An Algebraic Theory of Multiple Clocks

Rance Cleaveland[1,⋆], Gerald Lüttgen[2,⋆⋆], and Michael Mendler[3,⋆⋆⋆]

[1] Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA, e-mail: rance@eos.ncsu.edu
[2] Fakultät für Mathematik und Informatik, Universität Passau, D–94030 Passau, Germany, e-mail: {luettgen,mendler}@fmi.uni-passau.de

Abstract. This paper develops a temporal process algebra, CSA, for reasoning about *distributed* systems that involve *qualitative* timing constraints. It is a conservative extension of Milner's CCS that combines the idea of *multiple clocks* from the algebra PMC with the assumption of *maximal progress* familiar from timed process algebras such as TPL. Using a typical class of examples drawn from hardware design, we motivate why these features are useful and in some cases necessary for modeling and verifying distributed systems. We also present fully-abstract behavioral congruences based on the notion of strong bisimulation and observational equivalence, respectively. For temporal strong bisimulation we give sound and complete axiomatizations for several classes of processes.

1 Introduction

Process algebras [10,12] provide a well-studied framework for modeling and verifying concurrent systems [6,8]. These theories typically consist of a simple language with a rigorously defined semantics mapping terms to labeled transition systems. They also usually support equational reasoning as a basis for system verification: an equivalence on processes is defined that equates systems on the basis of their observable behavior, and this relation is used to relate specifications, which describe desired system behavior, and implementations. In order to support *compositional reasoning*, researchers have typically concentrated on equivalences that are also congruences for the given languages.

Traditionally, process algebras have been developed with a view toward modeling the nondeterministic behavior of concurrent and distributed systems. More recent work has incorporated other aspects of system behavior, including real time [1,9,13,14,16]. Most of this later work, however, has been devoted to modeling centralized, as opposed to distributed systems; the real-time work, in particular, has (implicitly or explicitly) focused on systems with a single clock. In this

⋆ Research supported by NSF/DARPA grant CCR-9014775, NSF grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, NSF Young Investigator Award CCR-9257963, NSF grant CCR-9402807, and AFOSR grant F49620-95-1-0508.

⋆⋆ Research support partly provided by the German Academic Exchange Service under grant D/95/09026 (Doktorandenstipendium HSP II / AUFE).

⋆⋆⋆ Author supported by the *Deutsche Forschungsgemeinschaft*.

paper we present a temporal process algebra, called CSA (Calculus for Synchrony and Asynchrony), which is aimed at modeling distributed, timed systems that contain a number of independent clocks. Technically, CSA extends Hennessy and Regan's TPL [9] with constructs from PMC [1] that enable the management of multiple clocks. In doing so we replace the global notion of *maximal progress* found in TPL with a local one that is more appropriate for distributed systems. This combination of features yields a convenient formalism for modeling distributed timed systems; it also introduces semantic subtleties the solutions to which constitute the body of this paper.

It should be noted that clocks in CSA are intended to capture qualitative timing constraints, in which it is not the absolute occurrence time or duration of actions that is constrained but their relative ordering and sequencing with respect to clocks. This contrasts with other theories of real-time, which typically focus on precisely measuring the time that elapses between different system events. In this respect CSA follows the philosophy advocated by Nicollin and Sifakis [14] and others, as well as synchronous languages such as ESTEREL [3].

2 Motivation

One standard hardware architecture consists of a number of cooperating synchronous systems which are distributed over different modules, e.g. chips or boards. Typically, each module possesses its own central clock to update all of its registers in a synchronous fashion. The clocks of different modules are independent, so that the modules change their states asynchronously with respect to each other. Such architectures are also called *globally-asynchronous, locally-synchronous* [4]. They not only arise through physical distribution, e.g. in computer networks where different sites cannot be synchronized by the same clock, but are also typical for heterogeneous real-time applications. A concrete example is the Brüel & Kjær 2145 Vehicle Signal Analyzer reported in [2].

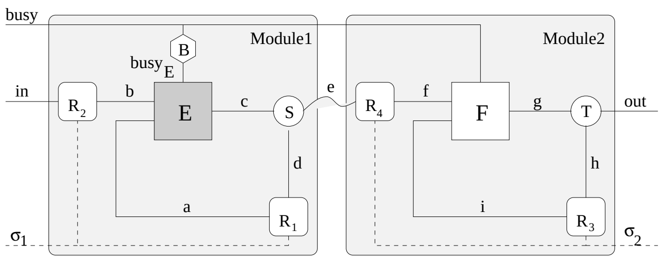

Fig. 1. A globally-asynchronous, locally-synchronous system

A generic example for a globally-asynchronous, locally-synchronous system is depicted in Fig. 1, where solid lines represent communication channels and dashed lines symbolize channels of clocks. Both modules, `Module1` and `Module2`, have their own local clocks σ_1 and σ_2, respectively, and their own function blocks, registers, and buffers. In every clock cycle, the function block `E` computes a new value from the current values of the state registers $\mathtt{R}_1$ and $\mathtt{R}_2$, obtained through channels a and b, and outputs it on channel c to be propagated further through `S` to register $\mathtt{R}_1$ via channel d and to the environment via channel e. External input enters the computation through channel *in*. Register $\mathtt{R}_2$ stores the most recent input value from the environment and, thus, ensures that `E` never has to wait for the environment. Component `S` and busy buffer `B` are explained later. `Module2` operates in a similar fashion, with its external input being fed by the output of `Module1`.

The example clearly suggests how we can benefit from a concept of multiple clocks to model real-world distributed systems. The question of what is an adequate notion of clock leads us to the second characteristic of our process algebra: the maximal progress assumption. The fundamental feature of the clock σ_1 is that it must tick only after the previous clock cycle has been completed, i.e. after the function block `E` has finished its internal computations and the new value has arrived at register $\mathtt{R}_1$. Otherwise, the value stored into $\mathtt{R}_1$ upon the tick of σ_1 is undefined. If `Module1` has more than one register reading d they may all take different values, and an inconsistent state may arise. The maximal progress assumption guarantees that a clock tick is delayed until all internal computations or communications have come to an end.

To take account of distribution, the maximal progress property must be "localized" and imposed on every module independently. For instance, the clock σ_1 of `Module1` must be able to tick as soon as the previous cycle of σ_1 has been completed, regardless of the state of `Module2`, which operates asynchronously with respect to `Module1`. In contrast, the traditional global version of maximal progress would imply that *all* clocks have to wait for *all* computations to complete, whence the system would be globally synchronous.

The combined concept of multiple clocks and local maximal progress is quite powerful. It supports *horizontal* and *vertical* forms of synchronous decomposition that correspond to temporal abstractions with synchronized and nested scales of time. The horizontal form has already been made explicit above. The vertical form arises when we implement, say, the function `E` of `Module1` as a whole synchronous system in itself, with its own local clock (see Sect. 4).

3 Syntax and Semantics

In this section we define the syntax and semantics of our language CSA, which is inspired by the process algebras TPL [9] and PMC [1], which both descend from ATP [14]. The syntax of CSA is essentially the same as in PMC; it extends Milner's CCS [12] with a *timeout operator* and a clock *ignore operator*. The timeout operator occurs in other real-time process algebras [1,9] and was originally introduced in ATP, where it is called *unit-delay*. The ignore operator originates with PMC, though here it is a primitive operation, not derivable as in PMC.

The semantical framework of CSA is based on a notion of transition system that involves two kinds of transitions, *action* transitions and *clock* transitions, modeling two different mechanisms of synchronization and communication in distributed systems. Action transitions, like in CCS, are local handshake communications in which two processes synchronize to take a joint state change together. A clock represents the progress of time, which manifests itself in a recurrent global synchronization event, the clock transition, in which all process components that are in the regime, or in the scope, of this clock are forced to take part.

In CSA action and clock transitions are not orthogonal concepts that can be specified independently from each other, but are connected in line with the following intuitions: (1) A clock records the *progress* of time, with two successive clock events marking an interval of time. (2) The *passage* of time is determined by internal computations that are within the regime of the clock. This yields the very specific semantic connection between actions and clocks, known as the *maximal progress assumption* [9,16]. Maximal progress usually is read as the condition that "communications must occur whenever they are possible," i.e. a process cannot be intercepted by a clock as long as it is able to perform internal computations.

The last feature of CSA is *clock scoping*. Since we are dealing with distributed systems and multiple clocks, it is natural to localize the maximal progress assumption with respect to clocks and to limit the scope of clocks. A communication that reaches outside the scope is an external computation that must be considered asynchronous with respect to the clock. Different clocks, which represent different local views of time, may have disjoint, overlapping, or nested scopes, and amount to different abstractions of time.

Note that clocks in our setting are abstract in the sense that we do not prejudice any particular way to interpret them. We are free to think of a clock as the ticking of a global real-time watch measuring off absolute process time in constant or non-constant intervals, as the system clock of a synchronous processor, as a recurrent external interrupt, or as the completion signal of a distributed synchronization protocol. Thus, clocks can be used as a general and flexible means for bundling asynchronous behavior into sequenced intervals, and to give local meaning to the notions of "before," "after," and "state."

3.1 Syntax of CSA

Formally, let Λ be a countable set of action labels, not including the so-called *silent* or *internal* action τ. With every $a \in \Lambda$ we associate a *complementary action* $\overline{a}$. We define $\overline{\Lambda} \stackrel{\mathrm{df}}{=} \{\overline{a} \mid a \in \Lambda\}$ and take $\mathcal{A}$ to denote the set of all actions $\Lambda \cup \overline{\Lambda} \cup \{\tau\}$, where $\tau \notin \Lambda \cup \overline{\Lambda}$. Complementation is lifted to $\Lambda \cup \overline{\Lambda}$ by defining $\overline{\overline{a}} = a$. As in CCS [12] an action a communicates with its complement $\overline{a}$ to produce the internal action τ. We let $a, b, \dots$ range over $\Lambda \cup \overline{\Lambda}$ and $\alpha, \beta, \dots$ over $\mathcal{A}$. Besides the set $\mathcal{A}$ of actions, CSA is parameterized in a set $\mathcal{T} = \{\sigma, \sigma', \rho, \dots\}$

of *clocks*. The syntax of our language is defined by the following BNF

$$P \ ::= \ \mathbf{0} \mid x \mid \alpha.P \mid P+P \mid P|P \mid P[f] \mid P\setminus L \mid P\!\uparrow\!\sigma \mid \lfloor P\rfloor\sigma(P) \mid \mu x.P$$

where x is a *variable* taken from a countably infinite set of variables $\mathcal{V}$, $f:\mathcal{A}\to\mathcal{A}$ is a *finite relabeling*, and $L\subseteq\mathcal{A}\setminus\{\tau\}$ is a *restriction set*. For convenience, we define $\overline{L}\stackrel{\text{df}}{=}\{\overline{a}\,|\,a\in L\}$. A finite relabeling satisfies the properties $f(\tau)=\tau$, $f(\overline{a})=\overline{f(a)}$, and $|\{\alpha\,|\,f(\alpha)\neq\alpha\}|<\infty$. Moreover, $\uparrow$ is called the (static) *ignore operator* and $\lfloor\cdot\rfloor\sigma(\cdot)$ the *timeout operator*. Further, we use the standard definitions for the *sort* of a term P, $\mathsf{sort}(P)\subseteq\Lambda\cup\overline{\Lambda}$, *static* and *dynamic* operators, *free* and *bound* variables, *open* and *closed* terms, and *contexts*. A process variable is called *guarded* in a process term if each occurrence of the variable is in the scope of a prefix or of the second argument of a timeout (see below). We refer to closed and guarded terms as *processes*. Let $\mathcal{P}$ be the set of all processes, ranged over by P,Q,R, and denote syntactic equality on $\mathcal{P}$ by $\equiv$. We extend the timeout operator to sequences of clocks by defining $\lfloor P\rfloor\stackrel{\text{df}}{=}P$ and $\lfloor P\rfloor\sigma_1(Q_1)\ldots\sigma_n(Q_n)\stackrel{\text{df}}{=}\lfloor\lfloor P\rfloor\sigma_1(Q_1)\ldots\sigma_{n-1}(Q_{n-1})\rfloor\sigma_n(Q_n)$. We often further abbreviate sequences $\sigma_1\ldots\sigma_n$ of clocks by $\boldsymbol{\sigma}$ and sequences $Q_1\ldots Q_n$ of processes by $\mathbf{Q}$. In this vein, $\lfloor P\rfloor\boldsymbol{\sigma}(\mathbf{Q})$ is a shorthand for $\lfloor P\rfloor\sigma_1(Q_1)\ldots\sigma_n(Q_n)$.

Table 1. Clock scoping

$\mathrm{I}_\sigma(\alpha.P)\stackrel{\text{df}}{=}\{\alpha\}$	$\mathrm{I}_\sigma(\mu x.P)\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P[\mu x.P/x])$
$\mathrm{I}_\sigma(P+Q)\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P)\cup\mathrm{I}_\sigma(Q)$	$\mathrm{I}_\sigma(P\|Q)\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P)\cup\mathrm{I}_\sigma(Q)\cup\{\tau\,\|\,\mathrm{I}_\sigma(P)\cap\overline{\mathrm{I}_\sigma(Q)}\neq\emptyset\}$
$\mathrm{I}_\sigma(P[f])\stackrel{\text{df}}{=}\{f(\alpha)\,\|\,\alpha\in\mathrm{I}_\sigma(P)\}$	$\mathrm{I}_\sigma(P\setminus L)\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P)\setminus(L\cup\overline{L})$
$\mathrm{I}_\sigma(\lfloor P\rfloor\sigma'(Q))\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P)$	$\mathrm{I}_\sigma(P\!\uparrow\!\sigma')\stackrel{\text{df}}{=}\mathrm{I}_\sigma(P)\quad$ if $\sigma\neq\sigma'$

3.2 Semantics of CSA

The *operational semantics* of a CSA process $P\in\mathcal{P}$ is given by a labeled transition system $\langle\mathcal{P},\mathcal{A}\cup\mathcal{T},\longrightarrow,P\rangle$ where $\mathcal{P}$ is the set of states, $\mathcal{A}\cup\mathcal{T}$ the alphabet, $\longrightarrow$ the transition relation, and P the start state. We refer to transitions with labels in $\mathcal{A}$ as *action transitions*, and to those with labels in $\mathcal{T}$ as *clock transitions*. The transition relation $\longrightarrow\subseteq\mathcal{P}\times(\mathcal{A}\cup\mathcal{T})\times\mathcal{P}$ for CSA is defined in Table 2 using operational rules. For the sake of simplicity, let us use γ for a representative of $\mathcal{A}\cup\mathcal{T}$, and write $P\stackrel{\gamma}{\to}P'$ instead of $\langle P,\gamma,P'\rangle\in\longrightarrow$ and $P\stackrel{\gamma}{\to}$ for $\exists P'\in\mathcal{P}.\,P\stackrel{\gamma}{\to}P'$.

To ensure maximal progress the operational rules involve side conditions on *initial action sets*. Beside the usual definition of $\mathrm{I}(P)$ for the initial action set of a process P – where $\mathrm{I}(P\!\uparrow\!\sigma)$ and $\mathrm{I}(\lfloor P\rfloor\sigma(Q))$ are given by $\mathrm{I}(P)$ – we define the set $\mathrm{I}_\sigma(P)\subseteq\mathrm{I}(P)$ of all initial actions of P *within the scope* of the clock σ

as the smallest set satisfying the equations in Table 1. Note that the sets $\mathrm{I}(P)$ and $\mathrm{I}_\sigma(P)$ are well-defined since all processes are closed and guarded. Moreover, $\mathrm{I}_\sigma(P) = \mathrm{I}(P)$ whenever P does not contain any ignore operator. Finally, we define initial visible action sets by $\mathbb{I}(P) \stackrel{\mathrm{df}}{=} \mathrm{I}(P) \setminus \{\tau\}$ and $\mathbb{I}_\sigma(P) \stackrel{\mathrm{df}}{=} \mathrm{I}_\sigma(P) \setminus \{\tau\}$.

Table 2. Operational semantics for CSA

Act	$\dfrac{-}{\alpha.P \xrightarrow{\alpha} P}$	tAct	$\dfrac{-}{a.P \xrightarrow{\sigma} a.P}\ \sigma \in \mathcal{T}$
Sum1	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	tNil	$\dfrac{-}{\mathbf{0} \xrightarrow{\sigma} \mathbf{0}}\ \sigma \in \mathcal{T}$
Sum2	$\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$	tSum	$\dfrac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P + Q \xrightarrow{\sigma} P' + Q'}$
Rel	$\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$	tRel	$\dfrac{P \xrightarrow{\sigma} P'}{P[f] \xrightarrow{\sigma} P'[f]}$
Res	$\dfrac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L}\ \alpha \notin L \cup \overline{L}$	tRes	$\dfrac{P \xrightarrow{\sigma} P'}{P \setminus L \xrightarrow{\sigma} P' \setminus L}$
Com1	$\dfrac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$	tCom	$\dfrac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\sigma} Q'}{P\|Q \xrightarrow{\sigma} P'\|Q'}\ \tau \notin \mathrm{I}_\sigma(P\|Q)$
Com2	$\dfrac{Q \xrightarrow{\alpha} Q'}{P\|Q \xrightarrow{\alpha} P\|Q'}$	tIgn1	$\dfrac{-}{P{\uparrow}\sigma \xrightarrow{\sigma} P{\uparrow}\sigma}$
Com3	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$	tIgn2	$\dfrac{P \xrightarrow{\sigma'} P'}{P{\uparrow}\sigma \xrightarrow{\sigma'} P'{\uparrow}\sigma}\ \sigma \neq \sigma'$
Ign	$\dfrac{P \xrightarrow{\alpha} P'}{P{\uparrow}\sigma \xrightarrow{\alpha} P'{\uparrow}\sigma}$	tTO1	$\dfrac{-}{\lfloor P \rfloor \sigma(Q) \xrightarrow{\sigma} Q}\ \tau \notin \mathrm{I}_\sigma(P)$
TO	$\dfrac{P \xrightarrow{\alpha} P'}{\lfloor P \rfloor \sigma(Q) \xrightarrow{\alpha} P'}$	tTO2	$\dfrac{P \xrightarrow{\sigma'} P'}{\lfloor P \rfloor \sigma(Q) \xrightarrow{\sigma'} P'}\ \sigma \neq \sigma'$
Rec	$\dfrac{P[\mu x.P/x] \xrightarrow{\alpha} P'}{\mu x.P \xrightarrow{\alpha} P'}$	tRec	$\dfrac{P[\mu x.P/x] \xrightarrow{\sigma} P'}{\mu x.P \xrightarrow{\sigma} P'}$

The operational semantics for action transitions extends the one of CCS by rules dealing with the ignore and the timeout operator. More precisely, the process $\alpha.P$ may engage in action α and then behave like P. The *summation operator* $+$ denotes nondeterministic choice, i.e. the process $P + Q$ may either behave like P or Q. The *restriction operator* $\setminus L$ prohibits the execution of actions in $L \cup \overline{L}$ and thus permits the scoping of actions. $P[f]$ behaves exactly as P where ordinary actions are renamed by the *relabeling* f. The process $P|Q$ stands for the *parallel composition* of P and Q according to an interleaving semantics with synchronized communication on complementary actions resulting in the internal action τ. The processes $P{\uparrow}\sigma$ and $\lfloor P \rfloor \sigma(Q)$ behave like P for action transitions. The timeout operator disappears as soon as P engages in an action transition, thereby observably changing its state. Finally, $\mu x.P$ denotes *recursion*, i.e. $\mu x.P$ is a process which behaves as a distinguished solution of the equation $x = P$.

With respect to clock transitions the operational semantics is set up such that if $\tau \in \mathrm{I}_\sigma(P)$ then the clock σ is inhibited. We refer to this kind of pre-emption as *local maximal progress*. Its local nature lies in the facts that, in general, $\mathrm{I}_\sigma(P) \neq \mathrm{I}(P)$ and that the sets $\mathrm{I}_\sigma(P)$ may be different for different clocks. Accordingly, the process $\alpha.P$ may idle for each clock σ whenever $\alpha \neq \tau$. Time has to proceed equally on both sides of summation, i.e. $P+Q$ can engage in a clock transition and, thus, delay the nondeterministic choice if and only if both P and Q can engage in the clock tick. Also both argument processes of a parallel composition have to synchronize on clock transitions according to Rule tCom. Its side condition implements local maximal progress and can alternatively be written as $\mathbb{I}_\sigma(P) \cap \overline{\mathbb{I}_\sigma(Q)} = \emptyset$, i.e. there is no pending communication between P and Q on an action that lies in the scope of σ. Regarding the ignore operator, the process $P\!\uparrow\!\sigma$ is capable of performing a σ-loop, i.e. P *ignores* σ, regardless if $\tau \in \mathrm{I}_\sigma(P)$ or not. This is consistent with our definition $\mathrm{I}_\sigma(P\!\uparrow\!\sigma) \stackrel{\mathrm{df}}{=} \emptyset$, which means that none of the initial actions of P is in the scope of clock σ. Thus, $\uparrow$ is actually not a *scoping* but a *co-scoping* operator, i.e. all processes are assumed to be within the scope of all clocks unless explicitly excluded using an ignore. Using co-scoping instead of scoping simplifies the operational rules when dealing with multiple local clocks, since the traditional rules for summation and parallel composition with respect to timed transitions need not be changed. Moreover, the process $\lfloor P \rfloor \sigma(Q)$ can perform a σ-transition to Q provided P cannot engage in an internal action which is in the scope of clock σ. Since a clock transition too represents an observable change of state, the timeout operator disappears as soon as P engages in such a transition. This intuition is the same as for the corresponding unit-delay operator in ATP [14]. For multiple clocks this leads to rule tTO2 in which the timeout for σ is dropped when a different σ' ticks. The idea is that the ordering of the σ and σ' ticks is observable and the first one determines the state change. Note, however, that by using recursion to insert explicit clock idling persistent versions of the timeout can be obtained.

The operational semantics for CSA possesses several important properties. First, the summation and the parallel operator of CSA are associative and commutative. Second, a process can always engage in a clock transition provided it cannot perform an internal action which is in the scope of this clock. Formally, $\tau \notin \mathrm{I}_\sigma(P)$ implies $P \stackrel{\sigma}{\rightarrow}$. Third, the semantics satisfies the *local maximal progress* and the *local time determinacy* property. Both are generalizations of the well-known maximal progress and time determinacy properties, for global time, to a local notion of time in terms of multiple local clocks. Local maximal progress states that $P \stackrel{\sigma}{\rightarrow}$ implies $\tau \notin \mathrm{I}_\sigma(P)$. Time determinacy, which is a common feature of all real-time process algebras, states that processes react in a deterministic way to clock ticks, reflecting the intuition that mere progress of time does not resolve choices. Formally, $P \stackrel{\sigma}{\rightarrow} P'$ and $P \stackrel{\sigma}{\rightarrow} P''$ implies $P' \equiv P''$.

It is not difficult to see that CSA is a conservative extension of TPL if we drop TPL's *undefined* process Ω, which has been introduced to define a semantics based on *testing* [7]. Restricting $\mathcal{T}$ to a single clock, say $\mathcal{T} = \{\sigma\}$, and dropping the ignore operator, gives us precisely the syntax and operational semantics of TPL. Note that in this single-clock version of CSA the timed prefixing of TPL can be derived as $\sigma.P \overset{\text{df}}{=} \lfloor \mathbf{0} \rfloor \sigma(P)$. Moreover, CCS [12] can be identified as the subcalculus of CSA which is obtained by defining $\mathcal{T} = \emptyset$.

Finally, it is worth mentioning that CSA allows us to express *clock constraints* by processes. For example, if we want to relate the speeds of clocks we can do so by composing the system under consideration in parallel with a process expressing the corresponding constraint. As this issue is not central to this paper, however, we do not address it further.

4 Example (revisited)

Now we formally describe the example presented in Sect. 2 in our algebra CSA. We refer to Fig. 1 and assume that we refine the function module `E` by a complete synchronous subsystem with its own local clock $\rho \in \mathcal{T} \overset{\text{df}}{=} \{\sigma_1, \sigma_2, \rho\}$, as depicted in Fig. 2. At top level the structure of the overall system `System` is $(\mathtt{Module1} \uparrow \sigma_2 \mid \mathtt{Module2} \uparrow \sigma_1 \uparrow \rho) \setminus \{e\}$. This captures the asynchronous parallel composition of `Module1` and `Module2`. The ignore operators $\uparrow \sigma_1$ and $\uparrow \sigma_2$ are introduced so as to make both modules ignore each other's clocks. The clock ρ is internal to `Module1`, whence `Module2` ignores it with $\uparrow \rho$. The channel e connecting both modules is internal to `System`, and thus restricted by $\setminus\{e\}$.

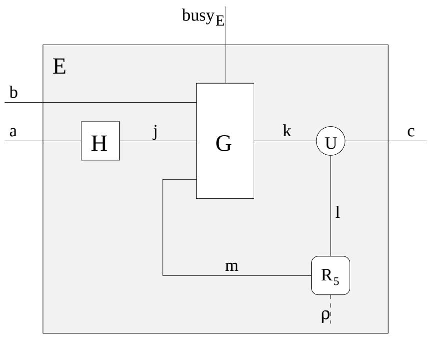

Fig. 2. Component E (refined)

At the next structural level we break up the two asynchronous modules, each of which is a synchronous subsystem. Let us look at the internals of `Module1`, which is $(\mathtt{E} \mid (\mathtt{R}_1 \mid \mathtt{R}_2 \mid \mathtt{S} \mid \mathtt{B}) \uparrow \rho) \setminus \{a, b, c, d, \mathtt{busy_E}\}$. It is a parallel composition of a function block `E`, state registers $\mathtt{R}_1$ and $\mathtt{R}_2$, a busy buffer `B`, and a special fork component `S`. These components communicate via the channels $\{a, b, c, d, \mathtt{busy_E}\}$ which are internal to `Module1` and hence are restricted away. The parallel subsystem $(\mathtt{R}_1 \mid \mathtt{R}_2 \mid \mathtt{S} \mid \mathtt{B}) \uparrow \rho$ ignores the clock ρ, making ρ local to the function block `E`. This function block finally is decomposed to the synchronous system $\mathtt{E} \overset{\text{df}}{=} (\mathtt{H} \mid \mathtt{G} \mid \mathtt{R}_5 \mid \mathtt{U}) \setminus \{j, k, l, m\}$.

Now let us turn our attention to the synchronous subsystem `E` (cf. Fig. 2). Block `E` should read its inputs from channels a and b, take an arbitrary number of cycles of clock ρ to compute a result that is then passed over to the environment on output channel c. The algorithm for this computation is contained in function block `G`. The register $\mathtt{R}_5$ stores the intermediate values, i.e. represents the local state on which the algorithm works. The function block `H` may be a preprocessing stage, and the component $\mathtt{U}$ is assumed to be a fork process that distributes the data on its input k to outputs c and l, i.e. $\mathtt{U} \stackrel{\mathrm{df}}{=} \mu x_0 .\, k.\overline{l}.(\mu x_1 .\, \overline{c}.x_0 + \tau.x_1)$. The τ-loop indicates waiting for external output through channel c, which will inhibit the local clock ρ as long as the output has not been delivered.

In CSA the register might be specified as $\mathtt{R}_5 \stackrel{\mathrm{df}}{=} \mu x.\, \lfloor l.x \rfloor \rho(\mu y.\, \overline{m}.x + l.y)$. It continuously accepts an updating input on channel l, and when the clock ρ ticks it changes its state to $\mu y.\, \overline{m}.\mathtt{R}_5 + l.y$. In this state the output action $\overline{m}$ starts the next computation cycle, while the l-loop makes sure that the register is always input enabled. If our channels would carry real data then the new value injected into the next cycle with $\overline{m}$ would be the last value read in from input l *before* the clock tick. This means that the l-loop *after* the clock must not change the registered value. From the value supplied by $\overline{m}$ after each clock tick, the function blocks compute a next state value that eventually ends up being latched into $\mathtt{R}_5$ again through l. Then the cycle is completed and ρ may tick again. To indicate the simplest case of a function block let $\mathtt{H}$ be the trivial iteration $\mathtt{H} \stackrel{\mathrm{df}}{=} \mu x.\, a.\tau.\overline{j}.x$. Accordingly, $\mathtt{H}$ first reads an external input from a, then performs an internal computation represented by τ, and finally outputs on j, whereupon it returns to the initial state. For a function block with more than one input more complicated input-output pattern are possible. For instance, we want $\mathtt{G}$ to implement an algorithm that reads its inputs b and j and initial state m and then computes its function in a number of steps, storing intermediate results in register $\mathtt{R}_5$. The following CSA process specifies such a behavior:

$$\mathtt{G} \stackrel{\mathrm{df}}{=} \mu x_0 .\, m.j.b.\mathtt{G}_1 \qquad \mathtt{G}_1 \stackrel{\mathrm{df}}{=} \mu x_1 .\, \tau.(\overline{k}.x_0 + \overline{k}.\mathtt{G}_2) \qquad \mathtt{G}_2 \stackrel{\mathrm{df}}{=} \mu x_2 .\, m.x_1 + \overline{\mathtt{busy}_{\mathtt{E}}}.x_2 \ .$$

$\mathtt{G}$ consumes the register value on m and the result of function $\mathtt{H}$ on j, reads an input from the environment through channel b, and then passes to $\mathtt{G}_1$, which is the actual computation state. After a finite amount of internal computation, indicated by the leading τ, a result is computed that may be output with action $\overline{k}$. Now two possibilities arise: either the algorithm is completed, in which case we pass back to the initial state $\mathtt{G}$ (variable x_0), or we carry on with another clock cycle, in which case we move to state $\mathtt{G}_2$. This decision, of course, depends on the data, but since we do not consider values, we model this as a nondeterministic choice. In $\mathtt{G}_2$ we have reached a final state of a single ρ clock cycle, but only an intermediate state of the algorithm implemented by $\mathtt{E}$. The $\overline{\mathtt{busy}_{\mathtt{E}}}$-loop signals this to the environment of $\mathtt{E}$ in order to inhibit the outer clock σ_1. In the intermediate state $\mathtt{G}_2$ of the algorithm we do not need to read new input data, but only get the next state by m and continue with $\mathtt{G}_1$.

The above specification example shows how the ignore operator can be used to localize clocks, and the timeout operator to model the synchronized updating of registers. Maximal progress controls when a clock tick is possible and when it is delayed. The only way to stop a clock is by internal divergence, e.g. arising from a feed-back loop that does not contain any clocked register. In this case of a violated design rule, the functional blocks produce divergence and the local clock of that module is never able to tick. This relationship between design error, internal divergence, and conceptual time stop is very natural for synchronous hardware, which stresses the adequacy of the maximal progress model.

5 Temporal Strong Bisimulation

The transition systems produced by the operational semantics are a rather fine-grained semantic view of processes. Therefore, we develop a semantic theory based on bisimulation [12]. Our aim in this section is to characterize the largest congruence contained in the "naive" strong bisimulation [12] where we treat clocks as actions.

Definition 1. A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is called *naive strong bisimulation* if for every $\langle P, Q\rangle \in \mathcal{R}$, $\gamma \in \mathcal{A} \cup \mathcal{T}$, the following condition holds: $P \stackrel{\gamma}{\rightarrow} P'$ implies $\exists Q'.\, Q \stackrel{\gamma}{\rightarrow} Q'$ and $\langle P', Q'\rangle \in \mathcal{R}$. We write $P \sim_{\mathrm{n}} Q$ if there exists a naive strong bisimulation $\mathcal{R}$ such that $\langle P, Q\rangle \in \mathcal{R}$.

It is straightforward to establish that $\sim_{\mathrm{n}}$ is the *largest* naive strong bisimulation and that $\sim_{\mathrm{n}}$ is an equivalence relation. Unfortunately, $\sim_{\mathrm{n}}$ is *not* a congruence. The reason is that the transition system of a process P does not contain the clock scoping information $\mathrm{I}_\sigma(P)$ needed to determine the transition system of $C[P]$ for all contexts $C[X]$. For instance, $a.\mathbf{0} \sim_{\mathrm{n}} a.\mathbf{0}\!\uparrow\!\sigma$ but $a.\mathbf{0} \,|\, \overline{a}.\mathbf{0} \not\sim_{\mathrm{n}} (a.\mathbf{0}\!\uparrow\!\sigma) \,|\, \overline{a}.\mathbf{0}$ since the right-hand process can do a σ-transition while the corresponding σ-transition of the left-hand process is pre-empted due to maximal progress. In this example $a.\mathbf{0}$ and $a.\mathbf{0} \uparrow \sigma$ have identical transition systems but different clock scoping, effecting different pre-emption of clock transitions in parallel contexts. In order to find the largest congruence contained in $\sim_{\mathrm{n}}$ we have to take into account the scope of clocks.

Definition 2. A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *temporal strong bisimulation* if for every $\langle P, Q\rangle \in \mathcal{R}$, $\alpha \in \mathcal{A}$, and $\sigma \in \mathcal{T}$ the following conditions hold.

1. $P \stackrel{\alpha}{\rightarrow} P'$ implies $\exists Q'.\, Q \stackrel{\alpha}{\rightarrow} Q'$ and $\langle P', Q'\rangle \in \mathcal{R}$.
2. $P \stackrel{\sigma}{\rightarrow} P'$ implies $\mathbb{I}_\sigma(Q) \subseteq \mathbb{I}_\sigma(P)$ and $\exists Q'.\, Q \stackrel{\sigma}{\rightarrow} Q'$ and $\langle P', Q'\rangle \in \mathcal{R}$.

We write $P \simeq Q$ if $\langle P, Q\rangle \in \mathcal{R}$ for some temporal strong bisimulation $\mathcal{R}$.

The definition of $P \simeq Q$ requires not only that all clock transitions in P and Q must match each other, but also that with respect to all these clocks σ the pre-emption potential of both P and Q must be identical, i.e. $\mathbb{I}_\sigma(P) = \mathbb{I}_\sigma(Q)$.

Theorem 3. *The relation $\simeq$ is the* largest congruence *contained in $\sim_{\mathrm{n}}$.*

Axiomatic Characterization

In this section, we provide an axiomatization of $\simeq$ for *regular* processes, i.e. a class of finite-state processes that do not contain static operators inside recursion. In order to develop the axiomatization, it is convenient to add a new ignore operator $\downarrow$ to CSA, called *dynamic ignore*, which is compositional with respect to temporal strong bisimulation. Its semantics is defined by the following operational rules.

$$\text{DIgn}\ \frac{P \stackrel{\alpha}{\to} P'}{P{\downarrow}\sigma \stackrel{\alpha}{\to} P'} \qquad \text{tDIgn1}\ \frac{-}{P{\downarrow}\sigma \stackrel{\sigma}{\to} P{\downarrow}\sigma} \qquad \text{tDIgn2}\ \frac{P \stackrel{\sigma'}{\to} P'}{P{\downarrow}\sigma \stackrel{\sigma'}{\to} P'{\downarrow}\sigma}\ \sigma \neq \sigma'$$

Moreover, we extend the definition of $\mathrm{I}_\sigma(\cdot)$ by $\mathrm{I}_\sigma(P{\downarrow}\rho) \stackrel{\text{df}}{=} \mathrm{I}_\sigma(P{\uparrow}\rho)$.

A process $P \in \mathcal{P}$ is called *regular* if it is built from nil, prefix, summation, timeout, dynamic ignore, variables, and recursion. We say that P is *rs-free*, where *rs* abbreviates *recursion through static operators*, if every subterm $\mu x.\, Q$ of P is regular. Finally, a process P is *finite* if it does not contain the recursion operator.

Table 3. Axiomatization of $\simeq$ (Part I)

(A1)	$t + u = u + t$	(B1)	$\lfloor\lfloor t\rfloor\sigma(u)\rfloor\sigma(v) = \lfloor t\rfloor\sigma(v)$
(A2)	$t + (u + v) = (t + u) + v$	(B2)	$\lfloor\lfloor t\rfloor\sigma(u)\rfloor\sigma'(v) = \lfloor\lfloor t\rfloor\sigma'(v)\rfloor\sigma(u) \quad \sigma \neq \sigma'$
(A3)	$t + t = t$	(B3)	$\lfloor t\rfloor\sigma(u) + \lfloor v\rfloor\sigma(w) = \lfloor t + v\rfloor\sigma(u + w)$
(A4)	$t + \mathbf{0} = t$		
(D1)	$\mathbf{0}[f] = \mathbf{0}$	(C1)	$\mathbf{0} \setminus L = \mathbf{0}$
(D2)	$(\alpha.t)[f] = f(\alpha).(t[f])$	(C2)	$(\alpha.t) \setminus L = \mathbf{0} \quad \alpha \in L \cup \overline{L}$
(D3)	$(t + u)[f] = t[f] + u[f]$	(C3)	$(\alpha.t) \setminus L = \alpha.(t \setminus L) \quad \alpha \notin L \cup \overline{L}$
(D4)	$(\lfloor t\rfloor\sigma(u))[f] = \lfloor t[f]\rfloor\sigma(u[f])$	(C4)	$(t + u) \setminus L = t \setminus L + u \setminus L$
		(C5)	$(\lfloor t\rfloor\sigma(u)) \setminus L = \lfloor t \setminus L\rfloor\sigma(u \setminus L)$

Now, we turn to the axioms for temporal strong bisimulation. We write $\vdash P = Q$ if P can be rewritten to Q by using the axioms in the Tables 3, 4, and 5 which are sound for arbitrary CSA processes. Many axioms are identical to the ones presented in [1] for PMC. Axioms (L1)–(L8) and (I8) deal with the new dynamic ignore operator, where Axiom (I8) captures the relationship between the static and the dynamic ignore operator. Moreover, the expansion axiom, Axiom (E), has been adapted for our algebra. The new semantic extension compared to PMC is reflected by Axioms (P1), (P2), (S1), and (S2). Equations (P1) and (P2) deal with the (local) pre-emptive power of τ, and Equations (S1) and (S2) make the implicit idling of clocks explicit.

Axioms (L5) and (L6) allow us to introduce $P{\downarrow}T$, where $T = \{\sigma_1, \dots \sigma_n\}$ is a finite set of clocks, as a shorthand for $P \downarrow \sigma_1 \dots \downarrow \sigma_n$. The same is true if we replace the dynamic ignore operator by the static one (cf. Axioms (I5) and (I6)). Thus, the simplifying notation in Axioms (L8), (P1), (P2), and (E) is justified.

In order to prove the completeness of our axiomatization with respect to temporal strong bisimulation we introduce a notion of *normal form* that is based on the following definition. A term t is called *in summation form* if it is of the shape $t \equiv \lfloor\sum_{i\in I}(\sum_{j\in J_i} \alpha_i.x_{ij}){\downarrow}T_i\rfloor\boldsymbol{\sigma}(\mathbf{y})$, where $\sum$ is the indexed version of $+$,

Table 4. Axiomatization of $\simeq$ (Part II)

(I1)	$\mathbf{0}\!\uparrow\!\sigma = \mathbf{0}$	(I5)	$(t\!\uparrow\!\sigma)\!\uparrow\!\sigma = t\!\uparrow\!\sigma$
(I2)	$(t+u)\!\uparrow\!\sigma = t\!\uparrow\!\sigma + u\!\uparrow\!\sigma$	(I6)	$(t\!\uparrow\!\sigma)\!\uparrow\!\rho = (t\!\uparrow\!\rho)\!\uparrow\!\sigma$
(I3)	$(t \setminus L)\!\uparrow\!\sigma = (t\!\uparrow\!\sigma) \setminus L$	(I7)	$(\lfloor t \rfloor \rho(u))\!\uparrow\!\sigma = \lfloor t\!\uparrow\!\sigma \rfloor \rho(u\!\uparrow\!\sigma)\sigma((\lfloor t \rfloor \rho(u))\!\uparrow\!\sigma)$
(I4)	$(t[f])\!\uparrow\!\sigma = (t\!\uparrow\!\sigma)[f]$	(I8)	$(\alpha.t)\!\uparrow\!\sigma = (\alpha.(t\!\uparrow\!\sigma))\!\downarrow\!\sigma$
(L1)	$\mathbf{0}\!\downarrow\!\sigma = \mathbf{0}$	(L5)	$(t\!\downarrow\!\sigma)\!\downarrow\!\sigma = t\!\downarrow\!\sigma$
(L2)	$(t+u)\!\downarrow\!\sigma = t\!\downarrow\!\sigma + u\!\downarrow\!\sigma$	(L6)	$(t\!\downarrow\!\sigma)\!\downarrow\!\rho = (t\!\downarrow\!\rho)\!\downarrow\!\sigma$
(L3)	$(t \setminus L)\!\downarrow\!\sigma = (t\!\downarrow\!\sigma) \setminus L$	(L7)	$(\lfloor t \rfloor \rho(u))\!\downarrow\!\sigma = \lfloor t\!\downarrow\!\sigma \rfloor \rho(u\!\downarrow\!\sigma)\sigma((\lfloor t \rfloor \rho(u))\!\downarrow\!\sigma)$
(L4)	$(t[f])\!\downarrow\!\sigma = (t\!\downarrow\!\sigma)[f]$	(L8)	$(\alpha.t)\!\downarrow\!T + (\alpha.u)\!\downarrow\!T' = (\alpha.t + \alpha.u)\!\downarrow\!(T \cap T')$
(S1)	$\mathbf{0} = \lfloor \mathbf{0} \rfloor \sigma(\mathbf{0})$	(P1)	$(\tau.t)\!\downarrow\!T + u\!\downarrow\!\sigma = (\tau.t)\!\downarrow\!T + u \quad \sigma \notin T$
(S2)	$\alpha.t = \lfloor \alpha.t \rfloor \sigma(\alpha.t)$	(P2)	$\lfloor (\tau.t)\!\downarrow\!T + u \rfloor \sigma(v) = (\tau.t)\!\downarrow\!T + u \quad \sigma \notin T$

the x_{ij} and $\mathbf{y} = y_1, y_2, \ldots, y_n$ are process variables, $\boldsymbol{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_n$ clocks, and $\alpha_i \in \mathcal{A}$. The index sets I and J_i, $i \in I$, are assumed to be finite, possibly empty, initial intervals of the natural numbers. By definition, $\sum_{i \in \emptyset} t_i \equiv \mathbf{0}$ is in summation form.

Table 5. Axiomatization of $\simeq$ (Part III)

(E) Let $t \equiv \lfloor \sum_{i \in I} (\sum_{j \in J_i} \alpha_i.t_{ij})\!\downarrow\!T_i \rfloor \boldsymbol{\sigma}(\mathbf{v})$, $u \equiv \lfloor \sum_{i \in I} (\sum_{k \in K_i} \alpha_i.u_{ik})\!\downarrow\!U_i \rfloor \boldsymbol{\sigma}(\mathbf{w})$ and $\boldsymbol{\sigma} \equiv \sigma_1, \ldots, \sigma_n$ .Then $t \,|\, u = \lfloor r \rfloor \sigma_1(v_1 \,|\, w_1) \ldots \sigma_n(v_n \,|\, w_n)$ where
$r \equiv \sum_{i \in I} ((\sum_{j \in J_i} \alpha_i.(t_{ij} \,|\, u))\!\downarrow\!T_i + (\sum_{k \in K_i} \alpha_i.(t \,|\, u_{ik}))\!\downarrow\!U_i) +$
$\sum_{i,i' \in I} \{ \sum_{j \in J_i} \sum_{k \in K_i} (\tau.(t_{ij} \,|\, u_{i'k}))\!\downarrow\!(T_i \cup U_{i'}) \mid \alpha_i = \overline{\alpha}_{i'} \}$

(R0) $\mu x.t = \mu y.(t[y/x])$ $\quad$ y does not occur in t
(R1) $\mu x.t = t[\mu x.t/x]$
(R2) $\vdash u = t[u/x]$ implies $\vdash u = \mu x.t$ $\quad$ x guarded in t

The Expansion Axiom (E) in Table 5 shows how we can eliminate the parallel composition operator. The timeout part of $t|u$ is defined componentwise for each clock. The summation part r splits up into two summands. The summand in the first line considers action transitions performed by one side alone, while the summand in the second line deals with the communication case. The dynamic ignore operators are determined naturally by our clock-scoping semantics. Specifically, the dynamic ignore set $\downarrow T_i \cup U_{i'}$ leaves the internal action τ in the scope of a clock σ if and only if $\sigma \notin T_i$ and $\sigma \notin U_{i'}$, i.e. σ is connected to each of the communicating actions α_i and $\overline{\alpha}_{i'}$.

Definition 4. The term $t \equiv \lfloor \sum_{i \in I} (\sum_{j \in J_i} \alpha_i.x_{ij})\!\downarrow\!T_i \rfloor \boldsymbol{\sigma}(\mathbf{y})$ in summation form is in *normal form* if it satisfies the following: (1) $\forall i \in I.\ \alpha_i = \tau$ iff $i = 0$,

(2) $\forall i, i' \in I.\ i \neq i'$ implies $\alpha_i \neq \alpha_{i'}$, and (3) $\forall \sigma \in \mathcal{T}.\ J_0 \neq \emptyset \ \wedge\ \sigma \notin T_0$ implies $\sigma \notin \boldsymbol{\sigma} \ \wedge\ \forall i \in I. \sigma \notin T_i$.

The completeness proof adapts Milner's technique [11] in characterizing recursive processes uniquely by systems of equations in normal form. A *normal form equation system*, into which every regular process can be unrolled, is a sequence $\langle y_i = t_i \mid i < n \rangle$ $(n \geq 1)$ of equations such that all t_i are in normal form and the free variables of all t_i are among $\mathbf{y}$.

Theorem 5. *For regular processes P and Q we have: $\vdash P = Q$ iff $P \simeq Q$.*

The completeness result can be extended to the class of rs-free processes by eliminating the static operators, using the Expansion Axiom (E) to get rid of parallel composition, eliminating restriction by Axioms (C1)–(C5), (L3), and (I3), and renaming by Axioms (D1)–(D4), (L4), and (I4). Finally, leaving out Axioms (R0)–(R2) for recursion, we obtain a complete axiomatization for *finite* processes. The corresponding completeness proof follows the standard lines (cf. [12]). It is based on a notion of normal form for terms, which corresponds to the one in Definition 4 where we substitute the variables x_{ij} and y_k by terms that are again in normal form.

6 Temporal Observational Congruence

The semantic congruence developed in the previous section is too fine for verifying systems in practice since it requires that two equivalent systems must match each other's internal transitions exactly. Consequently, we want to abstract from internal actions and develop a semantic congruence from the point of view of an external observer.

Observational equivalence is a notion of bisimulation in which any sequence of internal τ's may be skipped. For $\gamma \in \mathcal{A} \cup \mathcal{T}$ we define $\hat{\gamma} \stackrel{\mathrm{df}}{=} \epsilon$ if $\gamma = \tau$ and $\hat{\gamma} \stackrel{\mathrm{df}}{=} \gamma$, otherwise. Further, let $\stackrel{\epsilon}{\Rightarrow} \stackrel{\mathrm{df}}{=} \stackrel{\tau}{\rightarrow}^*$ and $P \stackrel{\gamma}{\Rightarrow} Q$ iff there exist processes R and S such that $P \stackrel{\epsilon}{\Rightarrow} R \stackrel{\gamma}{\rightarrow} S \stackrel{\epsilon}{\Rightarrow} Q$. Carrying over Milner's weak bisimulation [12] to CSA naively would suggest the following definition.

Definition 6. A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *naive temporal weak bisimulation* if for every $\langle P, Q \rangle \in \mathcal{R}$, $\gamma \in \mathcal{A} \cup \mathcal{T}$, the following condition holds: $P \stackrel{\gamma}{\rightarrow} P'$ implies $\exists Q'. Q \stackrel{\hat{\gamma}}{\Rightarrow} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$. We write $P \approx_{\mathrm{n}} Q$ if there exists a naive temporal weak bisimulation $\mathcal{R}$ such that $\langle P, Q \rangle \in \mathcal{R}$.

It is not surprising that $\approx_{\mathrm{n}}$ is not a congruence, for the same reason that weak bisimulation equivalence is not a congruence for CCS. In contrast to CCS, however, $\approx_{\mathrm{n}}$ is not even a congruence for parallel composition. The problem is that, again, the relation fails to account for clock scoping. The following refinement of the above definition is needed for the static contexts.

Definition 7. A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *temporal weak bisimulation* if for every $\langle P, Q \rangle \in \mathcal{R}$, $\alpha \in \mathcal{A}$, and $\sigma \in \mathcal{T}$ the following conditions hold.

1. $P \xrightarrow{\alpha} P'$ implies $\exists Q'.\, Q \stackrel{\hat{\alpha}}{\Rightarrow} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$.
2. $P \xrightarrow{\sigma} P'$ implies
   $\exists Q', Q'', Q'''.\, Q \stackrel{\epsilon}{\Rightarrow} Q'' \xrightarrow{\sigma} Q''' \stackrel{\epsilon}{\Rightarrow} Q'$, $\mathbb{I}_\sigma(Q'') \subseteq \mathbb{I}_\sigma(P)$, and $\langle P', Q' \rangle \in \mathcal{R}$.

We write $P \approx Q$ if $\langle P, Q \rangle \in \mathcal{R}$ for some temporal weak bisimulation $\mathcal{R}$.

Proposition 8. *The relation $\approx$ is a congruence with respect to prefixing and the static* CSA *operators. It is characterized as the largest congruence contained in $\approx_{\mathrm{n}}$, in the subalgebra of* CSA *induced by these operators.*

In order to identify the largest equivalence contained in $\approx_{\mathrm{n}}$ that is also a congruence for the other dynamic operators, the summation fix of CCS is not sufficient due to the special nature of clock transitions.

Definition 9. A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *temporal observational congruence* if for every $\langle P, Q \rangle \in \mathcal{R}$, $\alpha \in \mathcal{A}$, and $\sigma \in \mathcal{T}$ the following conditions hold:

1. $P \xrightarrow{\alpha} P'$ implies $\exists Q'.\, Q \stackrel{\alpha}{\Rightarrow} Q'$ and $P' \approx Q'$.
2. $P \xrightarrow{\sigma} P'$ implies $\mathbb{I}_\sigma(Q) \subseteq \mathbb{I}_\sigma(P)$ and $\exists Q'.\, Q \xrightarrow{\sigma} Q'$ and $\langle P', Q' \rangle \in \mathcal{R}$.

We write $P \approxeq Q$ if $\langle P, Q \rangle \in \mathcal{R}$ for some temporal observational congruence $\mathcal{R}$.

Theorem 10. *The relation $\approxeq$ is the* largest congruence *contained in $\approx_{\mathrm{n}}$.*

For details as well as the proofs of our results we refer the reader to [5].

7 Conclusions

We have presented the temporal process algebra CSA with multiple clocks and a local maximal progress assumption. CSA is closely related to the process algebras TPL and PMC which both are inspired by ATP. Whereas TPL does not deal with multiple clocks, and the semantics of PMC does not ensure maximal progress, CSA combines both features under the special consideration of the distribution of systems. By means of a generic example we have demonstrated the utility of CSA as a semantic framework for dealing with synchrony and asynchrony in which we can express various levels of time and synchronization. We have developed a fully-abstract semantic theory based on the notion of bisimulation. Alternative characterizations of our behavioral relations (see [5]) allow us to adapt standard partition refinement algorithms [15] for their computation.

Moreover, our results show that CSA is a conservative extension of TPL not only in terms of operational semantics but also in terms of strong and weak bisimulation. This means that our main theorems also apply to TPL. In particular, specializing Theorem 10 to the TPL fragment yields a characterization of observational congruence for TPL.

Future work will especially focus on two aspects. On the one hand, CSA should be implemented in the Concurrency Workbench of North Carolina [6], an automatic verification tool. On the other hand, an axiomatic characterization of temporal observational congruence may be interesting since it would support a better understanding of the underlying semantic theory and simplify a comparison with other temporal process algebras.

References

1. H.R. Andersen and M. Mendler. An asynchronous process algebra with multiple clocks. In D. Sannella, editor, *European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 58–73. Springer-Verlag, 1994.
2. H.R. Andersen and M. Mendler. Describing a signal analyzer in the process algebra PMC — A case study. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *Theory and Practice of Software Development, TAPSOFT'95*, volume 915 of *Lecture Notes in Computer Science*, pages 620–635. Springer-Verlag, 1995.
3. G. Berry and G. Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.
4. D.M. Chapiro. Reliable high-speed arbitration and synchronization. *IEEE Transaction on Computers*, C-36(10):1251–1255, October 1987.
5. R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. Technical report, North Carolina State University, Raleigh, NC, USA, 1997. To appear.
6. R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In R. Alur and T. Henzinger, editors, *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, New Brunswick, New Jersey, July 1996. Springer-Verlag.
7. R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1983.
8. W. Elseaidy, J. Baugh, and R. Cleaveland. Verification of an active control system using temporal process algebra. *Engineering with Computers*, 12:46–61, 1996.
9. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.
10. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1985.
11. R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.
12. R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.
13. F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415, Amsterdam, August 1990. Springer-Verlag.
14. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.
15. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.
16. W. Yi. CCS + time = an interleaving model for real time systems. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Automata, Languages and Programming (ICALP '91)*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228, Madrid, July 1991. Springer-Verlag.