Semantics foundations of PsyC based on synchronous Logical Execution Time

Fabien Siron, Dumitru Potop-Butucaru, Robert de Simone, Damien Chabrol, Amira Methni

Semantics foundations of PsyC based on synchronous Logical Execution Time

Fabien Siron
Krono-Safe
Université Côte d'Azur, Inria
France
fabien.siron@krono-safe.com

Dumitru Potop-Butucaru
Robert de Simone
dumitru.potop_butucaru@inria.fr
robert.de_simone@inria.fr
Inria
France

Damien Chabrol
Amira Methni
damien.chabrol@krono-safe.com
amira.methni@krono-safe.com
Krono-Safe
France

ABSTRACT

Task models for Real-Time Scheduling (RTS) and Synchronous Reactive (SR) languages are two prominent classes of formalisms for the design and analysis of time-critical embedded systems. Task models allow providing deadlines, periods, or other such kinds of interval time boundaries that make the system description fit for schedulability analysis. Synchronous reactive languages use logical clocks to be activation condition triggers in languages providing programmability. We consider here synchronous LET (sLET) extensions that intend to re-use notions of logical clocks and logical time, for the purpose of providing schedulability boundaries. As its name indicates, sLET borrows deeply from Logical Execution Time ideas, where timing dimensions are all provided at logical design time, but they extend asynchronous events as in xGiotto with SR-inspired programmability and "first-class citizen" logical clock constructs. Our work results in a two-level semantics of the programming language PsyC. The benefits are to reuse techniques from both RTS and SR. Big-step RTS models provide inputs for task model schedulability analysis and implementation. Meanwhile, SR small-step models provide methodological tools to view any events as a time base (logical clock) and verification technologies (but they do not consider the WCET of tasks to be kept within time boundaries by the scheduling). We show the semantic equivalence of those two semantics at visible time interval boundaries.

CCS CONCEPTS

• **Computer systems organization** → **Real-time languages**; **Real-time system specification**.

KEYWORDS

Multiform Logical Time, Synchronous languages, Logical Execution Time, Real-Time Systems

ACM Reference Format:
Fabien Siron, Dumitru Potop-Butucaru, Robert de Simone, Damien Chabrol, and Amira Methni. 2023. Semantics foundations of PsyC based on synchronous Logical Execution Time. In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3587495

CPS-IoT Week Workshops '23, May 09–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0049-1/23/05…$15.00
https://doi.org/10.1145/3576914.3587495

1 INTRODUCTION

Real-Time Systems have to handle time in a predictable manner. Specified temporal requirements from a system specification should still hold in the final system. Thus, time must be considered from the very start of the design. However, exact computation durations are usually not available in early design phases as they depend on the target. In answer, various formalisms based on the Multiform Logical Time concept have been introduced to abstract [4] those target-dependent durations (expressed in physical time) with logical time constraints (expressed in logical time). Then, specific analysis, usually called schedulability analysis (or time safety analysis) can be used to fill the gap between logical and physical time, that is, they ensure that physical computations satisfy their logical time constraints. In RTS models this usually translates into the fact that real-time constraints are specified as *requirements*, while worst-case execution time (WCET) of tasks is provided as guarantees. Schedulability analysis then breaks down in satisfying the contract that WCET guarantees imply real-time constraints. Although the PsyC language defined later includes WCET modeling and scheduling resolution, it is out of the scope of the current paper, which focuses on its specification expressiveness combining influences from RTS, SR and LET.

The synchronous approach introduced a discretized abstraction of time based on logical clocks in which computations and reactions happen in discrete atomic instants, and so, logically instantaneously [5]. While multiple logical clocks may be used for specification expressiveness of sophisticated event-based timing patterns, the traditional operational semantics of synchronous languages usually imposes to expand the behaviors on a unique parent clock. Consequently, compilation of synchronous languages may suffer from the "Long Task Problem" [12]. When different tasks of different rhythms are compiled, physical execution time variability is usually limited to a common cycle rate. More recently, the *Logical Execution Time* (LET) paradigm [13] has been introduced to give a compromise between the strong expressiveness of the synchronous approach and the efficiency of traditional task scheduling. For that, LET mandates to specify the actual logical duration a task has to fulfill based on a uniform pseudo-physical time. This forms LET intervals in which communications can only happen on its bounds. Inputs can only be consulted at the start of the interval and outputs are displayed to other tasks at the end. Consequently, as communication can only be made at predefined instants, LET ensures the *temporal determinism* property.

Previous work has introduced early results on the definition of synchronous LET (sLET) as being a variation of LET based on

the Multiform Logical Time approach [19]; it allows defining LET intervals with respect to multiple logical clocks. Therefore, the duration of an interval is specified as being *up to the next n^{th} tick of a clock*. It should be noted that such logical duration is not necessarily constant across all reactions as it is specified *relatively* to a clock. In the simple case where there is only one global clock used as in the original LET paradigm, all durations become constants. Hence, sLET is a generalization of the original LET paradigm. Moreover, as the sLET allows multiple valid schedules taking into account physical time that are mutually equivalent with respect to logical clocks, the classical synchronous semantics is actually one of those in which computations happen logically instantaneously at the start of sLET intervals and outputs are delayed at the end. This article goes further than our previous work in [19], introducing an equivalence relation theorem between semantics.

To illustrate this approach, this article gives in section 3 the abstract syntax and the formal semantics of PsyC, an industrial language developed by the company *Krono-Safe*, which implements the sLET paradigm. The approach may benefit from the fact that it is an industrial-scale language, with a user community of embedded engineers. The formal semantics of PsyC is then defined using two approaches, one native semantics in section 3.2 and an other semantics defined by translation into the Esterel language in section 4, thereafter called the *synchronous semantics*. Both semantics are observationally equivalent with respect to sLET interval boundaries formalized in section 4.3. Indeed, while the synchronous semantics covers a whole PsyC application, our native semantics covers only individual agents (i.e. PsyC sequential tasks). Through a discussion in section 5, this article shows how synchronous techniques such as formal verification could then be re-used for (s)LET based languages.

2 RELATED WORK

The synchronous approach has been implemented in languages such as Esterel or Lustre [5]. It is based on the Multiform Logical Time approach and thus, totally abstracts execution time to focus on logical instants, allowing both determinism and concurrency. Nonetheless, compilation can be non-trivial given non-negligible (physical) execution times, as described in the *Long Task Problem* to which some answers have been proposed [12]. Still, today, the common workaround in the industry is to slice long tasks into sub-tasks that fit in the instant period.

The Logical Execution Time approach has been introduced with the Giotto language [13], later extended with the Timing Definition Language (TDL), which allows describing applications with a fixed set of periodic tasks and some global modes mechanism. However, those two languages do not consider logical time as being anything else than a simple abstraction of physical time. Similarly to PsyC, TimedC extends C with timing primitives, although also using an abstraction of physical time [15]. xGiotto [11] is closer to our work as it extends Giotto with events handled by a mechanism called event scoping. While our sLET paradigm can express the same kind of patterns, it treats any event or time(s) basis in the same way through logical clocks.

sLET also shares similarities with the Sparse Synchronous Model (SSM) [10]; both dedicated to applications with sparse and potentially irregular computations. However, sLET still handles time with logical time units as in the Multiform Logical Time approach while SSM handles temporal expressions based on abstracted physical time units. Nonetheless, as SSM, the sLET semantics is also inspired by discrete event model such as Lingua Franca [14].

Although related, sLET also differs from k-periodic networks or N-synchronous systems [8]. These formalisms aim at providing a certain flexibility in exact execution instants based on a global timing framework in which synchrony can be relaxed. In sLET the focus is extended to placing in time behaviors with a certain duration in terms of number of instants spent in a single behavior.

3 THE PSYC LANGUAGE

3.1 Informal Description of PsyC

The industrial language PsyC is a language developed by the French company Krono-Safe [1], which provides a set of tools for the design and the integration of safety-critical real-time applications. Such applications can then be certified at the highest level of criticality for the avionic domain (DAL-A with the DO-178C standard). PsyC stands for *Parallel SYnchronous* and has been initially presented as a model based on the timed-triggered approach [9]. As stated in the introduction, previous work has introduced an early definition of the synchronous LET paradigm as being a generalization of the LET paradigm. As the modern version of PsyC can now use multiple logical clocks, it naturally implements the synchronous LET paradigm.

Similarly to Multiform Logical Time concept, time is defined through the use of logical clocks, that is, totally-ordered sequences of ticks. The PsyC language allows describing two levels of logical clocks:

- *sources* are externally-defined logical clocks, they can be mapped to a timer (i.e. pseudo-physical time) or any events (e.g. the rotation of an engine crankshaft);
- *clocks* define a sub-sampling of sources; they are defined through an affine relation $p \times c + o$ with p and o being respectively the period and the offset with respect to another *clock* or *source* c.

```
1 body start {
2   y1 = f(x1);
3   advance 2 with A;
4   y2 = g(x2);
5   advance 3 with B;
6 }
```

Listing 1: Simple PsyC example

A PsyC application is composed of multiple concurrent components, called *agents*. Each agent defines an infinite sequential behavior using a syntax based on the C language. A special statement called *advance* allows synchronizing on a tick of a given PsyC clock. They specify both the deadline of preceding code, and the activation of code following it. Moreover, following the LET semantics, the *advance* specifies the instants in which communication can happen. As an example, the Listing 1 describes an infinite loop

application	::=	*decl**
decl	::=	*source*
	|	*clock*
	|	*temporal*
	|	*agent*
source	::=	source c
temporal	::=	temporal id = v with c
clock	::=	clock $c_1 = n_1 \times c_2 + n_2$

agent	::=	agent id (starttime n with c) *body+*
body	::=	body id *stmt*
stmt	::=	id := $f(exp*)$
	|	$stmt_1$; $stmt_2$
	|	while exp do $stmt$
	|	if (exp) $stmt_2$ else $stmt_3$
	|	advance n with c
	|	next b
	|	skip

Figure 1: Abstract syntax of our PsyC subset

of two functions $f()$ and $g()$ in which the former interval takes 2 ticks of clock A while the latter takes 3 ticks of clock B.

agents can communicate with each other through dedicated deterministic communication channels. The main one, called *temporal variable*, is an implicit one-to-several real-time data flow. The task owner of the temporal variable updates this flow at a predetermined rhythm. Moreover, its value is sampled with respect to a *clock* allowing an additional sampling level between different *agents*, but we will not consider it in this article. Determinism of communication is ensured by the visibility principle, defined as follows:

- a data can only be timestamped with a date greater or equal to its corresponding deadline;
- a data can be consulted only if its timestamp is lower or equal to the current activation date;

In the Listing 1, in the first interval, $y1$ is made visible at the deadline specified by the next `advance` at line 3 while in the second interval, $x2$ should be visible from the activation specified by the preceding `advance`, also at line 3.

```
1  source source_ms;
2  clock c20ms = 20 * source_ms;
3  clock c50ms = 50 * source_ms;
4
5  agent GNC(starttime 0) {
6    consult sensors, mode; display commands;
7    body start {
8      if (mode == NOMINAL) {
9        commands = GNC(sensors);
10       advance 2 with c20ms;
11     }
12     advance 1 with c50ms;
13   }
14 }
```

Listing 2: PsyC implementation of a GNC task [6]

This mechanism, along with agent synchronization, is an implementation of the synchronous Logical Execution Time approach. As a simple example, consider the Listing 2. It implements a modified version of the conditional triggering of a Guidance, Navigation and Control System (GNC) as described in [6]. The example uses three clocks: a source clock, expected to have a period of `1 ms`, and two periodic clocks `c20ms` and `c50ms` which have a period of respectively 20 and 50 `ms`. The example shows only one agent called `GNC` which has two *consulting* inputs: `sensors` and `mode`, and one

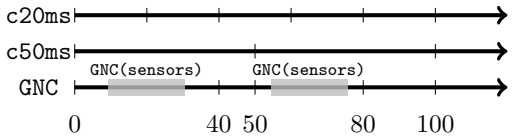

Figure 2: Possible timeline of the `GNC` task in *nominal* mode.

displaying output: `commands`. When *not in* nominal mode, the condition of line 8 is false and the agent waits for the next tick of clock `c50ms`. When *in* nominal mode, the condition of line 8 is true and the computation is constrained with a deadline of 2 ticks of clock `c20ms`. Then, afterward, the agent waits for the next tick of clock `c50ms` as shown in Figure 2.

In this article, we consider a subset of PsyC which grammar is described in Figure 1. The left column specifies the PsyC constructions defined at the application level while the right column describes the PsyC syntax at the agent level. The syntax is quite abstract with respect to the concrete one which is based on the C language. However, the objective here is to highlight the reactive part of PsyC.

3.2 Native Operational Semantics of PsyC

This section gives a native formal semantics of PsyC agents based on the Structural Operational Semantics (S.O.S.) approach introduced by Plotkin [16] and adapted later by Berry [17] to reactive languages. A more detailed version of the semantics will be made available in a research report [18]. The global approach is to successively rewrite the program such that each rewriting represents a logical instant. Transition rules (or relations) describe valid rewritings of the program. The following section describes the notations used by these rules.

3.2.1 Configuration and Transitions syntax. In this paper, a configuration (i.e. a program state) of an agent is defined by the following tuple:

$$\langle E, b, T_{output} \rangle$$

where E is the private environment of the agent, b is the identifier of the next body to be executed and T_{ouput} is the public environment of the agent. This allows to distinguish values that can be accessed by other tasks (through the temporal variable mechanism) and values that shouldn't be accessed.

Two different transitions are considered in the semantics: transitions that explicitly consume logical time and ones that are executed

in the instant. We shall call the former *temporal transitions* and the latter *non-temporal transitions*.

Non-temporal transitions are expressed using the following syntax:

$$C \vdash t \longrightarrow C' \vdash t'$$

where C (resp. C') denotes the agent configuration before (resp. after) the transition and t (resp. t') denotes the agent program before (resp. after) the transition. Temporal transitions are defined similarly:

$$C \vdash t \Longrightarrow_{n \times s} C' \vdash t'$$

where $n \in \mathbb{N}^*$ and s is a source denoting a temporal transition that has a duration of n ticks of the source s.

Moreover, to form sLET intervals, we can combine non-temporal transitions followed by a temporal one:

$$\frac{C \vdash t \longrightarrow C_1 \vdash t_1 \longrightarrow \ldots C_{n-1} \vdash t_{n-1} \Longrightarrow C_n \vdash t_n}{C \vdash t \Longrightarrow C_n \vdash t_n}$$

3.2.2 Sources and Clocks. While sources are considered as inputs in the semantics, PsyC clocks can be defined using three operators:

- *Source*(c) which gives the source from which clock c is derived;
- $\Pi(c)$ which gives the absolute period (in source ticks) of clock c with respect to its source;
- $\Phi(c)$ which gives the absolute offset (in source ticks) of clock c with respect to its source.

Additionally, to avoid keeping an unbounded date for each source, we assume that d_c gives the (current) date of c modulo its period. It is defined as $d_c = (d_s - \Phi(c)) \ mod \ \Pi(c)$ with respect to its corresponding source date d_s.

3.2.3 Semantics rules of agent. The three following rules describe the basic statements of PsyC. skip does not make time progress and is rewritten to itself while the assignment and the next are defined in the same way but update their environment accordingly.

$$C \vdash \mathsf{skip} \longrightarrow C \vdash \mathsf{skip} \qquad \text{(nothing)}$$

$$E, b, T \vdash x := f(exp+) \longrightarrow E[x \leftarrow [\![f(exp+)]\!]_E], b, T \vdash \mathsf{skip} \qquad \text{(assign)}$$

$$E, b_1, T \vdash \mathsf{next}\ b_2 \longrightarrow E, b_2, T \vdash \mathsf{skip} \qquad \text{(next)}$$

The advance statement however is more complex. It performs the two following steps:

- it makes time progress with a duration computed using the corresponding clock state d_c.
- and it updates the available outputs to the ones computed during the interval.

$$\frac{N = n \times \Pi(c) - d_c \qquad T' = \text{UpdateOutputs}(E) \quad s = \text{Source}(c)}{E, b, T \vdash \mathsf{advance}\ n\ \mathsf{with}\ c \Longrightarrow_{N \times s} E, b, T' \vdash \mathsf{skip}} \qquad \text{(advance)}$$

Based on these basic rules, control statements can be described easily. Rules if-1 and if-2 describe the rewriting of the condition statement when respectively the condition expression is true or false. Similarly, rule while-1 and while-2 describe the rewriting

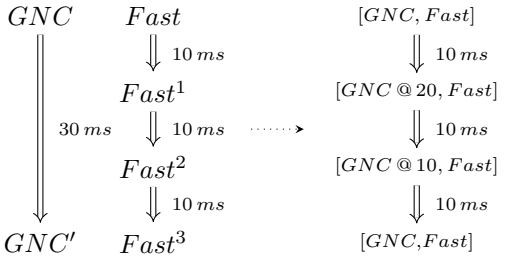

Figure 3: Agent network semantics

of the while statement. Finally, rule seq describes the rewriting of the sequence statement which rewrites its first part until it is terminated. Then, it is rewritten to its second part.

$$\frac{[\![exp]\!]_E \neq 0}{E, b, T \vdash \mathsf{if}\ (exp)\ s_1\ \mathsf{else}\ s_2 \longrightarrow E, b, T \vdash s_1} \qquad \text{(if-1)}$$

$$\frac{[\![exp]\!]_E = 0}{E, b, T \vdash \mathsf{if}\ (exp)\ s_1\ \mathsf{else}\ s_2 \longrightarrow E, b, T \vdash s_2} \qquad \text{(if-2)}$$

$$\frac{[\![exp]\!]_E = 0}{E, b, T \vdash \mathsf{while}\ exp\ \mathsf{do}\ s \longrightarrow E, b, T \vdash \mathsf{skip}} \qquad \text{(while-1)}$$

$$\frac{[\![exp]\!]_E \neq 0}{E, b, T \vdash \mathsf{while}\ exp\ \mathsf{do}\ s \longrightarrow E, b, T \vdash s\ ;\ \mathsf{while}\ exp\ \mathsf{do}\ s} \qquad \text{(while-2)}$$

$$\frac{C \vdash s_1 \longrightarrow^* C' \vdash \mathsf{skip}}{C \vdash s_1\ ;\ s_2 \longrightarrow C' \vdash s_2} \qquad \text{(seq)}$$

The statement rules defined above can then be used to describe the semantics of agents. For that, body can be rewritten to a loop containing a sequence of if statements (one for each body). The research report [18] describes the whole semantics of the PsyC language considering more advanced body constructs like early exit (e.g. equivalent to Esterel trap).

The parallel product is very similar to the synchronous one in the general case considering that the start and the end of sLET interval are synchronous instants. However, when all clocks are based on a unique source, then it can be defined as a sequence of intervals with some duration in which each agent is either in a global state (i.e. the start or the end of one of its interval) or in an intermediate state due to the overlapping with an other agent as illustrated in Figure 3.

4 SYNCHRONOUS SEMANTICS OF SLET

4.1 Description

The global idea comes from the following observation: for a given task, the sLET paradigm is equivalent to a synchronous model in which the communication model is delayed. By moving up all computations to the start of the sLET interval, the small-step semantics masks the ability to dispatch them all along. The main role of small-step semantics is therefore to allow the reuse of development environments based on SR languages. More generally, part of this work must be understood as an attempt to draw a precise semantic link between *existing* languages, emphasizing the place of synchronous LET in the making. The synchronous translation

is based on the following pattern: 1) on the activation instant, the inputs are read, the computation is done synchronously, and the output is saved in an internal state; 2) we then wait for the specified duration; and 3) finally, the outputs generated during the interval are displayed and made visible to other tasks on the last instant.

4.2 Structural Translation of PsyC to Esterel

To illustrate the approach above we propose in this article a structural translation of PsyC to the synchronous language Esterel. We chose Esterel because its syntax is very close to PsyC. It's mainly control-flow and imperative. In particular, the `advance` statement is very similar to the Esterel `await` statement. While a data-flow synchronous language such as Lustre has more available and up-to-date tools today for analysis, the translation of PsyC to Lustre is far more complicated as the control-flow needs to be flattened. However, the circuit semantics of Esterel describes a data-flow representation of its semantics, which could be represented in Lustre.

The global idea of the translation is to represent PsyC clock and source ticks with Esterel signals and PsyC temporal variables with Esterel valued signals as well as Esterel variables to handle private agent copies. Based on that, agent can be translated, almost as is, with PsyC `advance` statement translated to the Esterel `await` statement. The translation rules are as follows:
PsyC skip is just translated by:

```
nothing
```

PsyC assignation of temporal x, $x := exp$ is translated by:

```
private_temporal_x := T(exp)
```

PsyC assignation of variable x, $x := exp$ is translated by:

```
var_x := T(exp)
```

PsyC `next` statement, next b, is translated by:

```
next_body := b
```

PsyC statement's advance, of the form advance n with c and assuming `var1`, `var2`...`varN` are all the temporal variables in output of the agent, is translated by:

```
await n c;
emit temporal_var1(private_temporal_var1);
emit temporal_var2(private_temporal_var2);
...
emit temporal_varN(private_temporal_varN);
```

Control structures are translated into their equivalent form in Esterel without any difficulties, so they are not given here. The global translation scheme of an agent then directly results from the translation patterns defined above. Its module interface is composed of temporal variables (inputs and outputs) which are translated to Esterel valued signals and PsyC clocks which are translated to Esterel pure signals. As an illustration, the Listing 3 shows how the PsyC implementation of the task *GNC* described in Listing 2 is translated using the rules above.

```
1  var private_commands : t_cmd in
2    loop
3      if ?mode = NOMINAL then
4        private_commands := GNC(?sensors);
5        await 2 c20ms;
6        emit commands(private_commands);
7      end if;
8      await 1 c50ms;
9    end loop
10 end var
```

Listing 3: Esterel translation of task GNC

4.3 Equivalence relation with the native semantics

Based on the introduced PsyC semantics and its Esterel translation, this section gives an observational equivalence between them for individual agents. First of all, let us define the Esterel operational semantics coming from [17] as following:

Definition 4.1 (Esterel semantics). Given a constructive Esterel program P and a set of $data$, the semantics of its macro-step is given by the following relation:

$$P, data \xrightarrow[E_i]{E_o}\!\!\!\!\!\rightarrow P', data'$$

where E_i and E_o are respectively the input and output signal set active on the current reaction.

The equivalence theorem yields naturally from both semantics (PsyC and Esterel) and some data equivalence relation. Considering a sLET interval, the theorem states that both the PsyC and the Esterel representation have an equivalent behavior on the boundaries of the interval. The PsyC semantics yields only one transition while the Esterel semantics yields a sequence of unitary transitions (i.e. with respect to some source). If both data representations are equivalent at the start of the interval, then, they are also equivalent at the end. An extended proof will be available in the report [18].

Theorem 4.2. *For all PsyC agent p_{ag} and the Esterel translation T and for any agent transition of the form:*

$$C \vdash p_{ag} \Longrightarrow_{n \times s} C' \vdash p'_{ag}$$

Assuming, $C \approx data$ and $s \in E$, we have an equivalent sequence of Esterel transitions:

$$T(p_{ag}), data \xrightarrow[E]{}\!\!\!\!\!\rightarrow P^1, data^1 \ldots \xrightarrow[E]{}\!\!\!\!\!\rightarrow P^n, data^n$$

with $T(p'_{ag}) = P^n$ and $C' \approx data^n$

Proof. By structural induction on native rules structure □

5 DISCUSSION

As mentioned earlier, the dual semantic presentation (native and synchronous) was meant to target two distinct further design paths: real-time scheduling models lead to efficient multicore *compilation* based on schedulability analysis; synchronous reactive extensions yield verification techniques mostly based on model-checking of synchronous models. Note that, currently, each path is somehow weak for covering the other goal: SR expansion do not preserve WCET features and computations become instantaneous; RTS models do not exactly fit with existing verification formalisms and need

further encoding. We now comment briefly on experiments in these two directions.

5.1 Compilation

Currently, the efficient PsyC compilation chain restricts to *mono-source* programs, corresponding to a single global clock (as in Lustre or LET), but where affine clock down sampling is still allowed in multi-rate specifications. The compiler may then flatten all interval durations on the single source, and then checks real-time schedulability along with WCET inputs. A last issue remains when the actual interval value is data-dependent, as in the following case:

```
if (...)
  advance 1 with c10ms;
else
  advance 3 with c10ms;
```

The current compiler treats this case by looking for a uniform solution that satisfies the shortest delay (i.e. 10 ms), relaxing this constraint afterward (i.e. 30 ms when going to the "else" branch). However, we view this problem has a general issue for future work, especially in the case of multi-source systems.

5.2 Verification

Since PsyC systems cannot deadlock or support clock preemptions, verification efforts consist mainly in establishing the correctness of logical timed properties such as latencies. These are expressed as synchronous observers, in our case using the Clock Constraint Specification Language CCSL [3]. In the "simple efficient compilation case" as described above, verification models such as simple Timed Automata or even Integer Linear Programming solvers could be considered. However, in the general case the mix of boolean logic and integer duration constraints will require mixed-techniques, such as SAT/SMT or BDD. As ongoing work, we focused so far primarily on NuSMV [7] and Prover PSL model-checkers [2]. PSL is an industrial model-checker, developed by Prover, heavily used to demonstrate safety of railway systems.

Partial results are listed in Figure 4 conducted on two simple but representative PsyC case studies (both composed of 5 agents) from a benchmark suite coming from [20]: an Anti-Lock Braking System (ABS) and a Temperature Control System (TCS). The results for both tools are satisfying, even in the more complicated setting of ABS1-3 in which a slow mode involves long durations. More advanced results are expected in future work.

Requirement	NuSMV (s)	Prover PSL (s)
TCS1 *(Causality)*	4.223	3.005
TCS2 *(Periodicity)*	1.897	0.037
TCS3 *(Latency)*	0.149	0.022
ABS1 *(Causality)*	13.365	13.674
ABS2 *(Periodicity)*	0.687	11.777
ABS3 *(Latency)*	2.344	61.14

Figure 4: Verification results in seconds of a subset of requirements of two use-cases coming from [20].

6 CONCLUSION

We showed how semantic background from Synchronous Reactive Languages and Real-Time Scheduling task models could respectively allow providing a small-step and big-step meaning to PsyC, or alternately how the PsyC language could be seen as an exploitation of these notions in a commercial framework. We showed the two semantics to coincide on agent synchronization points (i.e. interval boundaries). These ideas are strongly indebted to the Logical Execution Time design philosophy, emphasizing the potential use of sporadic events as logical clocks syntactically involved in reactive language constructs. We hope this common ground will help better understand efficient multicore compilation as well as efficient timed verification and analysis in the future.

REFERENCES

[1] 2023. *Krono-Safe.* https://www.krono-safe.com/
[2] 2023. *Prover Technology.* https://www.prover.com/
[3] Charles André. 2010. *Verification of clock constraints: CCSL Observers in Esterel.* Research Report. INRIA.
[4] Charles André, Frédéric Mallet, and Marie-Agnès Peraldi-Frati. 2007. A multiform time approach to real-time system modeling; Application to an automotive system. In *2007 International Symposium on Industrial Embedded Systems.* 234–241.
[5] Albert Benveniste, Paul Caspi, Stephen Edwards, Nicolas Halbwachs, and Robert de Simone. 2003. The Synchronous Languages 12 Years Later. *Proc. IEEE* 91 (2003).
[6] Thomas Carle, Dumitru Potop-Butucaru, Yves Sorel, and David Lesens. 2015. From Dataflow Specification to Multiprocessor Partitioned Time-triggered Real-time Implementation *. *Leibniz Transactions on Embedded Systems* (2015).
[7] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. 2000. NuSMV: a new symbolic model checker. *International journal on software tools for technology transfer* 2, 4 (2000), 410–425.
[8] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. 2006. N-synchronous Kahn networks: a relaxed model of synchrony for real-time systems. *ACM SIGPLAN Notices* (2006).
[9] Vincent David, Jean Delcoigne, Evelyne Leret, Alain Ourghanlian, Philippe Hilsenkopf, and Philippe Paris. 1998. Safety Properties Ensured by the OASIS Model for Safety Critical Real-Time Systems. In *SAFECOMP.*
[10] Stephen A Edwards and John Hui. 2020. The sparse synchronous model. In *2020 Forum for Specification and Design Languages (FDL).* IEEE, 1–8.
[11] Arkadeb Ghosal, Thomas Henzinger, Christoph Kirsch, and Marco Sanvido. 2004. Event-Driven Programming with Logical Execution Times. In *International Workshop on Hybrid Systems: Computation and Control*, Vol. 2993. 357–371.
[12] Alain Girault and Xavier Nicollin. 2003. Clock-Driven Automatic Distribution of Lustre Programs. In *Embedded Software.* Springer, 206–222.
[13] Christoph Kirsch and Ana Sokolova. 2012. The Logical Execution Time Paradigm. In *Advances in Real-Time Systems.*
[14] Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A Lee. 2021. Toward a Lingua Franca for deterministic concurrent systems. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 4 (2021), 1–27.
[15] Saranya Natarajan and David Broman. 2018. Timed C: An Extension to the C Programming Language for Real-Time Systems. In *Real-Time and Embedded Technology and Applications Symposium.* Los Alamitos, CA, USA, 227–239.
[16] Gordon Plotkin. 2004. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.* (2004).
[17] Dumitru Potop-Butucaru, Stephen A Edwards, and Gérard Berry. 2007. Constructive Operational Semantics. *Compiling Esterel* (2007), 79–102.
[18] Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. 2022. *Formal Semantics of the PsyC language.* Research Report. INRIA Sophia Antipolis - Méditerranée (France). to appear.
[19] Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. 2022. The synchronous Logical Execution Time paradigm. In *ERTS 2022 - Embedded Real Time Systems.* Toulouse, France.
[20] Jagadish Suryadevara, Cristina Seceleanu, Frédéric Mallet, and Paul Pettersson. 2013. Verifying MARTE/CCSL mode behaviors using UPPAAL. In *International Conference on Software Engineering and Formal Methods.* Springer, 1–15.