Modèle n-Synchrone

Polytech Paris-Sud

Cycle ingénieur de la filière étudiant

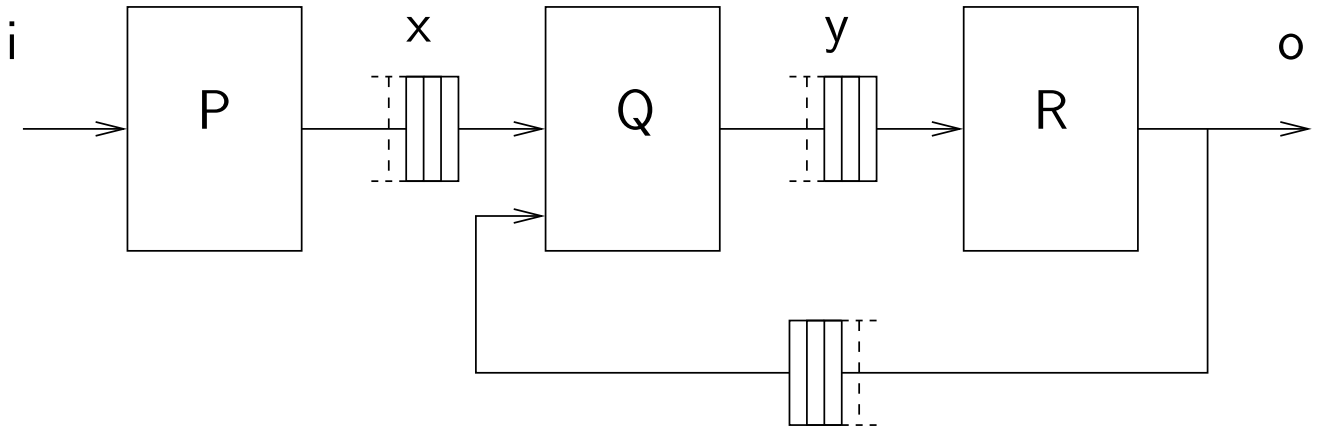
Louis Mandel et Florence Plateau

Université Paris-Sud 11

Louis.Mandel@lri.fr

année 2009/2010

Réseaux de Kahn [Gilles Kahn, 1974]

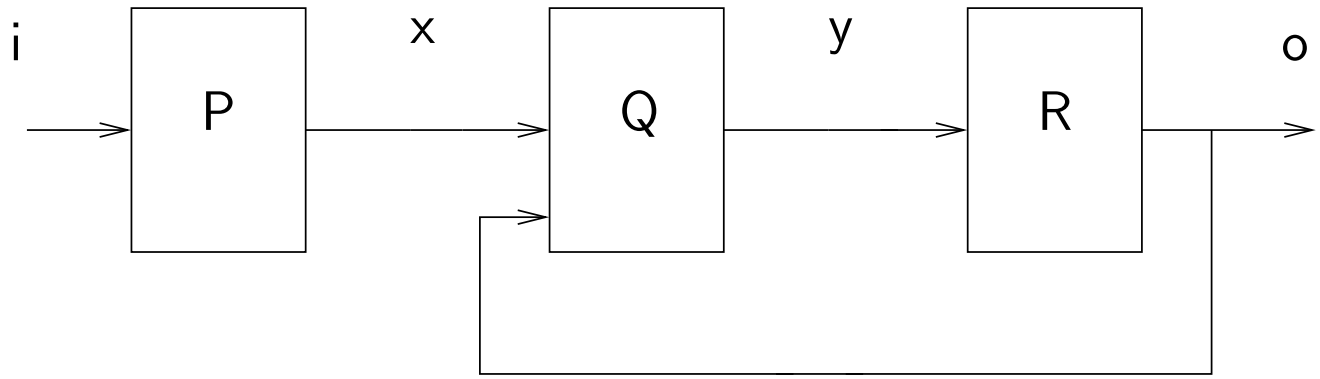


Modèle de concurrence

- processus s'exécutant en parallèle
- communication par FIFO :
  - écriture non-bloquante
  - lecture bloquante sur buffer vides

Si l'on dispose de buffers de taille suffisante, et si les programmes sont déterministes, alors le réseau est déterministe.

Le modèle synchrone flot de données

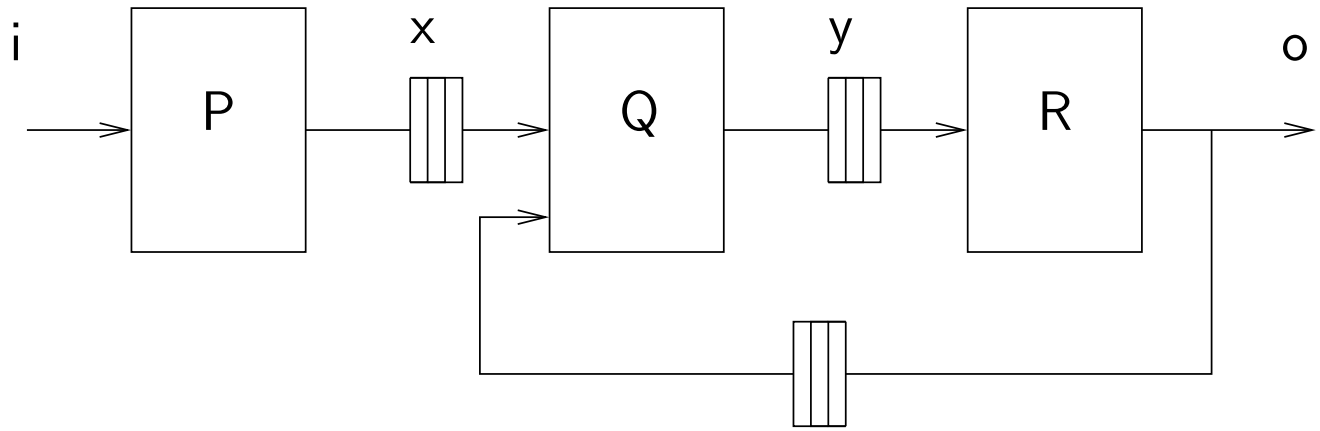


Programmation de réseaux de Kahn sans buffers :

- langages de programmation Lustre, Signal, Lucid Synchrone
- consommation instantanée des données produites
- garanties fortes : mémoire bornée, absence de blocage

Mais : communication sans buffers parfois trop restrictive
(e.g. applications multimédia)

Le modèle n-synchrone



Programmation de réseaux de Kahn avec buffers bornés :

- accepter de stocker les données dans des buffers
- rejeter les réseaux qui nécessitent une mémoire infinie
- calculer les rythmes d'activation des différents nœuds de calcul
- calculer les tailles des buffers nécessaires

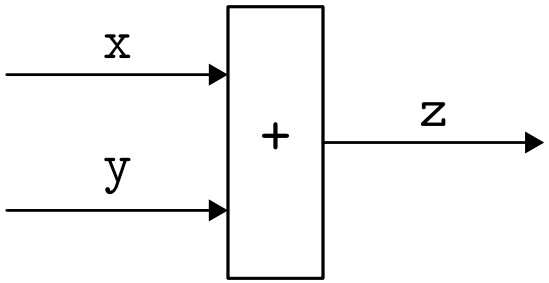
Plus de flexibilité, autant de garanties

Plan

1. Lucy-n : une extension n-synchrone de Lustre
2. Horloges périodiques
3. Horloges abstraites
4. Conclusion et perspectives

Lucy-n : une extension n-synchrone de Lustre

Un noyau synchrone flot de données



flot	valeurs								horloge
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...

Un noyau synchrone flot de données

flot	valeurs								horloge
`x`	5	7	3	6	2	8	1	...	`111111...`
`y`	3	2	1	5	4	1	7	...	`111111...`
`pre y`	?	3	2	1	5	4	1	...	`111111...`
`x -> pre y`	5	3	2	1	5	4	1	...	`111111...`

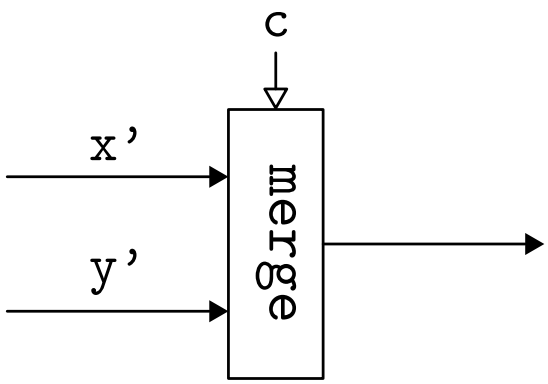
```
nat = 0 -> (pre nat + 1)
```

flot	valeurs	horloge
x	5 7 3 6 2 8 1 ...	1111111...
c	1 0 1 0 1 0 1 ...	
x when c	5 3 2 1 ...	1010101...
c'	1 0 1 1 ...	
(x when c) when c'	5 2 1 ...	1000101...

$$\mathit{horloge}(\texttt{(x when c) when c'}) = \mathit{horloge}(\texttt{x when c})\ \mathit{on}\ \texttt{c'}$$

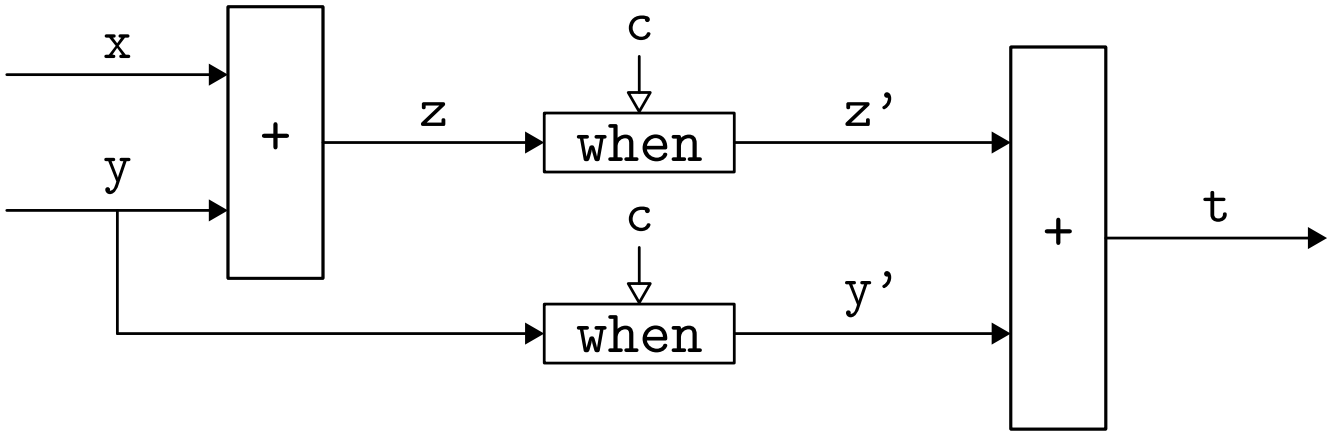
Opérateur *on* :

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



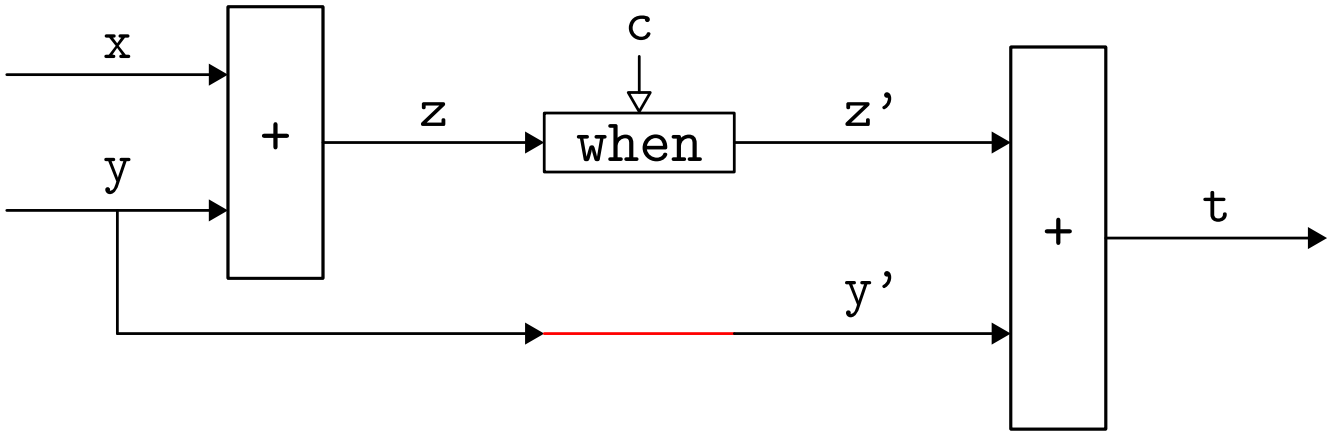
flot	valeurs								horloge
x'	5		3		2		1	...	1010101...
y'		2		5		1		...	0101010...
c	1	0	1	0	1	0	1	...	
merge c x' y'	5	2	3	5	2	1	1	...	1111111...

Un noyau synchrone flot de données



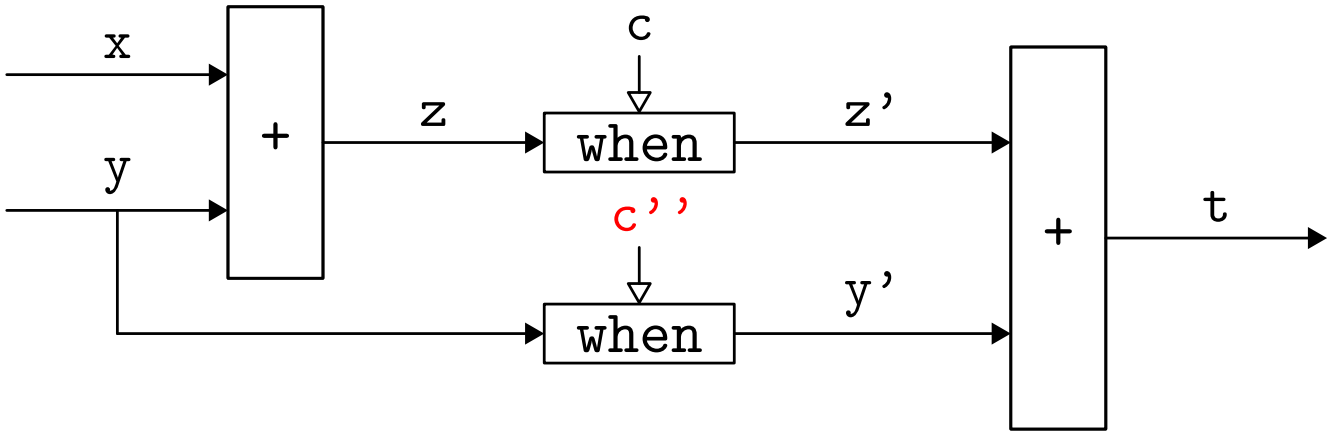
flot	valeurs								horloge
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
z' = z when c	8		4		6		8	...	101010...
y' = y when c	3		1		4		7	...	101010...
t = z' + y'	11		5		10		15	...	101010...

Un noyau synchrone flot de données



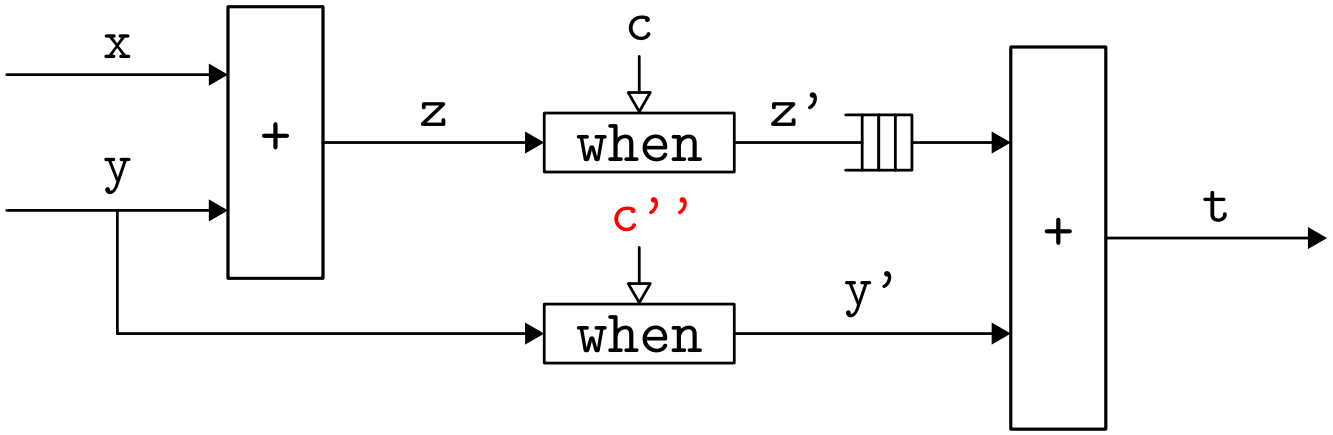
flot	valeurs								horloge
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
z' = z when c	8		4		6		8	...	101010...
y' = y	3	2	1	5	4	1	7	...	111111...
t = z' + y'	rejeté								

Un noyau synchrone flot de données



flot	valeurs								horloge
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
z' = z when c	8		4		6		8	...	101010...
y' = y when c''		2		5		1		...	010101...
t = z' + y'	rejeté								

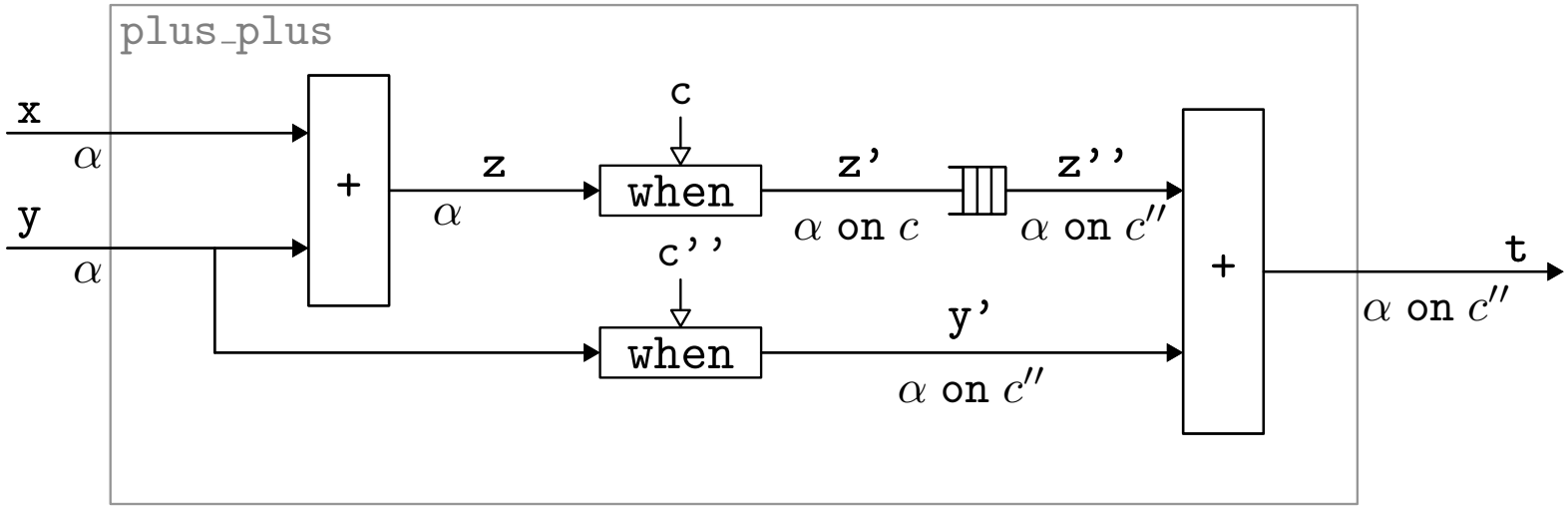
Extension n-synchrone : opérateur de bufferisation



flot	valeurs	horloge
z' = z when c	8 4 6 8 ...	101010...
z'' = buffer(z')	8 4 6 ...	010101...
y' = y when c''	2 5 1 ...	010101...
t = z'' + y'	10 9 7 ...	010101...

- Relation d'adaptabilité $\Rightarrow$ communication par buffer borné
- Exemple : 101010... $<:$ 010101...

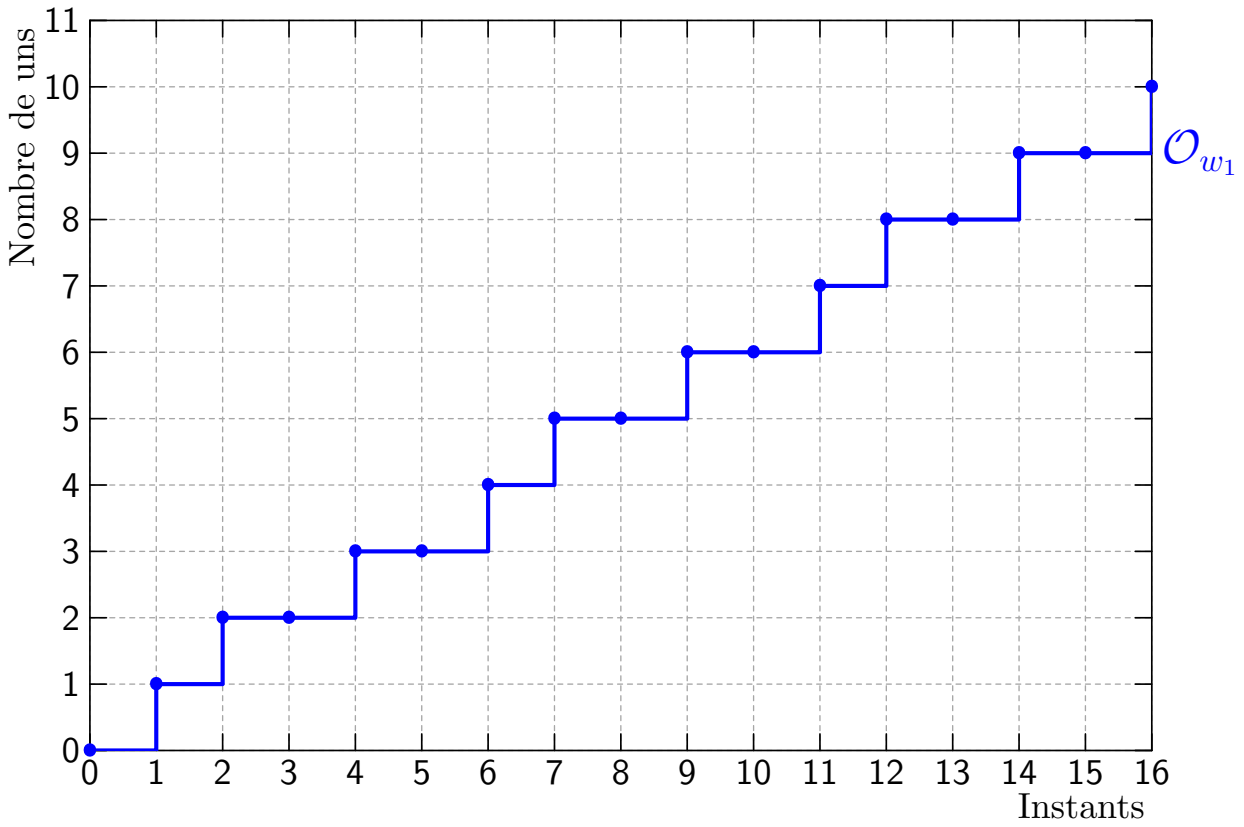
Typage




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

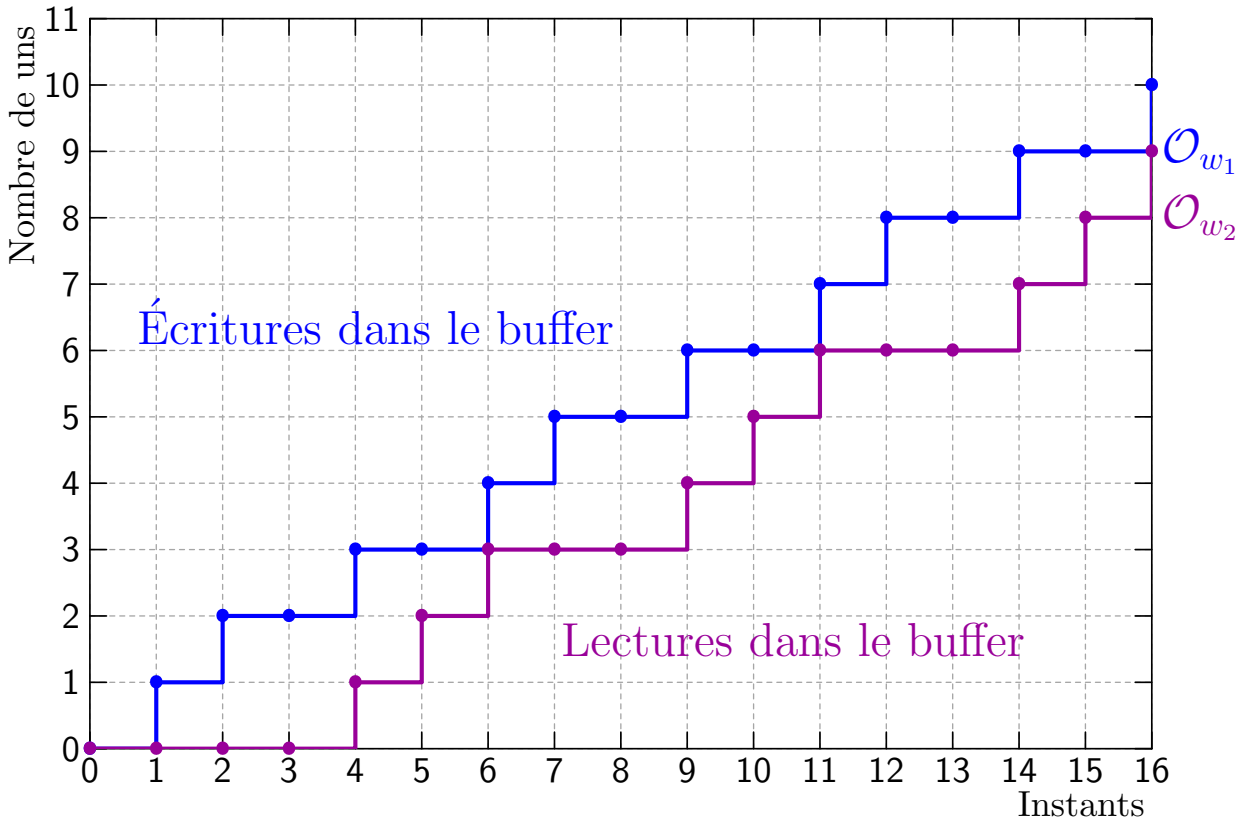
```
val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on c''
Buffer line 7, characters 11-21: size = 1
```

Relation d'adaptabilité



$\mathcal{O}_{w_1}$: fonction de cumul du mot w_1

Relation d'adaptabilité



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

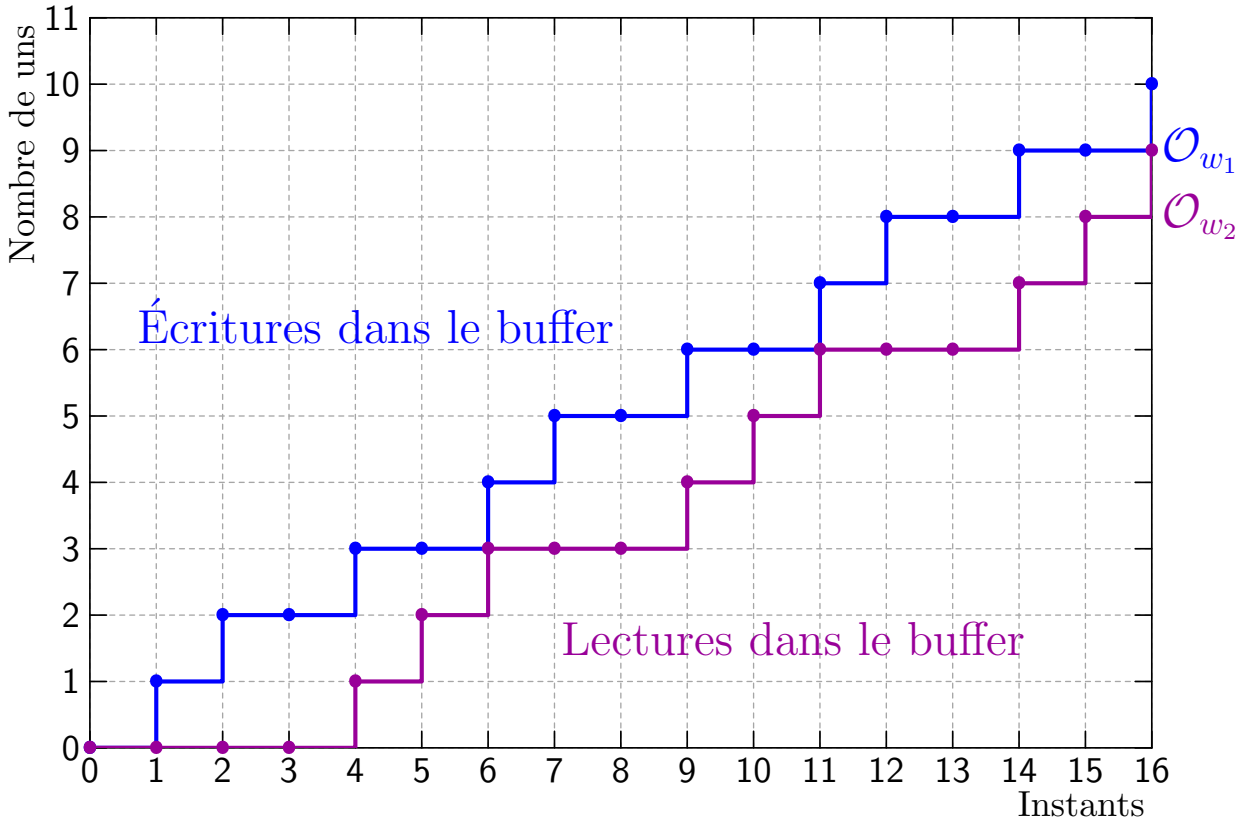
adaptabilité $w_1 <: w_2 \overset{def}{\Leftrightarrow} \exists n \in \mathbb{N}, \forall i,\ 0 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq n$

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

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

Horloges périodiques

Horloges ultimement périodiques

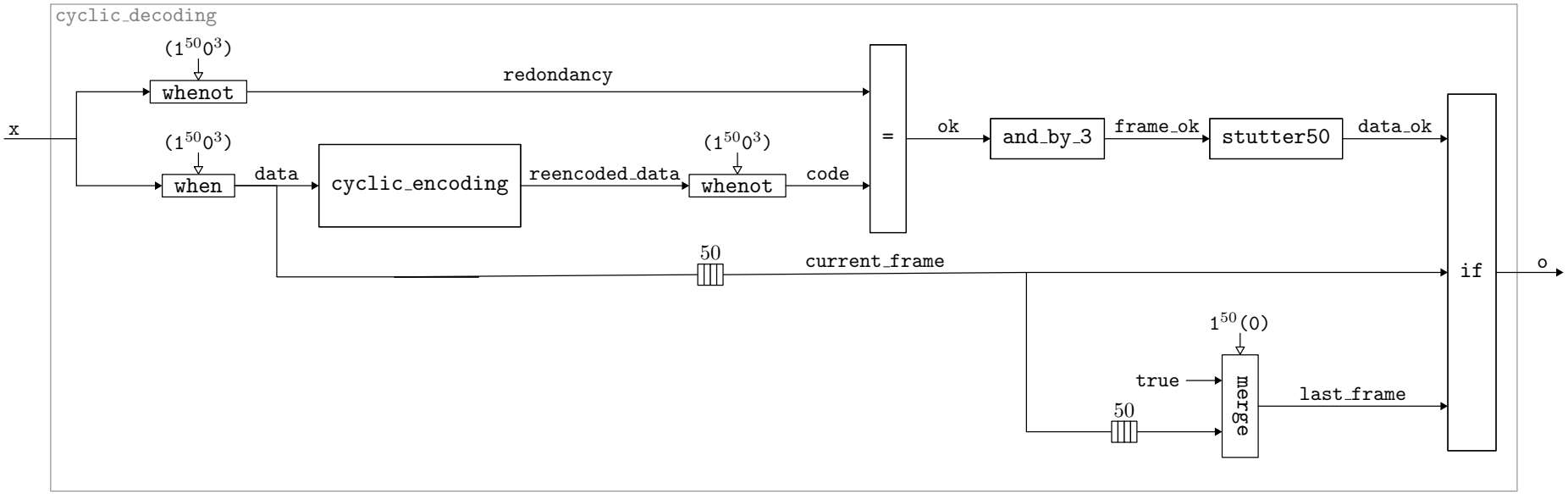


Exemple : $0(00111) = 0\,00111\,00111\ldots$

Vérification des relations sur les horloges

- Test de synchronisabilité : $(11010) \bowtie 0(00111)$
- Test de précédence : $(11010) \preceq 0(00111)$

Extrait d'un décodeur de canal GSM



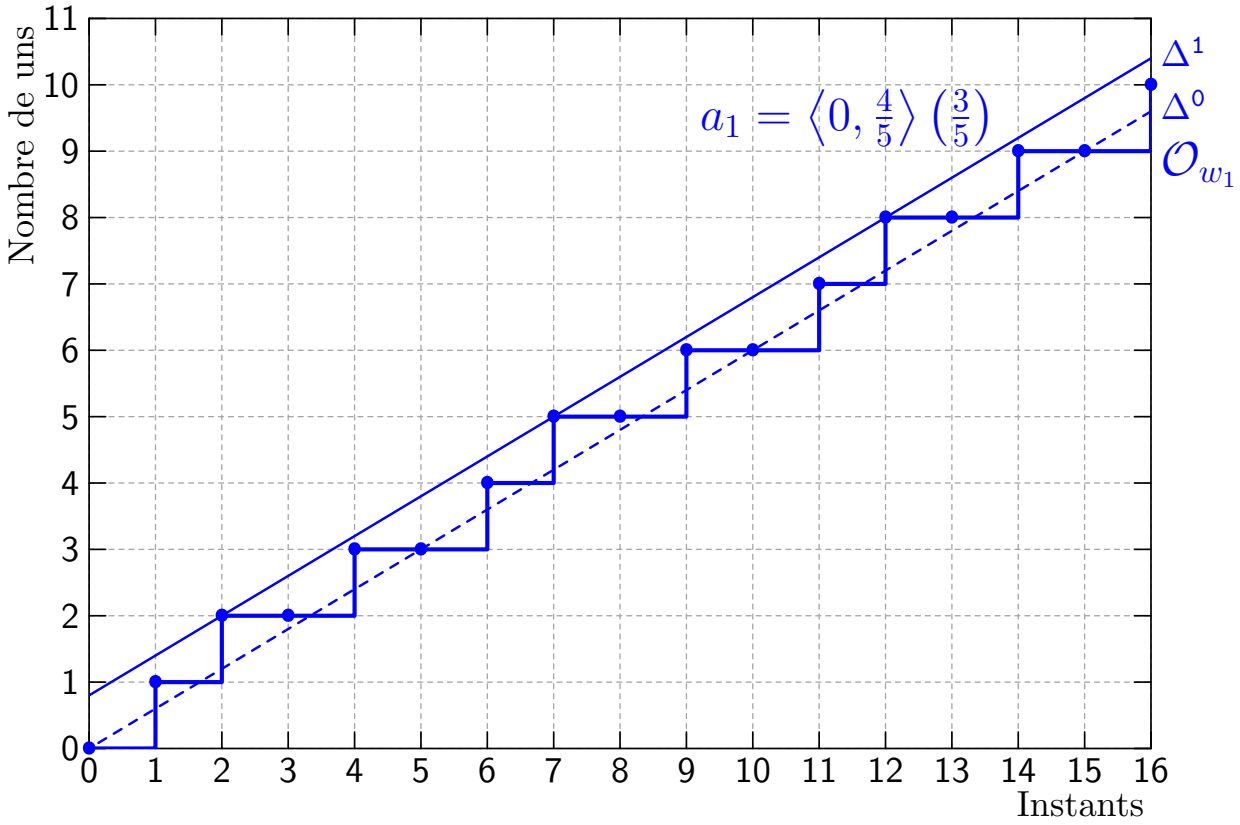
Extrait d'un décodeur de canal GSM

```
56  let node cyclic_decoding x = o where
57    rec (data, redondancy) =
58      (x when (1^50 0^3) , x whenot (1^50 0^3))
59    and reencoded_data = cyclic_encoding data
60    and code = reencoded_data whenot (1^50 0^3)
61    and ok = (code = redondancy)
62    and frame_ok = and_by_3 ok
63    and data_ok = stutter50 frame_ok
64    and current_frame = buffer(data)
65    and last_frame = merge 1^50(0) true (buffer(current_frame))
66    and o = if data_ok then current_frame else last_frame
```

```
cyclic_decoding ::
forall 'a. 'a -> 'a on 0^52 (1^50 0^3 )
Buffer line 64, characters 22-34: size = 50
Buffer line 65, characters 39-60: size = 50
```

Horloges abstraites

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

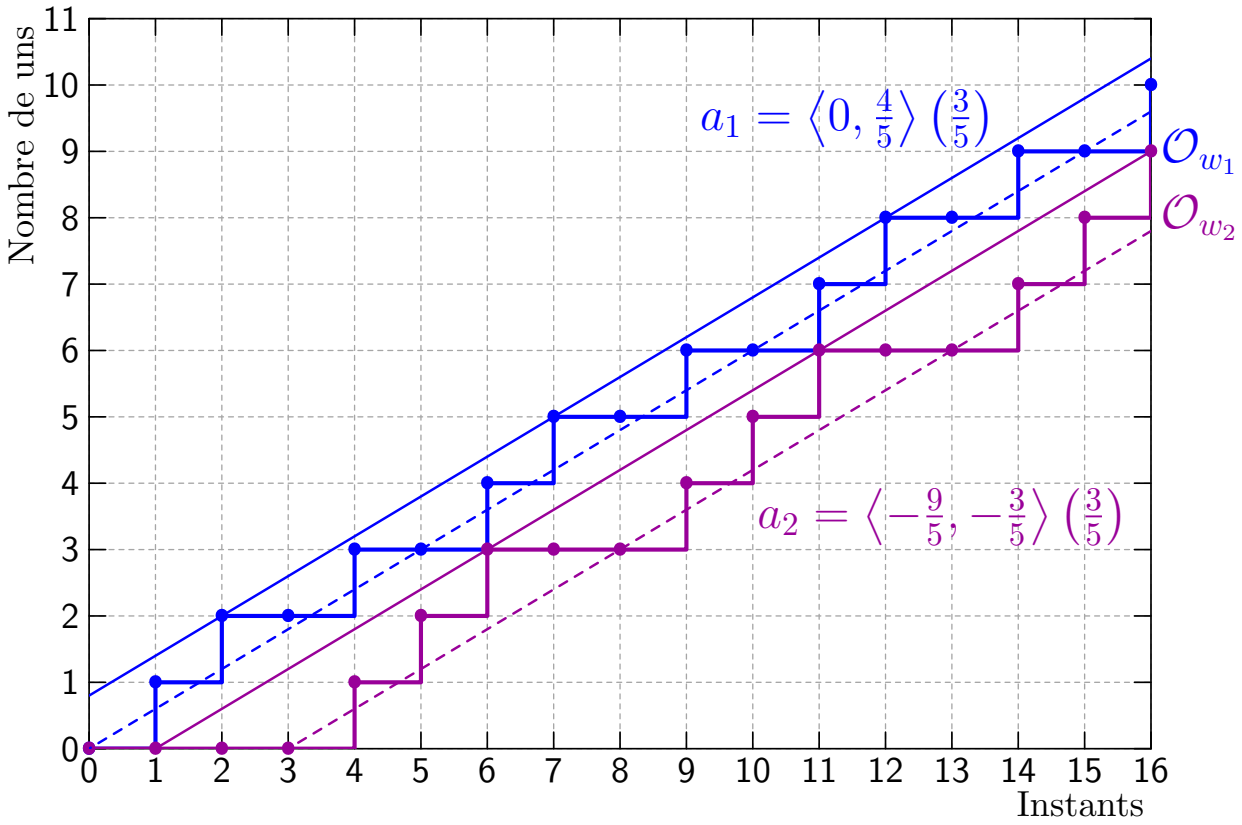


$$\Delta^1 : r \times i + b^1$$

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

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

Adaptabilité d'horloges abstraites



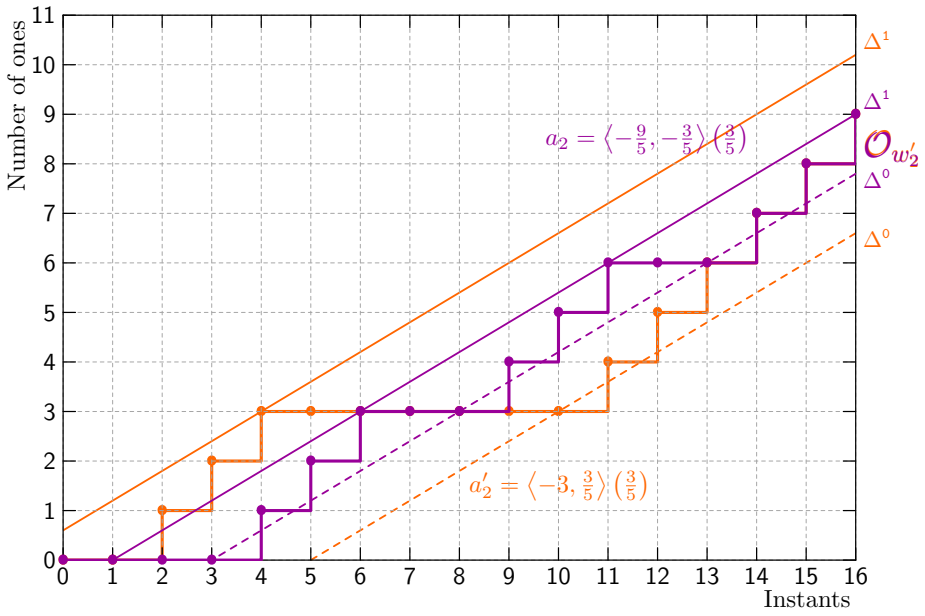
synchronisabilité $\quad \langle {b^0}_1, {b^1}_1 \rangle (r_1) \bowtie^{\sim} \langle {b^0}_2, {b^1}_2 \rangle (r_2) \overset{def}{\Leftrightarrow} r_1 = r_2$

précédence $\quad \langle {b^0}_1, {b^1}_1 \rangle (r) \preceq^{\sim} \langle {b^0}_2, {b^1}_2 \rangle (r) \overset{def}{\Leftrightarrow} {b^1}_2 - {b^0}_1 < 1$

Propriété : $\quad a_1 \bowtie^{\sim} a_2 \Rightarrow \forall w_1 \in concr(a_1), \forall w_2 \in concr(a_2), w_1 \bowtie w_2$

$\quad a_1 \preceq^{\sim} a_2 \Rightarrow \forall w_1 \in concr(a_1), \forall w_2 \in concr(a_2), w_1 \preceq w_2$

Abstraction d'horloges



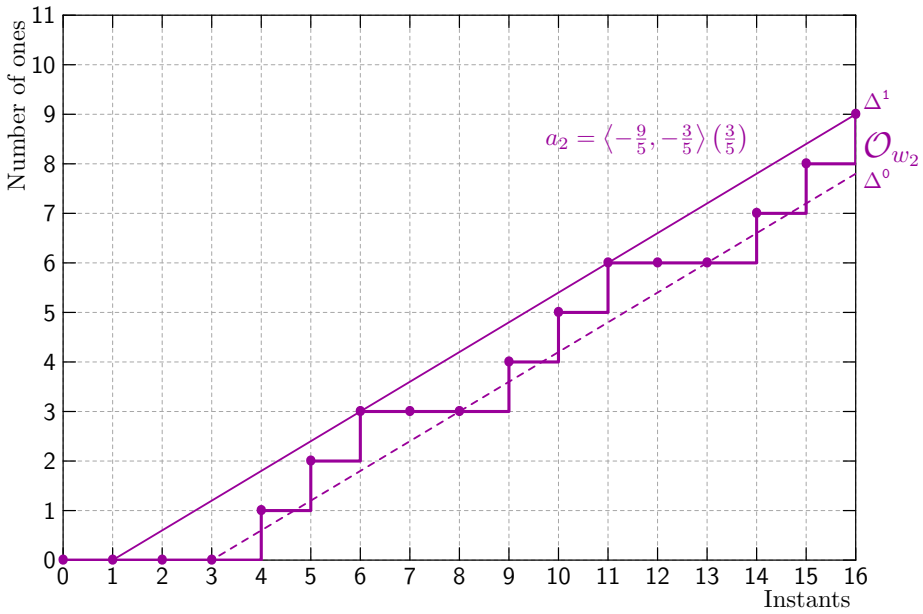
Abstraction des mots périodiques :

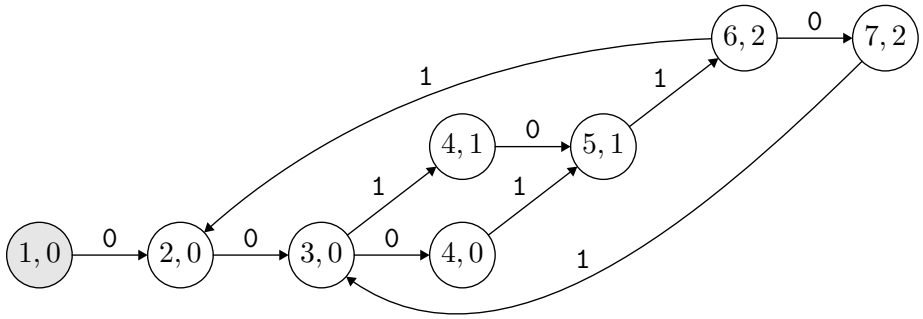
$$abs\left(\texttt{0(00111)}\right) = \left\langle -\frac{9}{5}, -\frac{3}{5} \right\rangle \left(\frac{3}{5}\right)$$

Abstraction de mots périodiques soumis à une certaine gigue :

$$abs\left(jitter(\texttt{0(00111)}, 2)\right) = \left\langle -\frac{15}{5}, \frac{3}{5} \right\rangle \left(\frac{3}{5}\right)$$

Horloges abstraites

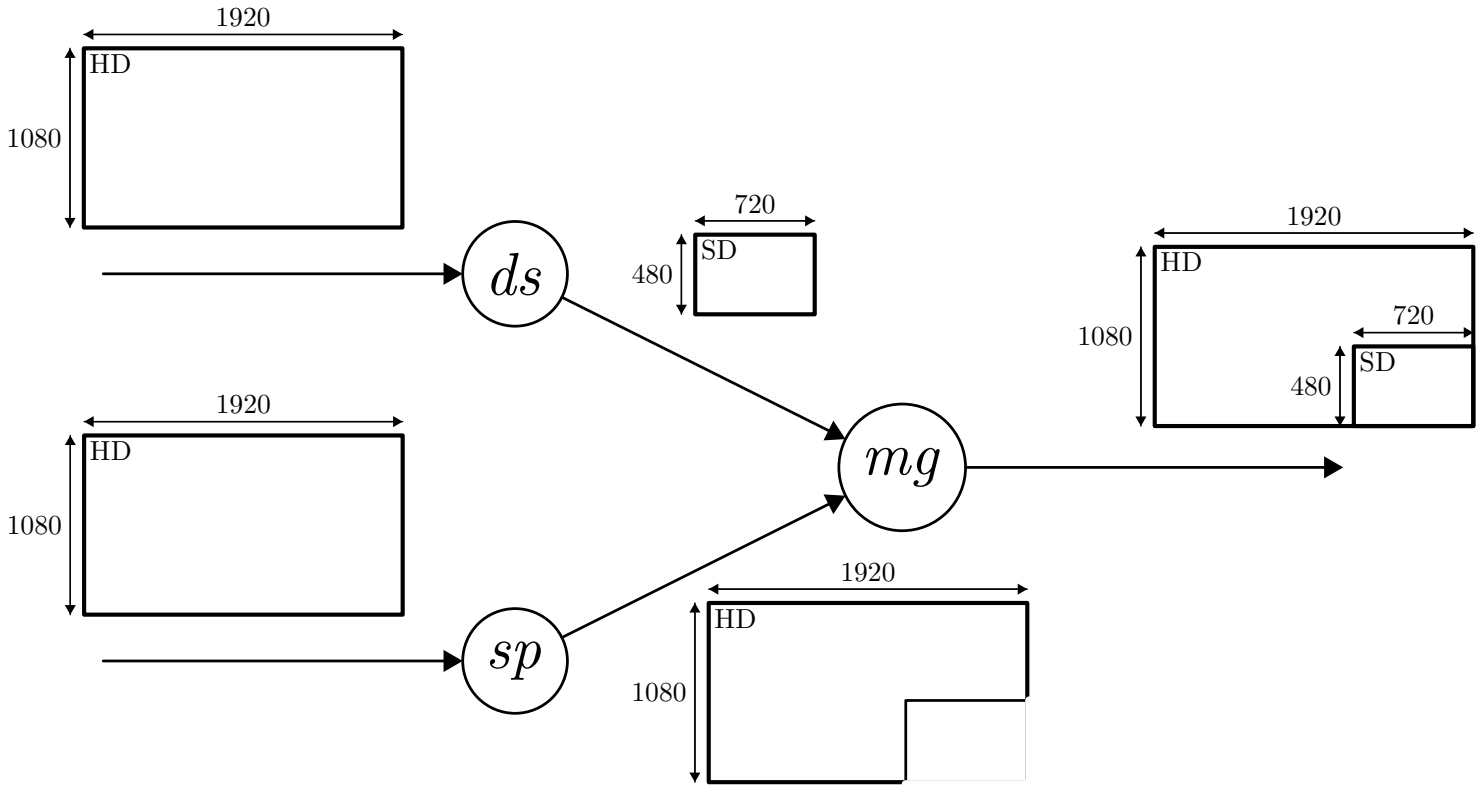




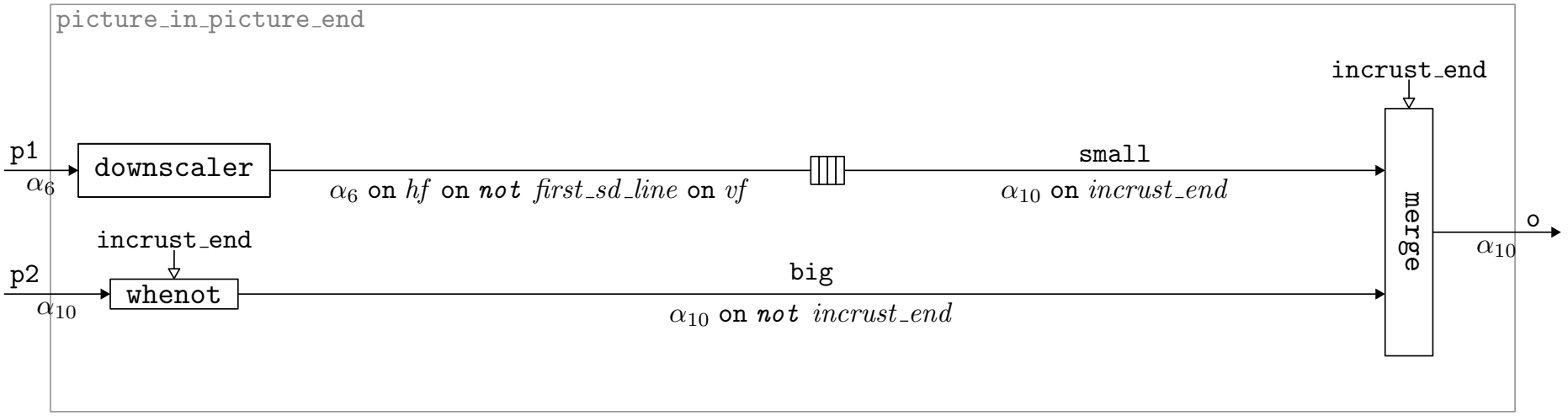
Automates accepteurs et générateurs :

- États (i, j) : coordonnées dans le graphique
- Transitions : $\begin{cases} 1 \text{ si } j + 1 \leq \Delta^1(i) \\ 0 \text{ si } j \geq \Delta^0(i) \end{cases}$
- Nombre fini de classes d'états équivalents

Application vidéo



Application vidéo




```
51  (* picture in picture *)
52  let clock incrust_end =
53    (0^(1920 * (1080 - 480)) {0^1200 1^720}^480)
54
55  let node picture_in_picture_end (p1, p2) = o where
56    rec small = buffer(downscaler p1)
57    and big = (p2 whenot incrust_end)
58    and o = merge incrust_end small big
```

Application vidéo

Système à résoudre :

$$\{\alpha_6 \text{ on } hf \text{ on not } \mathit{first\_sd\_line} \text{ on } vf \ \texttt{<:}\ \alpha_{10} \text{ on } \mathit{incrust\_end}\}$$

Solution : $\alpha_6 = \alpha \text{ on } (1), \quad \alpha_{10} = \alpha \text{ on } 0^{4315}(1)$

$$\mathit{picture\_in\_picture\_end} :: \forall\alpha.\,(\alpha \times \alpha \text{ on } 0^{4315}(1)) \to \alpha \text{ on } 0^{4315}(1)$$

Bilan :

	délais	taille de buffer
résultat minimal	$1\,920$ ($\approx$ 1 ligne HD)	$191\,970$ (≈ 266.6 lignes SD)
résultat abstrait	$4\,315$ ($\approx$ 2 lignes HD)	$193\,079$ (≈ 268.1 lignes SD)

- Prouvé correct
- Incomplet par nature
- Bonnes performances