Methodology for the formal verification of temporal properties for real-time safety-critical applications based on logical time

PhD Defense of Fabien Siron
Paris, December 11, 2023

Pr. Reinhard von Hanxleden, Rapporteur
Pr. Pierre-Loic Garoche, Rapporteur
Dr. Timothy Bourke, Examinateur
Dr. Dumitru Potop-Butucaru, Directeur
Dr. Robert de Simone, Co-directeur
Drs. D. Chabrol & A. Methni, Encadrants industriels

Safety-critical real-time systems

In this work, we consider systems that are:

- **Safety-critical**: failure may have an impact on the system's safety
- **Real-time** (and time-safety): system's correctness (and thus its overall safety) depends on a set of timing constraints

Typical example in the avionics domain: the control system of an aircraft engine (FADEC)

Formal design for real-time systems

Multiform Logical Time: Deal with a formal abstraction of time through **logical timing constraints** dictated by high-level requirements and independant from hardware platform.

Formal design for real-time systems

Multiform Logical Time: Deal with a formal abstraction of time through **logical timing constraints** dictated by high-level requirements and independant from hardware platform.

Typical methodology:

1. **Logical Time Design: Specification & Requirements**
   - Refine timing requirements to produce a program based only on **logical time** constraints.

Formal design for real-time systems

Multiform Logical Time: Deal with a formal abstraction of time through **logical timing constraints** dictated by high-level requirements and independant from hardware platform.

Typical methodology:

1. **Logical Time Design: Specification & Requirements**
   - Refine timing requirements to produce a program based only on **logical time** constraints.
2. **Physical Time Design: Implementation**
   - Add **physical time** platform provisions (e.g., WCET) and ensure that **logical time** constraints can be satisfied.

Formal design for real-time systems

Multiform Logical Time: Deal with a formal abstraction of time through **logical timing constraints** dictated by high-level requirements and independant from hardware platform.

Typical methodology:

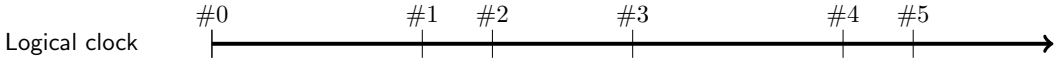
1. **Logical Time Design: Specification & Requirements**
   - Refine timing requirements to produce a program based only on **logical time** constraints.
2. **Physical Time Design: Implementation**
   - Add **physical time** platform provisions (e.g., WCET) and ensure that **logical time** constraints can be satisfied.

▷ We focus on the **Logical Time Design** phase.

Formal design for real-time systems

Logical timing constraints rely on **logical clocks** to express time.

- Sequence of specific instants to abstract a specific time base.
- Used to replace physical dates by logical sequencing.[1]

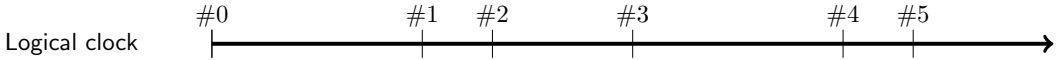


[1] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications ACM* (1978).

Formal design for real-time systems

Logical timing constraints rely on **logical clocks** to express time.

- Sequence of specific instants to abstract a specific time base.
- Used to replace physical dates by logical sequencing.[1]



Two main cases of multi-clock systems:

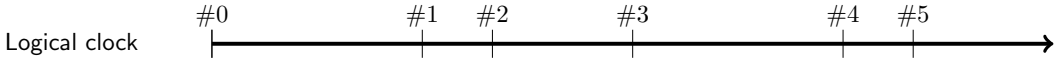
1. **Mono-Source**: all logical clocks are derived from a unique global **clock source**.

[1] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications ACM* (1978).

Formal design for real-time systems

Logical timing constraints rely on **logical clocks** to express time.

- Sequence of specific instants to abstract a specific time base.
- Used to replace physical dates by logical sequencing.[1]



Two main cases of multi-clock systems:

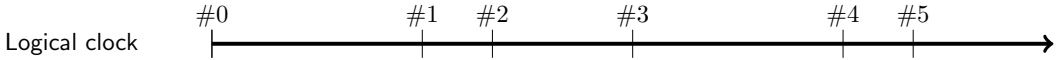
1. **Mono-Source**: all logical clocks are derived from a unique global **clock source**.
2. **Multi-Source**: logical clocks might rely on multiple independant **clock sources**.

[1] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications ACM* (1978).

Formal design for real-time systems

Logical timing constraints rely on **logical clocks** to express time.

- Sequence of specific instants to abstract a specific time base.
- Used to replace physical dates by logical sequencing.[1]



Two main cases of multi-clock systems:

1. **Mono-Source**: all logical clocks are derived from a unique global **clock source**.
2. **Multi-Source**: logical clocks might rely on multiple independant **clock sources**.

▷ In practice, most industrial systems fall in the first category due to safety considerations. However, the second one has also interesting use-cases (e.g., automotive Powertrain [5]).

[1] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". In: *Communications ACM* (1978).

Industrial Context

ASTERIOS technology:

- Software tool suite dedicated to the design and integration of safety-critical systems
- Created by the CEA research institute [2] (from 90's to 2011, formerly Oasis[3]):
  - ▷ Used for modern nuclear plant control systems
- Produced by the KRONO-SAFE company (since 2011):
  - ▷ Now renamed ASTERIOS TECHNOLOGIES due to the takeover by SAFRAN in 2023.

[2] French Alternative Energies and Atomic Energy Commission

[3] Stéphane Louise et al. "The OASIS kernel: A framework for high dependability real-time systems". In: *HASE*. IEEE. 2011.

Industrial Context

ASTERIOS technology:

- Software tool suite dedicated to the design and integration of safety-critical systems
- Created by the CEA research institute [2] (from 90's to 2011, formerly Oasis[3]):
  - ▷ Used for modern nuclear plant control systems
- Produced by the KRONO-SAFE company (since 2011):
  - ▷ Now renamed ASTERIOS TECHNOLOGIES due to the takeover by SAFRAN in 2023.

The PsyC language:

- Parallel SYnchronous dialect of the C language
- Fully integrated compilation with an in-house real-time kernel

[2] French Alternative Energies and Atomic Energy Commission

[3] Stéphane Louise et al. "The OASIS kernel: A framework for high dependability real-time systems". In: *HASE*. IEEE. 2011.

Problem statement

Objectives:

1. **Define semantics foundations for the PsyC language based on existing logical time approaches.**

Problem statement

Objectives:

1. **Define semantics foundations for the PsyC language based on existing logical time approaches.**

2. **Define a formal verification methodology for the PsyC language based on the defined semantics foundations.**

Proposed Methodology: contributions

Synchronous Logical Execution Time program in **PsyC**. [2]

Proposed Methodology: contributions

Synchronous Logical Execution Time program in **PsyC**. **2**

Native Semantics

→ Preservation of durations.

→ Defined as a "big-step" operational semantics. **3.1**

Proposed Methodology: contributions

Synchronous Logical Execution Time program in **PsyC**. **2**

Synchronous Semantics

→ Synchronous expansion of durations.
→ Translation to ESTEREL.
→ Yield a "small-step" operational semantics. **3.2**

Native Semantics

→ Preservation of durations.
→ Defined as a "big-step" operational semantics. **3.1**

Proposed Methodology: contributions

Synchronous Logical Execution Time program in **PsyC**. **2**

Synchronous Semantics

→ Synchronous expansion of durations.
→ Translation to ESTEREL.
→ Yield a "small-step" operational semantics. **3.2**

Observational equivalence 3.3

Native Semantics

→ Preservation of durations.
→ Defined as a "big-step" operational semantics. **3.1**

Proposed Methodology: contributions

CCSL encoding of timing requirements
→ Synchronizations.
→ Causality.
→ Latencies (including end-to-end). **4.1**

Synchronous Logical Execution Time program in **PsyC**. **2**

Synchronous Semantics
→ Synchronous expansion of durations.
→ Translation to ESTEREL.
→ Yield a "small-step" operational semantics. **3.2**

Observational equivalence 3.3

Native Semantics
→ Preservation of durations.
→ Defined as a "big-step" operational semantics. **3.1**

Proposed Methodology: contributions

CCSL encoding of timing requirements
→ Synchronizations.
→ Causality.
→ Latencies (including end-to-end). **4.1**

Synchronous Semantics
→ Synchronous expansion of durations.
→ Translation to ESTEREL.
→ Yield a "small-step" operational semantics. **3.2**

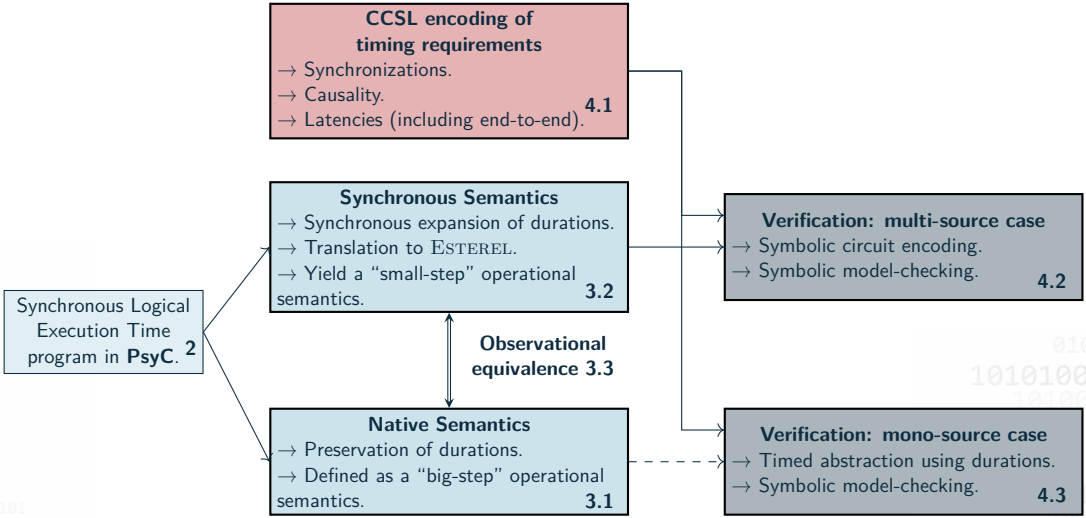
Synchronous Logical Execution Time program in **PsyC**. **2**

Observational equivalence 3.3

Native Semantics
→ Preservation of durations.
→ Defined as a "big-step" operational semantics. **3.1**

Verification: multi-source case
→ Symbolic circuit encoding.
→ Symbolic model-checking. **4.2**

Proposed Methodology: contributions



Outline

Outline

The Synchronous-Reactive approach

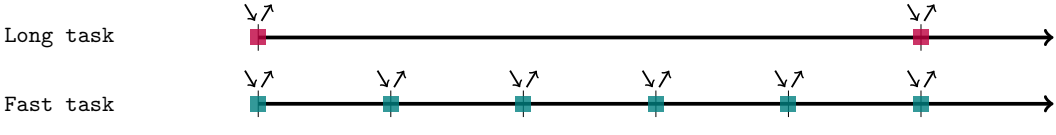


Figure: Synchronous-Reactive approach[4]

- Each reaction is triggered by the instants of a logical clock
- Synchronous communication model

[4] Albert Benveniste et al. "The Synchronous Languages 12 Years Later". In: *Proceedings of the IEEE* 91 (2003).

The Synchronous-Reactive approach

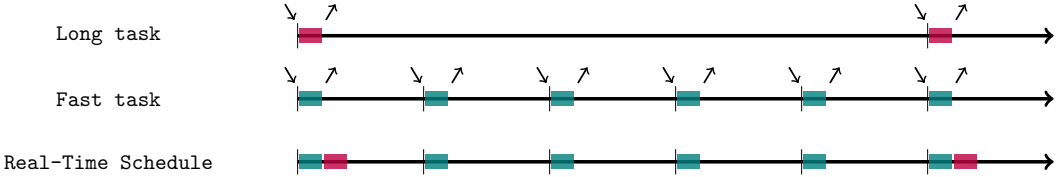


Figure: Synchronous-Reactive approach[5]

- Each reaction is triggered by the instants of a logical clock
- Synchronous communication model
- The *synchronous hypothesis* states that each computation should terminate before the next tick of a global base clock

[5] Albert Benveniste et al. "The Synchronous Languages 12 Years Later". In: *Proceedings of the IEEE* 91 (2003).

The Synchronous-Reactive approach

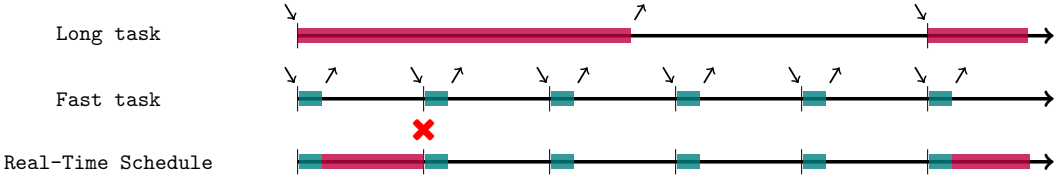


Figure: Synchronous-Reactive approach[6]

- Each reaction is triggered by the instants of a logical clock
- Synchronous communication model
- The *synchronous hypothesis* states that each computation should terminate before the next tick of a global base clock

[6] Albert Benveniste et al. "The Synchronous Languages 12 Years Later". In: *Proceedings of the IEEE* 91 (2003).

The Logical Execution Time approach

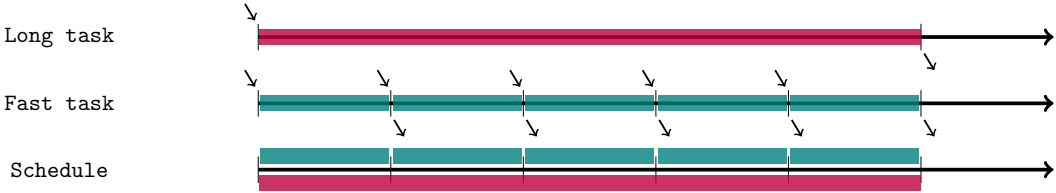


Figure: Logical Execution Time approach[7]

- Computation is abstracted by a specified constant duration based on a unique chronometrical time base (no multiple logical clocks)
- Delayed communication model

[7] Christoph Kirsch and Ana Sokolova. "The Logical Execution Time Paradigm". In: *Advances in Real-Time Systems*. 2012.

The Logical Execution Time approach

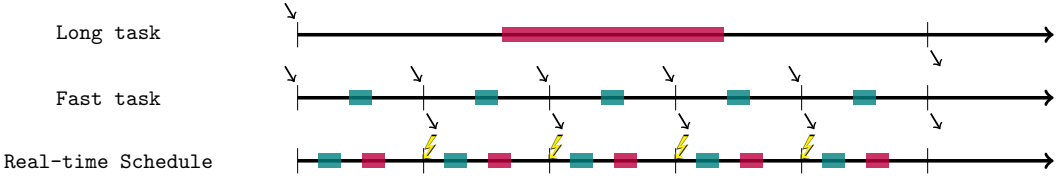


Figure: Logical Execution Time approach[8]

- Computation is abstracted by a specified constant duration based on a unique chronometrical time base (no multiple logical clocks)
- Delayed communication model
- Real-time analysis based on preemptive scheduling

[8] Christoph Kirsch and Ana Sokolova. "The Logical Execution Time Paradigm". In: *Advances in Real-Time Systems*. 2012.

A synchronous generalization of LET

	Synchronous-Reactive	Logical Execution Time
Modeling	**Logical clocks**	**Logical intervals**
Communication	Synchronous	**Delayed**
Implementation	Synchronous hypothesis	**Relaxed synchronous hypothesis**[9]
Languages	ESTEREL, LUSTRE	GIOTTO, TDL
	PsyC	

Synchronous LET extends LET with logical clock synchronisation

[9] Adrian Curic. "Implementing Lustre programs on distributed platforms with real-time constrains". PhD thesis. Université Joseph Fourier, Grenoble, France, 2005.

A synchronous generalization of LET

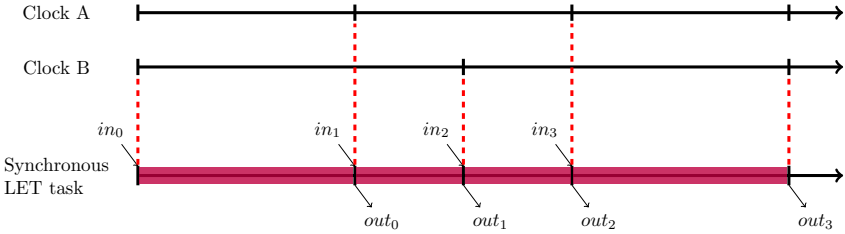


Figure: Synchronous Logical Execution Time approach

- Both computation triggering and duration instants are modeled respectively to logical clock ticks
- Communication is delayed similarly to LET
- Published at ERTS 2022[10]

[10] Fabien Siron et al. "The synchronous Logical Execution Time paradigm". In: *ERTS 2022 - Embedded Real Time Systems*. Toulouse, France, June 2022.

A synchronous generalization of LET

Related formalisms:

1. xGiotto [8] extends LET with events :
   - In principle, both triggering and duration can be modeled by events;
   - However, in practice, xGiotto relies on event scoping to encode implicitely inter-arrival time.

[11] Edward A Lee and Marten Lohstroh. "Generalizing Logical Execution Time". In: *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Springer, 2022, pp. 160–181.

A synchronous generalization of LET

Related formalisms:

1. xGiotto [8] extends LET with events :
   - In principle, both triggering and duration can be modeled by events;
   - However, in practice, xGiotto relies on event scoping to encode implicitely inter-arrival time.
2. Lingua Franca [12] is based on a multiform logical time framework called *Tagged Signal Model* :
   - Actors computations are triggered by logical tags
   - LET can be modeled using delays between dataflow actors[11]:
     `a.out -> b.in after X ms`
   - However, those delays rely on a chronometrical time base, not tags.

[11] Edward A Lee and Marten Lohstroh. "Generalizing Logical Execution Time". In: *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Springer, 2022, pp. 160–181.

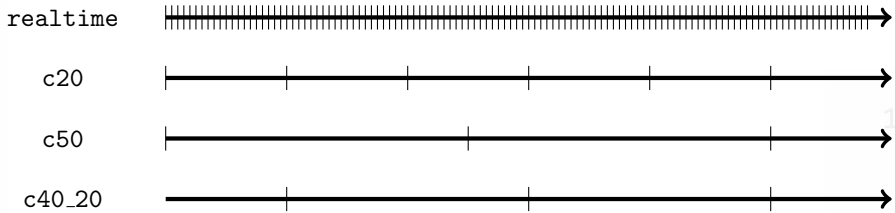
Outline

PsyC overview: clocks

PsyC defines two types of clocks:

- `source`, which are source clocks
- `clock`, which are periodically refined clocks (with period and offset)

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
clock c40_20 = 2 * c20 + 1;
// similar to 40 * realtime + 20
```

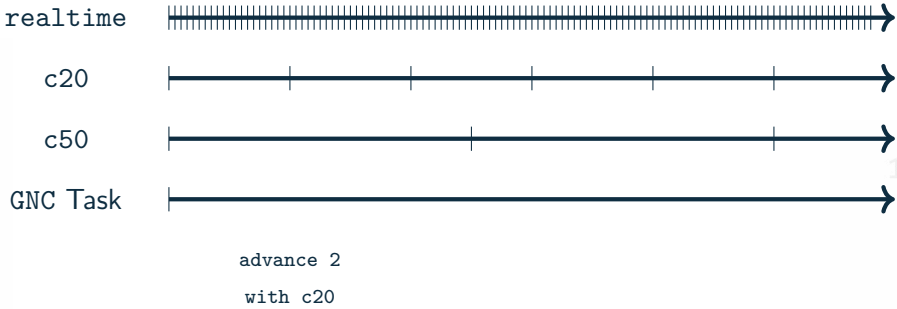


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

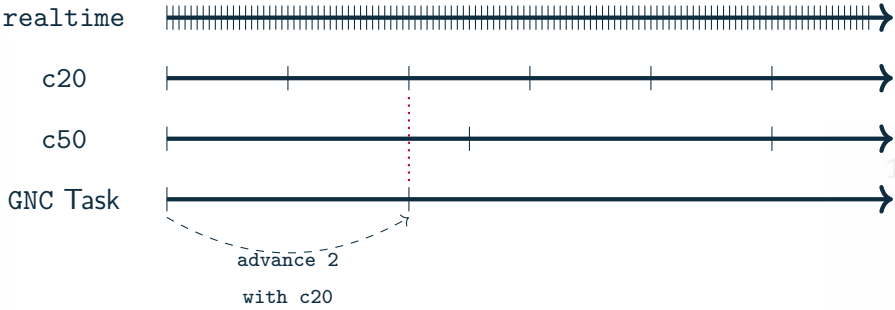


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

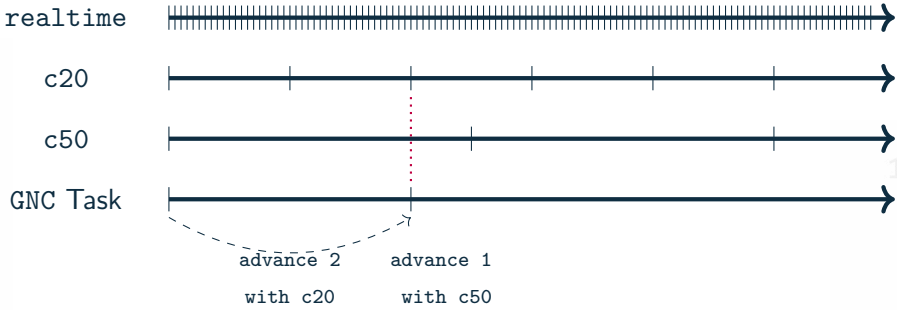


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

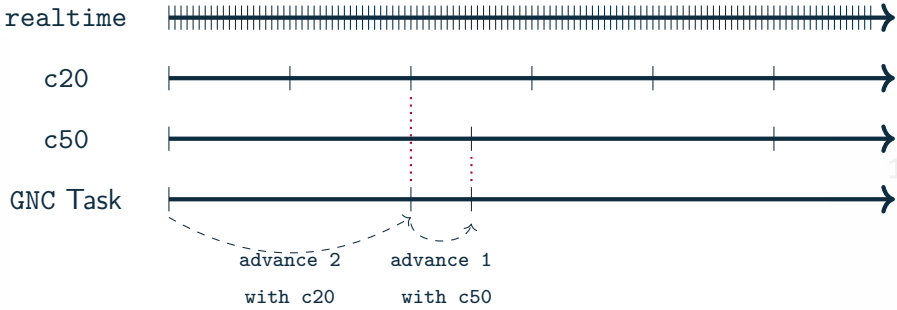


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

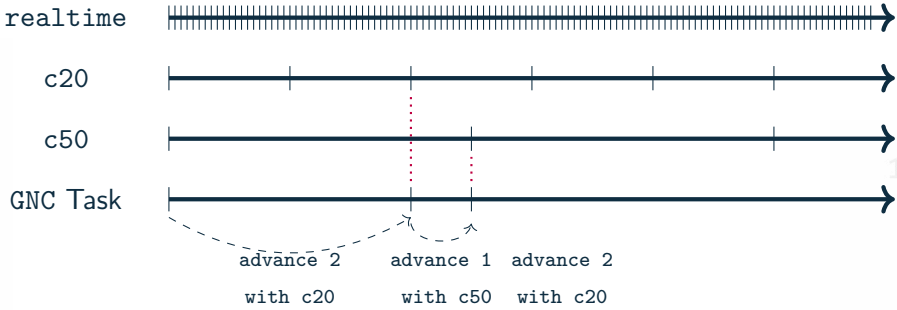


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

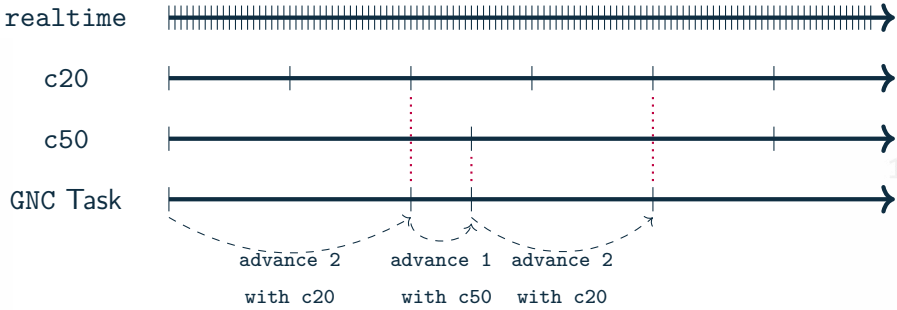


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

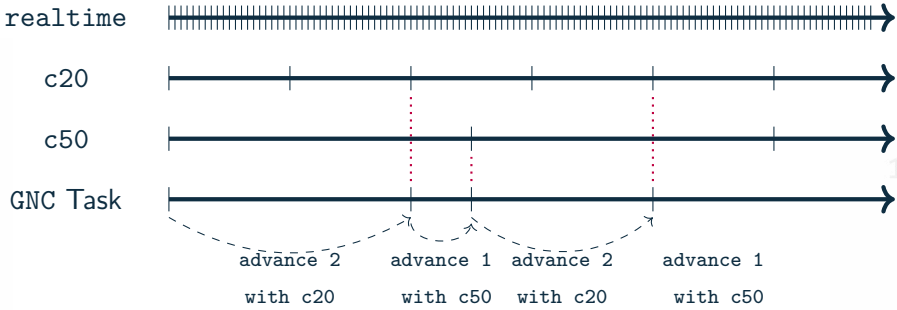


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```

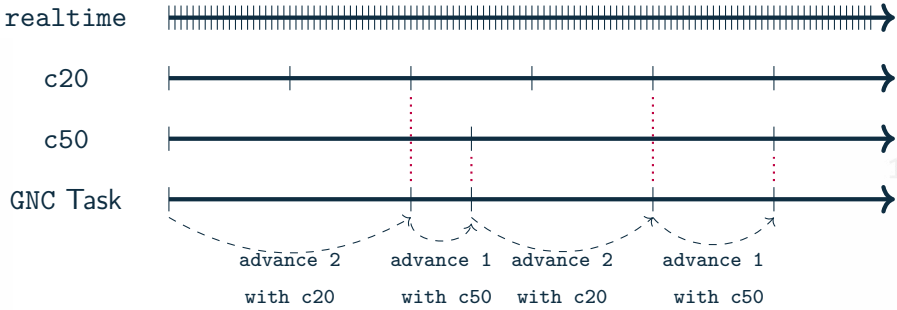


PsyC overview: synchronization

PsyC defines a dedicated synchronization statement:

- advance awaits for a specified number of clock ticks

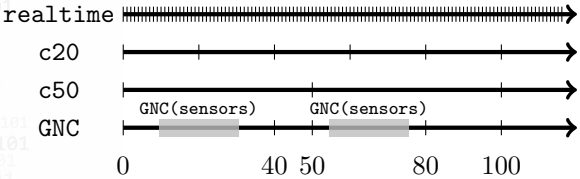
```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;
```



PsyC overview: agents

A PsyC agent defines:

- a task name;
- inputs (consult) and outputs (display);
- a looping body containing C code extended with advance statements.



▷ **Hypothesis:** all execution paths should be constrained by an advance.

```
source realtime;
clock c20 = 20 * realtime;
clock c50 = 50 * realtime;

agent GNC {
  /* inputs*/
  consult sensors, mode;
  /* outputs */
  display order;
  body { /* infinite loop */
    if (mode == NOMINAL) {
      order = GNC(sensors);
      advance 2 with c20;
    }
    advance 1 with c50;
  }
}
```

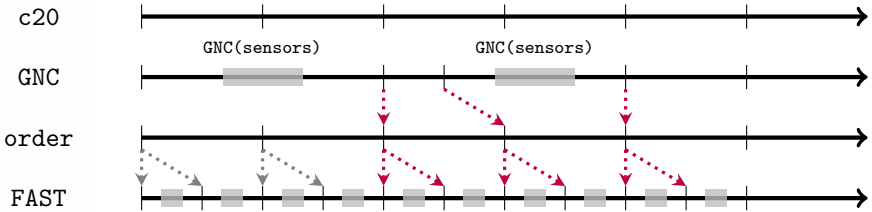
PsyC overview: communication

PsyC inter-task communication is done through dedicated temporal variables:

- One writer but multiple possible readers;
- Data is persistent;
- Values are sampled according to a clock.

```
/* ... */
temporal float order = 0.0f with c20;

agent GNC { /* ... */ }
agent FAST { /* ... */ }
```



Outline

Outline

Native (or big-step) Semantics

Based on structural operational semantics approach, for individual agents:

$$E, ag \Longrightarrow_{n \texttt{ with } s} E', ag'$$

with ag, ag' agent term, E, E' environments and $n \times s$ logical duration (extended on source clock).

[12] Fabien Siron et al. *Formal Semantics of the PsyC language*. Research Report. INRIA Sophia Antipolis - Méditerranée (France), 2022.

Native (or big-step) Semantics

Based on structural operational semantics approach, for individual agents:

$$E, ag \Longrightarrow_{n \texttt{ with } s} E', ag'$$

with ag, ag' agent term, E, E' environments and $n \times s$ logical duration (extended on source clock).

For individual agents, very classical rules but temporal rule for `advance` statement[12]:

$$E, \texttt{advance n with c} \Longrightarrow_{n \times \Pi(c) - d_c \texttt{ with } Source(c)} E, \texttt{nothing}$$

Where d_c denotes the corresponding clock state according to its period and $\Pi(c)$ its absolute period.

[12] Fabien Siron et al. *Formal Semantics of the PsyC language*. Research Report. INRIA Sophia Antipolis - Méditerranée (France), 2022.

Native (or big-step) Semantics

Classical synchronous composition cannot be used directly as agents do not have a common projection.

- ▷ Hence, we consider **global states**, for synchronization, along with **intermediate states**, due to the projection of **global states** from other agents.

[13] (Christophe Aussagues and Vincent David. "A method and a technique to model and ensure timeliness in safety critical real-time systems". In: *ICECCS*. 1998)

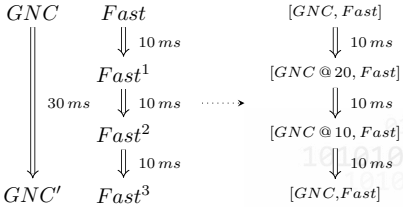
Native (or big-step) Semantics

Classical synchronous composition cannot be used directly as agents do not have a common projection.

- ▷ Hence, we consider **global states**, for synchronization, along with **intermediate states**, due to the projection of **global states** from other agents.

Then, agent composition can be split in two cases:

1. **Mono-Source**:
   - ▷ Every transition is now a partial interval generated by the agent projection. At each state, $N_{program} = min(N_{ag_1}, N_{ag_2}, \ldots)$, similar to Oasis synchronized product [13]
2. **Multi-Source**:
   - ▷ More complex due to multiple source interleaving



[13] (Christophe Aussagues and Vincent David. "A method and a technique to model and ensure timeliness in safety critical real-time systems". In: *ICECCS*. 1998)

Outline

Synchronous (or small-step) Semantics

A simpler semantics can be defined by translation to a **Synchronous-Reactive** language, ESTEREL. Translation principle:

- All control statements can be translated with similar constructs in ESTEREL
- `advance` statement is translated by the ESTEREL synchronization statement followed by the *display* of temporal variables.

```
private_variable := f(...);
...
await n c;
emit temporal(private_variables);
```

Synchronous (or small-step) Semantics

A simpler semantics can be defined by translation to a **Synchronous-Reactive** language, ESTEREL. Translation principle:

- All control statements can be translated with similar constructs in ESTEREL
- `advance` statement is translated by the ESTEREL synchronization statement followed by the *display* of temporal variables.

```
private_variable := f(...);
...
await n c;
emit temporal(private_variables);
```

▷ Durations are not preserved in the semantics as ESTEREL yields atomic semantics transitions:

$$p, data \xrightarrow[In]{Out}\!\!\!\!\rightarrow p', data'$$

Synchronous (or small-step) Semantics

A simpler semantics can be defined by translation to a **Synchronous-Reactive** language, ESTEREL.

Example:

```
body { /* infinite loop */
  if (mode == NOMINAL) {
    orders = GNC_step(sensors);
    advance 2 with c20;

  }
  advance 1 with c50;
}
```

PSYC agent example

```
loop

end loop
```

ESTEREL agent translation

Synchronous (or small-step) Semantics

A simpler semantics can be defined by translation to a **Synchronous-Reactive** language, ESTEREL.

Example:

```
body { /* infinite loop */
  if (mode == NOMINAL) {
    orders = GNC_step(sensors);
    advance 2 with c20;

  }
  advance 1 with c50;
}
```

PSYC agent example

```
loop
 if ?mode = NOMINAL then

 end if;
 await 1 c50;
end loop
```

ESTEREL agent translation

Synchronous (or small-step) Semantics

A simpler semantics can be defined by translation to a **Synchronous-Reactive** language, ESTEREL.

Example:

```
body { /* infinite loop */
  if (mode == NOMINAL) {
    orders = GNC_step(sensors);
    advance 2 with c20;

  }
  advance 1 with c50;
}
```

PSYC agent example

```
loop
 if ?mode = NOMINAL then
  private_orders := GNC_step(?sensors);
  await 2 c20;
  emit orders(private_orders);
 end if;
 await 1 c50;
end loop
```

ESTEREL agent translation

Outline

Equivalence relation

Theorem (equivalence)

Assuming source clock always present (no stuttering), for individual agents:

$$Ag \Longrightarrow_{n \times s} Ag' \quad \textit{iff} \quad \underbrace{P^0 \twoheadrightarrow P^1 \ldots \twoheadrightarrow P^n}_{n\ times}$$

assuming $Ag \approx P^0$, we have $Ag' \approx P^n$ (term and data equivalence)

Both semantics are observationally equivalent on the interval boundaries at the agent level.

- ▷ All properties depending only on (global) states in the synchronous semantics is preserved on the native semantics. Published in TCRS 2023[14].

[14] Fabien Siron et al. "Semantics fondations of PsyC based on synchronous Logical Execution Time". In: *TCRS 2023*. San Antonio, Texas, U.S.A., May 2023.

Outline

Outline

Specifying requirements: motivation

Motivation for verification activities:

- PsyC allows for the modeling of timing constraints *inside* agents.
- However, high-level timing requirements also concern timing constraints *across* multiple agents (e.g., synchronizations, latencies ...).

Specifying requirements: motivation

Motivation for verification activities:

- PsyC allows for the modeling of timing constraints *inside* agents.
- However, high-level timing requirements also concern timing constraints *across* multiple agents (e.g., synchronizations, latencies ...).

Multiple possible formalisms to specify those timing requirements:

- **Temporal Logic:** convential logic with temporal operators (next, always ...)
- **Contracts and Observers:** predicates on assumptions ("*assume*") and guarantees
- **Clock Constraint Specification Language:** constraint predicates between clocks

Specifying requirements: motivation

Motivation for verification activities:

- PsyC allows for the modeling of timing constraints *inside* agents.
- However, high-level timing requirements also concern timing constraints *across* multiple agents (e.g., synchronizations, latencies ...).

Multiple possible formalisms to specify those timing requirements:

- **Temporal Logic:** convential logic with temporal operators (next, always ...)
- **Contracts and Observers:** predicates on assumptions ("*assume*") and guarantees
- **Clock Constraint Specification Language:** constraint predicates between clocks

Clock Constraint Specification Language

CCSL is a specification language based on **clock constraints**[15]:

- A **CCSL clock** is (still) a sequence of ticks (or instants);

[15] Charles André. *Syntax and semantics of the clock constraint specification language (CCSL)*. Tech. rep. INRIA, 2009.

Clock Constraint Specification Language

CCSL is a specification language based on **clock constraints**[15]:

- A **CCSL clock** is (still) a sequence of ticks (or instants);
- **CCSL constraints** (or relations) constrain the tick occurence between multiple clocks:
  - e.g. `isPeriodicOn`, $\boxed{\preccurlyeq}$ (precedes) ...

[15] Charles André. *Syntax and semantics of the clock constraint specification language (CCSL)*. Tech. rep. INRIA, 2009.

Clock Constraint Specification Language

CCSL is a specification language based on **clock constraints**[15]:

- A **CCSL clock** is (still) a sequence of ticks (or instants);
- **CCSL constraints** (or relations) constrain the tick occurence between multiple clocks:
  - e.g. `isPeriodicOn`, $\boxed{\preccurlyeq}$ (precedes) ...
- **CCSL expressions** build new clocks from existing ones:
  - e.g. `sampledOn`, `DelayFor` ...

[15] Charles André. *Syntax and semantics of the clock constraint specification language (CCSL)*. Tech. rep. INRIA, 2009.

Specifying requirements: basic timing requirements

Description of basic timing requirements inspired from TADL2[16] and AUTOSAR timing extensions:

- **Repetition requirement**: period between successive instants
  - c `isPeriodicOn` *realtime* `period` n where $n \in \mathbb{N}^* *$ (Periodicity)

[16] Marie-Agnès Peraldi-Frati et al. "A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2". In: *IEEE 17th ICECCS*. 2012.

Specifying requirements: basic timing requirements

Description of basic timing requirements inspired from TADL2[16] and AUTOSAR timing extensions:

- **Repetition requirement**: period between successive instants
  - c `isPeriodicOn` *realtime* `period` n where $n \in \mathbb{N}^* *$ (Periodicity)
- **Synchronization requirements**: temporal invariants
  - $c_1 \boxed{\equiv} c_2$ (Synchronization)
  - $c_1 \boxed{\#} c_2$ (Exclusion)

[16] Marie-Agnès Peraldi-Frati et al. "A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2". In: *IEEE 17th ICECCS*. 2012.

Specifying requirements: basic timing requirements

Description of basic timing requirements inspired from TADL2[16] and AUTOSAR timing extensions:

- **Repetition requirement**: period between successive instants
  - c `isPeriodicOn` *realtime* `period` n where $n \in \mathbb{N}^* *$ (Periodicity)
- **Synchronization requirements**: temporal invariants
  - $c_1 \boxed{\equiv} c_2$ (Synchronization)
  - $c_1 \boxed{\#} c_2$ (Exclusion)
- **Causality requirements**: causal instant relations
  - $c_1 \boxed{\sim} c_2$ (Alternation)

[16] Marie-Agnès Peraldi-Frati et al. "A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2". In: *IEEE 17th ICECCS*. 2012.

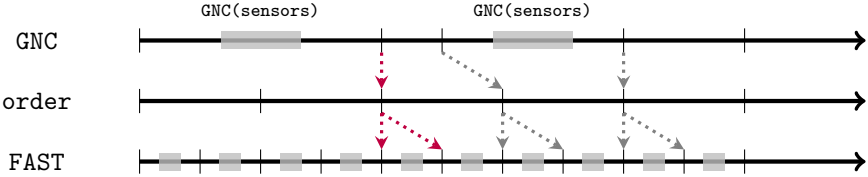
Specifying requirements: basic timing requirements

Description of basic timing requirements inspired from TADL2[16] and AUTOSAR timing extensions:

- **Repetition requirement**: period between successive instants
  - c `isPeriodicOn` *realtime* `period` n where $n \in \mathbb{N}^* *$ (Periodicity)
- **Synchronization requirements**: temporal invariants
  - $c_1 \boxed{\equiv} c_2$ (Synchronization)
  - $c_1 \boxed{\#} c_2$ (Exclusion)
- **Causality requirements**: causal instant relations
  - $c_1 \boxed{\sim} c_2$ (Alternation)
- **Delay requirements**: delay between stimuli and response clocks
  - *response* $\boxed{\prec}$ (*stimuli* `isDelayedFor` n `on` *realtime*) where $n \in \mathbb{N}^*$ (Delay)

[16] Marie-Agnès Peraldi-Frati et al. "A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2". In: *IEEE 17th ICECCS*. 2012.

Specifying requirements: end-to-end requirements

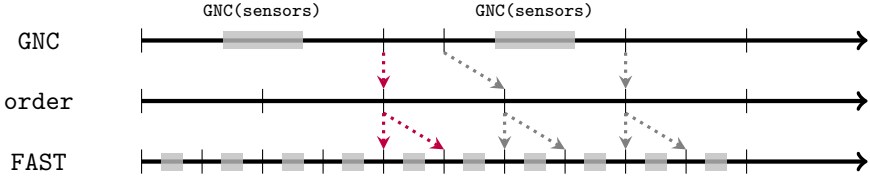


Delay between two instants across a *functional chain* of specific task instants $p^1, p^2, \ldots, p^n$ for tasks $t^1, t^2, \ldots, t^n$ where p^i *subclock* t^i:

- First, specify multiple propagation paths, function chains:
  - $p^i_{display} \preccurlyeq p^{i+1}_{consult}$ (Causality)

[17] Nico Feiertag et al. "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics". In: *RTSS*. 2009.

Specifying requirements: end-to-end requirements

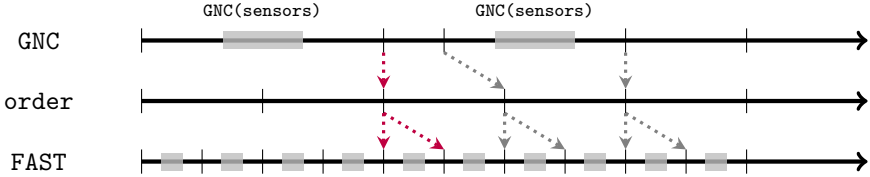


Delay between two instants across a *functional chain* of specific task instants $p^1, p^2, \ldots, p^n$ for tasks $t^1, t^2, \ldots, t^n$ where p^i *subclock* t^i:

- First, specify multiple propagation paths, function chains:
  - $p^i_{display} \preccurlyeq p^{i+1}_{consult}$ (Causality)
  - $p^{i+1}_{consult} \prec (p^i_{display}\texttt{sampledOn}\ t^i_{display})$ (Consistency)

[17] Nico Feiertag et al. "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics". In: *RTSS*. 2009.

Specifying requirements: end-to-end requirements

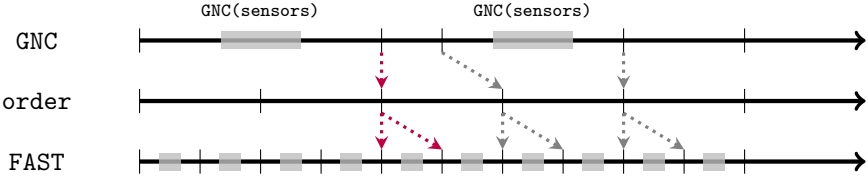


Delay between two instants across a *functional chain* of specific task instants $p^1, p^2, \ldots, p^n$ for tasks $t^1, t^2, \ldots, t^n$ where p^i *subclock* t^i:

- First, specify multiple propagation paths, function chains:
  - $p^i_{display} \prec p^{i+1}_{consult}$ (Causality)
  - $p^{i+1}_{consult} \prec (p^i_{display}\texttt{sampledOn}\ t^i_{display})$ (Consistency)
  - $p^{i+1}_{consult} \prec (p^i_{display}\texttt{sampledOn}\ t^{i+1}_{consult})$ (First[17])

[17] Nico Feiertag et al. "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics". In: *RTSS*. 2009.

Specifying requirements: end-to-end requirements

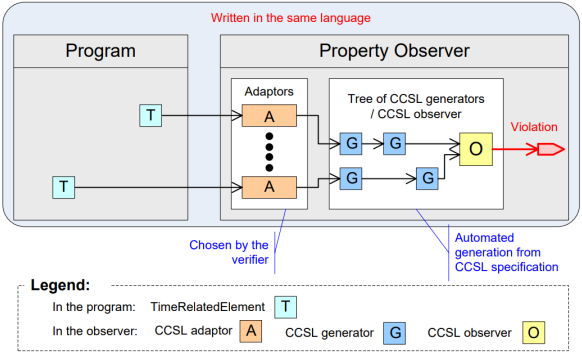


Delay between two instants across a *functional chain* of specific task instants $p^1, p^2, \ldots, p^n$ for tasks $t^1, t^2, \ldots, t^n$ where p^i *subclock* t^i:

- First, specify multiple propagation paths, function chains:
  - $p^i_{display} \boxed{\prec\!\!\prec} p^{i+1}_{consult}$ (Causality)
  - $p^{i+1}_{consult} \boxed{\prec} (p^i_{display}\texttt{sampledOn}\ t^i_{display})$ (Consistency)
  - $p^{i+1}_{consult} \boxed{\prec\!\!\prec} (p^i_{display}\texttt{sampledOn}\ t^{i+1}_{consult})$ (First[17])
- Then, use delay requirements on individual data propagation

[17] Nico Feiertag et al. "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics". In: *RTSS*. 2009.

Synchronous observer based verification

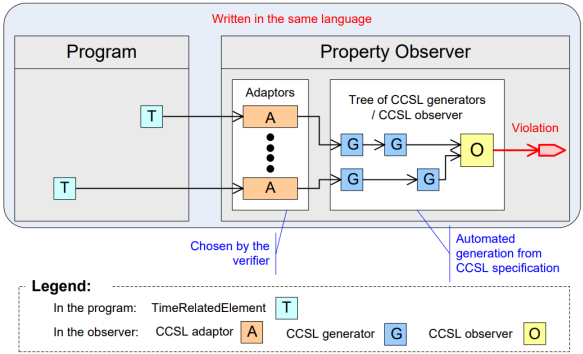


CCSL can be used to model synchronous observers [18]:

- CCSL expressions are translated to ESTEREL (*generators*);

[18] (Charles André. *Verification of clock constraints: CCSL Observers in Esterel.* Tech. rep. INRIA, 2010)

Synchronous observer based verification

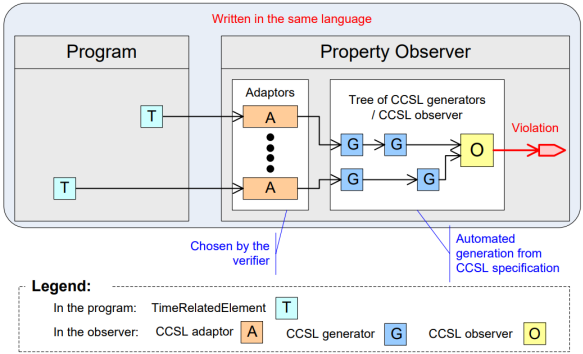


CCSL can be used to model synchronous observers [18]:

- CCSL expressions are translated to ESTEREL (*generators*);
- CCSL constraints are translated to ESTEREL (*observers*);

[18] (Charles André. *Verification of clock constraints: CCSL Observers in Esterel.* Tech. rep. INRIA, 2010)

Synchronous observer based verification

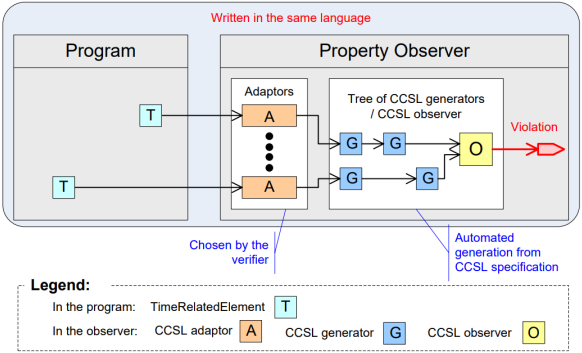


CCSL can be used to model synchronous observers [18]:

- CCSL expressions are translated to ESTEREL (*generators*);
- CCSL constraints are translated to ESTEREL (*observers*);
- Verification can be performed on the composition of an ESTEREL program and a set of translated CCSL expressions/constraints.

[18] (Charles André. *Verification of clock constraints: CCSL Observers in Esterel.* Tech. rep. INRIA, 2010)

Synchronous observer based verification



CCSL can be used to model synchronous observers [18]:

- CCSL expressions are translated to ESTEREL (*generators*);
- CCSL constraints are translated to ESTEREL (*observers*);
- Verification can be performed on the composition of an ESTEREL program and a set of translated CCSL expressions/constraints.

▷ This strategy can be used for both assume and guarantee patterns

[18] (Charles André. *Verification of clock constraints: CCSL Observers in Esterel.* Tech. rep. INRIA, 2010)

Outline

Verification techniques

Overview of verification techniques:

- **Enumerative**: explicit computation/traversal of the state-space
- **Timed Automata (TA)**: encoding of real-time constraints in automata using watches (called *"clocks"*)
- **Binary Decision Diagram (BDD)**: symbolic encoding of set of states using BDD data structures
- **Bounded Model-Checking (BMC)**: symbolic (and bounded) unfolding of the state-space using SAT solvers
- **K-Induction and Interpolation**: techniques to solve the completeness issue of BMC using SAT solvers

Verification methodology

Global methodology:

1. Translation of CCSL requirements into ESTEREL;

[19] Symbolic representation of mealy machines used as a generic representation among tool input formats

Verification methodology

Global methodology:

1. Translation of CCSL requirements into ESTEREL;
2. Translation of PSYC into ESTEREL;

[19] Symbolic representation of mealy machines used as a generic representation among tool input formats

Verification methodology

Global methodology:

1. Translation of CCSL requirements into ESTEREL;
2. Translation of PSYC into ESTEREL;
3. Re-use ESTEREL circuit semantics to encode the model in *Symbolic Transition Systems* [19];

[19] Symbolic representation of mealy machines used as a generic representation among tool input formats

Verification methodology

Global methodology:

1. Translation of CCSL requirements into ESTEREL;
2. Translation of PSYC into ESTEREL;
3. Re-use ESTEREL circuit semantics to encode the model in *Symbolic Transition Systems* [19];
4. Generate model and call specific symbolic model-checkers, we used mostly:
   - NUXMV to use BDD model-checking.
   - PROVER PSL to use SAT model-checking (both bounded and induction based).

[19] Symbolic representation of mealy machines used as a generic representation among tool input formats

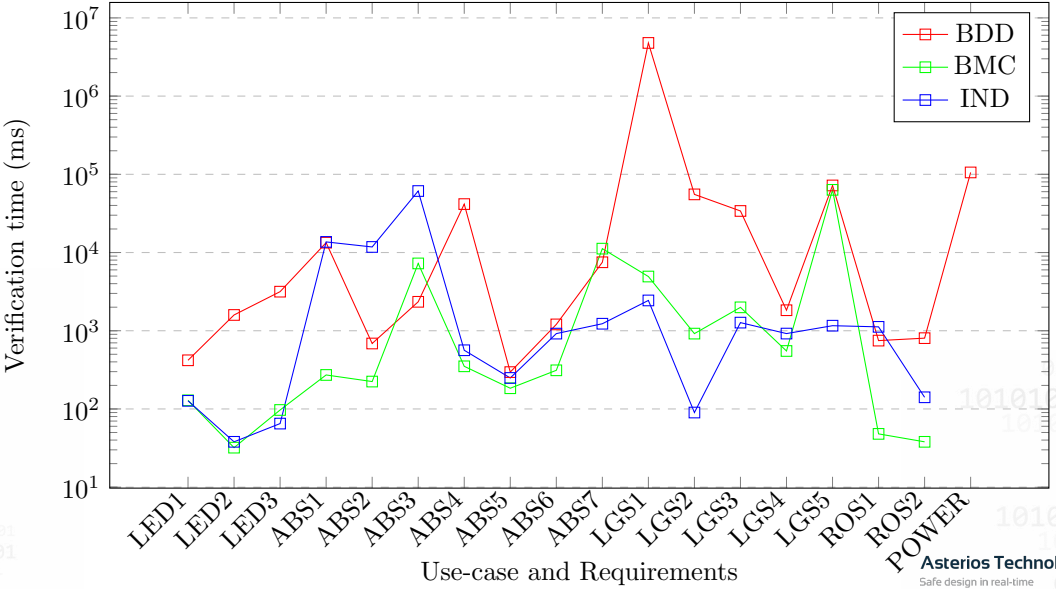
Benchmarks: use-cases

Multiple PsyC use-cases (or adapted from the litterature):

- **LED**: Basic LED controller with two blinking modes
- **ABS**: Automotive Anti-Lock Braking system with two modes
- **ROSACE**: Longitudinal flight controller with only periodic tasks
- **LGS**: Simplified landing gears controller with auxiliary tasks
- **POWER**: Automotive powertrain controller with multiple source clocks

Use-cases	#agents	#clocks	#sources	#decisions	#advance
LED	1	3	1	1	8
ABS	8	4	1	9	30
ROSACE	9	3	1	0	9
LGS	5	3	1	1	16
POWER	3	8	2	1	9

Benchmarks: results



Asterios Technologies
Safe design in real-time

Benchmarks: results

Asterios Technologies
Safe design in real-time

Benchmarks: results

Asterios Technologies
Safe design in real-time

Outline

Mono-source: long duration problem

Considering a basic periodic task example:

Task period	$1\ s$	$10\ s$	$100\ s$	$1000\ s$
#State vars	37	37	37	37
#States	42	402	4002	40002
Diameter	20	200	2000	20000

The state-space grows (linearly) with respect to the task period with, however, the same structure.

Mono-source: long duration problem

Considering a basic periodic task example:

Task period	$1\ s$	$10\ s$	$100\ s$	$1000\ s$
#State vars	37	37	37	37
#States	42	402	4002	40002
Diameter	20	200	2000	20000

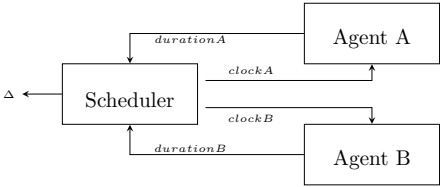
The state-space grows (linearly) with respect to the task period with, however, the same structure.

- ▷ Industrial use-cases often have long durations.
- ▷ We want to optimize the state-space for the **mono-source** scenario.

Optimization methodology

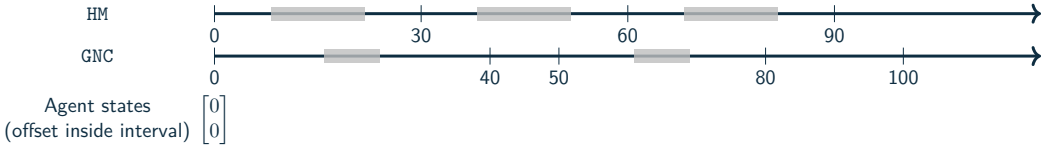
Solution: use a scheduler to jump only on instants in which at least an agent is in a **global state**

- Agents are now triggered by the scheduler and outputs their state to the scheduler
- The scheduler jumps multiple synchronous instants at once, denoted by Δ



Optimization methodology: example

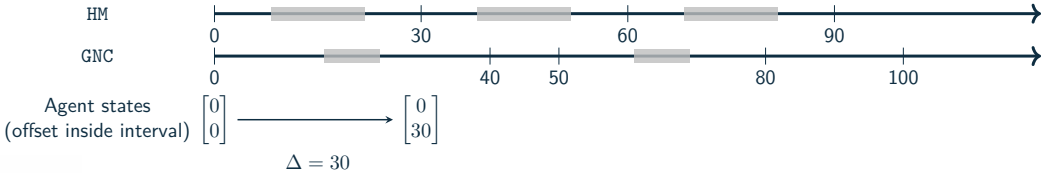
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

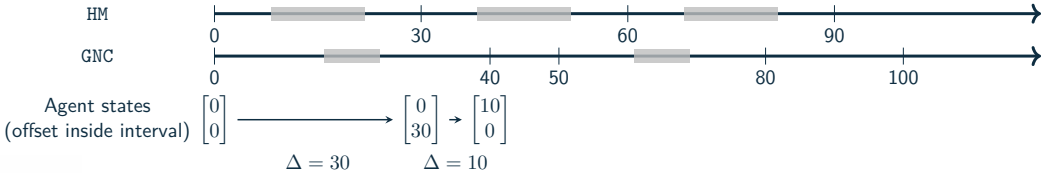
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

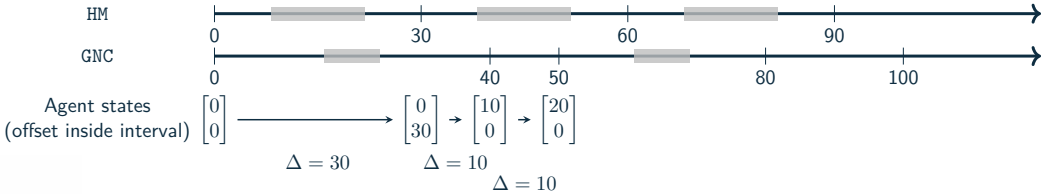
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

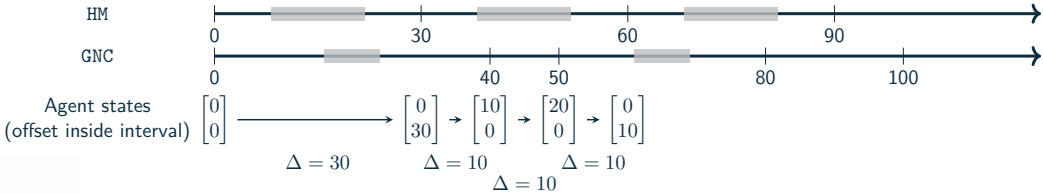
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

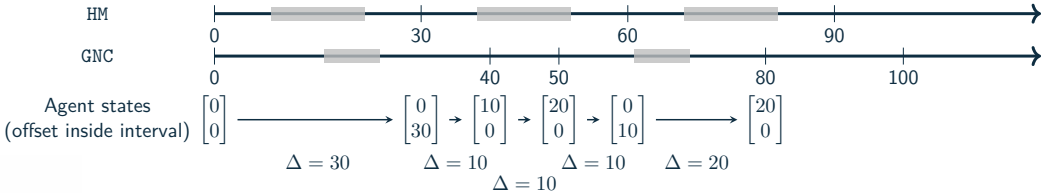
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

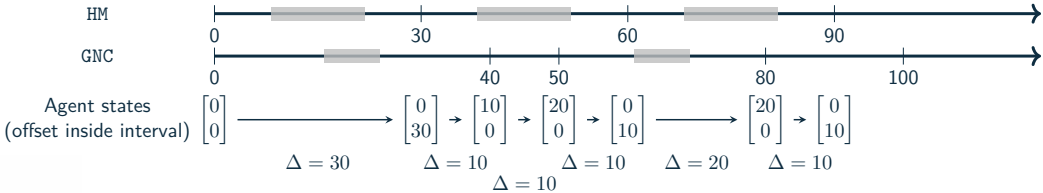
Example of an execution with the scheduler:



[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

Example of an execution with the scheduler:

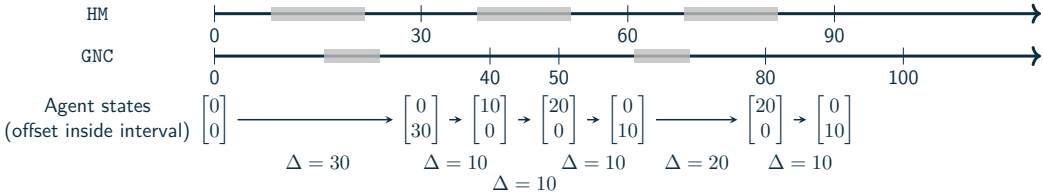


▷ Define a new notion of instant by abstracting durations

[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Optimization methodology: example

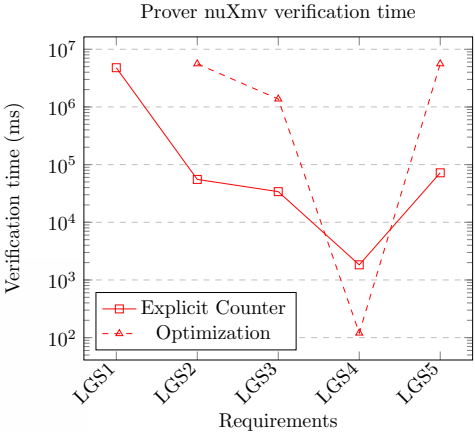
Example of an execution with the scheduler:

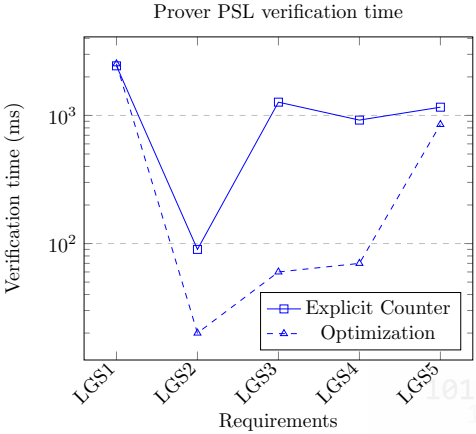


- ▷ Define a new notion of instant by abstracting durations
- ▷ Equivalent to the native semantics product, but computed dynamically! (similar to dynamic ticks[20])

[20] Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. "Real-time ticks for synchronous programming". In: *FDL*. IEEE. 2017.

Benchmarks: a (simplified) Landing Gears System





- ▷ Use-case: 5 tasks, long durations ($\times 50$), aperiodic timing constraints and 1 source
- ▷ Doesn't work with NUXMV using BDD model-checking
- ▷ Up to **95%** speed-up with SAT model-checking using PROVER PSL

Outline

Conclusion and Perspectives

Two main contributions:

- **Formal foundations** for the PsyC language with two formally equivalent semantics based on a new formalism, **synchronous Logical Execution Time**:
  - A native "big-step" semantics, used mainly for compilation.
  - A synchronous "small-step" semantics, now used for verification.
- **Formal verification** methodology for PsyC based on symbolic model-checking:
  - A general multi-source case based on the direct use of the synchronous semantics.
  - An optimization for the mono-source case based on abstracting the durations using the native semantics.

Conclusion and Perspectives

Perspectives:

- Transform the PsyC verification prototype in an industrial product candidate to help the design of PsyC programs (only during logical time phase)
- Extending semantics and verification methodology to PsyC null-latency communication by introducing *"fractional clocks"*
- Compositional model-checking through contracts on external C functions verified separately

Thank you

Thank you for your attention!

References I

- Charles André. *Syntax and semantics of the clock constraint specification language (CCSL)*. Tech. rep. INRIA, 2009.
- Charles André. *Verification of clock constraints: CCSL Observers in Esterel*. Tech. rep. INRIA, 2010.
- Christophe Aussagues and Vincent David. "A method and a technique to model and ensure timeliness in safety critical real-time systems". In: *ICECCS*. 1998.
- Albert Benveniste et al. "The Synchronous Languages 12 Years Later". In: *Proceedings of the IEEE* 91 (2003).
- Damien Chabrol et al. "Freedom from interference among time-triggered and angle-triggered tasks: a powertrain case study". In: *ERTS*. 2014.
- Adrian Curic. "Implementing Lustre programs on distributed platforms with real-time constrains". PhD thesis. Université Joseph Fourier, Grenoble, France, 2005.

References II

- Nico Feiertag et al. “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics”. In: *RTSS*. 2009.
- Arkadeb Ghosal et al. “Event-Driven Programming with Logical Execution Times”. In: *International Workshop on Hybrid Systems: Computation and Control*. Vol. 2993. 2004, pp. 357–371.
- Christoph Kirsch and Ana Sokolova. “The Logical Execution Time Paradigm”. In: *Advances in Real-Time Systems*. 2012.
- Leslie Lamport. “Time, Clocks, and the Ordering of Events in a Distributed System”. In: *Communications ACM* (1978).
- Edward A Lee and Marten Lohstroh. “Generalizing Logical Execution Time”. In: *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Springer, 2022, pp. 160–181.

References III

- Marten Lohstroh et al. “Toward a Lingua Franca for deterministic concurrent systems”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 20.4 (2021), pp. 1–27.
- Stéphane Louise et al. “The OASIS kernel: A framework for high dependability real-time systems”. In: *HASE*. IEEE. 2011.
- Marie-Agnès Peraldi-Frati et al. “A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2”. In: *IEEE 17th ICECCS*. 2012.
- Fabien Siron et al. *Formal Semantics of the PsyC language*. Research Report. INRIA Sophia Antipolis - Méditerranée (France), 2022.
- Fabien Siron et al. “Semantics fondations of PsyC based on synchronous Logical Execution Time”. In: *TCRS 2023*. San Antonio, Texas, U.S.A., May 2023.
- Fabien Siron et al. “The synchronous Logical Execution Time paradigm”. In: *ERTS 2022 - Embedded Real Time Systems*. Toulouse, France, June 2022.

References IV

Reinhard Von Hanxleden, Timothy Bourke, and Alain Girault. “Real-time ticks for synchronous programming”. In: *FDL*. IEEE. 2017.

Algorithm of the optimization methodology

Algorithm 1 Scheduler

procedure $\textsc{Schedule}(Duration_{ag_1},\ State_{ag_1}\ \ldots\ Duration_{ag_n},\ State_{ag_n})$

$\quad Remaining_{ag_i} \leftarrow Duration_{ag_i} - State_{ag_i} \quad \forall i \in [1\ ;\ n]$

$\quad \Delta \leftarrow min(Remaining_{ag_1},\ \ldots,\ Remaining_{ag_n})$

$$NewState_{ag_i} \leftarrow \begin{cases} 0, & \text{if } \Delta = Remaining_{ag_i} \\ State_{ag_i} + \Delta, & \text{otherwise} \end{cases} \quad \forall i \in [1\ ;\ n]$$

$\quad$ **return** $\Delta, NewState_{ag_1},\ \ldots\ NewState_{ag_n}$

end procedure

Equivalence criterion proof sketch

Theorem (equivalence)

Assuming source clock always present (no stuttering),

$$Ag \Longrightarrow_{n \times s} Ag' \quad \textit{iff} \quad \underbrace{P^0 \twoheadrightarrow P^1 \ldots \twoheadrightarrow P^n}_{n\ times}$$

assuming $Ag \approx P^0$, we have $Ag' \approx P^n$ (term and data equivalence)

Proof.

By structural induction on $\Longrightarrow$ definition.

- Instantaneous statements: trivial equivalence.
- Temporal statement (advance): by induction on the interval duration.
- Sequential composition: by the induction hypothesis.

$\square$