Abstraction of Clocks in Synchronous Data-flow Systems [⋆]

Albert Cohen[1], Louis Mandel[2], Florence Plateau[2], and Marc Pouzet[2][3]

[1] INRIA Saclay - Ile-de-France, Orsay, France
[2] LRI, Univ. Paris-Sud 11, Orsay, France and INRIA Saclay
[3] Institut Universitaire de France

Abstract. Synchronous data-flow languages such as LUSTRE manage infinite sequences or *streams* as basic values. Each stream is associated to a *clock* which defines the instants where the current value of the stream is present. This clock is a type information and a dedicated type system — the so-called clock-calculus — statically rejects programs which cannot be executed synchronously. In existing synchronous languages, it amounts at asking whether two streams have the same clocks and thus relies on clock equality only. Recent works have shown the interest of introducing some relaxed notion of synchrony, where two streams can be composed as soon as they can be synchronized through the introduction of a finite buffer (as done in the SDF model of Edward Lee). This technically consists in replacing typing by subtyping. The present paper introduces a simple way to achieve this relaxed model through the use of *clock envelopes*. These clock envelopes are set of concrete clocks which are not necessarily periodic. This allows to model various features in real-time embedded software such as bounded jitter as found in video-systems, execution time of real-time processes and scheduling resources or the communication through buffers. We present the algebra of clock envelopes and its main theoretical properties.

Keywords: Real-time systems; Synchronous languages; Kahn Process Networks; Compilation; Semantics; Type-systems.

1 Introduction

Synchronous data-flow languages such as LUSTRE [1] have been introduced in the 80's for the implementation of real-time critical software. Since then, they have been used in various industrial applications such as the fly-by-wire commands in Airbus planes. They were based on the objective to build a programming language close to the mathematical models used in embedded systems such as data-flow equations or the composition of finite-state machines. In these languages, synchrony finds a very practical justification: at a certain level of observation, time is considered *logically* as a sequence of instantaneous steps (or atomic reactions) of the system to external events and when processes are composed in

[⋆] This work was partially supported by the INRIA research project Synchronics.

parallel, they agree on those steps [2]. This also coincide with Milner's interpretation of synchrony in SCCS [3] or the synchronized product of automata [4]. In a data-flow language such as LUSTRE, synchrony is essentially the one of control-theory and can be interpreted as a typing constraint on the domain of sequences: when combining two sequences $(x_i)_{i\in D_1}$ and $(y_i)_{i\in D_2}$ as in $(x_i)_{i\in D_1} + (y_i)_{i\in D_2}$, their time-domain D_1 and D_2 must be compatible in some ways. This static checking is done by a specific type system, the so-called clock-calculus [5, 6] which impose D_1 and D_2 to be equal. Such analysis appears not to be bound to synchronous languages and are of a much wider interest. For example, clock analysis is done in modeling tools such as SIMULINK [7] or SCICOS [8].

The clock-calculus essentially consists in asking whether two streams have the same clock. For that, those clock information are abstracted by types. Consider the following type language (taken from [5]):

$$\begin{array}{lcl} \sigma & ::= & \forall\alpha.\forall X_1,...,X_m.ct \\ ct & ::= & ct \to ct \mid ct * ct \mid ck \mid (X : ck) \\ ck & ::= & ck \ \texttt{on}\ c \mid ck\ \texttt{on}\ \textit{not}\ c \mid \alpha \\ c & ::= & X|n \text{ where } n \text{ is a numerable set of names} \end{array}$$

As in the Hindley-Milner type system [9], types are separated into clock schemes (σ) and clock types (ct). A clock scheme is quantified over clock variables (α) or boolean variables (X). Then, the clock type of $(+)$ is $\forall\alpha.\alpha\times\alpha \to \alpha$ stating that if x and y are two integer streams with the same clock α then $x + y$ have also the clock α. Said differently, the addition expects its two arguments to be synchronous and produces a third one with the same clock. An other example of a synchronous primitive is the unitary delay which shifts its input. If x and y are two sequences $x_0\, x_1\, x_2 \ldots$ and $y_0\, y_1\, y_2 \ldots$ then $x\, \texttt{fby}\, y$ stands for $x_0\, y_0\, y_1\, y_2 \ldots$. x and y must have the same clock as well as $x\, \texttt{fby}\, y$ and we can give to `fby` the clock signature: $\forall\alpha.\alpha \times \alpha \to \alpha$.

Things become more interesting when sampling occurs. Two typical programming constructs are the sampler and the merge operator:

$$\begin{array}{ll} \texttt{when} & : \forall\alpha.\forall X.\alpha \to (X : \alpha) \to \alpha\ \texttt{on}\ X \\ \texttt{merge} & : \forall\alpha.\forall X.(X : \alpha) \to \alpha\ \texttt{on}\ X \to \alpha\ \texttt{on}\ \textit{not}\ X \to \alpha \end{array}$$

$x\, \texttt{when}\, y$ is well clocked when x and y have the same clock α. In that case, the result has a slower clock corresponding to the instant where y is true and we write it $\alpha\ \texttt{on}\ y$ (the meta-variable X is substituted with the actual one y). For example, if *half* is an alternating boolean sequence 10101010101... and x is a sequence $x_0\, x_1\, x_2 \ldots$ then $x\, \texttt{when}\, \textit{half}$ is a half frequency sampling of x, that is, $x_0\, x_2\, x_4 \ldots$. If x has some clock type ck, then $x\, \texttt{when}\, \textit{half}$ has clock type $ck\ \texttt{on}\ \textit{half}$. Then, the expression $x + (x\, \texttt{when}\, \textit{half})$ which would compute the sequence $(x_i + x_{2i})_{i\in\mathbb{N}}$ is not well clocked since ck is not equal to $ck\ \texttt{on}\ \textit{half}$ and is statically rejected by the compiler. The `merge` operator is symmetric: it expects two streams with complementary clocks and combines them to build a longer stream.

When comparing clocks, most implementations restrict the comparison to the equality of names: $ck\ \texttt{on}\ n_1$ and $ck\ \texttt{on}\ n_2$ can be unified only when $n_1 = n_2$. This strong restriction is justified by the graphical aspect of these languages. Two streams can be composed when they are sampled by the same condition coming from the very same source block, that is, the same wire. This restriction is reasonable when n_1 is the result of a complex computation (for which equality is non-decidable). This is nonetheless overly restrictive for an important variety of applications where clocks are periodic and it forbids to take their properties into account. In particular, recent works have shown the interest of a more *relaxed* model of synchrony allowing to compose streams as soon as they can be synchronized through the introduction of bounded buffers. This model is called the N-synchronous Kahn model [10] and pursues the foundational work of Edward Lee on Synchronous Data-Flow graphs (SDF) [11, 12]. Data-flow equations in SDF are not statically constrained with any notion of synchronous clock, yet *the existence of* a static synchronous schedule is guaranteed by periodicity constraints on production/consumption rates. From the typing point of view, N-synchrony amounts at turning the clock calculus into a type-system with a subtyping rule:

$$\text{(SUB)}\frac{H \vdash e : ck \quad ck <: ck'}{H \vdash e : ck'}$$

Intuitively, $ck <: ck'$ means that the $1s$ of ck arrive before the $1s$ of ck' and that ck and ck' have in average the same proportion of $1s$. In the particular case of *ultimately periodic clocks* [13], subtyping is decidable. For example, the clock *half* is written $(\mathtt{10})$ whereas $\mathtt{0000(10)}$ is the same clock with a prefix of four false values. Thus, $\alpha \texttt{ on } (\mathtt{10}) <: \alpha \texttt{ on } \mathtt{0000(10)}$. Algebraic properties allow to compare more clocks, e.g., $\alpha \texttt{ on } (\mathtt{10}) \texttt{ on } (\mathtt{10})$ equals $\alpha \texttt{ on } (\mathtt{1000})$. Note that those clocks correspond to simple linear circuits (and automata). The technical details shall be reminded in Section 2.

In the present paper, we adopt a slightly different and simpler point of view than in [10]. Instead of focusing on periodic clocks, we give the ability to reason on set of clocks or *clock envelopes* as abstractions of concrete clocks. Indeed, in various applications, exact synchrony with precise periodic clocks is not mandatory and it is sufficient to reason on clock intervals where bounds are nonetheless periodic. This is typically the case in three kind of applications, (1) video applications with bounded jitter, (2) the description of execution times when modeling physical resources, and (3) the communication through buffers (or cyclic arrays). For example, a stream x which is present in average 3 times over 7 according to a base clock ck and with a possible jitter of 4 will be given a clock type $ck \texttt{ on}^{\sim} [-2, 2](7/3)$ as a shortcut for $\exists n \in [-2, 2](7/3).\ ck \texttt{ on } n$. The existential quantifier hides the exact instant where the element is present but gives a bound on it. The intuition behind the notation $[-2, 2](7/3)$ is to account for all clocks whose $(j+1)^{\text{th}}$ $\mathtt{1}$ is between positions $(7/3) \times j - 2$ and $(7/3) \times j + 2$. Said differently, we consider all clocks ck' such that $ck \texttt{ on } \mathtt{1(1010100)} <: ck' <: ck \texttt{ on } (\mathtt{0010101})$. If f is of the form $\lambda x.x \texttt{ when } e$ for some complex boolean expression e but for which it can be proved that its value belongs to the envelope $[-2, 2](7/3)$ then a valid abstraction for f is $\forall\alpha.\alpha \rightarrow \alpha \texttt{ on}^{\sim} [-2, 2](7/3)$.

Another example appears when modeling the execution time of processes [14–16]. To state that a function f must be executed every ten cycles and that its computation takes between two and four cycles, we can give it the clock signature: $\forall \alpha . \alpha\ \mathtt{on}^{\sim}\ [0,0](10/1) \rightarrow \alpha\ \mathtt{on}^{\sim}\ [2,4](10/1)$. When composed twice, we get: $f\,o\,f : \forall \alpha .(\alpha\ \mathtt{on}^{\sim}\ [0,0](10/1)) \rightarrow \alpha\ \mathtt{on}^{\sim}\ [4,8](10/1)$.

Another typical example can be taken from *elastic circuits* [17]. For example, an elastic adder is a stream function which takes two streams which are not synchronous but are in the same envelope up to one delay, that is, they belong to some envelope $(ck\ \mathtt{on}\ c)\ \mathtt{on}^{\sim}\ [0,1](1/1)$ for some unknown boolean sequence c (not necessarily periodic).

Finally, it is also possible to mimic a common feature found in a video application: to read several inputs (or write several output) at once. Consider, for example a stream function f whose input has clock type $\alpha\ \mathtt{on}^{\sim}\ [0,4](1/1)$ connected to a stream x with clock type $ck\ \mathtt{on}^{\sim}\ [0,0](1/1)$. Then, at each instant, x produces a value whereas f can wait four instants before to start consuming the values. At the fifth instant, f can consume the first five values and then continue to read arrays by slides of five elements every five instants.

The main contribution of this paper is thus to introduce those clock envelopes and to study their algebraic properties. The paper is organized as follows. In Section 2, we remind the basic properties of infinite binary words. In Section 3, we introduce the clock envelopes as sets of clocks. Section 4 presents related work and we conclude in Section 5.

For lack of space, proofs are not given or only sketched. Proofs and complements are available at `www.lri.fr/~plateau/aplas08`.

2 Clocks as Infinite Binary Words

A subtyping relation can be checked only if clock types are expressed with respect to the same clock variable. Intuitively, this is because sampling is always relative to a clock ($ck\ \mathtt{on}\ c$ is relative to ck). In that case the subtyping relation corresponds to a relation on boolean sequences: $\alpha\ \mathtt{on}\ c <: \alpha\ \mathtt{on}\ c' \Leftrightarrow c <: c'$

In this section, we present infinite binary words and a boolean operation on for them such that $(ck\ \mathtt{on}\ c_1)\ \mathtt{on}\ c_2 = ck\ \mathtt{on}\ (c_1\ \mathit{on}\ c_2)$. We then present the subtyping relation on these words. As we mainly manipulate the boolean stream part c of a clock type we will also call it clock.

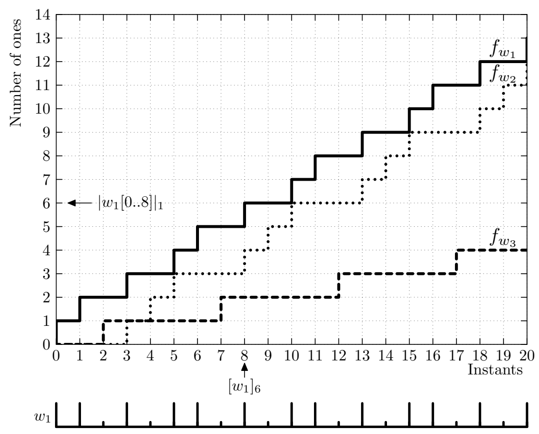
2.1 Definitions

A clock can be an infinite binary word or a composition of those. Identifying names to their values, clocks have the following grammar:

$$c \ ::= w \mid \mathit{not}\ w \mid c\ \mathit{on}\ c$$
$$w ::= 0.w \mid 1.w$$

$\mathit{not}\ w$ is the negation of w, $c_1\ \mathit{on}\ c_2$ is the sampled clock and w an infinite binary word. In infinite binary words, a 1 denotes the presence of a value on the flow, and a 0 the absence of value.

Remark 1. We will consider that in the clocks we manipulate, the maximal distance between two successive $1s$ is bounded (in particular, every word contains



In the 2D-chronograms, the discrete function $f_w(i) = |w[0..i]|_1$ associates to each instant i the number of 1*s* seen in w since the beginning. The function $f_w^{-1}(j) = \min i$ s.t. $f(i) = j$ gives the index of the j^{th} 1. A rising edge at instant i means that the element at index i of w is a 1 (i.e. $w[i] = 1$). In the contrary case (i.e. $f_w(i) = f_w(i-1)$), it is a 0 (i.e. $w[i] = 0$). The word w_1 is represented on the 1D-chronogram, which is a projection of f_{w_1}.

Fig. 1. Chronograms representing words $w_1 = (11010)$, $w_2 = 0(00111)$, $w_3 = (00100)$.

an infinity of 1*s*). So, the ***not*** operator cannot be applied on a clock that does not contain an infinity of 0*s*.

Notations: The concatenation of a finite binary word u and a binary word w is written $u.w$. We will sometimes note 0^n the concatenation of n values 0 and 1^n of n values 1. $w[i]$ is the element at index i of w, $w[0..i]$ is the prefix of w of length $i+1$, and $[w]_j$ the position of the j^{th} 1 in w. It is defined by: $[1.w]_1 = 0$, $[1.w]_{j+1} = 1 + [w]_j$, $[0^d.w]_j = d + [w]_j$, with $d \in \mathbb{N}, j \in \mathbb{N}^*$.[4] [5] Note that $\forall j \geq 1,\ [w]_j < [w]_{j+1}$. Finally, the number of 1*s* contained in a finite binary word v is denoted by $|v|_1$.

We will call *periodic binary words* and we will write $u(v)$, the words consisting of a finite prefix u, followed by the infinite repetition of a finite binary word v.

Fig. 1 shows some examples of infinite binary words, represented by chronograms. The discrete function $f_w(i) = |w[0..i]|_1$ associates to each instant i the number of 1 seen in w since the beginning. A rising edge at instant i means that $w[i] = 1$. If a flow is produced (resp. consumed) at clock w, then a token is produced (resp. consumed) at each rising edge of the chronogram.

Formally, the ***on*** operator is defined by:

$$\begin{aligned} 1.w_1 \ \textit{on}\ 1.w_2 &= 1.(w_1 \ \textit{on}\ w_2) \\ 1.w_1 \ \textit{on}\ 0.w_2 &= 0.(w_1 \ \textit{on}\ w_2) \\ 0.w_1 \ \textit{on}\ w_2 &= 0.(w_1 \ \textit{on}\ w_2) \end{aligned}$$

The elements of w_1 ***on*** w_2 correspond to the elements of w_2, when w_2 is traversed at the rhythm of the 1*s* of w_1. So if the j^{th} 1 of w_2 is the i^{th} element of w_2, we know that the j^{th} 1 of w_1 ***on*** w_2 is at the index of the i^{th} 1 in w_1.

Proposition 1. $[w_1 \ \textit{on}\ w_2]_j = [w_1]_{[w_2]_j+1}$

Corollary 1 (*on* associativity). $(w_1 \ \textit{on}\ w_2) \ \textit{on}\ w_3 = w_1 \ \textit{on}\ (w_2 \ \textit{on}\ w_3)$

[4] $\mathbb{N}$ is the set of natural numbers and $\mathbb{N}^*$ is the set of positive natural numbers.

[5] Note that for readability reasons and contrary to previous works, indexes of elements of w begin at 0. It involves a shift in some formulas, w.r.t those of [10].

2.2 Buffer Size

The minimal buffer size needed to communicate from an output with clock type α `on` w to an input with clock type α `on` w' is the maximum amount of data produced and not yet consumed in the course of the execution:

$$size(w, w') = \max_{i \in \mathbb{N}}(|w[0..i]|_1 - |w'[0..i]|_1)$$

Note that if the minimum of this difference is negative, then there will be at least one read in an empty buffer. Indeed, when a negative value is reached, more data have been consumed than produced.

In chronograms, the buffer size to communicate from α `on` w to α `on` w' is equal to the maximal difference $f_w(i) - f_{w'}(i)$. For example, in Fig. 1, the maximal amount of data produced and not yet consumed during the communication from α `on` w_1 to α `on` w_2 is 2, reached for the first time at instant 1. The amount of data to store during the communication from α `on` w_1 to α `on` w_3 grows infinitely.

2.3 Subtyping Relation

The subtyping relation is verified if the tokens are always produced before they are expected (precedence relation), and at a bounded distance of the instant they are consumed (synchronizability relation).

Definition 1 (precedence).
We say that w_1 precedes w_2 and we write $w_1 \preceq w_2$ iff $\forall j \geq 1,\ [w_1]_j \leq [w_2]_j$.

A word w_1 precedes a word w_2 if the j^{th} 1 of w_1 always comes before (or at the same time as) the j^{th} 1 of w_2. It permits to verify that the causality relation between flows is preserved, i.e. that the producer writes its outputs in the buffer before the consumer needs it.

In Fig. 1, the edges of the chronogram of w_1 occur earlier than the corresponding ones of w_2 and w_3, so $w_1 \preceq w_2$ and $w_1 \preceq w_3$ but chronograms of w_2 and w_3 are interleaved, so $w_2 \not\preceq w_3$ and $w_3 \not\preceq w_2$.

We can define the supremum $\sqcup$ and the infimum $\sqcap$ of a set of infinite binary words $W = \{w_1, ..., w_n\}$ for the $\preceq$ relation:
$\forall j \geq 1,\ [\sqcup W]_j = \max([w_1]_j, ..., [w_n]_j)$ and $[\sqcap W]_j = \min([w_1]_j, ..., [w_n]_j)$.
For instance, in Fig. 1, $w_2 \sqcup w_3$ is equal to w_2 until instant 3 and then equal to w_3.
For all $w \in W$, $\sqcap W \preceq w \preceq \sqcup W$.

Definition 2 (synchronizability). *We say that two words w_1 and w_2 are synchronizable and we write $w_1 \bowtie w_2$ iff there exists $d_1, d_2 \in \mathbb{N}$ such that $w_1 \preceq 0^{d_2}.w_2$ and $w_2 \preceq 0^{d_1}.w_1$.*

By definition of $\preceq$, it ensures that the j^{th} 1 of w_1 is at a bounded distance of the j^{th} 1 of w_2:

Proposition 2. $w_1 \bowtie w_2 \Leftrightarrow \exists d_1, d_2 \in \mathbb{N}, \forall j \geq 1, -d_1 \leq [w_1]_j - [w_2]_j \leq d_2$

For instance, in Fig. 1, $w_1 \bowtie w_2$, but $w_1 \not\bowtie w_3$ and $w_2 \not\bowtie w_3$.

When the producer of a flow communicates with the consumer through a buffer, it allows to verify that there exists a correct size for this buffer such that there will never be an overflow. Indeed, if $-d_1 \leq [w_1]_j - [w_2]_j \leq d_2$, the j^{th} 1 of w_2 occurs at worst d_1 instants after the j^{th} 1 of w_1. So when it occurs in w_2, at worst d_1 supplementary 1s have occurred in w_1, and the size $\max_i |w_1[0..i]|_1 - |w_2[0..i]|_1$ is lower or equal to d_1.

Definition 3 (subtyping). *The subtyping relation, written* $<:$ *is the conjunction of precedence and synchronizability:* $w_1 <: w_2 \overset{def}{=} w_1 \preceq w_2 \wedge w_1 \bowtie w_2$.

In fact, if the tokens are always produced before they are needed, and at a bounded distance of the time they are consumed, then the communication can be made synchronous by the insertion of a bounded size buffer. For instance, in Fig. 1, $w_1 <: w_2$.

Remark 2. By Def. of $\preceq$ and $\bowtie$, $w_1 <: w_2 \Leftrightarrow \exists d \in \mathbb{N}, \forall j \geq 1, 0 \leq [w_2]_j - [w_1]_j \leq d$

The *on* operator is monotonous with respect to the $<:$ relation:

Proposition 3 (*on* monotonicity).
$w_1 <: w_2 \wedge w'_1 <: w'_2 \Rightarrow w_1 \;\textit{on}\; w'_1 <: w_2 \;\textit{on}\; w'_2$

All definitions of this section can be lifted to clocks (c) by computation of *not* and *on* operators.

3 Abstraction of Clocks

Abstracting an infinite binary word w consists in keeping only (1) the average distance T between two 1s in w (the asymptotic rate of 1s of w is $\frac{1}{T}$) and (2) two phases d and D that bound indexes of 1s in w, with respect to the perfect repartition of one 1 every T instants. We note this abstraction $[d, D]\,(T)$ and we call it *envelope*.

Abstract clocks can be envelopes or compositions of those. They are defined by the following grammar:[6]

$$\begin{aligned} ac &::= a \mid \textit{not}^{\sim}\ a \mid ac\ \textit{on}^{\sim}\ ac \\ a &::= [d, D]\,(T) \ \text{ with } d, D, T \in \mathbb{Q}, D \geq 0 \text{ and } T \geq 1 \end{aligned}$$

In this section, we explain what set of words is represented by an envelope and we show that this language is recognizable by a finite automaton. Then we define *on*$^{\sim}$ and *not*$^{\sim}$ operators that can be computed efficiently, allowing to always reduce an abstract clock to an envelope. Finally, we show that relations presented in Sec. 2 can be easily checked, and that buffer size can be efficiently computed.

[6] $\mathbb{Q}$ is the set of rational numbers.

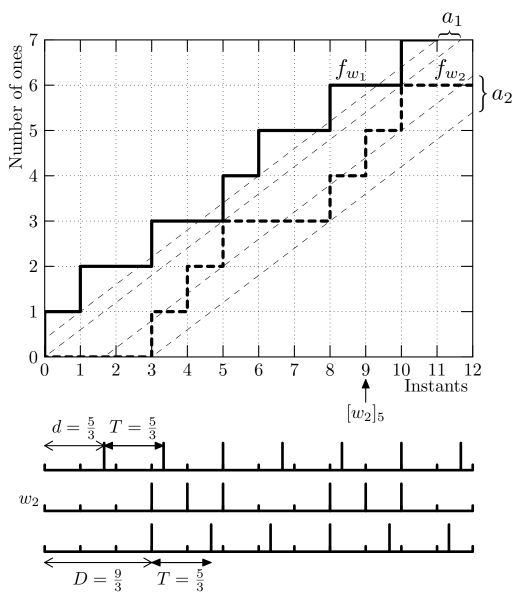


Fig. 2. If the rising edges of a word w all start between the lines of equation $\frac{i-d}{T}$ and $\frac{i-D}{T}$, then w is in the envelope $[d, D]\,(T)$, because for all $j \geq 0, T \times j + d \leq [w]_{j+1} \leq T \times j + D$. Thus w_1 is in $a_1 = \left[-\frac{2}{3}, 0\right]\left(\frac{5}{3}\right)$ and w_2 is in $a_2 = \left[\frac{5}{3}, \frac{9}{3}\right]\left(\frac{5}{3}\right)$. For instance, $\frac{5}{3} \times 4 + \frac{5}{3} \leq [w_2]_5 \leq \frac{5}{3} \times 4 + \frac{9}{3}$.

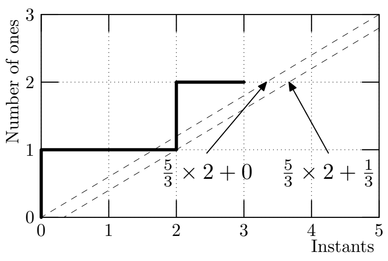


Fig. 3. The concretization set of $\left[0, \frac{1}{3}\right]\left(\frac{5}{3}\right)$ is empty: there is no valid discrete index for the third 1.

3.1 Abstraction of Infinite Binary Words

An envelope $[d, D]\,(T)$, with $d, D, T \in \mathbb{Q}$, $D \geq 0$, $T \geq 1$, represents the following set of infinite binary words:

Definition 4 (concretization).
$$concr\,([d, D]\,(T)) \overset{def}{=} \{w,\ \forall j \geq 0,\ T \times j + d \leq [w]_{j+1} \leq T \times j + D\}$$

D will always be positive or null, otherwise we would have $[w]_1 < 0$. A $T < 1$ would represent clocks that have in average more that one 1 per instant, which are not considered (as mentioned in Sec. 2.1 $\forall j \geq 1,\ [w]_j < [w]_{j+1}$).

In 2D-chronograms, the envelope $[d, D]\,(T)$ can be represented by two lines that bound the rising edges starting points of the words it contains. The equations of these lines are $\frac{i-d}{T}$ and $\frac{i-D}{T}$. For instance, in Fig. 2 we can see the word $w_2 = 0(00111)$ and $a_2 = \left[\frac{5}{3}, \frac{9}{3}\right]\left(\frac{5}{3}\right)$, which is an abstraction of w_2.

The lines can be interpreted as "ideal clocks" such that from instant $\frac{5}{3}$ (resp. $\frac{9}{3}$), a tick occurs every $\frac{5}{3}$ of instant (i.e. each time the line crosses a y-axis discrete value). The word w_2 is bounded by these two ideal clocks (see the 1D-chronograms of Fig. 2).

Notice that every word staying at a bounded distance of its asymptotic rate can be abstracted by an envelope.

An envelope can always be normalized into the form $\left[\frac{k}{n}, \frac{K}{n}\right]\left(\frac{l}{n}\right)$ with $k \in \mathbb{Z}$, $K \in \mathbb{N}$, and $gcd(l, n) = 1$ without changing the concretization set:

Proposition 4 (normal form).
*$\forall a = [d, D]\left(\frac{l'}{n'}\right),\ \exists k \in \mathbb{Z},\ K \in \mathbb{N},\ l \in \mathbb{N}, n \in \mathbb{N}^*$ with $gcd(l, n) = 1$ such that $concr\left(\left[\frac{k}{n}, \frac{K}{n}\right]\left(\frac{l}{n}\right)\right) = concr(a)$.
$l = \frac{l'}{gcd(l',n')}$, $n = \frac{n'}{gcd(l',n')}$, $k = \lceil d \times n \rceil$ and $K = \lfloor D \times n \rfloor$.*[7]

The concretization set is empty if there exists a $j \geq 0$ such that there is no natural number between the bounds of the $(j+1)^{\text{th}}$ 1. Actually, in this case, there is no valid index for the $(j+1)^{\text{th}}$ 1.

Proposition 5 (empty concretization set).
Let $a = [d, D]\,(T)$ be an envelope.

$$\begin{aligned} concr(a) = \varnothing &\Leftrightarrow \exists j \geq 0,\ \{m \in \mathbb{N},\ T \times j + d \leq m \leq T \times j + D\} = \varnothing \\ &\Leftrightarrow \exists j \geq 0,\ \lceil T \times j + d \rceil > \lfloor T \times j + D \rfloor \end{aligned}$$

For instance, the concretization set of $\left[0, \frac{1}{3}\right]\left(\frac{5}{3}\right)$ is empty. Indeed, we can see in Fig. 3 that there is no valid index for the third 1, that must occur between the instants $\frac{5}{3} \times 2 + 0$ and $\frac{5}{3} \times 2 + \frac{1}{3}$.

The following proposition gives a sufficient condition on an envelope a, to ensure that its concretization set of is not empty. If a is in normal form, this condition is necessary.

Proposition 6 (non-emptiness test).
*Let $a = \left[\frac{k}{n}, \frac{K}{n}\right]\left(\frac{l}{n}\right)$ be an envelope. $\frac{K}{n} - \frac{k}{n} \geq 1 - \frac{1}{n} \Rightarrow concr(a) \neq \varnothing$
Additionally, if a is in normal form (i.e. $gcd(l, n) = 1$), then the converse holds.*

A length $1 - \frac{1}{n}$ for the interval $[\frac{k}{n}, \frac{K}{n}]$ ensures that for all i, there is a natural number between the bounds of the j^{th} 1 index. If furthermore the envelope is in normal form, then this length is the minimal length such that the concretization set is not empty.

The concretization set contains one and only one element iff for all j there is exactly one natural number between the lower bound and the upper bound, i.e. iff $\forall j, \lceil T \times j + d \rceil = \lfloor T \times j + D \rfloor$. Indeed, in that case there is only one choice for the index of each 1 of the binary word. This occurs when $D - d = 1 - \frac{1}{n}$, and only when this condition is verified in the case of abstract clocks in normal form. For instance, in Fig. 2, w_1 is the unique element in the concretization set of a_1. Indeed, $a_1 = \left[-\frac{2}{3}, 0\right]\left(\frac{5}{3}\right)$ and $0 - (-\frac{2}{3}) = 1 - \frac{1}{3}$. We can check on the 2D-chronogram that for each j on the y-axis, there is exactly one valid index i in the envelope.

Otherwise, the concretization set is infinite. In fact, in that case there are several integers between the bounds of certain indexes, thus several choices for them. Then, it is the case for an infinity of indexes. This occurs iff $\forall j \geq 0, \lceil T \times j + d \rceil \leq \lfloor T \times j + D \rfloor$ (non-emptiness condition) and $\exists j \geq 0, \lceil T \times j + d \rceil < \lfloor T \times j + D \rfloor$ (several choices for at most one index). This occurs when $D - d > 1 - \frac{1}{n}$, and

[7] $\lceil x \rceil$ (resp. $\lfloor x \rfloor$) is the notation for the ceiling (resp. floor) function.

only when this condition is verified in the case of envelopes in normal form. For instance in Fig. 2, if we consider the concretization set of a_2, we note that the fourth edge can occur at index 8 (as in w_2) or 7.

Remark 3. The chronogram never passes on the right of the envelope. Indeed, passing on the right of it leads to no more allow rising edges in the word, and thus to have clocks with a finite number of presence instants, which are not considered in this work.

Let us consider the concretization set of a simpler example: $a_4 = [2,3]\,(2)$.
$concr(a_4) = \{0^2(10), 0^2(1001), 0^2(0110), 0^2(01), 0^2(011001), 0010(01), \ldots\}$
To simplify the presentation, we only give here some periodic elements of the concretization set, but it contains an infinity of periodic and non-periodic infinite binary words, all of the form $00(10\ +\ 01)^*$. We show in Sec. 3.2 a complete representation of the concretization sets.

The infimum and supremum of the concretization set (with respect to the $\preceq$ relation) are periodic binary words:

Proposition 7 ($\sqcap$, $\sqcup$). *Let $a = [d, D]\,(T)$ with $concr(a) \neq \varnothing$.*
$w_{\text{inf}} = \sqcap(concr(a)) \Leftrightarrow \forall j \geq 0, [w_{\text{inf}}]_{j+1} = \lceil T \times j + d \rceil$
$w_{\text{sup}} = \sqcup(concr(a)) \Leftrightarrow \forall j \geq 0, [w_{\text{sup}}]_{j+1} = \lfloor T \times j + D \rfloor$

If $T = \frac{l}{n}$, then the periodic pattern will be of length l and will contain n ones.

Proof. The formulas come from the definitions of $\sqcap$ and $\sqcup$. Let $T = \frac{l}{n}$. The word w_{inf} is periodic because for all $j \geq 0$, $[w_{\text{inf}}]_{j+n+1} = \lceil \frac{l}{n} \times (j + n) + d \rceil = \lceil \frac{l}{n} \times j + d \rceil + l$, so $\forall j \geq n, [w_{\text{inf}}]_{j+1} = [w_{\text{inf}}]_{j+1-n} + l$. □

For instance in Fig. 2, $w'_2 = 00(10110)$ is the infimum of the concretization set of a_2, the rising edges occur as soon as possible, and $w''_2 = 00(01101)$ is the supremum, the rising edges occur as late as possible. As we illustrate in Sec. 3.2, these two particular clocks can be efficiently computed with a synchronous circuit with linear size (w.r.t $max(d, D, T)$) instead of the size of the period and they are perfectly balanced.

We have a partial order relation on abstract clocks:

Definition 5 (order relation $\sqsubseteq^{\sim}$). $ac_1 \sqsubseteq^{\sim} ac_2 \overset{def}{=} concr(ac_1) \subseteq concr(ac_2)$

It can be tested efficiently.

Proposition 8 ($\sqsubseteq^{\sim}$ test). *Let $a_1 = [d_1, D_1]\,(T_1)$ and $a_2 = [d_2, D_2]\,(T_2)$ be envelopes such that $concr(a_1) \neq \varnothing$ and $concr(a_2) \neq \varnothing$. Then,*

$$T_1 = T_2 \text{ and } [d_1, D_1] \subseteq [d_2, D_2] \Rightarrow a_1 \sqsubseteq^{\sim} a_2$$

Additionally, if a_1 and a_2 are in normal form, then the converse holds.

If we interpret this on the 2D-chronograms, $a_1 \sqsubseteq^{\sim} a_2$ if the lines representing the envelope of a_1 are between the ones of a_2.

Proof (Intuition). To stay between the lines of a_2, the lines of a_1 must have the same slope as the ones of a_2, thus $\frac{1}{T_2} = \frac{1}{T_1}$. Concerning the delays, the proof that the converse holds relies on the fact that a_1 and a_2 are in normal form. □

We will write $abs(w)$ any function such that $abs(w) = a \Rightarrow w \in concr(a)$.

3.2 Abstract Clocks as Automata

We have seen that the concretization set of an abstract clock can be empty, contain a unique element or an infinity of elements. It can be represented by a deterministic finite automaton recognizing all binary words of the set, and only them. We first define an infinite automaton such that the language recognized is the concretization set, then we show that it is equivalent to a finite automaton.

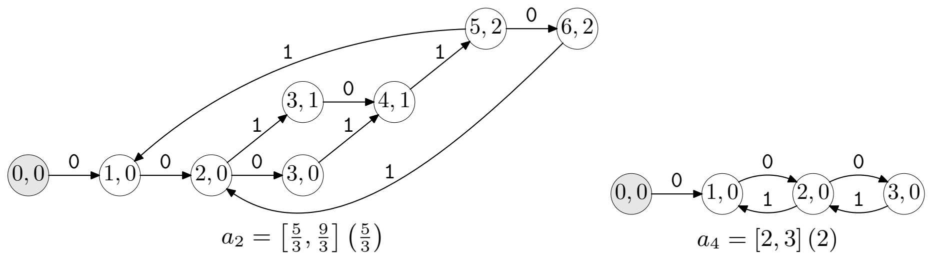
Definition 6 (automaton associated to an envelope). *Let $a = [d, D]\,(T)$ be an envelope. The infinite automaton associated to a is $I_a = \langle Q, \Sigma, \delta, q_o \rangle$ with:*

- *The set of states Q is a set of pairs $(i, j) \in \mathbb{N}^2$*
- *The initial state q_o is $(0, 0)$*
- *The alphabet Σ is $\{\mathsf{0}, \mathsf{1}\}$*
- *The transition function δ is defined by:*
  $\delta(\mathsf{1}, (i, j)) = (i + 1, j + 1)$ *if* $T \times j + d \leq i \leq T \times j + D$
  $\delta(\mathsf{0}, (i, j)) = (i + 1, j)$ *if* $i + 1 \leq T \times j + D$
  It is undefined otherwise.

Proposition 9. *Let $a = [d, D]\,(T)$ be an envelope, and I_a its associated infinite automaton. The language recognized by I_a is $concr(a)$.*

The labels of the states correspond to coordinates in 2D-chronograms. The value i is the index of the current instant, and j is the number of $\mathsf{1}s$ seen before the current instant. A transition from (i, j) to $(i+1, j+1)$ corresponds to a rising edge starting at (i, j), i.e. to the occurrence of the $(j + 1)^{\text{th}}$ $\mathsf{1}$ at index i. It can be taken if (i, j) is between the bounding lines, i.e. if $T \times j + d \leq i \leq T \times j + D$. A transition from (i, j) to $(i+1, j)$ corresponds to a flat edge in the chronogram. It can be taken if the destination state $(i + 1, j)$ is not on the right-side of the bounding lines (see Rem. 3), i.e. if $i + 1 \leq T \times j + D$ (it ensures that i is not the last valid index for the $(j + 1)^{\text{th}}$ $\mathsf{1}$).

Let us now define the set of reachable states. A state (i, j) is reachable if the two following conditions are verified: (1) if $j > 0$, it is possible that the j^{th} $\mathsf{1}$ has occurred before i, i.e. the earliest possible index for the j^{th} $\mathsf{1}$ ($T \times (j - 1) + d$) is smaller or equal to $i - 1$, and (2) it is still possible for the $(j + 1)^{\text{th}}$ $\mathsf{1}$ to occur, i.e. i is less or equal to the latest possible index for the $(j + 1)^{\text{th}}$ $\mathsf{1}$ ($T \times j + D$). So we can restrict the set of states to the set of reachable states which is
$Q = \{(i, j) \in \mathbb{N}^2, (j = 0 \text{ or } d - T + 1 \leq i - T \times j) \text{ and } (i - T \times j \leq D)\}$.
This infinite automaton can be transformed into a finite one, by noticing that the transition function only depends on the value of $i - T \times j$. We thus identify all states (i, j), (i', j') such that $i - T \times j = i' - T \times j'$. As states are such that $j = 0$ and $i \leq D$, or $d - T + 1 \leq i - T \times j \leq D$, the set of states obtained is finite. The transition function of the finite automaton A_a equivalent to the infinite automaton I_a is:
$\delta(\mathsf{1}, (i, j)) = nf(i + 1, j + 1)$ if $T \times j + d \leq i \leq T \times j + D$
$\delta(\mathsf{0}, (i, j)) = nf(i + 1, j)$ if $i + 1 \leq T \times j + D$
with $T = \frac{l}{n}$, $nf(i, j) = (i - x \times l, j - x \times n)$, $x = \max\{x \in \mathbb{N}, (x \times l \leq i) \wedge (x \times n \leq j)\}$



$a_2 = \left[\frac{5}{3}, \frac{9}{3}\right]\left(\frac{5}{3}\right)$
$a_4 = [2,3]\,(2)$

Fig. 4. Automata recognizing the clocks of $concr(a_2)$ and $concr(a_4)$.

We thus are able to represent infinite concretization sets by finite automata. Fig. 4 shows the automata associated to $a_2 = \left[\frac{5}{3}, \frac{9}{3}\right]\left(\frac{5}{3}\right)$ of Fig. 2, and to $a_4 = [2,3]\,(2)$.

The infimum of the concretization set (with respect to $\preceq$) corresponds to the path taking in priority the 1-transitions, and the supremum corresponds to the path taking in priority the 0-transitions. All paths have the same asymptotic rate of $1s$ (e.g. choosing 1-transitions first only delays the corresponding 0-transitions).

It is interesting because it allows to check dynamically or statically (with model checking) that a clock is in the envelope specified by the user, and throw an error message if it's not the case. Note that it is not necessary to build explicitly the automaton. Here is a Lustre program that checks that a clock `clk` is in an envelope $\left[\frac{k}{n}, \frac{K}{n}\right]\left(\frac{l}{n}\right)$.

```
node norm(const l, n: int; i, j: int) returns (ni, nj: int);
let
  (ni, nj) = if i >= l and j >= n then (i - l, j - n) else (i,j);
tel

node check(const k, K, l, n: int; clk: bool) returns (ok: bool);
var i, j, v: int;
let
  (i,j) = (0,0) -> pre norm(l, n, i+1, if clk then j + 1 else j);
  v = i * n - j * l;
  ok = if clk then (k <= v and v <= K) else v <= K - n;
tel
```

The function `norm` incrementally computes the normal form of the state `(i,j)` using `l` and `n`. The function `check` maintains the value of the current state. It is initialized to `(0,0)`, then at each instant, `i` is incremented, and `j` is incremented if the clock `clk` was true at the preceding instant. The normalization function is applied to the new state. Then, if the current value of `clk` is true, we check that a 1-transition is allowed, and in the contrary case, we check that a 0-transition is allowed.
The same principle allows to generate clocks within a certain abstraction, for simulation purposes. To generate the earliest clock we take a 1-transition each time it is allowed.

```
node early(const k, K, l, n: int) returns (clk: bool);
var i, j, v: int;
let
  (i, j) = (0,0) -> pre norm(l, n, i+1, if clk then j + 1 else j);
  v = i * n - j * l;
  clk = (k <= v and v <= K);
tel
```

Similarly, to generate the latest clock we take a 1-transition each time a 0-transition is not allowed.

3.3 Abstract Operators

We define in this section operators on envelopes, corresponding to operators on words defined in Sec. 2. Computing these operators in the abstract domain is in constant time and memory. Moreover, they are correct, i.e. the result of the operation in the abstract domain contains the result of the operation in the concrete one.

Definition 7 ($\textit{on}^{\sim}$ operator).
We define an abstract $\textit{on}$ *operator, written* $\textit{on}^{\sim}$*:*

$$[d_1, D_1]\,(T_1)\ \textit{on}^{\sim}\ [d_2, D_2]\,(T_2) = [d_{12}, D_{12}]\,(T_{12})$$

with: $T_{12} = T_1 \times T_2, \quad d_{12} = d_1 + d_2 \times T_1, \quad D_{12} = D_1 + D_2 \times T_1.$

We have seen in Sec. 2 that the elements of $w_1\ \textit{on}\ w_2$ are the elements of w_2, traversed at the pace of the 1*s* of w_1. So if the distance between the 1*s* of w_2 is on average equal to T_2, and the distance between those of w_1 is on average equal to T_1, then the distance between the 1*s* of $w_1\ \textit{on}\ w_2$ is on average equal to $T_2 \times T_1$. Sampling w_1 with w_2 keeps intact the delays of w_1, and adds to it the delay of w_2 multiplied by T_1, because w_2 is traversed at the pace T_1.
Thus, this abstract operator has the expected property: for all words of the respective concretization sets, the result of the concrete $\textit{on}$ operation is in the concretization set of the result of the abstract operation.

Proposition 10.
$\forall w_1 \in \textit{concr}(a_1),\ \forall w_2 \in \textit{concr}(a_2),\ w_1\ \textit{on}\ w_2 \in \textit{concr}(a_1\ \textit{on}^{\sim}\ a_2)$

Remark 4. The fact that a_1 and a_2 are in normal form does not necessary lead to a result $a_3 = a_1\ \textit{on}^{\sim}\ a_2$ in normal form.

Definition 8 ($\textit{not}^{\sim}$ operator).
We define an abstract $\textit{not}$ *operator, written* $\textit{not}^{\sim}$*:*

$$\textit{not}^{\sim}\ ([d, D]\,(T)) = \left[\tfrac{-D+1}{T-1}, \max\left(0, 1 - \tfrac{d}{T-1}\right)\right] \left(\tfrac{T}{T-1}\right)$$

with $T > 1$.

The intuition behind this formula is the following. Let $a = [d, D]\left(\frac{l}{n}\right)$ and $w \in \textit{concr}(a)$. It means that over l elements of the word, there are on average n 1*s* in w. So over l instants, there are on average $(l-n)$ 0*s* in w, thus $(l-n)$ 1*s* in $\textit{not}\ w$.

The average distance between two $1s$ in $\texttt{not}\ w$ is $\frac{l}{l-n}$, i.e. $\frac{T}{T-1}$. Concerning the delays, the maximal amount of $0s$ that appear before the first 1 in w is defined by D, and the maximal amount of $1s$ that appear before the first 0 in w is function of d. Thus it is not surprising that in the negation of w, the lower bound of the interval is function of D, and the upper bound of the interval is function of d.

This abstract operator is correct:

Proposition 11. $\forall w \in concr(a),\ \texttt{not}\ w \in concr(\texttt{not}^{\sim}\ a)$

Remark 5. If a is in normal form, then by definition of the $\texttt{not}^{\sim}$ operator, $a' = \texttt{not}^{\sim}\ a$ is also in normal form.

Remark 6. Applying the abstract negation looses some information about the abstracted word: $a \sqsubseteq^{\sim} \texttt{not}^{\sim}\ \texttt{not}^{\sim}\ a$. It is due to the fact that the abstraction can represent words that will eventually begin with d $0s$, by setting the minimum of the interval to d, but cannot represent their negation without loss of information, i.e. words that will eventually begin with d $1s$ (D cannot be negative).

In fact, only the first negation looses information: $\texttt{not}^{\sim}\ a = \texttt{not}^{\sim}\ \texttt{not}^{\sim}\ \texttt{not}^{\sim}\ a$. For example, if $a = [2,3]\left(\frac{5}{3}\right)$ then $\texttt{not}^{\sim}\ a = [-3,0]\left(\frac{5}{2}\right)$, $\texttt{not}^{\sim}\ \texttt{not}^{\sim}\ a = \left[\frac{2}{3},3\right]\left(\frac{5}{3}\right)$ and $\texttt{not}^{\sim}\ \texttt{not}^{\sim}\ \texttt{not}^{\sim}\ a = [-3,0]\left(\frac{5}{2}\right) = \texttt{not}^{\sim}\ a$.

3.4 Abstraction of a Clock

Given an abstraction of words $(abs(w))$, we can compute the abstraction of clocks composed by these words. It is recursively defined as follows:

Definition 9 (clocks abstraction function).

$$\begin{aligned} abs(\texttt{not}\ w) &\overset{def}{=} \texttt{not}^{\sim}\ abs(w) \\ abs(c_1\ \texttt{on}\ c_2) &\overset{def}{=} abs(c_1)\ \texttt{on}^{\sim}\ abs(c_2) \end{aligned}$$

This abstraction of clocks is correct:

Proposition 12. $c \in concr(abs(c))$

3.5 Abstract Relations

We now define the relations on envelopes, corresponding to the relations on words defined in Sec. 2. A relation is verified in the abstract domain if it is verified for all couple of words in the respective concretization sets.

Definition 10 (abstract synchronizability).
$ac_1 \bowtie^{\sim} ac_2 \overset{def}{=} \forall w_1 \in concr(ac_1), w_2 \in concr(ac_2),\ w_1 \bowtie w_2$

We are able to check the synchronizability on envelopes.

Proposition 13 (synchronizability test).
$[d_1, D_1]\,(T_1) \bowtie^{\sim} [d_2, D_2]\,(T_2) \Leftrightarrow T_1 = T_2$

Indeed, if two clocks stay at a bounded distance of there asymptotic rate, then the $1s$ of the first clock stay at a bounded distance of the $1s$ of the second clock iff their rates are equal.

Remark 7. A corollary of this proposition is that $\bowtie^{\sim}$ is reflexive, so every abstract clock is synchronizable with itself. That means that in a concretization set, all couple of words are synchronizable.

The abstraction contains all necessary information to exactly check the synchronizability of two clocks on their abstraction:

Proposition 14. $abs(c_1) \bowtie^{\sim} abs(c_2) \Leftrightarrow c_1 \bowtie c_2$

Proof. From Rem. 7 and by transitivity of $\bowtie$. □

Definition 11 (abstract precedence).
$ac_1 \preceq^{\sim} ac_2 \overset{def}{=} \forall w_1 \in concr(ac_1), w_2 \in concr(ac_2),\ w_1 \preceq w_2$

The precedence on abstract clocks is verified iff the 1*s* of the latest concrete clock in the first envelope arrive before or at the same instant as the 1*s* of the earliest concrete clock in the second envelope:

Proposition 15.
Let $a_1 = [d_1, D_1]\,(T_1)$ *and* $a_2 = [d_2, D_2]\,(T_2)$*.*

$$\begin{aligned} a_1 \preceq^{\sim} a_2 &\Leftrightarrow \sqcup(concr(a_1)) \preceq \sqcap(concr(a_2)) \\ &\Leftrightarrow \forall j \geq 0, [\sqcup(concr(a_1))]_{j+1} \leq [\sqcap(concr(a_2))]_{j+1} \\ &\Leftrightarrow \forall j \geq 0, \lfloor T_1 \times j + D_1 \rfloor \leq \lceil T_2 \times j + d_2 \rceil \end{aligned}$$

When abstract clocks are synchronizable, Prop. 15 can be checked by a sufficient condition. If abstract clocks are in normal form, this condition is also necessary.

Proposition 16 (precedence test).
Let $a_1 = \left[\frac{k_1}{n}, \frac{K_1}{n}\right]\left(\frac{l}{n}\right)$ *and* $a_2 = \left[\frac{k_2}{n}, \frac{K_2}{n}\right]\left(\frac{l}{n}\right)$ *be two envelopes. Then:*

$$\frac{K_1}{n} - \frac{k_2}{n} \leq 1 - \frac{1}{n} \Rightarrow a_1 \preceq^{\sim} a_2$$

Additionally, the converse holds if a_1 *and* a_2 *are in normal form (i.e. here* $gcd(l, n) = 1$*).*

Proof (Intuition). An overlap of less than $1 - \frac{1}{n}$ between $\left[\frac{k_1}{n}, \frac{K_1}{n}\right]$ and $\left[\frac{k_2}{n}, \frac{K_2}{n}\right]$ ensures that $\forall j \geq 0, \lfloor \frac{l}{n} \times j + \frac{K_1}{n} \rfloor \leq \lceil \frac{l}{n} \times j + \frac{k_2}{n} \rceil$. If furthermore a_1 and a_2 are in normal form, then this length is the maximal overlap such that this property is verified. □

To check the precedence relation between clocks on their abstraction, the lack of information about the positions of the 1*s* enforces us to consider the worst case of concretization. This verification on the abstraction is thus correct, but not complete with respect to the verification on the concrete clocks:

Proposition 17. $abs(c_1) \preceq^{\sim} abs(c_2) \Rightarrow c_1 \preceq c_2$

We now define the subtyping relation on envelopes:

Definition 12 (abstract subtyping relation).
$a_1 <:^{\sim} a_2 \overset{def}{=} \forall w_1 \in concr(a_1),\ w_2 \in concr(a_2),\ w_1 <: w_2$

Proposition 18. $a_1 <:^{\sim} a_2 \Leftrightarrow a_1 \bowtie^{\sim} a_2 \wedge a_1 \preceq^{\sim} a_2$

3.6 Computing Buffers Size

To synchronize producers and consumers, buffers are inserted. The question addressed here is the buffer size needed to store a flow produced on a clock of abstraction ac_1, and consumed on a clock of abstraction ac_2, with $ac_1 <:^{\sim} ac_2$.

Proposition 19. *Let* $a_1 = \left[\frac{k_1}{n}, \frac{K_1}{n}\right]\left(\frac{l}{n}\right)$ *and* $a_2 = \left[\frac{k_2}{n}, \frac{K_2}{n}\right]\left(\frac{l}{n}\right)$ *be two envelopes such that* $a_1 <:^{\sim} a_2$*. The minimal buffer needed to be able to communicate from any clock of abstraction* a_1 *to any clock of abstraction* a_2 *is of size:*

$$size(a_1, a_2) = \left\lceil \frac{K_2-(n-1)-k_1}{l} \right\rceil$$

Proof (Intuition). The size of buffer needed to communicate from any clock of a_1 to any clock of a_2 is the size needed to communicate from the earliest clock of a_1 ($\sqcap(concr(a_1))$) to the latest clock of a_2 ($\sqcup(concr(a_2))$). The formula comes from the definition of the calculus of $size$ on concrete clocks and the formulas of $|w[0..i]|_1$ for the infimum and supremum of the concretization sets. $\square$

4 Discussion and Related Work

Back to Periodic Clocks. To be able to check the subtyping relation and compute buffer sizes, the exclusive use of periodic binary words has been proposed in [10].[8]

The periodic behavior of those words allows to statically compute the *on* and *not* operators (definitions become algorithms). In the same way, it allows to check the precedence relation (if it is verified until a certain rank, it will be verified forever) and the synchronizability relation which is equivalent to the equality of rates of 1s in the periodic behavior. Finally, the definition of the minimal buffers size also becomes an algorithm.

However, it can be interesting to avoid exact computations on periodic words because of their cost: for instance, the *on* operation needs a complete traversal of elements of the periods we compose. Moreover if operands have not a compatible size, the result is much longer that the operands. In contexts like video applications, this cost is a problem because the periods length can be huge: in the example cited in [10], a classical downscaler, the output clock has a periodic behavior of length 17280. Adding vertical blanking periods leads to a periodic behavior of size 2073600 (the size of a high definition frame). Computing on abstract values gives a solution to this drawback.

Notice that when periodic clocks are used, the abstraction of words can be automatically computed:

Proposition 20 (periodic binary words abstraction function). *Let* $w = u(v)$ *be an infinite binary word.* $abs(w) = [d, D]\,(T)$ *with* $T = \frac{|v|}{|v|_1}$, $d = \min_{j=0..(|u|_1+|v|_1-1)}([w]_{j+1} - T \times j)$, $D = \max_{j=0..(|u|_1+|v|_1-1)}([w]_{j+1} - T \times j)$.

For example, the abstraction of the downscaler's output clock is:
$abs((10100100)\ \mathit{on}\ 0^{3600}(1)\ \mathit{on}\ (1^{720}0^{720}1^{720}0^{720}0^{720}1^{720}0^{720}0^{720}1^{720}))$
$= \left[-\frac{2}{3}, 0\right]\left(\frac{8}{3}\right)\ \mathit{on}^{\sim}\ [3600, 3600]\,(1)\ \mathit{on}^{\sim}\ \left[\frac{-4315}{4}, \frac{3600}{4}\right]\left(\frac{9}{4}\right) = [6723, 12000]\,(6)$

[8] Those periodic binary words have been used since then to specify statically computed periodic schedules for Latency Insensitive Design [18].

A Particular Case of Periodic Clocks. Affine clocks, presented in [19, 20], are a subset of periodic clocks of the form $0^{\phi}(10^{l-1})$. They have been used to extend the clock calculus of the synchronous data-flow language SIGNAL [21], in the context of hardware/software co-design. Thanks to the simple regular form of those clocks, the extended clock calculus of SIGNAL has a more powerful unification algorithm.

In the case of affine clocks, the abstraction mechanism presented in Sec. 3 is correct and complete. Abstracting a word is trivial: $abs(0^{\phi}(10^{l-1})) = [\phi, \phi]\,(l)$ and doesn't loose any information. Indeed, the concretization set contains only one element (as specified in Sec. 3, $\phi - \phi = 1 - \frac{1}{1}$). Moreover, $abs(c)$ $\textit{\textbf{on}}^{\sim}$ $abs(c')$ has a singleton concretization set: c $\textit{\textbf{on}}$ c', and is in normal form. Clocks being in normal form and concretization sets being singletons, testing precedence relations on the abstraction is equivalent to testing them on the concrete clocks. As it is also the case for the synchronizability relation, we have
$abs(c) <:^{\sim} abs(c') \Leftrightarrow c <: c'$.

5 Conclusion

This paper generalizes the classical notion of clocks in synchronous data-flow languages by allowing to deal with sets of clocks. This is based on the introduction of *clock envelopes* which define intervals of clocks up to bounded buffering. We have focused on the algebraic properties of those clocks and illustrated their expressive power. The motivation behind them is essentially pragmatic and gives some answer to the need to model jittering phenomena, execution time and more generally communication through bounded buffering. The real novelty is to deal with quantitative properties during the clock calculus instead of simply strict synchrony as done usually. We have experimented the use of these clocks on several examples (e.g., video picture-and-picture or filters in software defined radio). The extension of the existing clock calculus of LUCID SYNCHRONE [22] compiler is under way.

Acknowledgements. Pascal Raymond showed us an unexpected (and very elegant) use of the Lustre compiler to generate clocks within an envelope. We also thank the anonymous reviewers for their helpful comments.

References

1. Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE **91**(1) (January 2003)
2. Benveniste, A., Berry, G. In: The synchronous approach to reactive and real-time systems. Kluwer Academic Publishers, Norwell, MA, USA (2002) 147–159
3. Milner, R.: Calculi for synchrony and asynchrony. Theoretical Computer Science **25**(3) (1983) 267–310
4. Arnold, A.: Systèmes de transitions et sémantique des processus communicants. Masson (1992)

5. Colaço, J.L., Pouzet, M.: Clocks as First Class Abstract Types. In: Third International Conference on Embedded Software, Philadelphia, USA (October 2003)
6. Amagbegnon, T., Besnard, L., Guernic., P.L.: Implementation of the data-flow synchronous language signal. In: Programming Languages Design and Implementation, ACM (1995) 163–173
7. Caspi, P., Curic, A., Maignan, A., Sofronis, C., Tripakis, S.: Translating Discrete-Time Simulink to Lustre. ACM Transactions on Embedded Computing Systems (2005) Special Issue on Embedded Software.
8. Scicos: `http://www-rocq.inria.fr/scicos`.
9. Milner, R.: A theory of type polymorphism in programming. Journal of Computer and System Sciences **17**(3) (1978) 348–375
10. Cohen, A., Duranton, M., Eisenbeis, C., Pagetti, C., Plateau, F., Pouzet, M.: N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems. In: ACM International Conference on Principles of Programming Languages. (January 2006)
11. Lee, E., Messerschmitt, D.: Synchronous dataflow. IEEE Trans. Comput. **75**(9) (1987)
12. Buck, J., Ha, S., Lee, E., Messerschmitt, D.: Ptolemy: A framework for simulating and prototyping heterogeneous systems. International Journal of computer Simulation (1994) special issue on Simulation Software Development.
13. Vuillemin, J.: On Circuits and Numbers. Technical report, Digital, Paris Research Laboratory (1993)
14. Curic, A.: Implementing Lustre Programs on Distributed Platforms with Real-time Constraints. PhD thesis, Université Joseph Fourier (2005)
15. Sofronis, C.: Embedded Code Generation from High-level Heterogeneous Components. PhD thesis, Université Joseph Fourier (2006)
16. Halbwachs, N., Mandel, L.: Simulation and verification of asynchronous systems by means of a synchronous model. In: Sixth International Conference on Application of Concurrency to System Design, Turku, Finland (June 2006)
17. Krstic, S., Cortadella, J., Kishinevsky, M., O'Leary, J.: Synchronous elastic networks. In: Proceedings of the Formal Methods in Computer Aided Design, Washington, DC, USA, IEEE Computer Society (2006) 19–30
18. Boucaron, J., de Simone, R., Millo, J.V.: Formal methods for scheduling of latency-insensitive designs. EURASIP Journal on Embedded Systems **Issue 1**(ISSN:1687-3955) (January 2007) 8 – 8
19. Smarandache, I.M., Guernic, P.L.: Affine transformations in SIGNAL and their application in the specification and validation of real-time systems. In: ARTS. (1997) 233–247
20. Smarandache, I.M., Gautier, T., Guernic, P.L.: Validation of mixed SIGNAL-ALPHA real-time systems through affine calculus on clock synchronisation constraints. In: World Congress on Formal Methods (2). (1999) 1364–1383
21. Benveniste, A., Guernic, P.L., Jacquemot, C.: Synchronous programming with events and relations: the SIGNAL language and its semantics. Sci. Comput. Program. **16**(2) (1991) 103–149
22. Pouzet, M.: Lucid Synchrone, version 3. Tutorial and reference manual. Université Paris-Sud, LRI. (April 2006) Distribution available at: `www.lri.fr/~pouzet/lucid-synchrone`.