When 1 Clock Is Not Enough

Gerald Lüttgen, University of York, UK
Michael Mendler, University of Bamberg, Germany

Abstract

Sometimes single–clock timed process algebras are insufficient for modelling systems in practice. For example, this is the case for systems–on–chip where a single clock cannot be physically implemented with the required accuracy, or globally–asynchronous locally–synchronous systems that are spatially distributed. This note revisits the few published approaches to multi–clock timed process algebras, namely PMC, CSA and CaSE which are all based on Nicollin and Sifakis' ATP and extend Milner's CCS. In contrast to timed automata and continuous–time process algebra, these algebras treat time as a qualitative rather than a quantitative concept: a clock is taken to be a synchronisation event with limited scope, orchestrating the computations conducted within its scope into a well–defined sequence of successive clock phases. However, PMC, CSA and CaSE differ in the choice of operators and semantic features; these shall be discussed here with the help of a novel, unified semantic framework.

1 Unified Semantic Framework

Let $\mathscr{C}$ and $\mathscr{A}$ be sets of *clocks* and *actions*, respectively, with $\mathscr{A}$ containing the special internal action τ. Clocks $\sigma \in \mathscr{C}$ synchronise in a broadcast fashion as in CSP, while actions $a, \overline{a}, \tau \in \mathscr{A}$ follow the handshake scheme of CCS.

The amount of computation sandwiched between two ticks of σ into a clock phase is variable and depends on the willingness of a process to accept a clock tick at a given state. We refer to this willingness to end a clock phase as *stability* and to the complement notion as *instability*. There are two kinds of instability: *unconditional* and *conditional* instability, which are controlled both by a process and its environment. Regarding unconditional instability, any sub–process contributing towards a particular clock phase may hold up σ for that phase, by remaining *unstable* until its part of the computation is completed. Regarding conditional instability, the clock phase may also be extended by *urgent* communications pending inside the scope of σ. Whether or not a communication is urgent for σ may depend on the local states of the participating processes. This localised priority scheme, known as *local maximal progress*, is one of the main features distinguishing synchronising clocks from the broadcast actions of CSP. Another is *time determinism*, which is common to all timed process algebras.

Our unified semantic domain $\mathscr{P}$ of multi–clock processes is defined as follows. A multi–clock process is a labelled transition system $p = (\mathscr{A}_p, \mathscr{C}_p, \mathsf{act}_p, \mathsf{clk}_p, \Sigma_p, \Pi_p) \in \mathscr{P}$ of *initial actions* $\mathscr{A}_p \subseteq \mathscr{A}$ and *initial clocks* $\mathscr{C}_p \subseteq \mathscr{C}$, the *transition relations* $\mathsf{act}_p : \mathscr{A}_p \to 2^{\mathscr{P}}$ for actions

and $\mathsf{clk}_p : \mathscr{C}_p \to \mathscr{P}$ for clocks, together with an *instability set* $\Sigma_p \subseteq \mathscr{C} \setminus \mathscr{C}_p$ and an *urgency relation* $\Pi_p : \mathscr{A}_p \to 2^{\mathscr{C}}$. Transitions may be written more suggestively as $p \xrightarrow{\gamma} p'$ when $p' \in \mathsf{act}_p(\gamma)$ or $p' = \mathsf{clk}_p(\gamma)$, where $\gamma \in \mathscr{A}_p \cup \mathscr{C}_p$. The *instability set* Σ_p comprises all clocks for which p is unstable and which are thus held up by p, i.e., $\Sigma_p \cap \mathscr{C}_p = \emptyset$. Set $\mathscr{C}_p$ includes the clocks for which p defines a deterministic initial transition. Hence, the set of clocks on which p synchronises with its environment is $\Sigma_p \cup \mathscr{C}_p$. Clocks outside of $\Sigma_p \cup \mathscr{C}_p$ are independent in that p neither stops them, nor reacts to them by changing state. The *urgency relation* Π_p associates with every action $\alpha \in \mathscr{A}_p$, a set $\Pi_p(\alpha) \subseteq \mathscr{C}$ of clocks in whose scope an occurrence of α takes place, i.e., $\sigma \in \Pi_p(\alpha)$ means that initial action α has higher priority than clock σ, so that σ is permitted to proceed only if the environment cannot communicate on α. As a special case, σ is blocked outright if $\sigma \in \Pi_p(\tau)$ for the internal action $\tau \in \mathscr{A}_p$ (a complete communication). This is a special form of unconditional instability, whence $\Pi_p(\tau) \subseteq \Sigma_p$, which is also known as *maximal progress*.

A multi–clock process algebra defines algebraic operators for specifying semantic structures $p = (\mathscr{A}_p, \mathscr{C}_p, \mathsf{act}_p, \mathsf{clk}_p, \Sigma_p, \Pi_p)$. One natural starting point is the standard syntax of CCS consisting of the nil process $\mathbf{0}$, prefixing $\alpha.p$, summation $p+q$, parallel composition $p|q$, restriction $p \setminus a$ and recursion $\mu x.\, p$. The standard operator for specifying clock transitions is the *timeout* $\lfloor p \rfloor \sigma(q)$ introduced with ATP [5]. It behaves like p for all actions and clock transitions different from σ. For σ it adds a clock transition to q (the "timeout step"), provided that p cannot engage in an urgent initial τ. All σ–transitions that may exist in p are pruned.

2 Controlling Clock Phasing

In the following we review the design decisions taken by the multi–clock process algebras PMC [1], CSA [2] and CaSE [7] and show how they fit into our unified semantic framework. The design choices relate to whether clock phasing is controlled explicitly or implicitly.

2.1 Explicit Control of Clock Phasing

Explicit control means that clock phasing is made explicit by the placement of timeout operators. In this scheme, such as employed in ATP [5] or PMC [1], clock phases are defined entirely by instability of process *state*, i.e., by way of the sets Σ_p. Clocks are stopped by default and thus cannot tick unless specified explicitly. Time progress is controlled locally by inserting clock ticks only in those (stable) states of a process where the process has finished all computations that are due to happen within the current clock phase.

As a consequence, processes are unstable for all clocks unless defined otherwise via timeouts. In particular, for action prefixes $\alpha.p$ and nil $\mathbf{0}$ one puts $\Sigma_{\alpha.p} =_{\mathrm{df}} \Sigma_{\mathbf{0}} =_{\mathrm{df}} \mathscr{C}$, whereas $\Sigma_{\lfloor p \rfloor \sigma(q)} =_{\mathrm{df}} \Sigma_p \setminus \{\sigma\}$ for timeouts $\lfloor p \rfloor \sigma(q)$. Since prefixes stop all clocks, they are called *insistent*. For pure CCS processes we have $\Sigma_p = \mathscr{C}$ and $\mathscr{C}_p = \emptyset$, while in general timed processes satisfy $\Sigma_p = \mathscr{C} \setminus \mathscr{C}_p$, where $\mathscr{C}_p$ are p's initial clock transitions specified by timeouts.

In a pure language of insistent prefixing, the urgency relations play no role and are fixed as $\Pi_p(\alpha) =_{\mathrm{df}} \emptyset$, for all $\alpha \in \mathscr{A}_p$. Since they never preempt any clock transition, actions are referred to as *patient*. In this combination of insistent prefixing with patient actions communication through actions and clocks are independent concepts.

2.2 Implicit Control of Clock Phasing

The other option is to control clock phasing implicitly, by way of *action urgency* and *maximal progress*, such as in CSA [2] and CaSE [7]. Here, it is not a decision of an individual process state if a clock is to be stopped but a feature of its interaction. A process p can adjust the urgency relation Π_p by specifying which actions are to fall within which clock regime. This is done via clock scoping and clock hiding operators.

Clock scoping and hiding. Suppose a subsystem p that runs under the regime of a clock σ is to be integrated into a larger system q, forming $p|q$. If clock σ is to run independently of the parallel context q, it needs to be decoupled. There are several solutions for making sure that σ inside p is not blocked by q. The first technique employed in PMC and CSA is to use an explicit static *ignore* operator $q\!\uparrow\!\sigma$ that adds σ–loops at all states reachable by q. This has the effect that σ cannot be used inside q since all σ–transitions are overridden. If its use is required, one may opt for *hiding* clock σ inside q, which turns σ into a non–synchronising action. This closes off q with respect to synchronisations on σ and preserve these to the outside. Hiding q/σ in CaSE [7] uses the urgent non–synchronising action τ for this, while hiding $q\langle\sigma\rangle$ in $\mathrm{CSA}^{\mathrm{ch}}_{\mathrm{att}}$ [4] introduces a new patient non–synchronising action $\imath$.

Since clock scopes Π_p are relations between initial actions and clocks, it is natural to start with a default scope at the point where actions are introduced, i.e., with prefixes. This can be managed in two ways. Firstly, we can assume that every action is maximally urgent and hence in the scope of every clock, until it is *detached* explicitly through other operators. This is done in CSA [2] where ignore $p\!\uparrow\!\sigma$ turns all initial actions patient for σ, i.e., $\Pi_{p\uparrow\sigma}(\alpha) = \Pi_p(\alpha) \setminus \{\sigma\}$ for all $\alpha \in \mathscr{A}_p$. Secondly, we can dually start with patient actions outside the scope of any clock and then use *attach* operators to bring them within the regime of a clock. This technique was introduced with $\mathrm{CSA}^{\mathrm{ch}}_{\mathrm{att}}$ [4] where the attach operator $p@\sigma$ is defined so that $\Pi_{p@\sigma}(\alpha) =_{\mathrm{df}} \Pi_p(\alpha) \cup \{\sigma\}$.

Maximal progress. The urgency relation is then used to implement the maximal progress assumption. This is essentially done via an operational rule demanding that the parallel composition $p|q$ can engage in a clock transition only (i) if both processes p, q can and (ii) if there is no handshake communication possible between some action a and its complement $\overline{a}$ within the scope of σ, i.e., $\sigma \notin \Pi_p(a) \cap \Pi_p(\overline{a})$. For a single clock this is the classic form of *global* maximal progress employed in TPL [3], whereas for multiple clocks with clock scoping we obtain the refined form of *local* maximal progress introduced with CSA [2].

Pure systems with urgent actions and maximal progress, such as TPL and CSA, typically take a *relaxed* view on process stability. A process p is stable for each clock unless it is preempted by a τ–transition in its scope, i.e., $\Sigma_p = \emptyset$ in case $\tau \notin \mathscr{A}_p$, and $\Sigma_p = \Pi_p(\tau)$ otherwise. In TPL and CSA the instability set Σ_p is modelled indirectly by clock *self–loops*.

In settings with maximal progress and employing a semantics based on observational equivalence, τ–loops can be used to stop clocks in specific process states [4]. Alternatively, this can be done using customised *time–stop* operators whose semantics directly influence the instability set Σ_p. Time stops, if judiciously inserted into a process, can be used for modelling the violation of real–time constraints in system verification, as shown in [7].

3 Summary and Challenges

The above design choices leave a large design space for defining multi–clock process algebras. Only a few points within this design space have so far been studied:

PMC	=	insistent prefixing + ignore + no maximal progress	[1]
CSA	=	relaxed prefixing + ignore + local maximal progress	[2]
$\mathrm{CSA}_{\mathrm{att}}^{\mathrm{ch}}$	=	relaxed prefixing + attach + hiding + local maximal progress	[4]
CaSE	=	relaxed prefixing + time stop + hiding + global maximal progress	[7]

Technical achievements include (i) a complete axiomatisation of strong bisimulation for regular processes in PMC and CSA, (ii) a fully–abstract characterisation of *observational congruence* in PMC, CSA and CaSE, and (iii) a complete axiomatisation of observational congruence for *finite* processes in PMC. Work on similar results for CaSE will be included in [6].

Future work should complete these theories by providing axiomatic characterisations of the observational congruences for *regular* processes. The challenge here is that the standard completeness–proof technique (Milner) is not generally applicable. This is because in the presence of time determinism, unguarded recursion can only be eliminated in the very special case of global maximal progress. For example, the PMC process $\mu x.\lfloor\tau.\lfloor\tau.x\rfloor\sigma(b.\mathbf{0})\rfloor\sigma(a.\mathbf{0})$ cannot be expressed without unguarded recursion [1].

A second challenge lies in exploring the sketched design space more fully and in general terms, rather than theories for particular points in this space. This should take into account ongoing efforts in the area of synchronous programming, such as defining a semantics for multi–clock Esterel.

Last, but not least, semantics other than those founded on bisimulation, such as *failure* semantics or *testing* semantics, should to be studied for multi–clock process algebras. To the best of our knowledge, this has not yet been done. The challenge here is to lift the approaches incorporated in Timed CSP and TPL from a single clock to multiple clocks, which is a task that can profit from the unified semantic framework sketched in this note.

References

[1] H.R. Andersen and M. Mendler. An asynchronous process algebra with multiple clocks. In *ESOP '94*, vol. 788 of *LNCS*, pp. 58–73.

[2] R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In *CONCUR '97*, vol. 1243 of *LNCS*, pp. 166–180.

[3] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[4] M. Kick. Modelling synchrony and asynchrony with multiple clocks. Master's thesis, University of Passau, 1999.

[5] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.

[6] B. Norton. *A Process Algebraic Theory for Synchronous Software Composition*. PhD thesis, University of Sheffield. To be submitted in June 2005.

[7] B. Norton, G. Lüttgen, and M. Mendler. A compositional semantic theory for synchronous component-based design. In *CONCUR 2003*, vol. 2761 of *LNCS*, pp. 461–476.