Relaxing Synchronous Composition with Clock Abstraction

Albert Cohen, Louis Mandel, Florence Plateau, Marc Pouzet

Relaxing Synchronous Composition with Clock Abstraction *

Albert Cohen Louis Mandel Florence Plateau Marc Pouzet
Université Paris-Sud 11 and INRIA Saclay - Ile-de-France

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 as illustrated in the following example ($*$ stands for the absence of relevant value as in [6]).

x	x_1	x_2	x_3	$*$	x_4	$*$	x_5	x_6	$*$	$*$	x_7	$*$	x_8	$*$	x_9	...
w_1	1	1	1	0	1	0	1	1	0	0	1	0	1	0	1	...

In synchronous languages [1], a binary composition like $x + y$ expects its two arguments x and y to be synchronous, i.e., to have the same clocks. Clocks are a mean to model different time-scales in the system. The purpose of the clock-calculus in these languages is to check that the actual clock input of a function matches the expected clock and this is mostly a typing problem. In existing synchronous languages, the clock calculus relies on clock equality only.

In [4] we have introduced a way to compose streams which have *almost the same clock* and can be synchronized through the use of a finite buffer. For example, a stream with clock $w_1 = 111(0101100101)$ can be used later, for example on clock $w_2 = 0(000111)$ since 1's in w_2 appears later than 1's in w_1 and we write $w_1 <: w_2$. This relaxed model is achieved by introducing a subtyping rule in the type system to localise places where some synchronization code must be inserted. Subtyping can be checked when clocks are defined by ultimately periodic infinite binary words [7]. Nevertheless, this check can be costly if the patterns are long, and is in any case restricted to periodic behaviors only.

To achieve this relaxed model on non periodic clocks, the notion of *clock envelopes* has been introduced [5]. These envelopes are abstractions of clocks that give bounds on the number of instants where the stream has been present since the beginning of the execution. An envelope is a set of (not necessarily periodic) clocks and subtyping can be checked by simple arithmetic comparison.

This work presents a new abstraction of clocks. Consider the binary word $w_2 = 0(000111)$. It can be represented graphically by its cumulative function: a rising edge at index i of the chronogram corresponds to the occurrence of a 1 in the word (Figure 1). The abstraction $a_2 = \langle -\frac{4}{2}, -\frac{3}{2}, \frac{1}{2} \rangle$ is represented by two lines, of equation $\frac{1}{2} \times i - \frac{3}{2}$ and $\frac{1}{2} \times i - \frac{4}{2}$. It thus consists in keeping a slope, here $\frac{1}{2}$ and two shifts here $-\frac{3}{2}$ and $-\frac{4}{2}$. The red line gives a upper bound to the occurrence of rising edges, and the green dotted line gives a lower bound to the absence of edge.

In Figure 2, consider that a stream is produced on clock $w_1 = 111(0101100101)$ and consumed on clock w_2. This is possible through a FIFO because the green line bounding w_1 is above the red line bounding w_2. The fact that the lines bounding w_1 and w_2 have the same slope ensures that the difference between the number of values produced and consumed is bounded. Thus, a bounded FIFO is enough and its size is the maximal vertical difference between w_1 and w_2. A correct over-approximation is the vertical distance between the red line of w_1 and the green line of w_2. Checking subtyping on abstractions is thus done very efficiently by testing the equality of the slopes and comparing the shifts. The size of the buffer is also computed efficiently and is the difference between two shifts.

The present work brings mostly three new contributions with respect to [4, 5]:

- The new definition of clock abstraction allows to abstract clocks which were not considered previously such as those containing a finite number of 1's. For example, the clock $11001011(0)$ is abstracted

*This work was partially supported by the INRIA research project Synchronics.

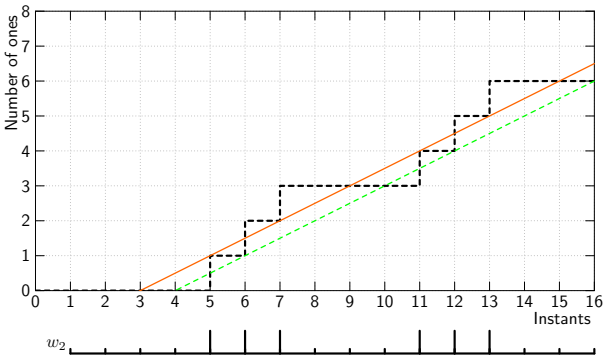


Figure 1: Chronogram of clock w_2 and its abstraction $a_2 = \langle -\frac{4}{2}, -\frac{3}{2}, \frac{1}{2} \rangle$.

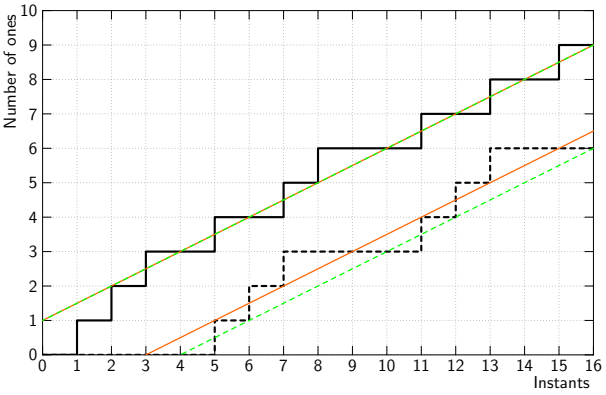


Figure 2: Communication through a bounded FIFO, from w_1 (abstracted by $a_1 = \langle 1, 1, \frac{1}{2} \rangle$) to w_2.

as $\langle 3, 5, 0 \rangle$. As a consequence, we are able to abstract any infinite binary words bounded by two ultimately periodic infinite binary words.

- The new abstraction is more precise than the previous one, i.e., the size of the abstraction of a clock is smaller.
- Finally, algebraic properties of this new abstraction have been formalized and proved in the proof assistant Coq [2]. It has been a strong support to the reflexion and gives confidence in the results we have obtained. These proofs are available at `http://www.lri.fr/~plateau/hfl09`.

Clocks defined as ultimately periodic infinite binary words were introduced to model features which are usually not considered in synchronous languages such as communication delay or the composition through bounded buffers. Clock abstraction are a simple mean allowing to reason in *average* on those clocks. These features are an important concern in hardware design. We are very interested in presenting our work to this community, and relate it to existing methods such as [3, 6].

References

[1] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1), January 2003.

[2] Yves Bertot and Pierre Castran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Springer-Verlag., 2004.

[3] Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli. Coping with latency in soc design. *IEEE Micro*, 22(5):24–35, 2002.

[4] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems. In *ACM International Conference on Principles of Programming Languages*, January 2006.

[5] Albert Cohen, Louis Mandel, Florence Plateau, and Marc Pouzet. Abstraction of Clocks in Synchronous Data-flow Systems. In *The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS 08)*, Bangalore, India, December 2008.

[6] Sava Krstic, Jordi Cortadella, Mike Kishinevsky, and John O'Leary. Synchronous elastic networks. In *Proceedings of the Formal Methods in Computer Aided Design*. IEEE Computer Society, 2006.

[7] Jean Vuillemin. On Circuits and Numbers. Technical report, Digital, Paris Research Laboratory, 1993.

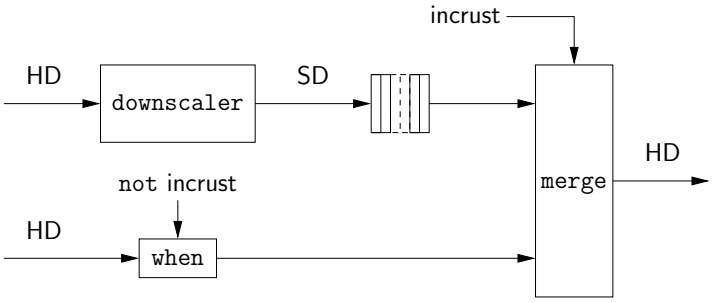
Relaxing Synchronous Composition with Clock Abstraction

Albert Cohen Louis Mandel Florence Plateau Marc Pouzet

Laboratoire de Recherche en Informatique
Université Paris-Sud 11
INRIA Saclay – Ile-de-France

HFL – 28-29/03/2009

Motivation: Picture in Picture Example



Incrustation of a Standard Definition (SD) image in a High Definition (HD) one

- `downscaler`: reduction of an HD image (1920×1080 pixels) to an SD image (720×480 pixels)
- `when`: removal of a part of an HD image
- `merge`: incrustation of an SD image in an HD image

Question: programming Kahn Networks with bounded buffers

- delay introduced by the picture in picture in the video processing chain?
- buffer size needed between the `downscaler` and the `merge` nodes?

Synchronous and n-Synchronous Model

Synchronous dataflow languages [CP96]

- programming Kahn Networks without buffers
- clocks define instants when data transfers occur
  - ▷ they can be arbitrarily complex
- clock calculus: dedicated type system
  - ▷ computes the activation paces
  - ▷ rejects programs that cannot be executed without buffers

n-Synchronous model [CDE$^+$06]

- programming Kahn Networks with bounded buffers
- clock calculus with a subtyping rule
  - ▷ rejects programs that cannot be executed with bounded buffers
  - ▷ computes the size of the buffers
  - ▷ computes initial delays to avoid reading in an empty buffer

Contribution

Previous work:

- n-synchrony can be checked on periodic clocks

Contribution:

- dealing with long patterns in periodic clocks
- modeling with jitter ("almost periodic" clocks)
- modeling nodes execution time

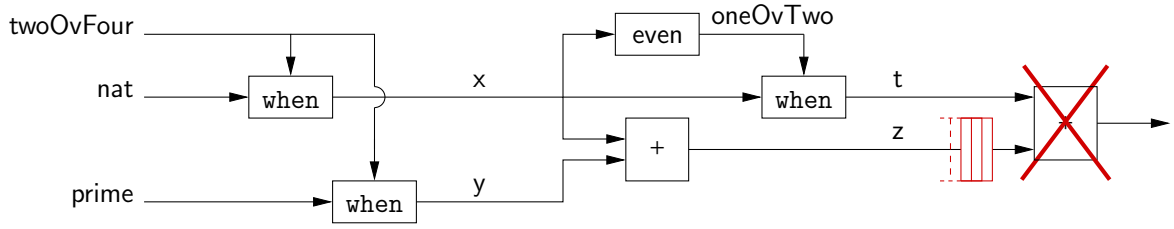
n-synchrony can be checked by abstracting clocks

Overview

1. Synchronous and n-Synchronous Models
2. n-Synchronous Model Formally
3. Abstraction of Clocks
4. Applications
5. Related Work

Synchronous and n-Synchronous Models

Synchronous Dataflow Languages (Lustre, Signal, Lucid Synchrone)

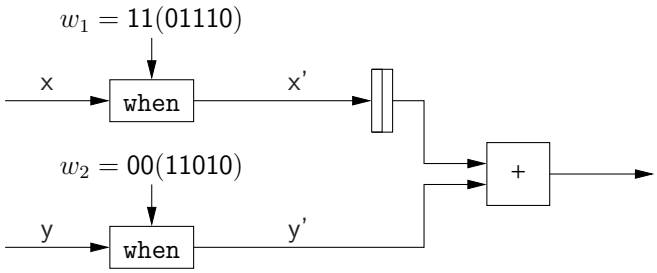


name	value							clock
nat	0	1	2	3	4	5	...	$(1)^\omega$
prime	2	3	5	7	11	13	...	$(1)^\omega$
twoOvFour	1	1	0	0	1	1	...	$(1)^\omega$
x = nat `when` twoOvFour	0	1			4	5	...	$(1)^\omega$ *on* twoOvFour
y = prime `when` twoOvFour	2	3			11	13	...	$(1)^\omega$ *on* twoOvFour
z = x + y	2	4			15	18	...	$(1)^\omega$ *on* twoOvFour
oneOvTwo = even x	1	0			1	0	...	$(1)^\omega$ *on* twoOvFour
t = x `when` oneOvTwo	0				4		...	$(1)^\omega$ *on* twoOvFour *on* oneOvTwo

synchronous operator cannot be applied to z and t: they do not have the same clock

here, an infinite buffer would be needed

Relaxing Synchronous Composition



x'	x_1	x_2		x_4	x_5	x_6			x_9	x_{10}
y'			y_3	y_4		y_6		y_8	y_9	
buffer(x') + y'			x_1+y_3	x_2+y_4		x_4+y_6		x_5+y_8	x_6+y_9	

x' and y' can be composed through the introduction of a two places buffer

In the synchronous model, this system is rejected. Adding buffer code by hand is feasible but hard and error-prone. We need a relaxed model of synchrony and a relaxed clock calculus.

Relaxing Synchronous Composition

$$\text{(SUB)}\quad \frac{H \vdash e : ck \text{ on } w_1 \quad w_1 <: w_2}{H \vdash e : ck \text{ on } w_2}$$

n-Synchronous model:

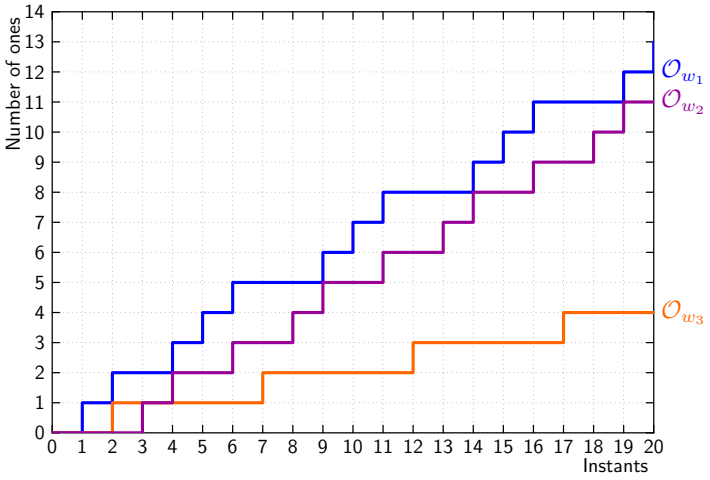
- w_1 is a subtype of w_2 ($w_1 <: w_2$) means that a flow of clock w_1 can be consumed on clock w_2 through the insertion of a bounded buffer
- buffers are inserted at each subtyping point
- buffer size is computed automatically

Previous example:

- as $11(01110) <: 00(11010)$ the consumption of x' on clock w_2 is accepted
- a two places buffer is automatically inserted

n-Synchronous Model Formally

Clocks as Infinite Binary Words



$\mathcal{O}_w(i) =$ cumulative function of 1s of w

If a flow is produced on clock w_1, buffer size needed to consume it on clock w_2 is:

$$size(w_1, w_2) = \max_{i \in \mathbb{N}} (\mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i))$$

Relations on Words

Synchronizability ($\bowtie$): buffer size is bounded

$$w_1 \bowtie w_2 \overset{def}{\Leftrightarrow} \exists b_1, b_2 \in \mathbb{Z}, \forall i,\ b_1 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq b_2$$

- previous figure: $w_1 \bowtie w_2$, $w_1 \not\bowtie w_3$ and $w_2 \not\bowtie w_3$

Precedence ($\preceq$): no read in an empty buffer

$$w_1 \preceq w_2 \overset{def}{\Leftrightarrow} \forall i,\ \mathcal{O}_{w_1}(i) \geq \mathcal{O}_{w_2}(i)$$

- previous figure: $w_1 \preceq w_2$, $w_1 \preceq w_3$ and $w_2 \not\preceq w_3$

Subtyping ($<:$): flow produced on w_1 can be consumed on w_2

$$w_1 <: w_2 \overset{def}{\Leftrightarrow} \exists n \in \mathbb{N}, \forall i,\ 0 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq n$$

- previous figure: $w_1 <: w_2$, $w_1 \not<: w_3$ and $w_2 \not<: w_3$

Operators on Words

Composed clocks: $c ::= w \mid \mathit{not}\ w \mid c\ \mathit{on}\ c$

Operators on flows

- negation:
$$\begin{cases} \mathit{not}\ 1.w &= 0.\mathit{not}\ w \\ \mathit{not}\ 0.w &= 1.\mathit{not}\ w \end{cases}$$
- sampling:
$$\begin{cases} 1.w_1\ \mathit{on}\ 1.w_2 &= 1.(w_1\ \mathit{on}\ w_2) \\ 1.w_1\ \mathit{on}\ 0.w_2 &= 0.(w_1\ \mathit{on}\ w_2) \\ 0.w_1\ \mathit{on}\ w_2 &= 0.(w_1\ \mathit{on}\ w_2) \end{cases}$$

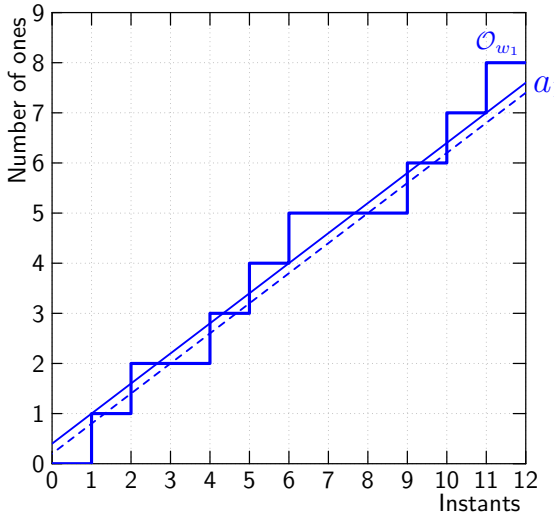
Operators on cumulative functions

- negation: $\mathcal{O}_{\mathit{not}\ w}(i) = i - \mathcal{O}_w(i)$
- sampling: $\mathcal{O}_{w_1\ \mathit{on}\ w_2}(i) = \mathcal{O}_{w_2}(\mathcal{O}_{w_1}(i))$

Abstraction of Clocks

This idea has been introduced in [CMPP08]. We present here an improved abstraction.

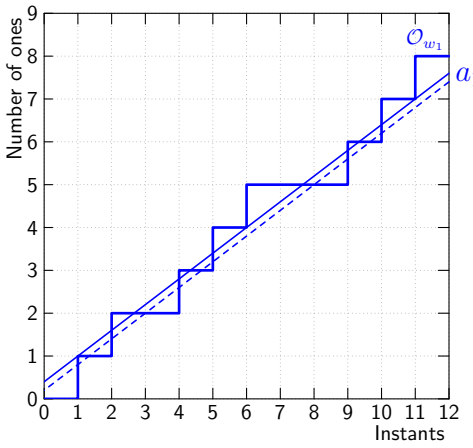
Abstraction of Infinite Binary Words

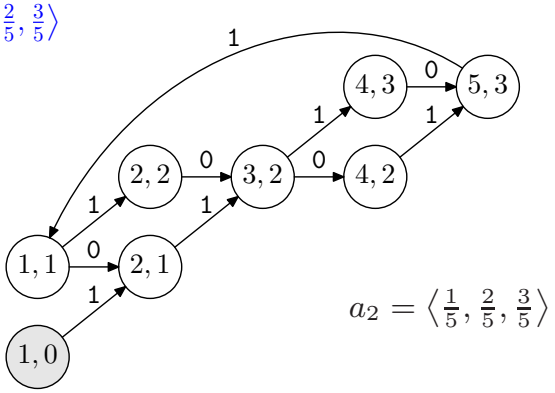


A word w can be abstracted by two lines: $abs(w) = \langle b^0, b^1, r \rangle$

$$concr\left(\langle b^0, b^1, r\rangle\right) \overset{def}{\Leftrightarrow} \left\{ w,\ \forall i \geq 1,\ \wedge \begin{array}{lcl} w[i] = 1 & \Rightarrow & \mathcal{O}_w(i-1) \leq r \times i + b^1 \\ w[i] = 0 & \Rightarrow & \mathcal{O}_w(i-1) \geq r \times i + b^0 \end{array} \right\}$$

Abstract Clocks as Automata





- set of states $\{(i,j) \in \mathbb{N}^2\}$: coordinates in the 2D-chronogram
- finite number of state equivalence classes
- transition function δ: $\begin{cases} \delta(1,(i,j)) = nf(i+1,j+1) & \text{if } j \leq r \times i + b^1 \\ \delta(0,(i,j)) = nf(i+1,j) & \text{if } j \geq r \times i + b^0 \end{cases}$
- allows to check/generate clocks

Clocks Generator (Lucid Synchrone)

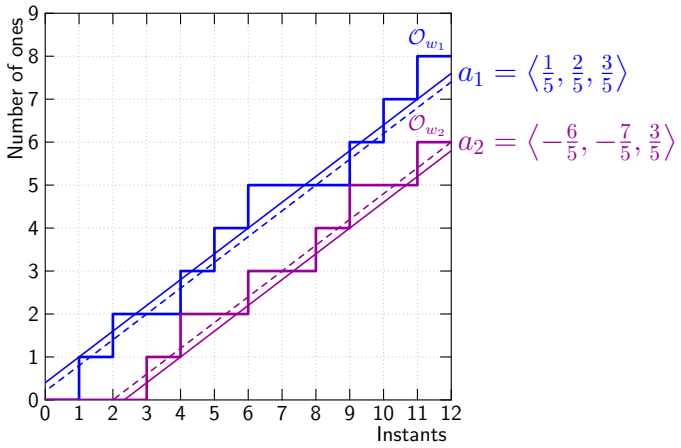
```
type rat = { num: int; den: int; }

let nf { num = n; den = l; } i j =
  if i >= l && j >= n then (i - l, j - n) else (i, j)

let node generate choice (b0, b1, r) = clk where
  rec i, j = (1,0) fby nf r (i+1) (if clk then j + 1 else j)
  and one = (rat_of_int j) <=: (r *: (rat_of_int i) +: b1)
  and zero = (rat_of_int j) >=: (r *: (rat_of_int i) +: b0)
  and clk = choice one zero

let node early a = generate (fun x y -> x) a
let node late a = generate (fun x y -> not y) a
```

Abstract Relations



Synchronizability: $r_1 = r_2 \Leftrightarrow \langle {b^0}_1, {b^1}_1, r_1 \rangle \bowtie^{\sim} \langle {b^0}_2, {b^1}_2, r_2 \rangle$

Precedence: ${b^0}_1 > {b^1}_2 \Rightarrow \langle {b^0}_1, {b^1}_1, r_1 \rangle \preceq^{\sim} \langle {b^0}_2, {b^1}_2, r_2 \rangle$

Subtyping: $a_1 <:^{\sim} a_2 \Leftrightarrow a_1 \bowtie^{\sim} a_2 \wedge a_1 \preceq^{\sim} a_2$

▷ proposition: $abs(w_1) <:^{\sim} abs(w_2) \Rightarrow w_1 <: w_2$

▷ buffer: $size(a_1, a_2) = \lfloor {b^1}_1 - {b^0}_2 \rfloor + 1$

Abstract Operators

Abstraction of a composed clock:

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

Operators definition:

- $\mathit{not}^{\sim} \langle b^0, b^1, r \rangle = \langle -b^1 - 1,\ b^0 - 1,\ 1 - r \rangle$
- $\langle {b^0}_1, {b^1}_1, r_1 \rangle\ \mathit{on}^{\sim} \langle {b^0}_2, {b^1}_2, r_2 \rangle = \langle {b^0}_{12},\ {b^1}_{12},\ r_1 \times r_2 \rangle$
  with $\quad {b^0}_{12} = {b^0}_1 \times r_2 + {b^0}_2, \quad {b^1}_{12} = ({b^1}_1 + 1) \times r_2 + {b^1}_2$
  and $\quad {b^0}_1 \leq 0, \quad {b^0}_2 \leq 0$

Correctness property:

$$\begin{aligned} \mathit{not}\ w &\in concr(\mathit{not}^{\sim}\ abs(w)) \\ c_1\ \mathit{on}\ c_2 &\in concr(abs(c_1)\ \mathit{on}^{\sim}\ abs(c_2)) \end{aligned}$$

Formalization in a Proof Assistant

Most of the properties have been proved in Coq

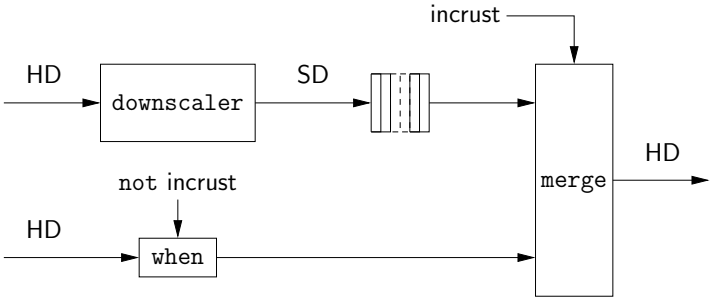
- available at: `http://www.lri.fr/~plateau/hfl09`
- example of property

```
Property on_absh_correctness:
  forall (w1:ibw) (w2:ibw),
  forall (a1:abstractionh) (a2:abstractionh),
  forall H_wf_a1: well_formed_abstractionh a1,
  forall H_wf_a2: well_formed_abstractionh a2,
  forall H_a1_eq_absh_w1: in_abstractionh w1 a1,
  forall H_a2_eq_absh_w2: in_abstractionh w2 a2,
  in_abstractionh (on w1 w2) (on_absh a1 a2).
```

- number of Source Lines of Code
  - specifications: about 1600 SLOC
  - proofs: about 5000 SLOC

Applications

Back to the Picture in Picture Example



- abstraction of downscaler output:

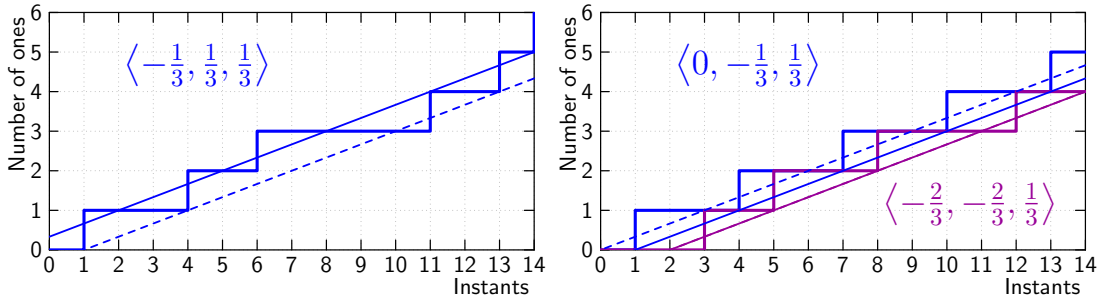
  $abs((10100100)\ on\ 0^{3600}(1)\ on\ (1^{720}0^{720}1^{720}0^{720}0^{720}1^{720}0^{720}0^{720}1^{720}))$

  $= \langle 0, -\frac{1}{8}, \frac{3}{8} \rangle\ on^{\sim}\ \langle -3600, -3601, 1 \rangle\ on^{\sim}\ \langle -400, 479, \frac{4}{9} \rangle = \langle -2000, -\frac{20171}{18}, \frac{1}{6} \rangle$

- minimal delay and buffer:

	delay	buffer size
exact result	$9\,598$ ($\approx$ time to receive 5 HD lines)	$192\,240$ (≈ 267 SD lines)
abstract result	$11\,995$ ($\approx$ time to receive 6 HD lines)	$193\,079$ (≈ 268 SD lines)

Applications



Modeling with jitter:

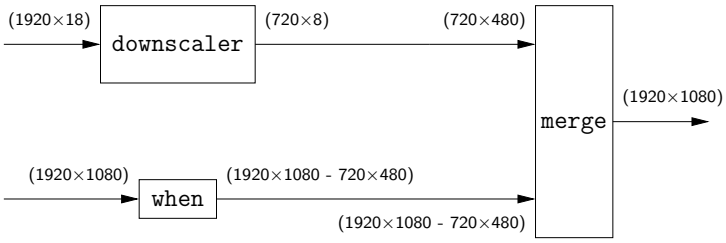
- set of clock of rate $r = \frac{1}{3}$ and jitter 1 can be specified by $\langle -\frac{1}{3}, \frac{1}{3}, \frac{1}{3} \rangle$

Modeling execution time:

- f must be executed every three cycles and its execution takes between one and two cycles
- $f :: \forall \alpha.\alpha\ \mathrm{on}^{\sim} \langle 0, -\frac{1}{3}, \frac{1}{3} \rangle \to \alpha\ \mathrm{on}^{\sim} \langle -\frac{2}{3}, -\frac{2}{3}, \frac{1}{3} \rangle$

Related Work

Synchronous Data Flow Graphs [LM87]



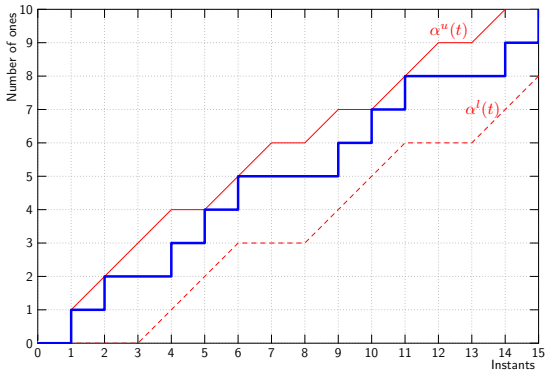
Synchronous Data Flow Graphs (SDF): special case of Kahn networks

- specify on the wires the number of values consumed and produced by each node at each activation
- *balance equations* allow to check that buffers needed are bounded
- static scheduling algorithm allows to compute buffer size

Cyclo Static Data Flow: generalization of SDF

- specify number of values produced and consumed as periodic sequences

Real-Time Calculus [TCG+01]



Method used to obtain bounds on delay and buffers in distributed embedded systems

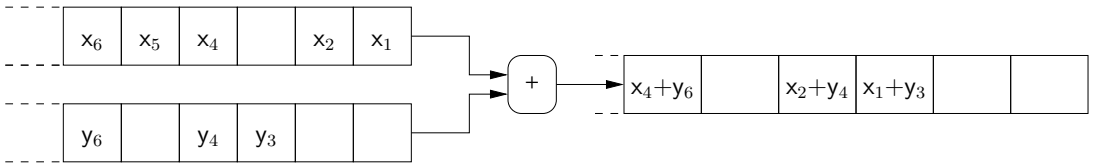
Based on bounding cumulative functions but with respect to a sliding window

- arrival curves $(\alpha^l(t), \alpha^u(t))$ characterize event streams
- service curves $(\beta^l(t), \beta^u(t))$ characterize nodes

Other similarities

- buffer size: maximal vertical distance between $\alpha^u(t)$ and $\beta^l(t)$
- delay: maximal horizontal distance between $\alpha^u(t)$ and $\beta^l(t)$

Latency Insensitive Design [CMSV01]



Method used to define synchronous circuits that tolerate any variation of the data transfer latency

- design synchronous IPs and interconnect them
- analysing the sum of delays and initial values on each cycle indicates whether the system is alive
- elastic circuits dynamic schedule [KCKO06]: to handle latency, every wire is transformed into a channel carrying data and control bits
- k-periodic static schedule [BdSM07]: maximize rate and minimize storage elements by insertion of fractional registers and computation of an explicit schedule

Conclusion

- n-synchronous model allows a more flexible composition of nodes in Synchronous Dataflow Systems
- abstraction of clocks allows to use this model on systems involving jitter and execution time, and to treat efficiently clocks with long periodic patterns
- correctness of abstract relations and operators have been proved in Coq
- Current work:
  integration in the Lucid Synchrone dataflow synchronous language

`http://www.lri.fr/~plateau/hfl09/`

References

[BdSM07] Julien Boucaron, Robert de Simone, and Jean-Vivien Millo, Formal methods for scheduling of latency-insensitive designs, EURASIP Journal on Embedded Systems **Issue 1** (2007), no. ISSN:1687-3955, 8 – 8.

[CDE+06] A. Cohen, M. Duranton, Ch. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet, N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems, ACM International Conference on Principles of Programming Languages, January 2006.

[CMPP08] A. Cohen, L. Mandel, F. Plateau, and M. Pouzet, Abstraction of Clocks in Synchronous Data-flow Systems, The Sixth ASIAN Symposium on Programming Languages and Systems, December 2008.

[CMSV01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, Theory of latency-insensitive design, IEEE Trans. on CAD of Integrated Circuits and Systems **20** (2001), no. 9, 1059–1076.

[CP96] P. Caspi and M. Pouzet, Synchronous Kahn Networks, ACM SIGPLAN International Conference on Functional Programming, May 1996.

[KCKO06] S. Krstic, J. Cortadella, M. Kishinevsky, and J. O'Leary, Synchronous elastic networks, Proceedings of the Formal Methods in Computer Aided Design, 2006.

[LM87] E. Lee and D. Messerschmitt, Synchronous dataflow, IEEE Trans. Comput. **75** (1987), no. 9.

[TCG+01] L. Thiele, S. Chakraborty, M. Gries, Er Maxiaguine, and J. Greutert, Embedded software in network processors – models and algorithms, In First Workshop on Embedded Software, LNCS 2211, Springer Verlag, 2001, pp. 416–434.