Lectures on Desynchronization

Albert Benveniste, Benoît Caillaud, and Paul Le Guernic

IRISA

September 2003
1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion
SYNCHRONY

1. programs progress via reactions: \( P = R^\omega \)

   \[ \text{run}(P) = \text{sequence of tuples of events} \]
   \[ = \{(x_i(1))_{i=1,...,k}, (x_i(2))_{i=1,...,k}, \ldots\} \]

2. Within a reaction, decisions can be taken based on testing for the absence of some signals (denoted by “\( x = \perp \)”):

   \[
   \begin{align*}
   &\text{present } S \text{ else ‘stat’} \\
   &y = \text{current } x \\
   &y := u \text{ default } v
   \end{align*}
   \]

3. communication: instantaneous broadcast

   \[
   P_1 \parallel P_2 =_{\text{def}} P_1 \cap P_2 = (R_1 \wedge R_2)^\omega
   \]
1. **no reaction**, no global clock, no global state:

\[
\text{run}(P) = \text{tuple of sequences of events} \\
= (\{x_i(1), x_i(2), \ldots\})_{i=1,\ldots,k} \\
\triangleq \text{tuple of channels}
\]

2. **absence** cannot be sensed and has no meaning (\(\neg [x = \bot] \) holds).

3. **communication**: channel-per-channel unification

\[
P_1 \parallel P_2 \triangleq P_1 \cap P_2
\]
relaxing synchrony, informal definition

for systems and communications
relaxing synchrony, informal definition

for systems and communications
How to ensure that the two communicating components do not see the difference?
How to ensure that the two communicating components do not see the difference?
The issue of variable communication delay

if ∃ no absent variable, that asynchronous communications desalign reactions in physical time can be easily dealt with by compensating, at receiver, for variable latency.
The issue of variable communication delay

squares become bananas, unfortunately bananas cannot be recovered!

if $\exists$ no absent variable, that asynchronous communications desalign reactions in physical time can be easily dealt with by compensating, at receiver, for variable latency.

problematic if some variables are absent at some reactions.
The issue of variable communication delay

assign to each signal a boolean clock that is always present

This is a straightforward solution. However it requires 1/ to assign one signalling wire to each signal, 2/ to run computations and communications at the fastest rate, even for an architecture with slow/fast components. In the sequel we develop an algebraic approach that allows to optimize this.
1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion
Synchronous Transition Systems: $\Phi = \langle V, \Theta, \rho \rangle$

$V$: finite set of typed variables $\in D \cup \{\bot\}$ (“absent”)
$V^{-}$: “previous” variables

$\Theta(V)$: assertion characterizing initial states

$\rho(V^{-}, V)$: transition relation

states: valuation of all variables, $s = (s[v])_{v \in V}$

run: $\sigma : s_0, s_1, s_2, \ldots$, sequence of states such that

$$s_0 \models \Theta(s_0) \land \forall i > 0 \ , \ (s_{i-1}, s_i) \models \rho(s_{i-1}, s_i)$$

signal: $\sigma[v] : s_0[v], s_1[v], s_2[v], \ldots$
$\Phi_i = \langle V_i, \Theta_i, \rho_i \rangle, i = 1, 2$

$\Phi = \Phi_1 \parallel \Phi_2$

$V = V_1 \cup V_2$

$\Theta = \Theta_1 \wedge \Theta_2$

$\rho = \rho_1 \wedge \rho_2$

$\rho_1 \wedge \rho_2$ is the conjunction of $\rho_1$ and $\rho_2$, defined as follows. Transitions $(s_1^-, s_1)$ and $(s_2^-, s_2)$ are unifiable, written $(s_1^-, s_1) \bowtie (s_2^-, s_2)$, iff they agree on the shared variables; then $(s^-, s)$ is the join of $(s_1^-, s_1)$ and $(s_2^-, s_2)$, and $\rho_1 \wedge \rho_2$ is the set of these $(s^-, s)$. 
Asynchronous Systems: $\Phi = \langle V, \Sigma \rangle$

$V$: finite set of typed variables $\in D$
(“absent” has no meaning)

local states: valuation of individual variables, $s[v]$ 

signal: $\sigma[v] : s_0[v], s_1[v], s_2[v], \ldots$

run: $\sigma = \{\sigma[v], v \in V\}$, tuple of signals

$\Sigma$: set of legal runs
\[ \Phi_i = \langle V_i, \Sigma_i \rangle, i = 1, 2 \]
\[ \Phi = \Phi_1 \parallel \Phi_2 \]
\[ V = V_1 \cup V_2 \]
\[ \Sigma = \Sigma_1 \land \Sigma_2 \]

\( \Sigma_1 \land \Sigma_2 \) is the conjunction of \( \Sigma_1 \) and \( \Sigma_2 \), defined as follows. Runs \( \sigma_1 \) and \( \sigma_2 \) are unifiable, written \( \sigma_1 \bowtie \sigma_2 \), iff they agree on the shared variables, meaning that signals associated with shared variables are equal; then \( \sigma \) is the join of \( \sigma_1 \) and \( \sigma_2 \), and \( \Sigma \) is the set of these \( \sigma \).
1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion
GALS = $\Phi_1 \| \text{asynch channel} \| \Phi_2$
GALS = \Phi_1 \parallel \text{asynch channel} \parallel \Phi_2

\sigma_1 \text{ run of } \Phi_1, \sigma_1^a \text{ desynchronized run of } \Phi_1, \\
c (\text{asynchronous}) \text{ run of } C; \text{ write } \sigma_1^a \bowtie c \text{ iff } \sigma_1^a \bowtie c; \\
\text{then } \Phi_1 \parallel_a C \text{ is the set of pairs } (\sigma_1, c) \text{ such that } \sigma_1^a \bowtie c.
GALS: preserving semantics?

\[ \Phi_1 \parallel \Phi_2 = \Phi_1 \parallel Identity \parallel \Phi_2 \]

\[ \equiv \Phi_1 \parallel C \parallel \Phi_2 \]

symbol \( \equiv \) means that both sides possess identical pairs of unifiable runs, for \( \Phi_1 \) and \( \Phi_2 \).
Two key problems

**Thm 1** [Emsoft2003]: $\Phi_1 \parallel \Phi_2 \equiv \Phi_1 \parallel C \parallel \Phi_2$ holds if Questions 1 & 2 below receive positive answers

**QUESTION 1:** is it possible, for $\Phi_i, i = 1, 2$, to reconstruct the successive reactions from a “desynchronised” run?

$$\Phi_i \iff \Phi_i^a \quad ? \quad \Phi_i$$

**QUESTION 2:** does desynchronization commute with parallel composition, for pair $(\Phi_1, \Phi_2)$?

$$\Phi_1^a \parallel \Phi_2^a \quad ? \quad (\Phi_1 \parallel \Phi_2)^a$$
QUESTION 1: endochrony [Benv. & al. 2000]

\[ \Phi \longrightarrow \Phi^a \quad ? \quad \Phi \]

\[ \emptyset \leftrightarrow V_1 \leftrightarrow V_2 \leftrightarrow \ldots \leftrightarrow V \]

where:

\[ V_1 \leftrightarrow V_2 : \text{infer presence of } V_2 \text{ from values of } V_1 \]

protocol for the on-line reconstruction of successive reactions, for endochronous STS:

\begin{align*}
\text{await } V_1 & \Rightarrow \text{val}(V_1) \Rightarrow \text{pres/abs}(V_2) \\
\text{await } \text{pres}(V_2) & \Rightarrow \text{val}(<\text{pres}(V_2)) \Rightarrow \text{pres/abs}(V_3) \\
\text{etc... until } V
\end{align*}
endochrony: (counter)examples

examples:

- a single-clocked STS
- if $b = T$ then get $u$

counterexample:

- $\Phi_1 \parallel \Phi_2$, where the $\Phi_i$ do not communicate at all ($V_1 \cap V_2 = \emptyset$) — internal asynchrony
QUESTION 2: isochrony [Benv. & al. 2000]

\( \Phi_1^a \parallel \Phi_2^a \overset{?}{=} (\Phi_1 \parallel \Phi_2)^a \)

\( \rho_1 \land \rho_2 \) : unifies on \{absence, present values\}

\( \rho_1 \land_a \rho_2 \) : unifies on \{present values\} only

**isochrony**: \( \rho_1 \land \rho_2 = \rho_1 \land_a \rho_2 \)
isochrony, details

\( (\Phi_1, \Phi_2) : W = V_1 \cap V_2, \) write \( t = (s^-, s) \)

\( t_1 \otimes t_2 : \forall w \in W, t_1[w] = t_2[w] \)

\( \rho_1 \land \rho_2 : \forall t_1 \otimes t_2, t[v] = \begin{cases} 
\text{if } v \in V_i \\
\text{then } t_i[v] 
\end{cases} \)

\( t_1 \otimes_a t_2 : s_1, s_2 \neq \bot \text{ and } \forall w \in W \begin{cases} 
\text{if } v \in V_i, s_1[w], s_2[w] \neq \bot \\
\downarrow \\
t_1[w] = t_2[w]
\end{cases} \)

\( \rho_1 \land_a \rho_2 : \forall t_1 \otimes_a t_2, t[v] = \begin{cases} 
\text{if } v \in V_i \text{ and } t_i[v] \neq \bot \\
\text{then } t_i[v] \text{ else } \bot
\end{cases} \)

**isochrony:** \( \rho_1 \land \rho_2 = \rho_1 \land_a \rho_2 \)

\( (s \neq \bot \) means that state \( s \) is non-silent, i.e., has some present variables in it)
isochrony: (counter)examples

examples:

- a pair \((\Phi_1, \Phi_2)\) with single-clocked communications (strongly synchronous)

- \(V_1 \cap V_2 = \emptyset\): a pair \((\Phi_1, \Phi_2)\) with no communication at all (fully asynchronous)

counterexample:

\[
[\text{await } X \ || \ \text{await } Y \ || \ Z = \text{merge}(X,Y) ]
\\
[ \text{emit } X \text{ when } C \ || \ \text{emit } Y \text{ when not } C ]
\]
Isochrony is local

\((\Phi_2, \Psi)\) isochronous
\((\Phi_1, \Psi)\) do not interact
\[\downarrow\]
\((\Phi_1 \parallel \Phi_2, \Psi)\) isochronous
\( \Phi_1 \) and \( \Phi_2 \) endochronous and \( (\Phi_1, \Phi_2) \) isochronous

\[ \Downarrow \quad \text{by Thm 1} \]

\[ \Phi_1 \parallel \Phi_2 \equiv \Phi_1 \parallel_{channel} \Phi_2 \]

from synchronous specification to GALS deployment while preserving semantics
endo/isochrony

checkable on synchronous program(s)

can be enforced by adding protocols
1. Synchrony vs. asynchrony

2. Synchronous/Asynchronous Transition Systems

3. Preserving semantics: Endochrony, Isochrony

4. Discussion
What about GALS with $\geq 2$ components?

- So far isochrony has been defined only for pairs; for $\geq 2$ components it is not enough to require isochrony for all pairs, because it is not true that $(\Phi_i, \Phi_j)$ for $(i, j) = (1, 2), (2, 3), (3, 1)$ implies $((\Phi_1 \parallel \Phi_2), \Phi_3)$ isochronous.

- Thus we need to define isochrony for tuples directly, a straightforward extension. Unfortunately, whereas isochrony is tight for pairs, is it too strong for tuples — many correct GALS deployments are not isochronous, the problem arises from components involving concurrent transitions.

- [Potop-Butucaru & Caillaud 2003] have provided the right generalization of endo/isochrony for GALS networks and components possibly involving concurrent transitions — beyond the scope of these notes.
Valid Architectures for the theory to apply

1. messages shall not be lost
2. ordering within each channel shall be preserved

YES (assuming nominal, not faulty behaviour):
- RTOS POSIX
- point to point comm. via fifos
- “synchronous comm.” in OS
- UNIX sockets
- inter-object comm., (dynamic instanciation)

NO (???):
- CAN
- ARINC
- field busses

BUT ... still feasible [Emsoft2002] ...
“loosely” time triggered bus

- clocks are independent and not synchronized
- read / write are clocked & independent
- periodic bus: reads at $n$, writes at $n + 1$
- non blocking communication
- **robust against failures**, but...
- communications can duplicate or loose messages
  $\Rightarrow$ our theory does not apply directly
loosely time triggered communications

Loosely time triggered bus

\[ \text{sustain} \rightarrow \text{scan} \]

encoding

decoding

\[ \text{y}_{\text{put}} \rightarrow \text{y}_{\text{get}} \]

Theorem:

if bus clock \( \gg \) write/read clocks (upsampling), then \( y_{\text{put}} \rightarrow y_{\text{get}} \) satisfies requirements for desynch.

- timeliness ? timing evaluation
- fault tolerance ? watchdogs & exceptions