Non-Simultaneity as a Design Constraint

TIME 2020

Jean Guyomarc’h, Krono-Safe
François Guerret, Krono-Safe
Bilal El Mejjiati, Krono-Safe
Emmanuel Ohayon, Krono-Safe
Bastien Vincze, Université Paris-Saclay
Alain Mérigot, Université Paris-Saclay

September 23, 2020

https://doi.org/10.4230/LIPIcs.TIME.2020.13
Invalid resource sharing between CPU cores is problematic:

- security issues (e.g. unexpected reads);
- safety issues (e.g. memory corruption).

Widespread solution: **critical sections** from non-temporized programming models (e.g. mutual exclusion).

- Lack of safety properties (e.g. possible deadlocks).
- Dynamic behavior (e.g. suboptimal predictability and determinism).
- Fine for most applications.
Safety-critical systems (e.g. airborne navigation):

- strong safety requirements (defects have consequences);
- strip dynamic behaviors (improve predictability and determinism);
- may require worst-case *timing analysis* (real-time);
- starting to use multi-cores instead of mono-cores.

Goal: **enable off-line design of temporized critical sections.**

- Fully static approach requiring no preliminary execution ("good by design").
- Suitable for safety-critical systems with "hard" timing requirements.
Motivations

System Model
A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
Motivations

System Model
A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
Sequences of computations are encompassed between temporal constraints:

- ▶️ (after): earliest start date;
- ▶️ (before): deadline;
- ▶️ (synchronization): a deadline defining an earliest start date.

Nodes denote the temporal constraints.

Arcs represent the computations that execute between two temporal constraints.
Scheduling schemes can be derived from TCAs.

Enable concrete execution from a temporal specification.

Original work by Lemerre et al. (2010).
Overview

Motivations

System Model

A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
New semantics added to TCAs: **simultaneity**.

- Applied to windows of computations that execute within a known and bounded time span (delimited by synchronous events).
- Two windows of computations are simultaneous if their execution may overlap in time.

Figure: Execution with simultaneity
Clear distinction between:

- **model of computation**: embodies the design space (based on TCA); and
- **model of execution**: embodies the run-time of the designed application.

Existing work aim at enforcing non-simultaneity as a constraint of the model of execution (i.e. when generating scheduling schemes).

- Adds another constraint to schedulability analysis.
- Difficult fallback when no feasible solution are found.
Our approach to verify non-simultaneity properties is based on the model of computation.

- Disentangle non-simultaneity analysis from schedulability analysis.
- Both problems can be solved independently.
- Clear separation between temporal design and execution times of computations.
System Model:

- **Time-Constrained Application**: fixed set of (a subset of) TCAs that share a same unique base clock.
- **Synchrony**: clock ticks occur simultaneously on all TCAs.
- **Exclusion groups**: fixed set of *temporal transitions* (i.e. named arcs) that shall not overlap in time.
- **Isochrony**: all TCA can be re-written to exhibit arcs of the same length.

**Non-simultaneity** is ensured by the safety property that, for each exclusion group of a time-constrained application, their temporal transitions never overlap in time.
Example 1 - Simple

One exclusion group: \( \{\tau_A_1, \tau_B_2, \tau_B_4\} \).

Superposition of “unfolded” graphs of TCAs A and B.
Superposition of "unfolded" graphs of TCAs A and B

One exclusion group: \( \{ \tau_{A_2}, \tau_{B_1}, \tau_{B_4} \} \).

Hints that the non-simultaneity property holds. Can we be sure?
Example 2 - Is it a non-simultaneous system?
Overview

Motivations

System Model
A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
Motivations

System Model
   A temporized model of computation
   Simultaneity in a model of computation

Validating the simultaneity constraints
   Formalization of the problem
   Determination of dates of reachability
   Intersection of dates

Conclusions and perspectives
Goal: determine dates of reachability for every temporal transition.

Figure: Finite automaton formalizing the set of dates at which state C is reachable.
Goal: determine dates of reachability for every temporal transition.

An isochronous TCA can be understood as a finite automaton, where:

▶ each state (but the initial one) can be marked as accepting;
▶ the increment of time, associated to every arc, can be seen as the symbol of a unary alphabet;
▶ the set of dates at which a state can be reached is given by the length of the words that lead to this state.

The set of dates at which a state can be reached is expressed as the regular language over a unary alphabet accepted by the automaton where only this state is marked as accepting.
Goal: **determine dates of reachability for every temporal transition.**

Each regular **unary language** can be represented as the union of a finite number of arithmetic progressions of the form \( \{ c + dk | k \in \mathbb{N} \} \), where:

- \( c \in \mathbb{N} \) is the offset;
- \( d \in \mathbb{N} \) is the period.

These are the sets of **word lengths**, and therefore are **dates**.
Motivations

System Model
- A temporized model of computation
- Simultaneity in a model of computation

Validating the simultaneity constraints
- Formalization of the problem
- Determination of dates of reachability
- Intersection of dates

Conclusions and perspectives
Original algorithm designed by Sawa (2013).

- Used to determine the dates at which **one state** reachable.
- Time complexity (for one automaton): $O(n^2(n + m))$.
- **As-is**, yields a total time complexity in $O(n^3(n + m))$.

Algorithm has been tailored to preserve the original complexity when processing $n - 1$ times the same "core" automaton where only the single accepting state changes.
Overview

Motivations

System Model
A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
Reminder - find overlapping arcs

Non-Simultaneity as a Design Constraint
An empty intersection of dates implies the non-simultaneity property holds. With:

- \( G \): exclusion group (set of temporal transitions).
- \( \mathcal{D}_\tau \): dates at which \( \tau \) is activated (set of arithmetic progressions).

Solving linear diophantine equation \( \alpha x + \beta y = \gamma \).

- \((\alpha, \beta, \gamma) \in \mathbb{N}^3\) given by the values of arithmetic transitions.
- Solution in \( \mathbb{Z}^2 \) iff. \( \gcd(\alpha, \beta) \mid \gamma \).
- We are interested in solutions in \( \mathbb{N}^2 \) (dates).
- Here, if we have solutions in \( \mathbb{Z}^2 \), there also exist infinite solutions in \( \mathbb{N}^2 \).
For each pair of temporal transitions in $G$, if there are no solution to the linear diophantine equation, then the intersection of dates is empty.

- Arcs originating from a state are reachable at this set of dates.
- Intersection of dates is empty $\Rightarrow$ non-simultaneity within $G$
Overview

Motivations

System Model
A temporized model of computation
Simultaneity in a model of computation

Validating the simultaneity constraints
Formalization of the problem
Determination of dates of reachability
Intersection of dates

Conclusions and perspectives
Conclusions

- **Model of computation** based on TCA to express **non-simultaneity** as a design constraint.
- Express a **safety property** over parallel systems, ensuring that litigious sequences of computations can **never run simultaneously**.
- **Robust**, standalone verification technique based on a formalization of reachable dates.

Help build systems with non-simultaneity as a design constraint, enabling safe resources sharing from the ground up.

- Perspectives: help in the identification of critical transitions.
Let $D_a = \{c_a + d_a k | k \in \mathbb{N}\}$ and $D_b = \{c_b + d_b k | k \in \mathbb{N}\}$ for set of dates for temporal transitions $a$ and $b$ in $G$. $D_a \cap D_b = \emptyset$ iff. the linear diophantine equation $\alpha x + \beta y = \gamma$ has a solution, with:

- $(x, y) \in \mathbb{Z}^2$;
- $\alpha = d_a$;
- $\beta = -d_b$;
- $\gamma = c_b - c_a$. 

Intersection of dates

Non-Simultaneity as a Design Constraint
Solution in $\mathbb{Z}^2$ iff. $\gcd(\alpha, \beta) \mid \gamma$.

- $D_a$ and $D_b$ have in common an infinite set of dates, since for any solution $(x_0, y_0)$ the set of solutions $\{(x_0 + d_b k, y_0 + d_a k) | k \in \mathbb{Z}\}$ can always be built.

- This set of solutions in $\mathbb{Z}^2$ contains an infinite number of pairs where both members are in $\mathbb{N}^2$.

This general form has simplifications when one (or both) set of dates are singletons.
References
