From synchrony to asynchrony

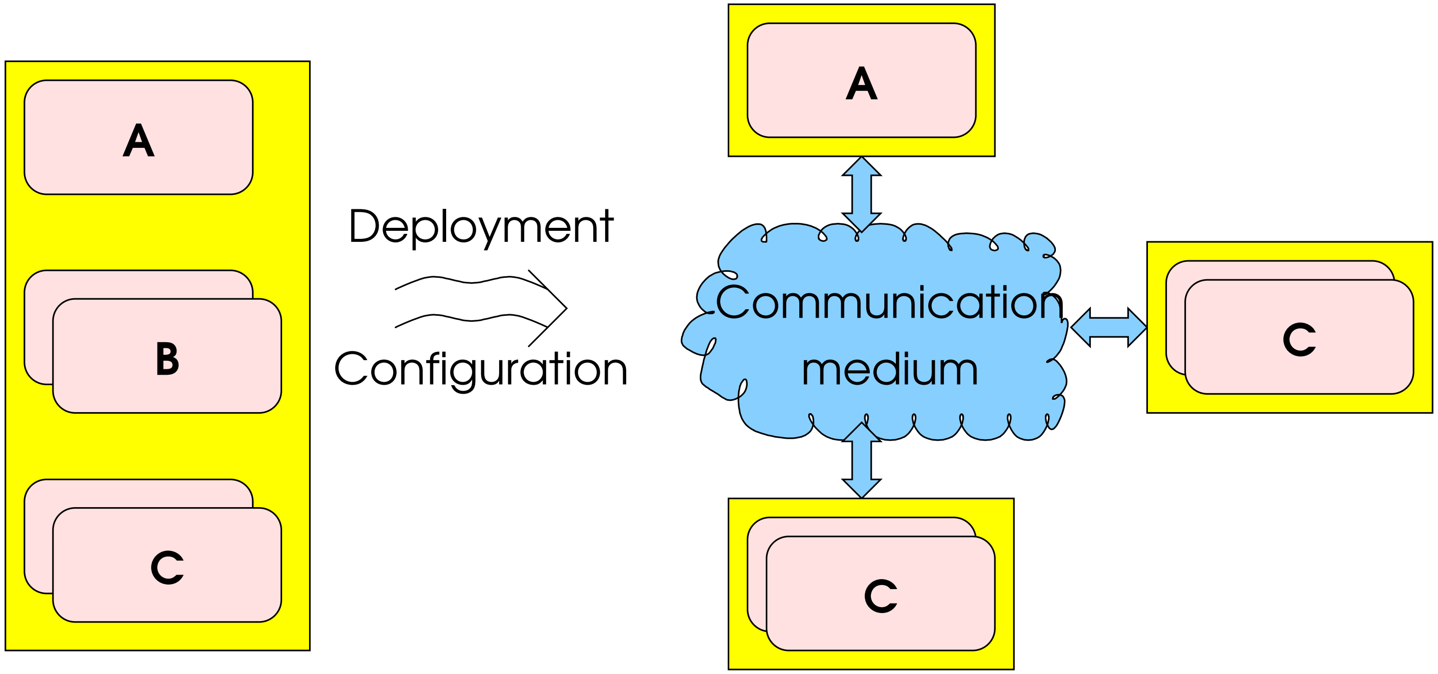
Albert Benveniste, Benoît Caillaud, Paul Le Guernic

- Synchronous design $\longrightarrow$ asynchronous implementation
- STS : Synchronous transition systems and asynchronous behaviours
- Modeling asynchronous behaviours, asynchronous behaviours
- Two questions:

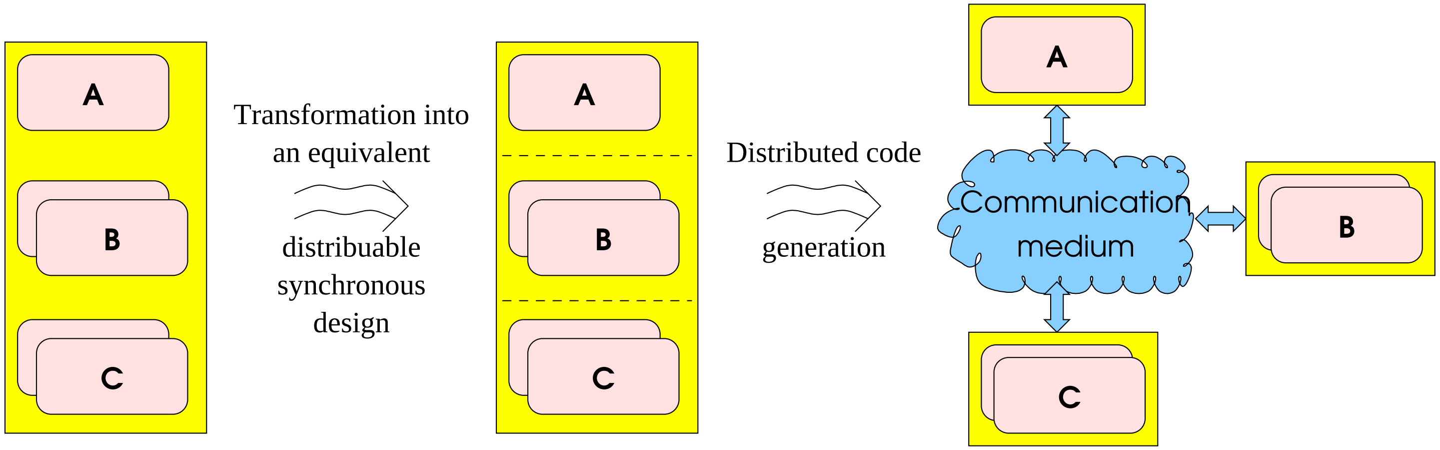
  Desynchronizing a single STS : Endochrony

  Desynchronizing two STS : Isochrony
- Desynchronizing networks of STS

Synchronous design → asynchronous implementation



Towards a design methodology...



- Clock analysis : computing a hierarchy of control clocks.
- Causality analysis : computing dependencies on variables and computations.

Synchrony

- Programs progress via reactions: $P = R^\omega$
- Within a reaction, decisions can be taken based on testing for absence:

```
y := u default v
y = current x
present S else ‘stat’
```

$$y = \begin{cases} \mathtt{u} & (\text{if } \mathtt{u} \text{ is present}) \\ \mathtt{v} & (\text{if } \mathtt{u} \text{ is absent}) \end{cases}$$

- Communication: instantaneous broadcast

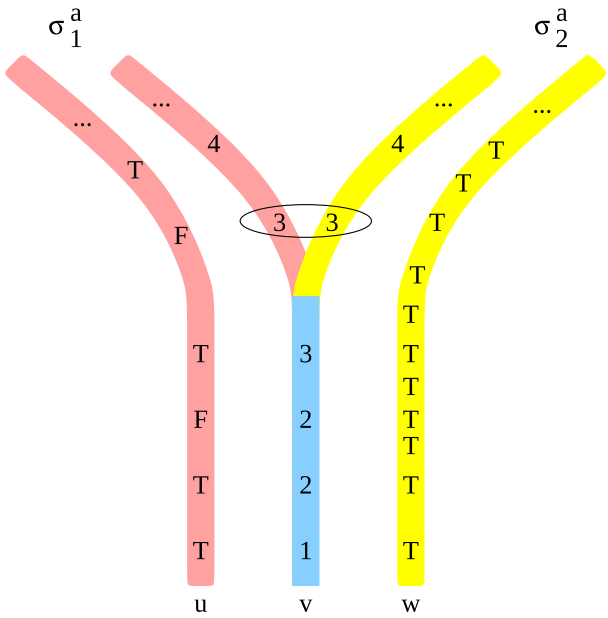
$$P_1 \parallel P_2 \quad =_{\mathrm{def}} \quad (R_1 \wedge R_2)^\omega$$

Asynchrony

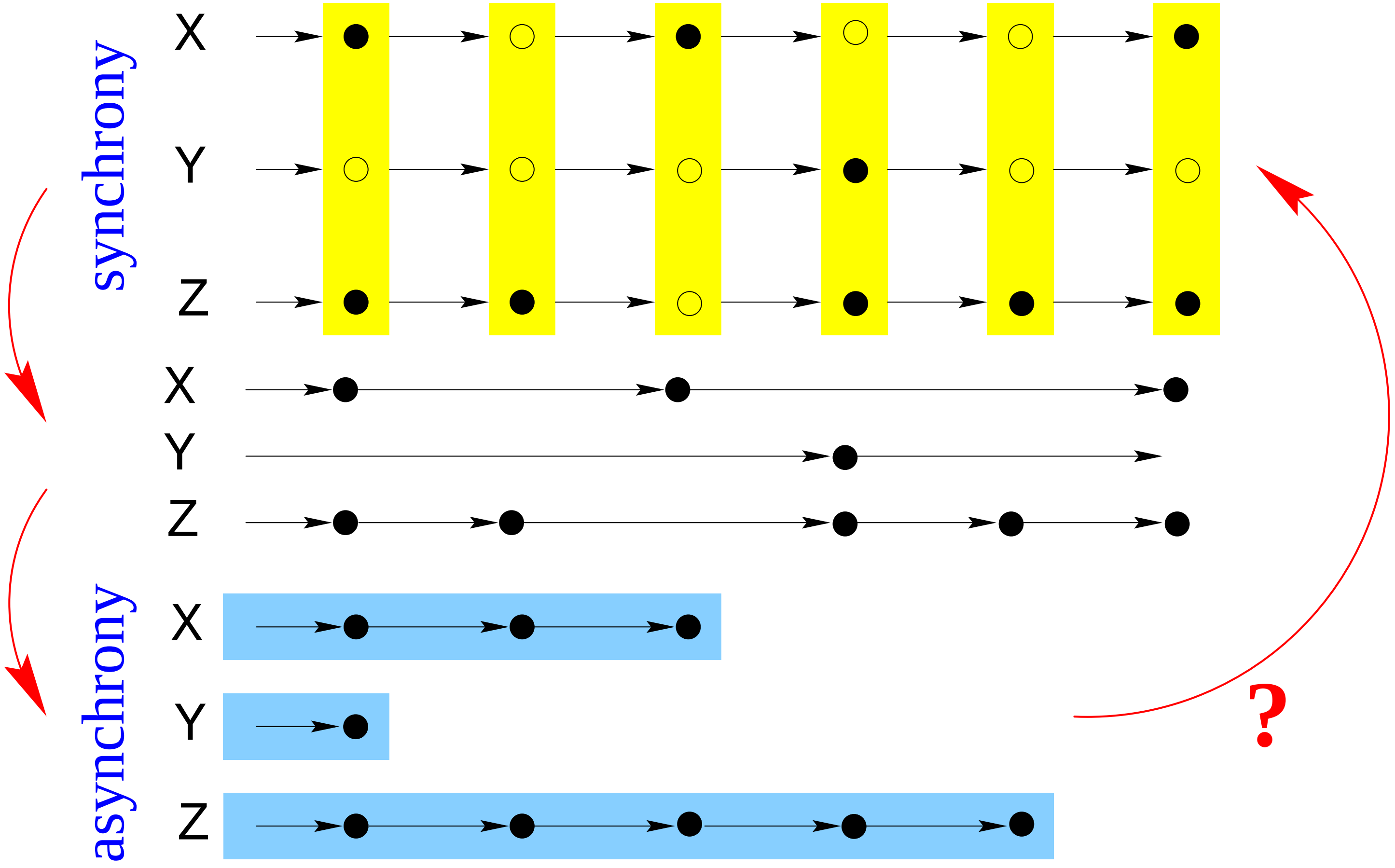
- No reaction, no global clock:

$$\mathbf{run}(P) = \text{tuple of sequences of values} = \text{tuple of channels}$$

- Absence cannot be sensed
- Communication: per channel unification of sequences



Relaxing synchrony: informal definition



Two questions

- Question 1:under which conditions is it possible to uniquely reconstruct the successive reactions from a desynchronized run?

$$\text{asynchronous run} \quad \stackrel{\text{unique ?}}{\longmapsto} \quad \text{synchronous run}$$

- Question 2:under which conditions synchronous and asynchronous compositions are equivalent?

$$P_1^a \parallel_a P_2^a \ \stackrel{?}{\equiv} \ (P_1 \parallel P_2)^a$$

STS : synchronous transition systems

- $\mathcal{V}$: finite set of typed variables with values $\in \mathcal{D} \cup \{\perp\}$, $\perp =$ absence
- STS $\Phi = \langle V, \Theta, \rho\rangle$ with $V \subseteq \mathcal{V}$
- State $s =$ valuation of a variables V
- Notations : $s[v]$, $s[U]$
- Initial states : $\{s | s \models \Theta\}$
- ρ is the transition relation $(s^-, s) \models \rho$
- Runs $\Sigma = \{\sigma = s_0, s_1, s_2, \ldots$ s.t. $s_0 \models \Theta \;\wedge\; \forall i > 0\,,\ (s_{i-1}, s_i) \models \rho\}$
- Composition $\Phi = \Phi_1 \parallel \Phi_2 = \langle V = V_1 \cup V_2, \Theta = \Theta_1 \wedge \Theta_2, \rho = \rho_1 \wedge \rho_2\rangle$

State, reactive and clock variables

- Reactive/state/clock variables: $V = V_r \uplus V_s \uplus V_h$, $v \in V_r$, $\xi_v \in V_s$, $h_v \in V_h$

$$s[\xi_v] = \begin{cases} s[v] & \text{if } s[v] \neq \perp \\ s^-[\xi_v] & \text{if } s[v] = \perp \end{cases} \qquad s[h_v] = \perp \iff s[v] = \perp$$

x	$\perp$	1	$\perp$	2	$\perp$
ξx	0	1	1	2	2
hx	$\perp$	T	$\perp$	T	$\perp$

Stuttering robustness

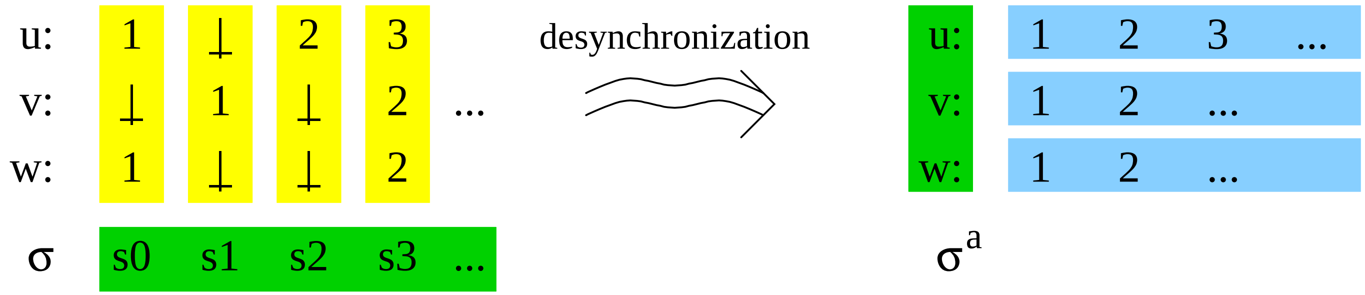
- A silent reaction is a state s s.t. $\forall v \in V_r, s[v] = \bot$
- $\bot_s$: silent reaction s.t. $\forall v \in V_r, s[\xi_v] = \bot_s[\xi_v]$
- Stuttering robustness : in any state, the STS can perform a silent reaction copying state variables and then can still perform the same reactions.

$$(s^-, s) \models \rho \Rightarrow (s^-, \bot_{s^-}) \models \rho \text{ and } (\bot_{s^-}, s) \models \rho$$

$$s_0, \ldots, s_i, s_{i+1}, \ldots \in \Sigma \Rightarrow s_0, \ldots, s_i, \bot_{s_i}, s_{i+1}, \ldots \in \Sigma$$

Modeling asynchronous behaviours

- Synchrony : absence can be used to exercise control.
- Asynchrony : absence cannot be detected.

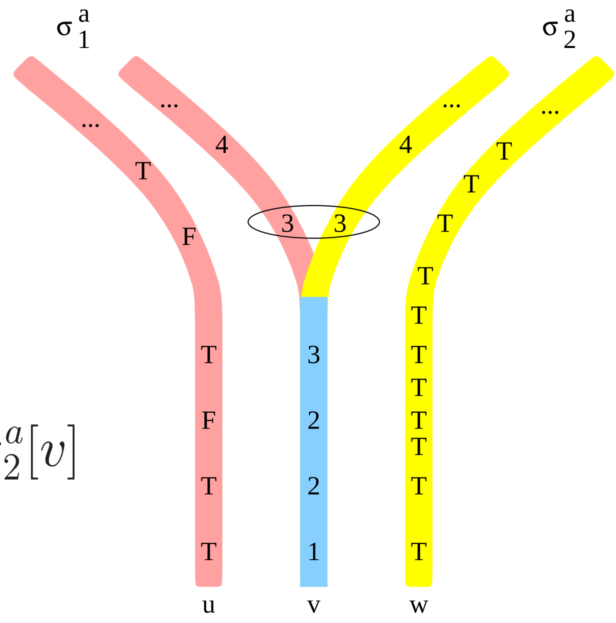


- asynchronous abstraction : $\Phi^a = \langle V, \Sigma^a = \{\sigma^a \text{ s.t. } \sigma \in \Sigma\}\rangle$

Asynchronous composition

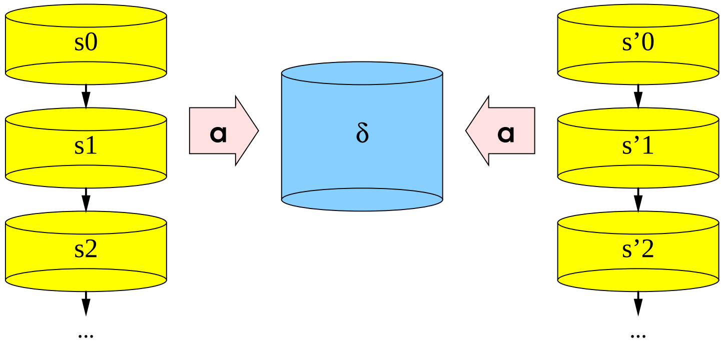
$$\Phi_1^a \parallel_a \Phi_2^a = \langle V = V_1 \cup V_2, \Sigma^a = \Sigma_1^a \ \wedge^a \ \Sigma_2^a \rangle$$
$$\Sigma_1^a \ \wedge^a \ \Sigma_2^a = \{\sigma_1^a \ \wedge^a \ \sigma_2^a \text{ s.t. } \sigma_1^a \in \Sigma_1^a, \sigma_2^a \in \Sigma_2^a \text{ unifiable}\}$$

- where two asynchronous runs are unifiable iff they have equal flows foreach shared variable,
- and $\wedge^a$ denotes the conjunction of asynchronous runs :



$$\forall v \in V_1 \cap V_2 \ : \ (\sigma_1^a \ \wedge^a \ \sigma_2^a)[v] = \sigma_1^a[v] = \sigma_2^a[v]$$
$$\forall v \in V_1 \setminus V_2 \ : \ (\sigma_1^a \ \wedge^a \ \sigma_2^a)[v] = \sigma_1^a[v]$$
$$\forall v \in V_2 \setminus V_1 \ : \ (\sigma_1^a \ \wedge^a \ \sigma_2^a)[v] = \sigma_2^a[v]$$

Desynchronizing a single STS : endochrony

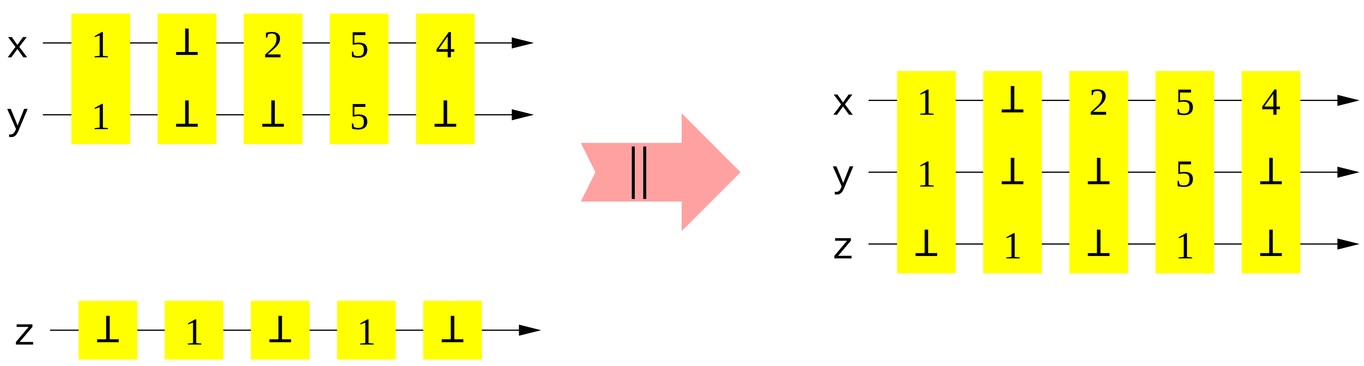


- Reconstructibility problem : unicity (up to stuttering) of the reconstruction of reactions from an asynchronous behaviour.

- Clock inference : $W \hookrightarrow_{s^-} W'$ with $W, W' \subseteq V_r$, iff $\forall s$ such that $(s^-, s) \models \rho$, $s[W]$ uniquely determines $s[h_{W'}]$.
- Endochrony : clock inference sequence converging to the set of all reactive variables: $\emptyset \hookrightarrow_{s^-} V_1 \hookrightarrow_{s^-} \ldots \hookrightarrow_{s^-} V_r$
- Endochrony $\Longrightarrow$ reconstructibility

Endochrony is not compositional

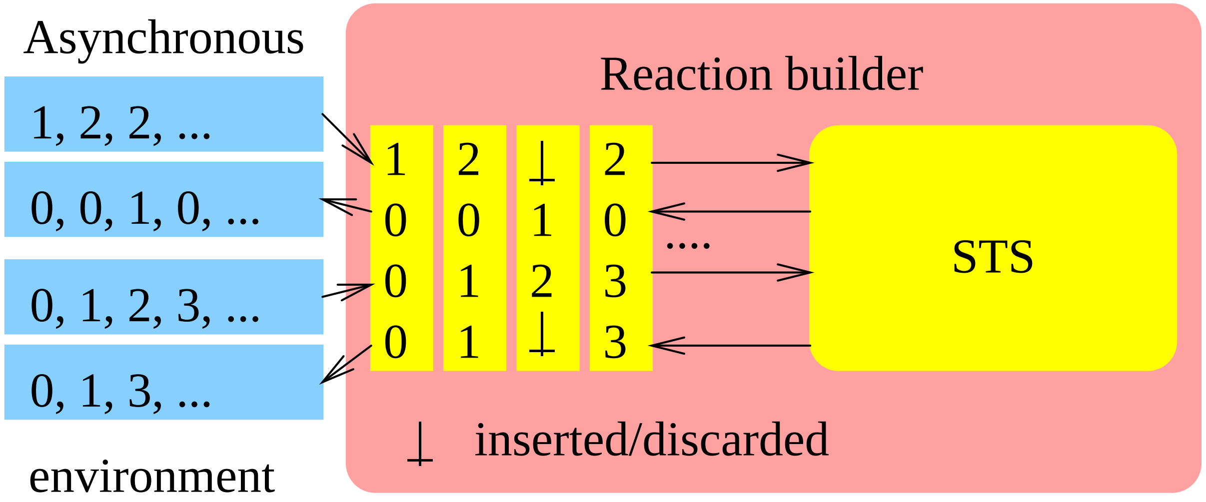
- $y = x$ when x is odd and $z = 1$ are both endochronous



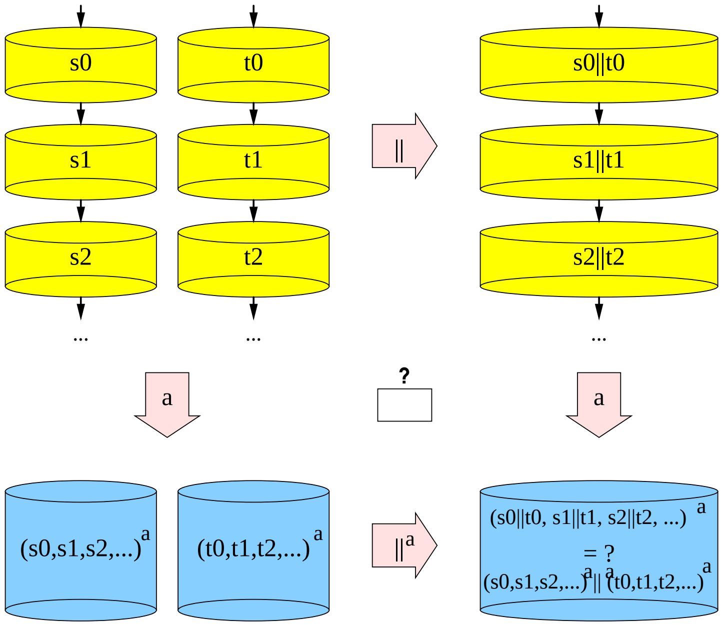
- However $(y = x$ when x is odd$) \parallel (z = 1)$ is not endochronous

Desynchronizing a single STS : application

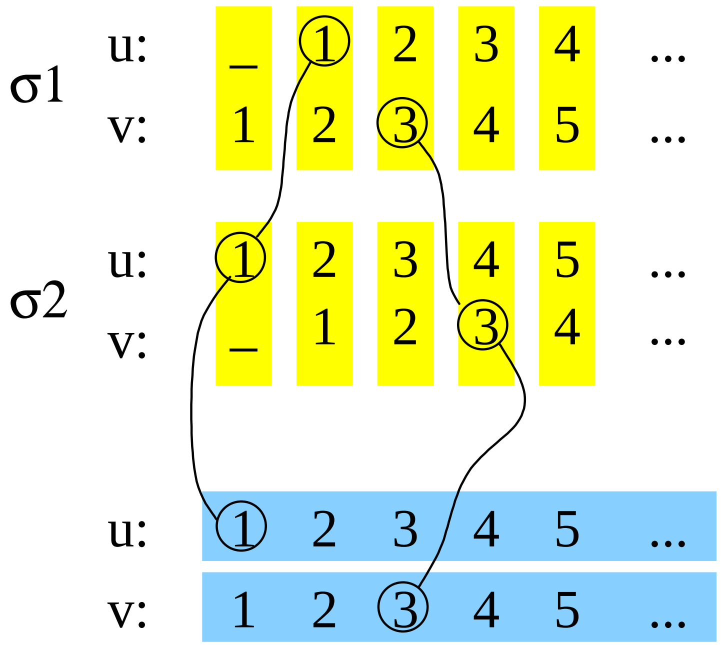
- Execution of a synchronous program in an asynchronous environment.
- In practice (SIGNAL compiler) : endochrony analysis on the boolean part of programs ; one clock inference sequence for all states.
- Compilation = clock hierarchy $\longrightarrow$ nested conditionals in a toplevel loop.



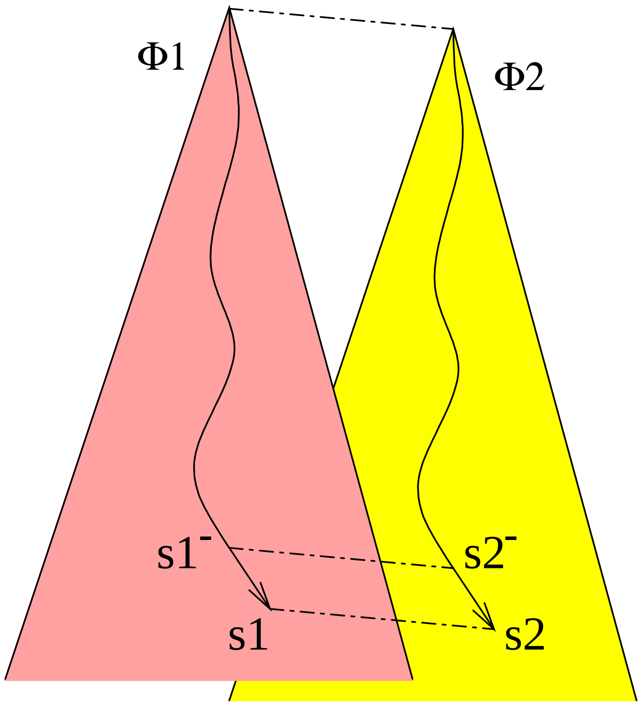
Desynchronizing two STS : synchronous-asynchronous equivalence



- Counter-example :



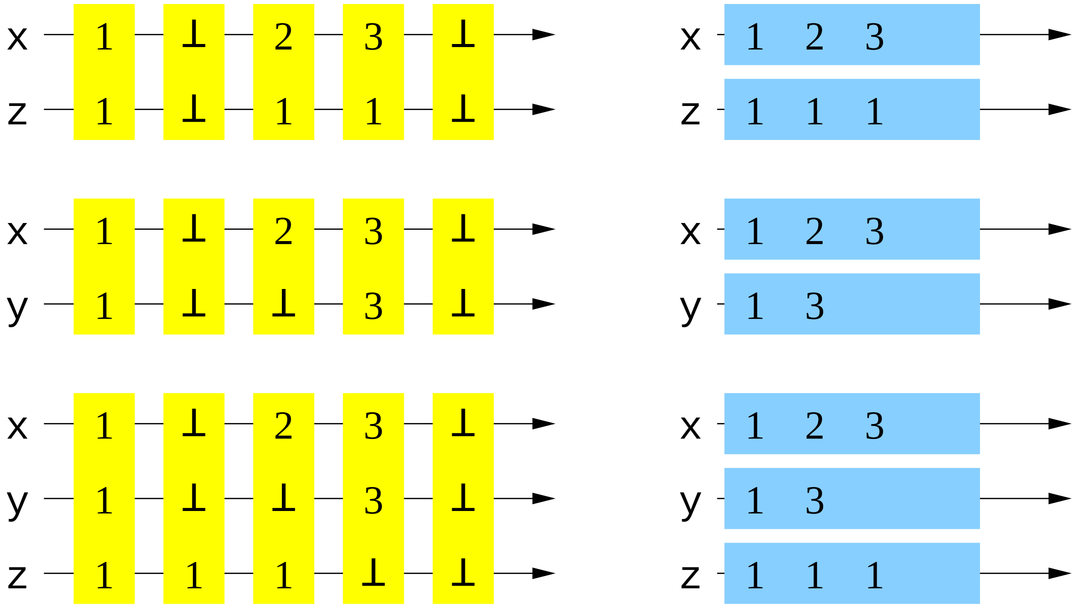
Desynchronizing two STS : Isochrony



- Isochrony :
  $\forall (s_1^-, s_2^-)$ accessible state of $\Phi_1 \| \Phi_2$,
  $\forall s_1$ non-silent reaction, s.t. $(s_1^-, s_1) \models \rho_1$
  $\forall s_2$ non-silent reaction, s.t. $(s_2^-, s_2) \models \rho_2$
  s_1 and s_2 unifiable over present variables
  $\Rightarrow$
  s_1 and s_2 unifiable
- Isochrony $\Rightarrow$ synchronous-asynchronous equivalence
- Endochrony $\wedge$ synchronous-asynchronous equivalence $\Rightarrow$ isochrony

(Counter-)Example

- $(x = \xi_x^- + z) \parallel (z = 1)$ and $y = x$ **when** x **is odd** are isochronous



- However $x = \xi_x^- + z$ and $(y = x$ **when** x **is odd**$) \parallel (z = 1)$ are not isochronous

Desynchronizing networks of STS

- Isochrony is compositional : (Ψ, Φ_1) and (Ψ, Φ_2) isochronous $\Rightarrow (\Phi, \Phi_1 \parallel \Phi_2)$ isochronous.
- Isochrony $\Rightarrow$ synchronous-asynchronous equivalence : $\Phi_1, \ldots, \Phi_n$ pair-wise isochronous :

$$(\Phi_1 \parallel \Phi_2 \parallel \ \ldots \ \parallel \Phi_n)^a = \Phi_1^a \parallel_a \Phi_2^a \parallel_a \ \ldots \ \parallel_a \Phi_n^a$$

- In practice (SIGNAL compiler) : isochrony analysis on the boolean part of programs ; generated code = dataflow network

Results

- Desynchronizing a single STS : endochrony $\Longrightarrow$ reconstructibility.
- Desynchronizing networks of STS :

  Isochrony $\Longrightarrow$ synchronous-asynchronous equivalence

  Endochrony of components $\wedge$ synchronous-asynchronous equivalence $\Longrightarrow$ isochrony.
- In practice : endo/isochrony required.
- Related problem :modularity and separate compilation.