as the least fixpoint of the function:
\[ X \mapsto \pi(X) \text{ [ Goal] } \]
Untimed Games

Backwards Fixed-Point Computation

\[ \text{cPred}(X) = \{ q2Q \mid \exists q' \text{ s.t. } X \cdot q \xrightarrow{c} q' \} \]
\[ \text{uPred}(X) = \{ q2Q \mid \exists q' \text{ s.t. } X \cdot q \xrightarrow{u} q' \} \]

\[ \pi(X) = \text{cPred}(X) \setminus \text{uPred}(X^c) \]

**Theorem:**
The set of winning states is obtained as the least fixpoint of the function:
\[ X \mapsto \pi(X) \ [ \text{Goal} \]
### Untimed Games

**Backwards Fixed-Point Computation**

\[
\begin{align*}
\text{cPred}(X) & = \{ q2Q \mid \exists q' \exists X. q \xrightarrow{c} q' \} \\
\text{uPred}(X) & = \{ q2Q \mid \exists q' \exists X. q \xrightarrow{u} q' \}
\end{align*}
\]

\[
\pi(X) = \text{cPred}(X) \setminus \text{uPred}(X^C)
\]

**Theorem:**
The set of winning states is obtained as the least fixpoint of the function:

\[
X \mapsto \pi(X) \ [ \text{Goal}]
\]

- **Uncontrollable**
- **Controllable**
Timed Games

Memoryless Strategy:
\[ F : Q \rightarrow E_c [ \lambda ] \]

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

Winning Strategy:
\[ \text{Runs}(F) \cup \text{WinRuns} \]
Timed Games

Memoryless Strategy:
$$F : Q \rightarrow E_c[\lambda]$$

Winning Run:
States($\rho$) $\not= \emptyset$

Winning Strategy:
Runs(F) $\mu$ WinRuns

Winning (memoryless) strategy

UCb
Timed Games – State-of-the-Art

Definitions

\[ c\text{Pred}(X) = \{ q2Q | \exists q'2X. q \xrightarrow{c} q' \} \]
\[ u\text{Pred}(X) = \{ q2Q | \exists q'2X. q \xrightarrow{u} q' \} \]
\[ \text{Pred}_t(X,Y) = \{ q2Q | \exists t. q^t2X \text{ and } \forall s. t. q^s2Y^c \} \]

\[ \pi(X) = \text{Pred}_t[X[c\text{Pred}(X), u\text{Pred}(X^c)] \]

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

\[ X \mapsto \pi(X) \] [Goal]
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

\[
x \cdot 1 \quad x > 1
\]

\[
x \cdot 2 \quad x < 1
\]

\[
x \cdot 1 \quad x = 0
\]
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

UCb
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

\[ x > 1 \Rightarrow x \cdot 1 \]

\[ x < 1 \Rightarrow x \cdot 2 \]

\[ x = 0 \Rightarrow x \cdot 1 \]
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

0 1 2
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

\[
\begin{align*}
\text{Backwards Fixed-Point Computation} & \\
x > 1 & \Rightarrow x \cdot 1 \\
x \cdot 2 & \Rightarrow x < 1 \\
x < 1 & \Rightarrow x \cdot 1 \\
x : = 0 & \Rightarrow x \cdot 1
\end{align*}
\]
Timed Games – State-of-the-Art

Backwards Fixed-Point Computation

UPPAAL Tiga

= On-the-fly algorithm for timed games

[CONCUR’05]
UPPAAL Tiga

CAV 2007
UPPAAL Tiga

CTL Control Objectives

- **Reachability properties:**
  - control: $A[ p \ U \ q ]$
  - control: $A<> q$ $\iff$ control: $A[ true \ U \ q ]$

- **Safety properties:**
  - control: $A[ p \ W \ q ]$ weak until
  - control: $A[] p$ $\iff$ control: $A[ p \ W \ false ]$

- **Time-optimality:**
  - control$_t^*(u,g)$: $A[ p \ U \ q ]$
    - $u$ is an upper-bound to prune the search
    - $g$ is the time to the goal from the current state
Train Crossing

\[ \phi : \text{Never two trains at the crossing at the same time} \]
A Buggy Brick Sorting Program

Exercise:
Design Controller so that only yellow boxes are being pushed out.

Boxes
Piston
Black Yellow

Controller

MAIN
PUSH
Conveyor Belt
eject

TaskMAIN

red?
black?
x=0, active=true

TaskPUSH

x==ctime
eject!
active==0

wait
x<=ctime
active==1
go?
Brick Sorting

Controller

Piston

Generic Plate

Brick Sorting

Strategy for EJECT

Controller

Informationsteknologi
## Production Cell

<table>
<thead>
<tr>
<th>Plates</th>
<th>Basic</th>
<th>Basic + inc + pruning</th>
<th>Basic + lose + inc + pruning</th>
<th>Basic + lose + inc + topt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>win</td>
<td>time</td>
<td>mem</td>
<td>time</td>
</tr>
<tr>
<td>2</td>
<td>lose</td>
<td>0.0s</td>
<td>1M</td>
<td>0.0s</td>
</tr>
<tr>
<td>3</td>
<td>win</td>
<td>0.5s</td>
<td>1M</td>
<td>0.1s</td>
</tr>
<tr>
<td>3</td>
<td>lose</td>
<td>1.1s</td>
<td>45M</td>
<td>0.2s</td>
</tr>
<tr>
<td>4</td>
<td>win</td>
<td>33.9s</td>
<td>1395M</td>
<td>0.1s</td>
</tr>
<tr>
<td>4</td>
<td>lose</td>
<td>-</td>
<td>-</td>
<td>0.4s</td>
</tr>
<tr>
<td>5</td>
<td>win</td>
<td>11.1s</td>
<td>61M</td>
<td>1.5s</td>
</tr>
<tr>
<td>5</td>
<td>lose</td>
<td>-</td>
<td>-</td>
<td>5.9s</td>
</tr>
<tr>
<td>6</td>
<td>win</td>
<td>699s</td>
<td>480M</td>
<td>3.8s</td>
</tr>
<tr>
<td>6</td>
<td>lose</td>
<td>-</td>
<td>-</td>
<td>317s</td>
</tr>
<tr>
<td>7</td>
<td>win</td>
<td>3256s</td>
<td>1183M</td>
<td>1181s</td>
</tr>
<tr>
<td>7</td>
<td>lose</td>
<td>-</td>
<td>-</td>
<td>16791s</td>
</tr>
</tbody>
</table>

### Model

<table>
<thead>
<tr>
<th>Plate</th>
<th>3</th>
<th>6</th>
<th>12</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old</td>
<td>c</td>
<td>0.1s</td>
<td>1M</td>
<td>12s</td>
<td>63M</td>
</tr>
<tr>
<td></td>
<td>u</td>
<td>0.2s</td>
<td>3M</td>
<td>235s</td>
<td>273M</td>
</tr>
<tr>
<td>New</td>
<td>c</td>
<td>0.05s</td>
<td>3.5M</td>
<td>0.05s</td>
<td>3.5M</td>
</tr>
<tr>
<td></td>
<td>u</td>
<td>0.02s</td>
<td>5M</td>
<td>0.04s</td>
<td>3.5M</td>
</tr>
</tbody>
</table>

UCb
Climate Control

Syvsten, Northern Jutland, DK

With Jan J. Jessen
Jacob I. Rasmussen
Obtaining executable code

Strategy for state:
Zone i-1: (Temp. lower/equal, wants flow)
Zone i+1: (Temp. lower/equal, no interaction)
Hottest neighbor: i-1
Objective: heat
is:
Wants flow from i-1
Wants flow from i+1
inlet closed
outlet off
heater on

Strategy for state:
Zone i-1: (Temp. greater, offers flow)
Zone i+1: (Temp. greater, offers flow)
Hottest neighbor: i+1
Objective: cool
is:
Has flow for i-1
Has flow for i+1
inlet open
outlet on
heater off

Strategy for state:
Zone i-1: (Temp. lower/equal, no interaction)
Zone i+1: (Temp. greater, no interaction)
Hottest neighbor: i+1
Objective: cool

control : A[]
((ZC.Init && objective) imply temp_derivative > 0) &&
((ZC.Init && !objective) imply temp_derivative < 0)
Obtaining executable code
Open Problems

- **Priced Timed Games**
  - Reachability & Model Checking
  - Safety
  - Negative cost rates
- **Probabilistic Priced Timed Automata**
- **Timed Automata**
  - Fully Symbolic
  - Static Analysis & Slicing of C-code
- **Timed Games**
  - Alternating Logics
  - Partial Observability
  - CEGAR

- **APPLICATIONS**
- Live Sequence Charts
- Gantt Chart
- Code Generation
  - From TA models of controllers
  - From strategies

Thanks for your attention!

kgl@cs.aau.dk
www.cs.aau.dk