Lucy-n: a n-Synchronous Extension of Lustre

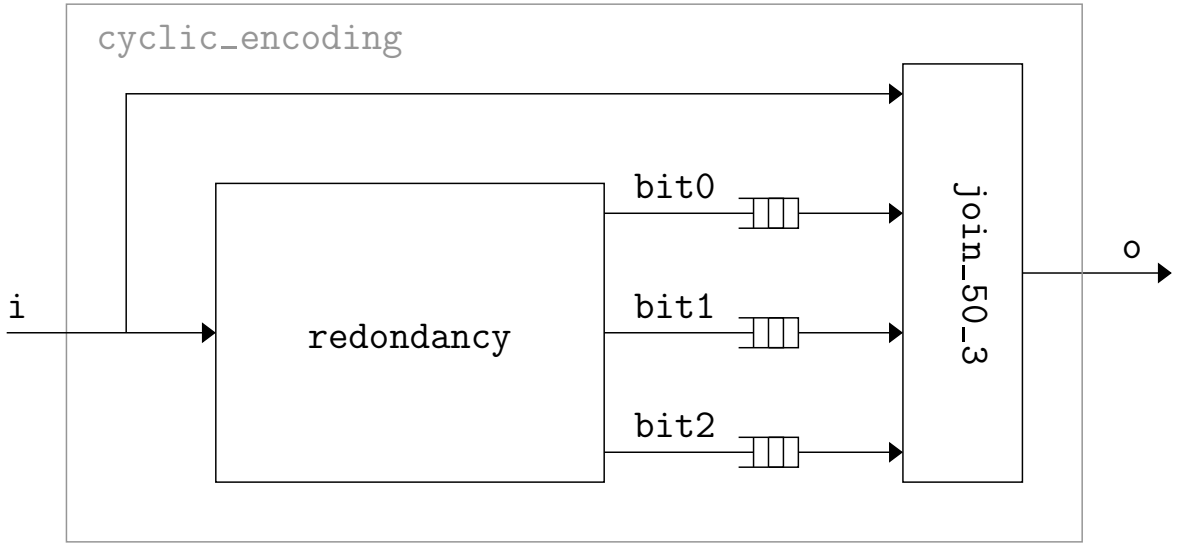
Louis Mandel Florence Plateau Marc Pouzet

Parkas Team

École Normale Supérieure
University Paris-Sud 11
INRIA

GGJJ 2011

Kahn Process Networks [Gilles Kahn, 1974]



Network of processes

- concurrent execution
- communication through buffers of sufficient size

If processes are deterministic then the network is deterministic

Programming Kahn Process Networks

Problem: computation of sufficient buffer sizes

- risk of data loss, of deadlock
- sometimes, need of infinite buffers

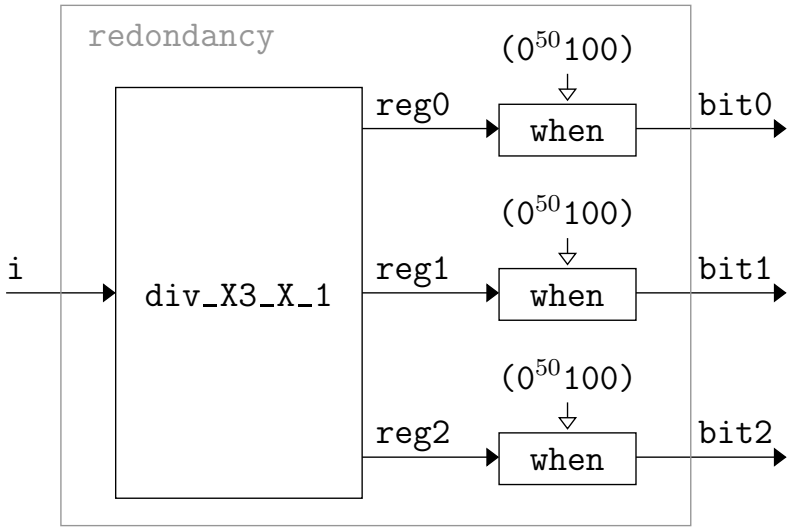
Goal:

- rejection of infinite buffers
- automatic sizing of buffers

Related work:

- Synchronous Data Flow and variants [Lee *et al.*] [Buck]
- scheduling [Carlier, Chretienne] [Baccelli, Cohen, Quadrat]
- Network Calculus [Cruz], Real-time Calculus [Thiele *et al.*]

Dataflow Synchronous Model

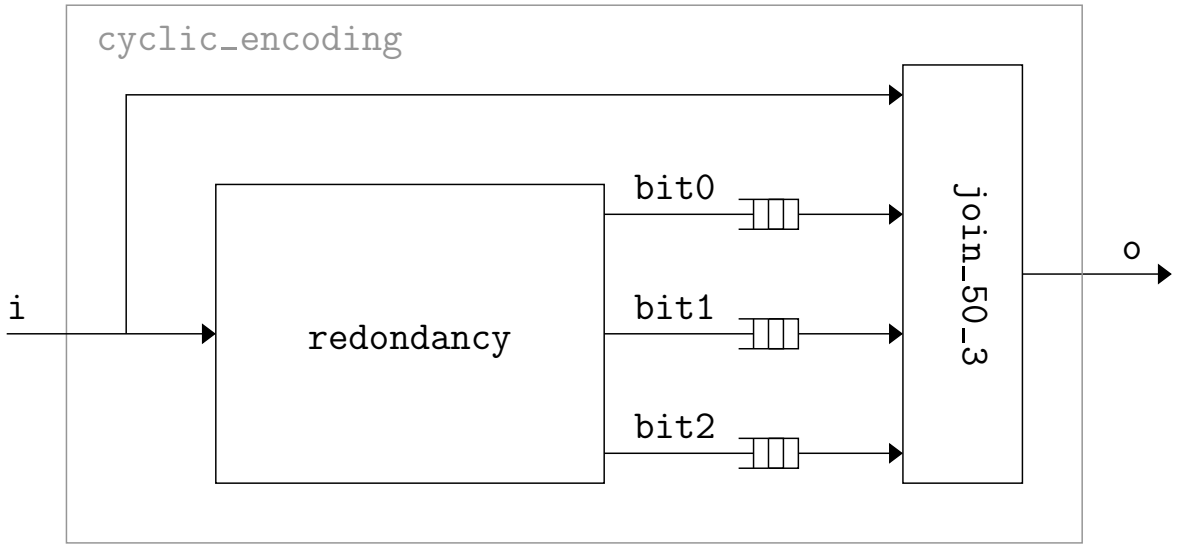


Programming Kahn networks without buffers:

- programming languages: Lustre, Signal, Lucid Synchrone
- instantaneous consumption of produced data
- strong guaranties: bounded memory, absence of deadlocks

But: communication without buffers sometimes too restrictive
(e.g. multimedia applications)

n-Synchronous Model



Programming Kahn networks with bounded memory

Automatic methods at compile time to:

- reject networks needing infinite memory
- compute activation paces of computations nodes
- compute sufficient buffers sizes

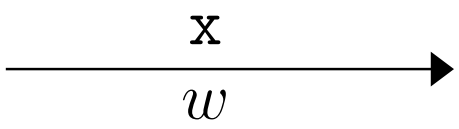
More flexibility with the same guaranties

Overview

1. Lucy-n: a n-Synchronous Extension of Lustre
2. Adaptability Relation
3. Clock Typing
4. Abstract Clocks

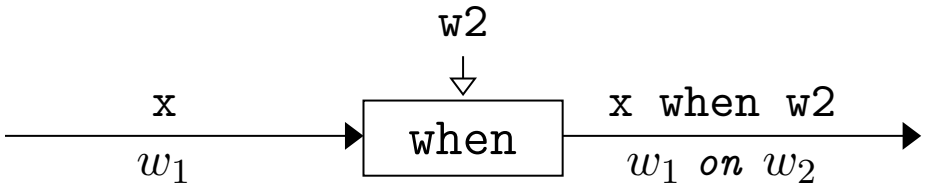
Lucy-n: a n-Synchronous Extension of Lustre

Flows and Clocks



x	2	5		3		7	9	4			6	...
$w = clock(\mathtt{x})$	1	1	0	1	0	1	1	1	0	0	1	...

Sampling



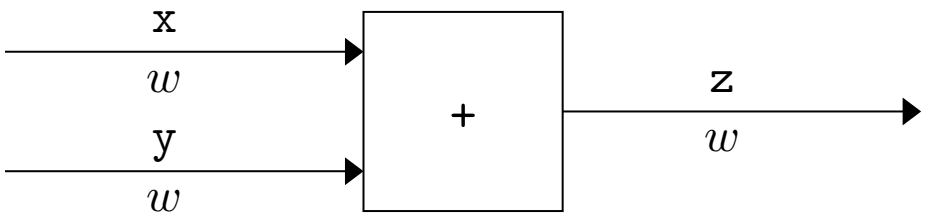
x	2	5	3	7	9	...
w2	1	0	1	1	0	...
x when w2	2		3	7		...

$$clock(\texttt{x when w2}) = clock(\texttt{x}) \textit{ on } w_2$$

Definition:

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

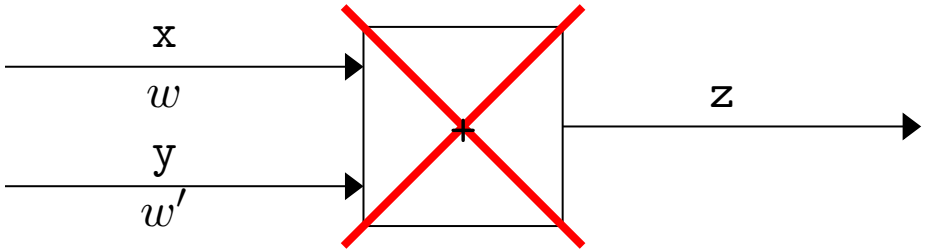
Composition



x	2	5	3	7	9	4	6	...
y	5	3	2	2	0	2	1	...
z = x + y	7	8	5	9	9	6	7	...

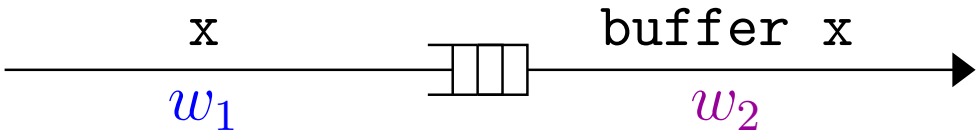
$$clock(\mathrm{x}) = clock(\mathrm{y}) = clock(\mathrm{z})$$

Composition



x	2	5		3		7	9	4			6	...
y	5		3		2	2	0			2	1	...
z = x + y													

Buffering



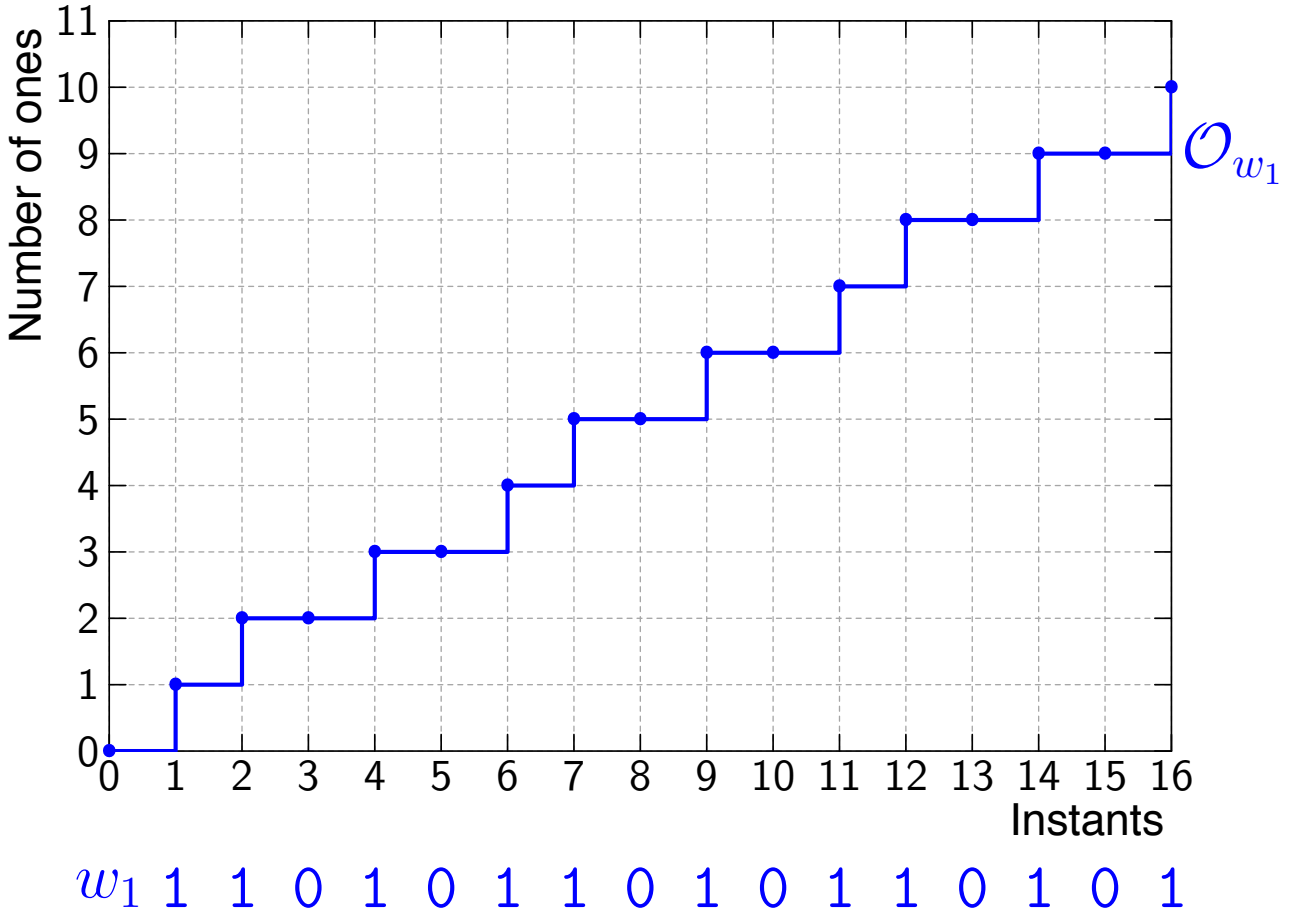
Communication through a bounded buffer:

the input's clock must be **adaptable** to the output's clock

$$w_1 \quad <: \quad w_2$$

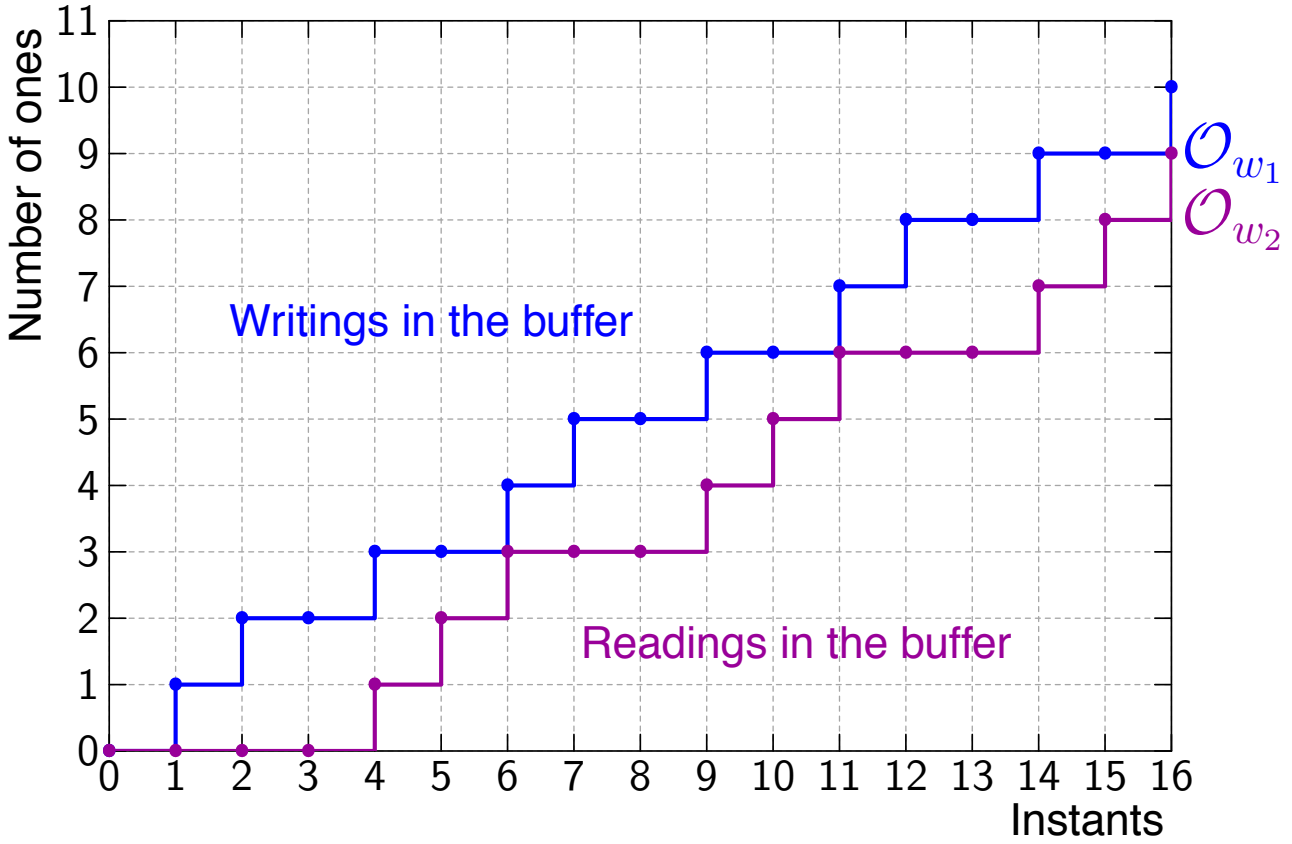
Adaptability Relation

Cumulative Function



$\mathcal{O}_{w_1}$: cumulative function of the word w_1

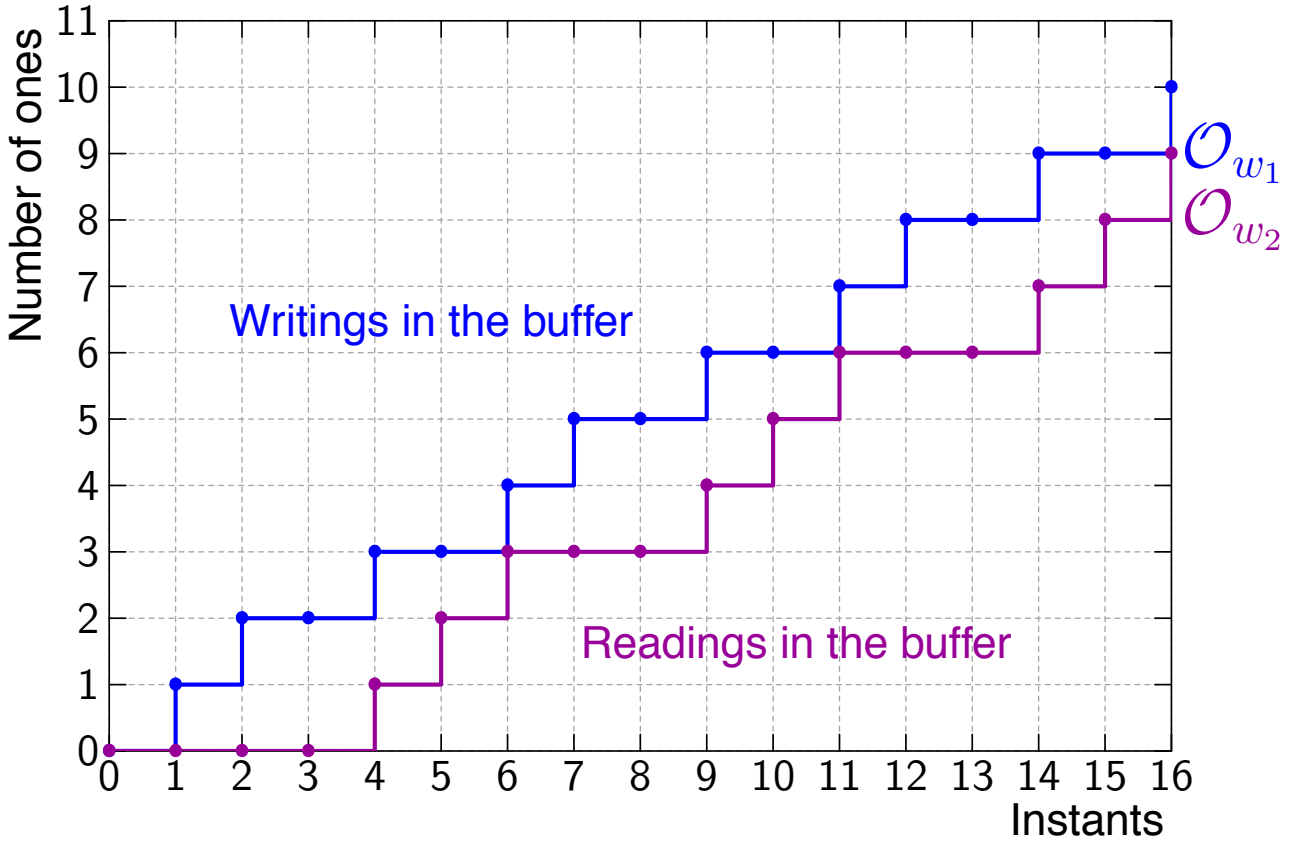
Adaptability Relation



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

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

Ultimately Periodic Clocks



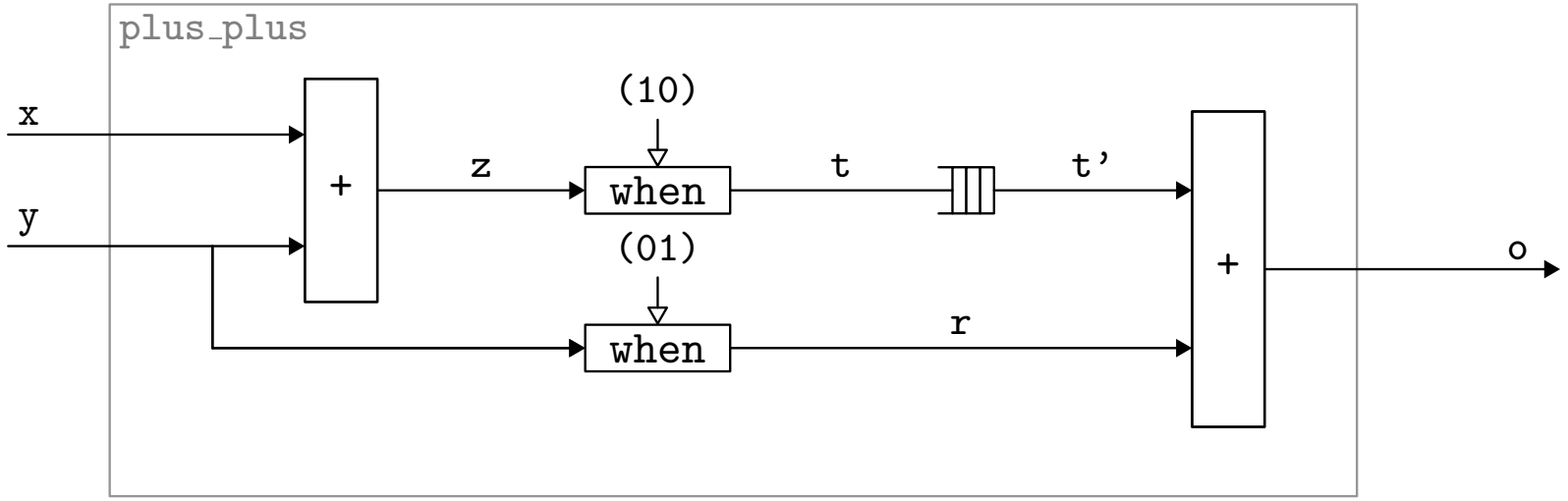
Example : $0(00111) = 0\,00111\,00111\,...$

Checking relations on clocks

- adaptability test: $(11010) <: 0(00111)$

Clock Typing

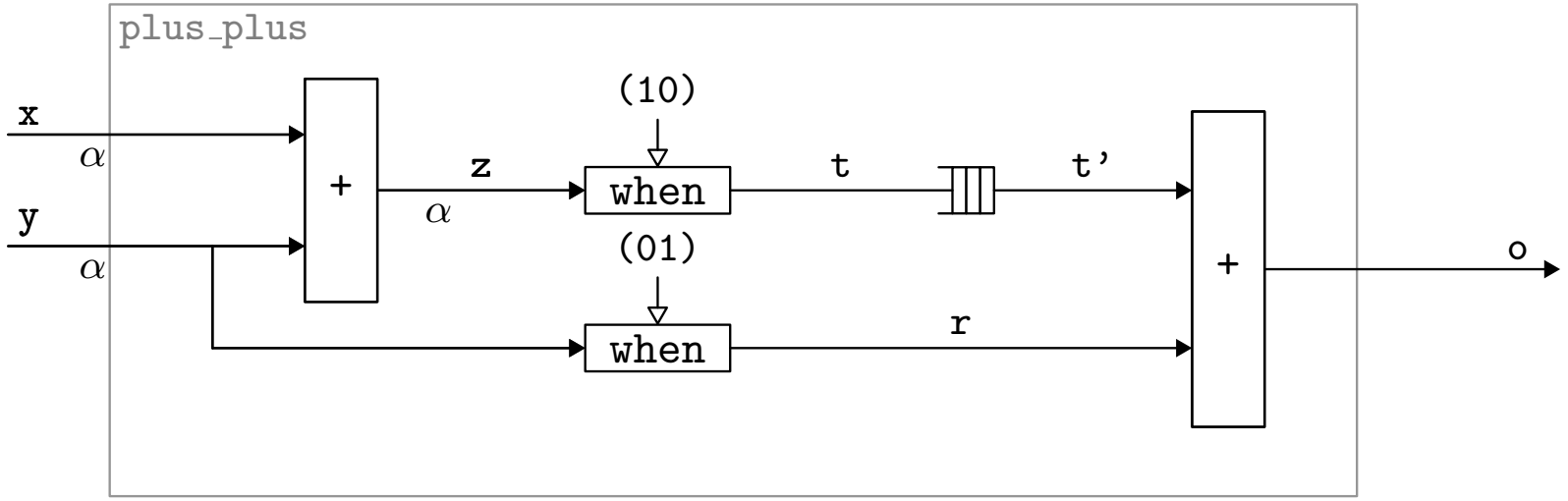
Typing




```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when (10)
7    and t'= buffer(t)
8    and r = y when (01)
9    and o = t' + r
```

```
val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on (01)
Buffer line 7, characters 11-21: size = 1
```

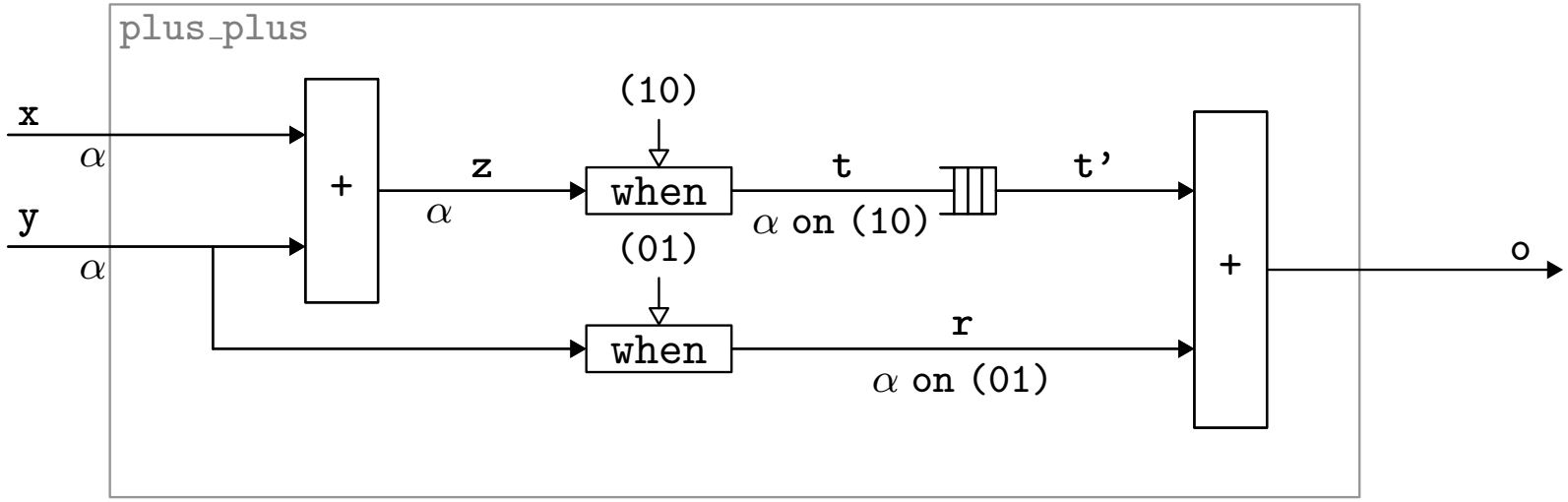
Typing




```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when (10)
7    and t'= buffer(t)
8    and r = y when (01)
9    and o = t' + r
```

```
val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on (01)
Buffer line 7, characters 11-21: size = 1
```

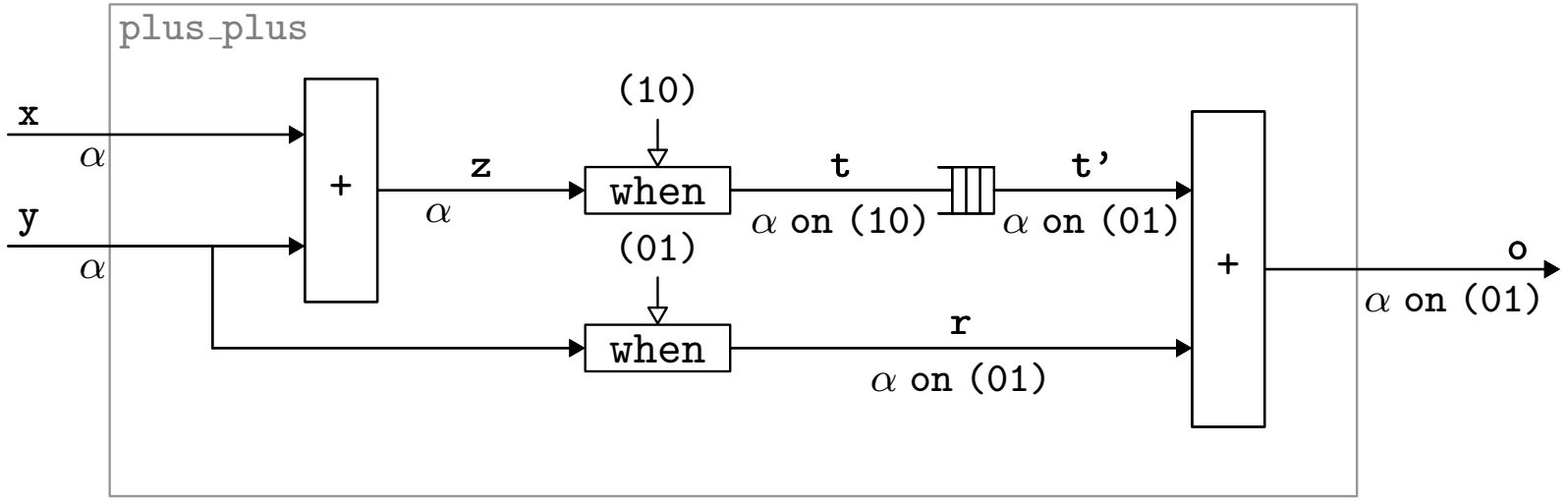
Typing




```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when (10)
7    and t'= buffer(t)
8    and r = y when (01)
9    and o = t' + r
```

```
val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on (01)
Buffer line 7, characters 11-21: size = 1
```

Typing

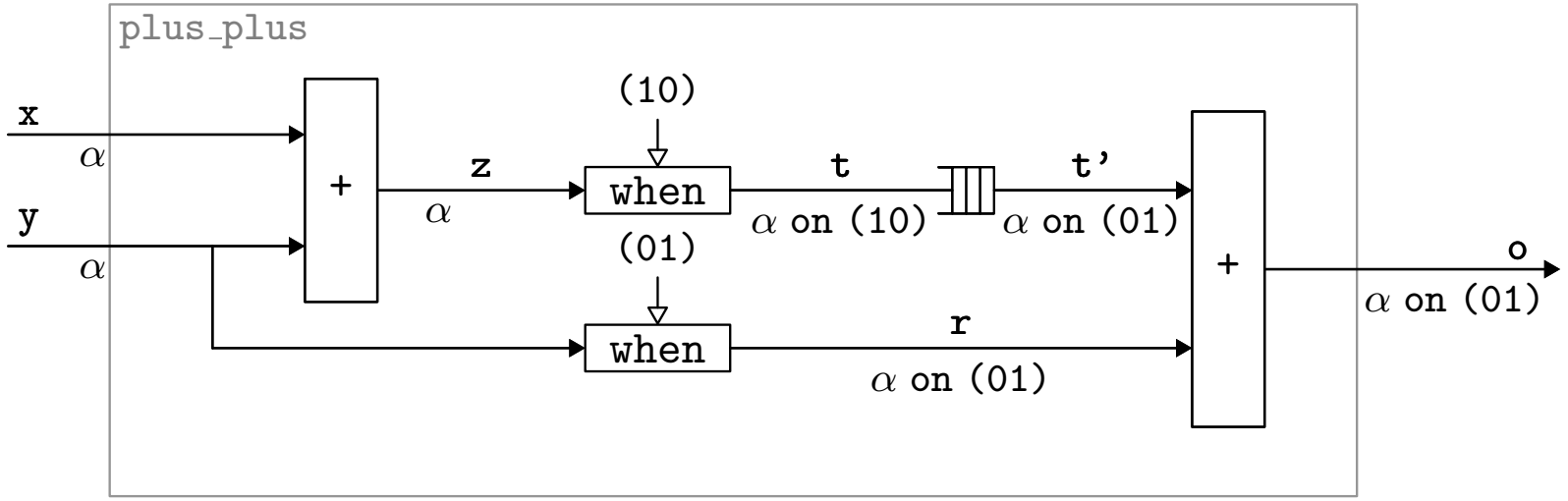


for all α such that $\{\alpha \text{ on } (10) <: \alpha \text{ on } (01)\}$, $(\alpha \times \alpha) \to \alpha \text{ on } (01)$

Subtyping constraint solving

- simple case: $\alpha \text{ on } w_1 <: \alpha \text{ on } w_2 \Leftrightarrow w_1 <: w_2$

Typing

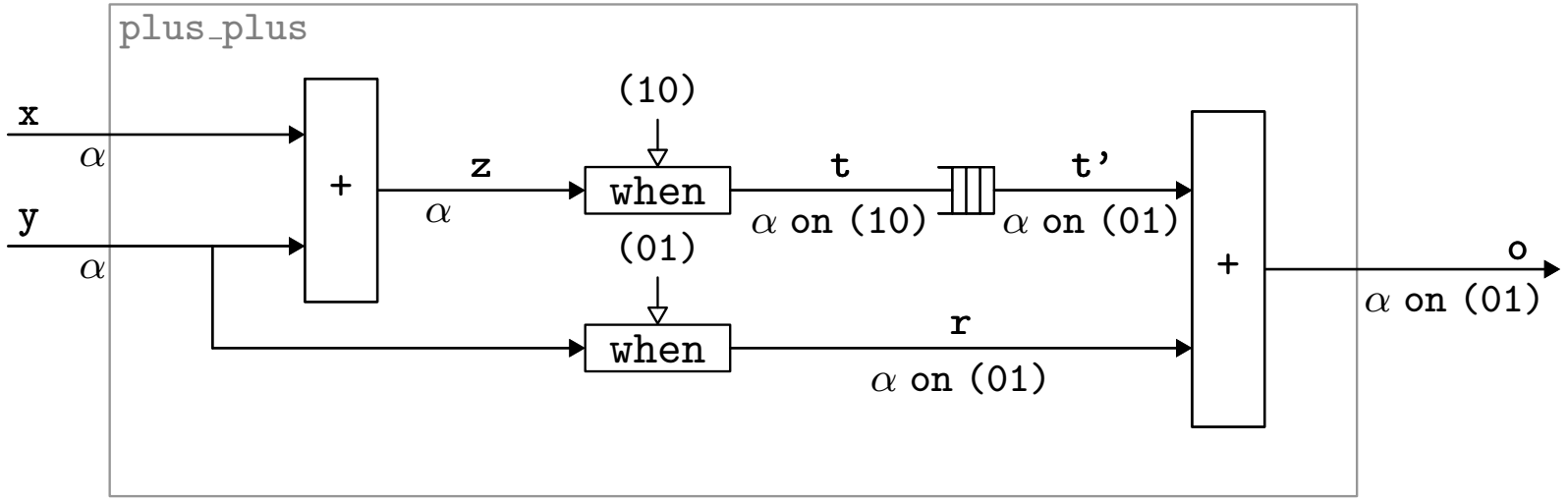


for all α such that $\{$ (10) <: (01)$\}$, $(\alpha \times \alpha) \to \alpha$ on (01)

Subtyping constraint solving

- simple case: α on w_1 <: α on w_2 $\Leftrightarrow$ $w_1 <: w_2$

Typing

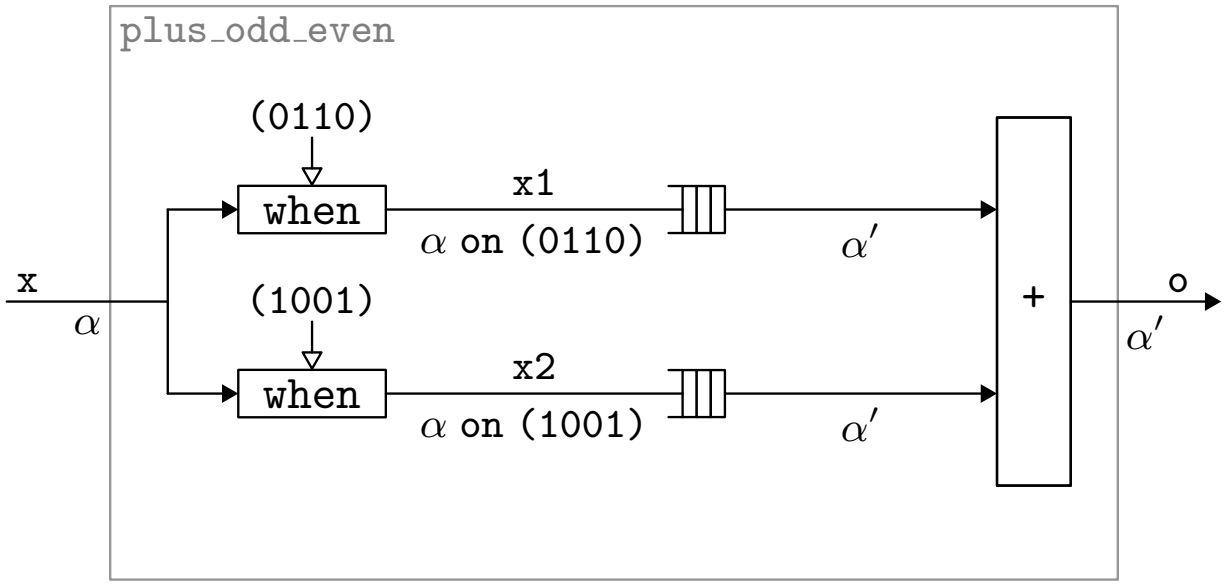


for all α , $(\alpha \times \alpha) \rightarrow \alpha$ on (01)

Subtyping constraint solving

- simple case: α on w_1 <: α on $w_2 \Leftrightarrow w_1 <: w_2$

Typing: More Difficult Case



for all α, α' such that $\left\{ \begin{array}{lll} \alpha \text{ on } (0110) & <: & \alpha' \\ \alpha \text{ on } (1001) & <: & \alpha' \end{array} \right\}, \alpha \to \alpha'$

Goal of the constraints resolution algorithm :

find instantiations for α and α' such that the constraints are always verified.

Results

Correct (and complete ?) algorithm

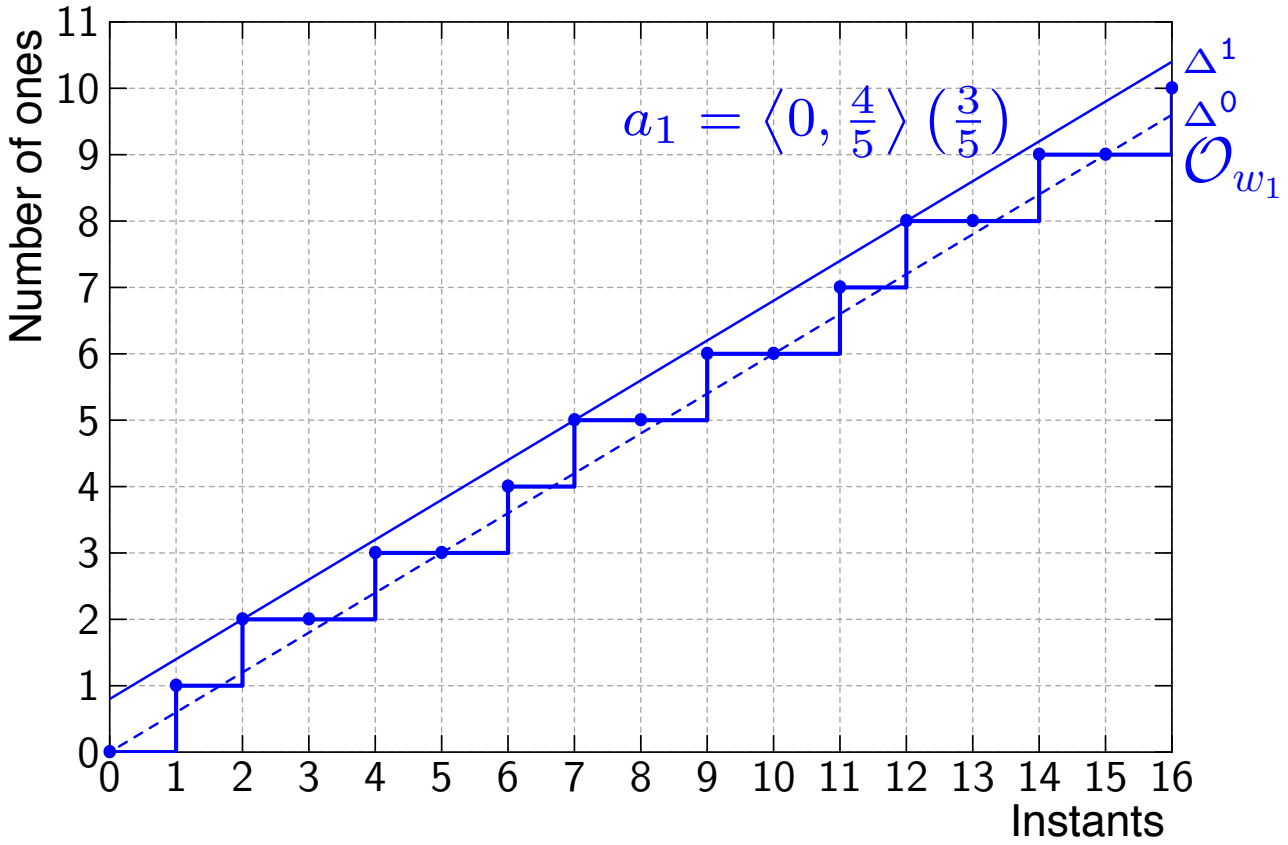
- ex: GSM encoding/decoding protocol

Can be costly

- ex: Picture in Picture video application

Abstract Clocks

Abstract Clocks: $abs\,(w) = \langle b^0, b^1 \rangle\,(r)$

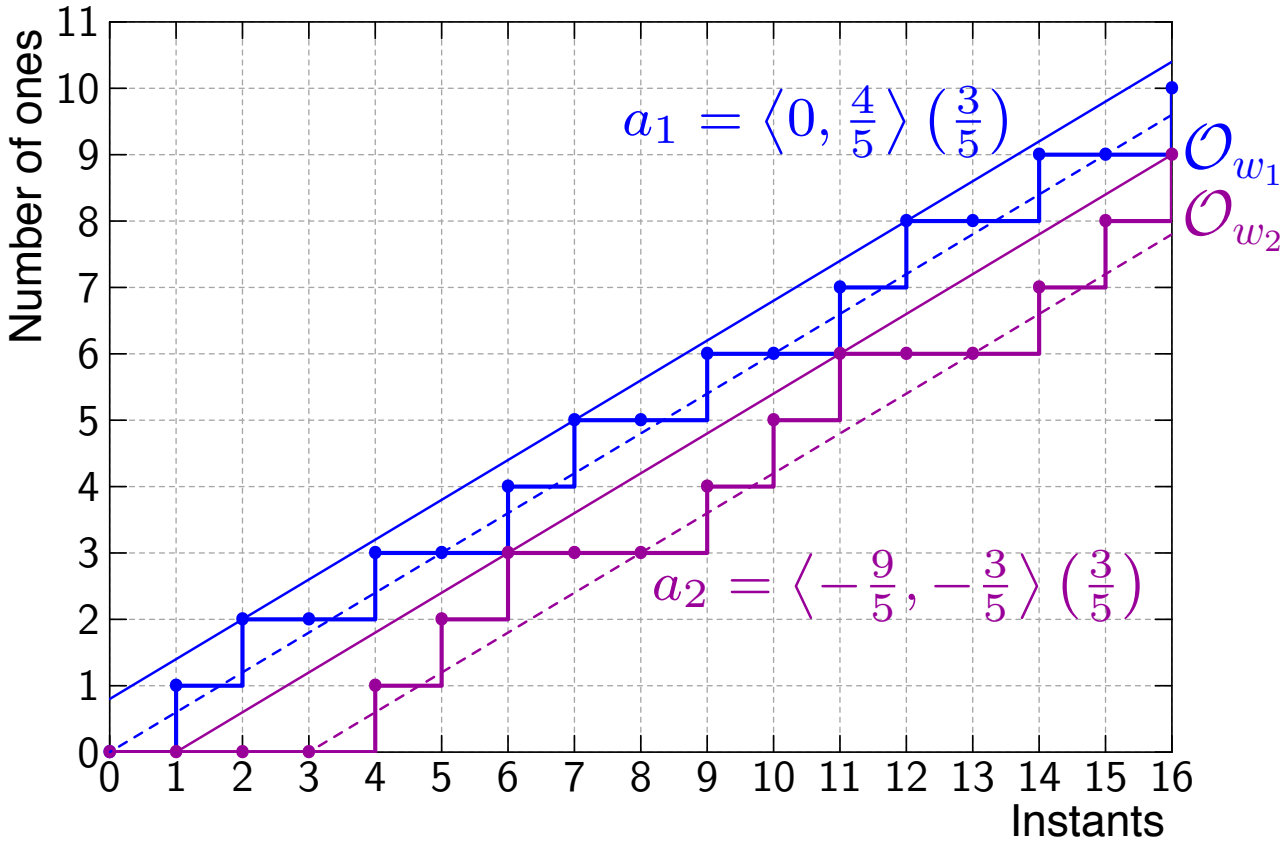


$$\Delta^1 : r \times i + b^1$$

$$\Delta^0 : r \times i + b^0$$

$$concr\,(\langle b^0, b^1 \rangle\,(r)) = \left\{ w \;\middle|\; \begin{array}{lcl} w[i] = 1 & \Rightarrow & \mathcal{O}_w(i) \leq \Delta^1(i) \\ w[i] = 0 & \Rightarrow & \mathcal{O}_w(i) \geq \Delta^0(i) \end{array} \right\}$$

Adaptability Relation on Abstract Clocks



adaptability $\langle {b^0}_1, {b^1}_1 \rangle (r_1) <:^{\sim} \langle {b^0}_2, {b^1}_2 \rangle (r_2) \overset{def}{\Leftrightarrow} r_1 = r_2 \ \wedge \ {b^1}_2 - {b^0}_1 < 1$

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

Conclusion and Future Work

Conclusion

- n-synchronous model:
  more flexible composition of nodes without loss of guaranties
- algorithm without abstraction:
  precise and working on constraints difficult to satisfy
- algorithm with abstraction:
  efficient and working on non periodic clocks

Future work:

- application to latency insensitive design
- clock gating / buffering

http://www.florenceplateau.fr