Soundness of the Quasi-Synchronous Abstraction

Guillaume Baudart  Timothy Bourke  Marc Pouzet

École normale supérieure, INRIA Paris, UPMC

FMCAD’16

Mountain View, 06-10-2016
**Example:** Flight Control System
Generate pitch and roll guidance commands
Distributed Embedded Systems

Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Example: Flight Control System
Generate pitch and roll guidance commands
Distributed Embedded Systems
Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Crew can switch from one to the other

Example: Flight Control System
Generate pitch and roll guidance commands

Example from [Miller et al. 2015]
Two redundant Flight Guidance Systems
Only one active side (pilot side)

Crew can switch from one to the other

Example: Flight Control System
Generate pitch and roll guidance commands
Distributed Embedded Systems
Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Example: Flight Control System
Generate pitch and roll guidance commands

The two modules must share their state to avoid control glitch

Example from [Miller et al. 2015]
Distributed Embedded Systems
Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Example: Flight Control System
Generate pitch and roll guidance commands

Run embedded application...

Crew can switch from one to the other

The two modules must share their state to avoid control glitch

Example from [Miller et al. 2015]
Distributed Embedded Systems
Distributed controllers for critical embedded systems

Two redundant Flight Guidance Systems
Only one active side (pilot side)

Run embedded application...
...on distributed architectures

Crew can switch from one to the other

Example: Flight Control System
Generate pitch and roll guidance commands

The two modules must share their state to avoid control glitch

Example from [Miller et al. 2015]
Synchronous Real-Time Model

For each process, activations are triggered by a **local clock**.

**Execution**: infinite sequence of activations

- For each process: known bounds for the time between two activations.
  \[ 0 \leq T_{\text{min}} \leq \kappa_i - \kappa_{i-1} \leq T_{\text{max}} \]
  \[(\kappa_i)_{i \in \mathbb{N}} \text{ clock activations} \]

- Buffered communication without message inversion or loss

- Bounded communication delay
  \[ 0 \leq \tau_{\text{min}} \leq \tau \leq \tau_{\text{max}} \]
Synchronous Real-Time Model

For each process, activations are triggered by a local clock

**Execution:** infinite sequence of activations

- For each process: known bounds for the time between two activations.

\[ 0 \leq T_{\text{min}} \leq \kappa_i - \kappa_{i-1} \leq T_{\text{max}} \]

\[(\kappa_i)_{i \in \mathbb{N}}\] clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay

\[ 0 \leq \tau_{\text{min}} \leq \tau \leq \tau_{\text{max}} \]
For each process: known bounds for the time between two activations.

\[ 0 \leq T_{\min} \leq \kappa_i - \kappa_{i-1} \leq T_{\max} \]

\((\kappa_i)_{i \in \mathbb{N}}\) clock activations

• Buffered communication without message inversion or loss

• Bounded communication delay

\[ 0 \leq \tau_{\min} \leq \tau \leq \tau_{\max} \]
Synchronous Real-Time Model

For each process, activations are triggered by a local clock

Execution: infinite sequence of activations

- For each process: known bounds for the time between two activations.
  \[
  0 \leq T_{\min} \leq \kappa_i - \kappa_{i-1} \leq T_{\max}
  \]
  \[(\kappa_i)_{i \in \mathbb{N}} \text{ clock activations}\]

- Buffered communication without message inversion or loss

- Bounded communication delay
  \[
  0 \leq \tau_{\min} \leq \tau \leq \tau_{\max}
  \]
Synchronous Real-Time Model

For each process, activations are triggered by a local clock

Execution: infinite sequence of activations

• For each process: known bounds for the time between two activations.
  
  \[ 0 \leq T_{\text{min}} \leq \kappa_i - \kappa_{i-1} \leq T_{\text{max}} \]

  \((\kappa_i)_{i \in \mathbb{N}}\) clock activations

• Buffered communication without message inversion or loss

• Bounded communication delay

  \[ 0 \leq \tau_{\text{min}} \leq \tau \leq \tau_{\text{max}} \]
Overview

Industrial practices observed at Airbus

[Caspi 2000]
Overview

Verification

Verifying safety critical applications running on quasi-periodic architectures

Quasi-Synchronous Abstraction

Industrial practices observed at Airbus

[Caspi 2000]
Overview

Verification

Verifying safety critical applications running on quasi-periodic architectures

Quasi-Synchronous Abstraction

Industrial practices observed at Airbus

[Caspi 2000]
Overview

Verification

Verifying safety critical applications running on quasi-periodic architectures

Quasi-Synchronous Abstraction

Contributions

Abstraction is not sound in general

Give exact conditions of application

Industrial practices observed at Airbus

[Caspi 2000]
The Big Picture

Discrete-time Model (DT)

\[ 0 < T_{\text{min}} \leq T_A, T_B \leq T_{\text{max}} \]
\[ 0 < \tau_{\text{min}} \leq \tau_A, \tau_B \leq \tau_{\text{max}} \]

Scheduler

Real-time Model (RT)
The Big Picture

Discrete-time Model (DT)

Real-time Model (RT)

\[ 0 < T_{\text{min}} \leq T_A, T_B \leq T_{\text{max}} \]
\[ 0 < \tau_{\text{min}} \leq \tau_A, \tau_B \leq \tau_{\text{max}} \]
The Big Picture

Discrete-time Model (DT)

\[ 0 < T_{\text{min}} \leq T_A, T_B \leq T_{\text{max}} \]
\[ 0 < \tau_{\text{min}} \leq \tau_A, \tau_B \leq \tau_{\text{max}} \]

Real-time Model (RT)

\[ RT \models \varphi \quad \overset{\text{Soundness}}{\iff} \quad DT \models \varphi \]
The Big Picture

Real-time Model (RT)  Discrete-time Model (DT)

\[ RT \models \varphi \quad \text{Soundness} \quad DT \models \varphi \]

Why discretize?
Verification in a simpler discrete-time model
Use discrete-time model checking tools (Lesar-Verimag, Kind2-Ulowa)

[Halbwachs et al 1992]
[Hagen, Tinelli 2008]
Abstracting Real Time
Abstracting execution time
Abstracting Real Time

Abstracting execution time

\[ \tau_{\text{exec}} \]

\[ \tau_{\text{send}} \]
Abstracting Real Time

Abstracting execution time

\[ \tau = \tau_{\text{exec}} + \tau_{\text{send}} \]
Abstracting Real Time

Abstracting execution time
Abstracting Real Time

Abstracting execution time
Abstracting Real Time

Abstracting execution time
Abstracting communication
Abstracting Real Time

Abstracting execution time
Abstracting communication
Abstracting Real Time

Abstracting execution time
Abstracting communication
Abstracting Real Time

Abstracting execution time
Abstracting communication

Problems:
• Lots of possible interleavings
• Too general
Abstracting Real Time

Abstracting execution time
Abstracting communication

Problems:
• Lots of possible interleavings
• Too general

Can we do better using real-time assumptions?
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays (one step of the logical clock)
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays (one step of the logical clock)
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays
   (one step of the logical clock)
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays (one step of the logical clock)

Replace transmission with precedence
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Reduce the state-space in two ways:

1. Transmissions as unit delays (one step of the logical clock)

2. Limit activations interleavings
   A process is at most twice as fast as another

Replace transmission with precedence
The Quasi-Synchronous Abstraction

Focus on 'almost' synchronous architectures with fast transmissions

Is this abstraction sound?

Reduce the state-space in two ways:

1. Transmissions as unit delays (one step of the logical clock)

2. Limit activations interleavings
   A process is at most twice as fast as another

Replace transmission with precedence
Unitary Discretization

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

Always possible if transmissions are not instantaneous.
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

> Always possible if transmissions are not instantaneous
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

Always possible if transmissions are not instantaneous.
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

*Always possible if transmissions are not instantaneous*
Unitary Discretization

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

> Always possible if transmissions are not instantaneous

[Diagram showing unitary discretization with τ_max delays and a cross indicating non-discretizability.]
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

![Diagram showing the concept of unitary discretization]

Always possible if transmissions are not instantaneous.
**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.
**Unitary Discretization**

**Definition:** A trace is unitary discretizable if there exist a discretization where transmission can be modeled as unit-delays.

**Theorem:** A real-time model with more than two processes is, in general, not unitary discretizable.

Some traces are not captured by the discrete abstraction.

Always possible if transmissions are not instantaneous.
Trace Graph

Gather all constraints on the unitary discretization $f$ in a weighted graph

After reception

$x \xrightarrow{1} y \implies f(x) < f(y)$

Before reception

$x \xrightarrow{0} y \implies f(x) \leq f(y)$
Gather all contraints on the unitary discretization $f$ in a weighted graph

**Lemma:** A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

**Definition:** A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable*. 

\[
\begin{align*}
x \xrightarrow{1} y & \implies f(x) < f(y) \\
x \xrightarrow{0} y & \implies f(x) \leq f(y)
\end{align*}
\]
Gather all constraints on the unitary discretization $f$ in a weighted graph.

**Lemma:** A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

**Definition:** A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable.*
Trace Graph

Gather all constraints on the unitary discretization $f$ in a weighted graph.

**After reception**

$x \xrightarrow{1} y \implies f(x) < f(y)$

**Before reception**

$x \xleftarrow{0} y \implies f(x) \leq f(y)$

**Lemma:** A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

**Definition:** A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable*.
Gather all constraints on the unitary discretization $f$ in a weighted graph.

**After reception**

$x \xrightarrow{1} y \implies f(x) < f(y)$

**Before reception**

$x \xrightarrow{0} y \implies f(x) \leq f(y)$

**Lemma:** A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

**Definition:** A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable*. 
Trace Graph

Gather all constraints on the unitary discretization $f$ in a weighted graph

After reception

$x \xrightarrow{1} y \implies f(x) < f(y)$

Before reception

$x \xrightarrow{0} y \implies f(x) \leq f(y)$

**Lemma:** A trace is *unitary discretizable* if and only if there is no cycle of positive weight in the associated trace graph.

**Definition:** A real-time model is *unitary discretizable* if all possible traces are *unitary discretizable*. 
Recovering Soundness

Forbidden topologies in the static communication graph

cycle

u-cycle

balanced u-cycle
Recovering Soundness

Forbidden topologies in the static communication graph

\begin{itemize}
  \item \textit{cycle} \hspace{1cm} \textit{u-cycle} \hspace{1cm} \textit{balanced u-cycle}
\end{itemize}
Recovering Soundness

Forbidden topologies in the static communication graph

- cycle
- u-cycle
- balanced u-cycle

can be allowed at the cost of additional timing constraints
Theorem: A quasi-periodic architecture is unitary discretizable if and only if, in the communication graph

1. All u-cycles are cycles of balanced u-cycle, or $\tau_{\text{max}} = 0$, and
2. There is no balanced u-cycle, or $\tau_{\text{min}} = \tau_{\text{max}}$, and
3. There is no cycle in the communication graph, or $T_{\text{min}} \geq L_c \tau_{\text{max}}$

$L_c$: size of the longest elementary cycle
Recovering Soundness

Proof: If there is a $u$-cycle, construction of a counter-example
Recovering Soundness

**Proof:** If there is a $u$-cycle, construction of a counter-example

$q = 3$: # ←→
$p = 2$: # →→
Recovering Soundness

Proof: If there is a $u$-cycle, construction of a counter-example

$q > p \implies \varepsilon = (q \tau_{\text{max}} - p \tau_{\text{min}})/q > 0$

$q = 3: \# \leftrightarrow$
$p = 2: \# \rightarrow$

Communications
Recovering Soundness

Proof: If there is a $u$-cycle, construction of a counter-example

$q > p \quad \Rightarrow \quad \varepsilon = \frac{q \tau_{\text{max}} - p \tau_{\text{min}}}{q} > 0$

Communications

$q = 3: \# \quad \leftarrow$
$p = 2: \# \quad \rightarrow$
Proof: If there is a $u$-cycle, construction of a counter-example

$$q > p \implies \varepsilon = \frac{q\tau_{\text{max}} - p\tau_{\text{min}}}{q} > 0$$
Recovering Soundness

Proof: If there is a \( u \)-cycle, construction of a counter-example

\[ q > p \implies \epsilon = \frac{q\tau_{\text{max}} - p\tau_{\text{min}}}{q} > 0 \]

Communications

\[ q = 3: \# \leftrightarrow \]
\[ p = 2: \# \rightarrow \]
Proof: If there is a $u$-cycle, construction of a counter-example

Communications

$q = 3$: # \leftrightarrow

$p = 2$: # \rightarrow

$q > p \implies \varepsilon = (q\tau_{\max} - p\tau_{\min})/q > 0$

Recovering Soundness
Recovering Soundness

**Proof:** If there is a $u$-cycle, construction of a counter-example

$q > p \implies \varepsilon = (q \tau_{\text{max}} - p \tau_{\text{min}})/q > 0$
**Recovering Soundness**

**Proof:** If there is a $u$-cycle, construction of a counter-example

$$q > p \implies \varepsilon = (q\tau_{\text{max}} - p\tau_{\text{min}})/q > 0$$
Proof: If there is a $u$-cycle, construction of a counter-example

$\varepsilon = \frac{(q\tau_{\text{max}} - p\tau_{\text{min}})}{q}$

$q > p \implies \varepsilon > 0$
Recovering Soundness

Proof: If there is a $u$-cycle, construction of a counter-example

$q > p \implies \varepsilon = (q \tau_{\text{max}} - p \tau_{\text{min}})/q > 0$

$q = 3: \# \quad \quad \quad p = 2: \#
Proof: If there is a $u$-cycle, construction of a counter-example

$q > p \implies \varepsilon = (q \tau_{\text{max}} - p \tau_{\text{min}})/q > 0$
Recovering Soundness

**Proof:** If there is a \( u \)-cycle, construction of a counter-example

\[
q > p \implies \epsilon = (q\tau_{\text{max}} - p\tau_{\text{min}})/q > 0
\]
Recovering Soundness

Proof: If there is a $u$-cycle, construction of a counter-example

$q > p \implies \varepsilon = (q\tau_{\text{max}} - p\tau_{\text{min}})/q > 0$
Recovering Soundness

**Proof:** If there is a $u$-cycle, construction of a counter-example

Communications

$q = 3$: # $\iff$ 
$p = 2$: # $\implies$

\[ q > p \implies \varepsilon = \left( q\tau_{\text{max}} - p\tau_{\text{min}} \right) / q > 0 \]

We built a cycle of positive weight!
Proof: On the other hand, by contraposition,
Recovering Soundness

**Proof:** On the other hand, by contraposition,

PC/u-cycle
Proof: On the other hand, by contraposition,
Proof: On the other hand, by contraposition,
Proof: On the other hand, by contraposition,

\[ \text{cycle} \quad \quad \quad \quad \quad \text{balanced} \quad \quad \quad \quad \quad \quad \longrightarrow \quad \tau_{\text{max}} = 0 \]
Recovering Soundness

**Proof:** On the other hand, by contraposition,

\[ \tau_{\text{max}} = 0 \]

PC/u-cycle

- cycle
- balanced

Condition 1.
Proof: On the other hand, by contraposition,

\[ \text{PC/u-cycle} \quad \text{cycle} \quad \text{balanced} \quad \Rightarrow \quad \tau_{\text{max}} = 0 \]

\[ \Rightarrow \quad \tau_{\text{min}} < \tau_{\text{max}} \]

Condition 1.
Proof: On the other hand, by contraposition,

\[ \text{PC/u-cycle} \quad \text{cycle} \quad \text{balanced} \quad \Rightarrow \quad \tau_{\text{max}} = 0 \]

\[ \text{Condition 1.} \]

\[ \text{balanced} \quad \Rightarrow \quad \tau_{\text{min}} < \tau_{\text{max}} \]

\[ \text{Condition 2.} \]
Recovering Soundness

Proof: On the other hand, by contraposition,

PC/\(u\)-cycle

---

cycle

balanced

---

\(\tau_{\text{max}} = 0\) 

Condition 1.

---

balanced

\(\tau_{\text{min}} < \tau_{\text{max}}\)

Condition 2.

---

cycle

\(T_{\text{min}} \geq L_c\tau_{\text{max}}\)
Proof: On the other hand, by contraposition,

\[ \text{PC/u-cycle} \]

- Cycle \[ \implies \tau_{\text{max}} = 0 \] (Condition 1.)
- Balanced \[ \implies \tau_{\text{min}} < \tau_{\text{max}} \] (Condition 2.)
- Cycle \[ \implies T_{\text{min}} \geq L_c \tau_{\text{max}} \] (Condition 3.)
**Topology Examples**

A ↔ B ↔ C ↔ D

daisy chain: $T_{\text{min}} \geq 2\tau_{\text{max}}$

B

A ↔ F ↔ C

star: $T_{\text{min}} \geq 2\tau_{\text{max}}$

E

D

bidirectional ring: $\tau_{\text{max}} = 0$

A ↔ B ↔ C

E ↔ D

fully connected: $\tau_{\text{max}} = 0$

E ↔ D

unidirectional ring: $T_{\text{min}} \geq 5\tau_{\text{max}}$

Communications of the application
Topoogy Examples

Daisy chain: $T_{\text{min}} \geq 2\tau_{\text{max}}$

Star: $T_{\text{min}} \geq 2\tau_{\text{max}}$

Unidirectional ring: $T_{\text{min}} \geq 5\tau_{\text{max}}$

Bidirectional ring: $\tau_{\text{max}} = 0$

Fully connected: $\tau_{\text{max}} = 0$

Require instantaneous communications

Communications of the application
“It is not the case that a component process executes more than twice between two successive executions of another process.”

For any node:
1. no more than 2 activations between 2 message receptions
2. no more than 2 message receptions between two activations

Condition 1.

Condition 2.
Quasi-Synchronous Systems

“It is not the case that a component process executes more than twice between two successive executions of another process.”

**Theorem:** A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}}$
Quasi-Synchronous Systems

“It is not the case that a component process executes more than twice between two successive executions of another process.”

Theorem: A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. \(2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}}\)

Worst-case scenario
Quasi-Synchronous Systems

“It is not the case that a component process executes more than twice between two successive executions of another process.”

**Theorem:** A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. \(2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}}\)

Worst-case scenario
“It is not the case that a component process executes more than twice between two successive executions of another process.”

**Theorem:** A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. \( 2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}} \)

Worst-case scenario
“It is not the case that a component process executes more than twice between two successive executions of another process.”

**Theorem:** A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. \(2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}}\)
Quasi-Synchronous Systems

“It is not the case that a component process executes more than twice between two successive executions of another process.”

**Theorem:** A real-time model is quasi-synchronous if and only if,

1. it is unitary discretizable
2. $2T_{\text{min}} + \tau_{\text{min}} \geq T_{\text{max}} + \tau_{\text{max}}$

Worst-case scenario
The quasi-synchronous abstraction:
1. Model transmission as unit delays
2. Constrain node activations interleavings

Contributions:
• Condition 1 is not sound in general
• Notion of unitary discretization
• Necessary and sufficient conditions to recover soundness
• Characterization of quasi-synchronous systems

Constrain both the communication graph and the real-time characteristics of the architecture to recover soundness of the quasi-synchronous abstraction.