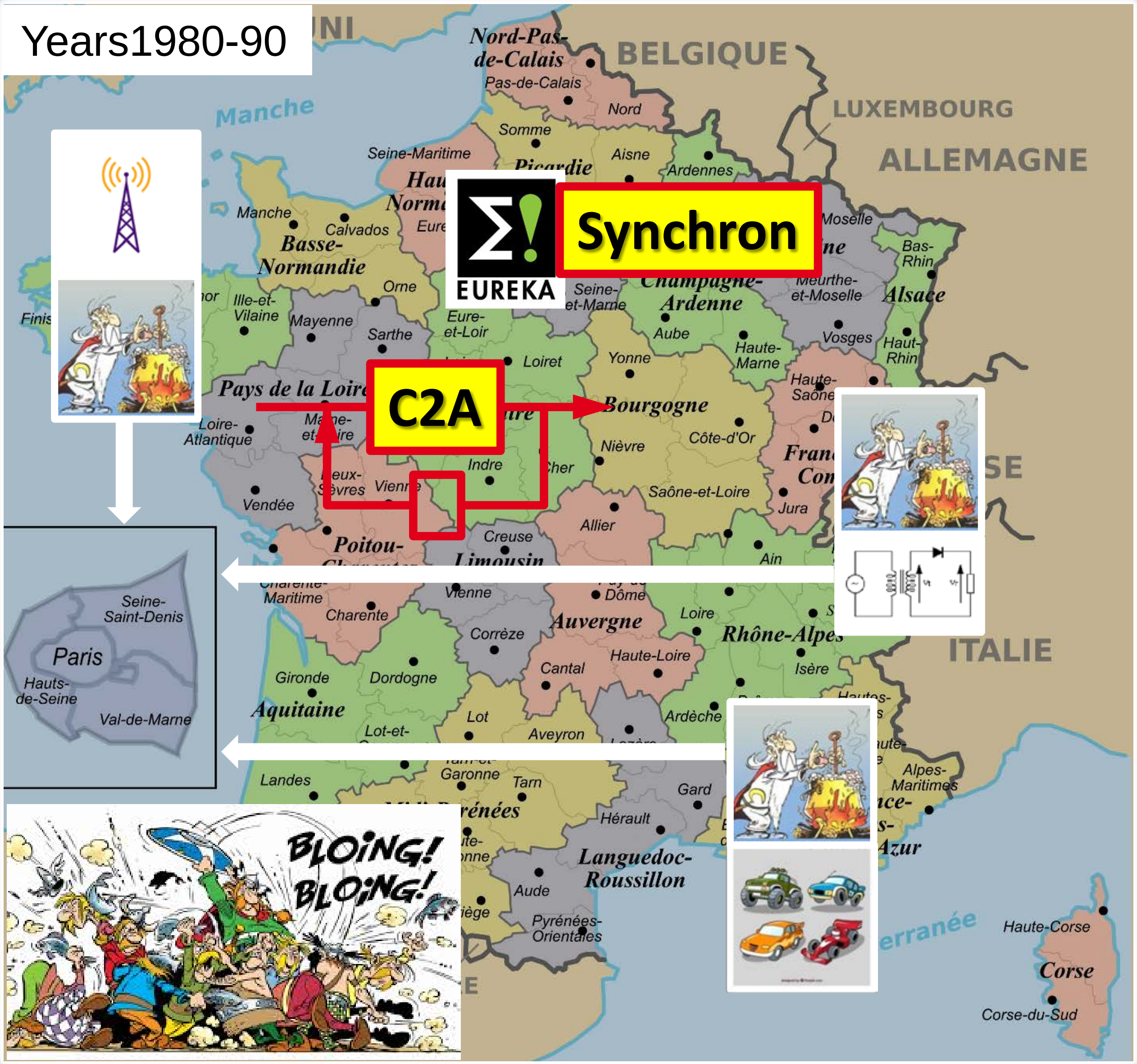
The Signal synchronous language: the principles beyond the language and how to exploit and extend them

Albert Benveniste and Thierry Gautier (Inria-Rennes)

Acknowledgement: Paul Le Guernic and Loïc Besnard

Nicolas Halbwachs Feria, June 2018

Years1980-90

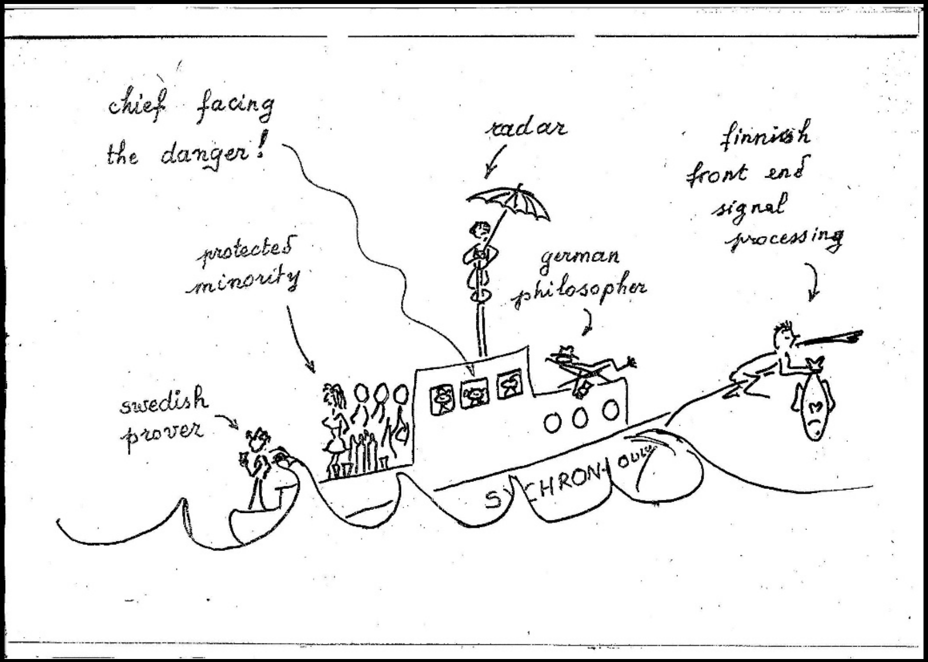

Years1980-90
EUREKA
Synchron
C2A
BLOING! BLOING!
BELGIQUE
LUXEMBOURG
ALLEMAGNE
ITALIE
Manche
Paris

Oulu

The official picture

Oulu

Hand written notes by Albert at a talk given by Nicolas



A picture taken by Nicolas

The official picture

Synchronous Guys
by Willem-Paul de Roever, 2002

Giving birth to Synchronous Languages

Are they *programming languages*? Yes, but…
Are they *modeling languages*? Well, cannot disagree…

Giving birth to Synchronous Languages

Are they *programming languages*? Yes, but…
Are they *modeling languages*? Well, cannot disagree…

Are they really *synchronous*? MMhhh, what about the *bananas*?

Giving birth to Synchronous Languages

Are they *programming languages*? Yes, but…

Are they *modeling languages*? Well, cannot disagree…

Are they really *synchronous*? MMhhh, what about the *bananas*?

What is *time* in synchrony? It's not time!

Giving birth to Synchronous Languages

Are they *programming languages*? Yes, but…
Are they *modeling languages*? Well, cannot disagree…

Are they really *synchronous*? MMhhh, what about the *bananas*?

What is *time* in synchrony? It's not time!

Is it simple? It can be
Is it powerful? It can be

Giving birth to Synchronous Languages

Are they *programming languages*?	Yes, but…
Are they *modeling languages*?	Well, cannot disagree…
Are they really *synchronous*?	MMhhh, what about the *bananas*?
What is *time* in synchrony?	It's not time!
Is it simple?	It can be
Is it powerful?	It can be
What about crowd-correcting?	It's all crowdless
Crowd-cleaning?	Semantics, semantics, semantics,
Crowd-debugging?	semantics, and more semantics

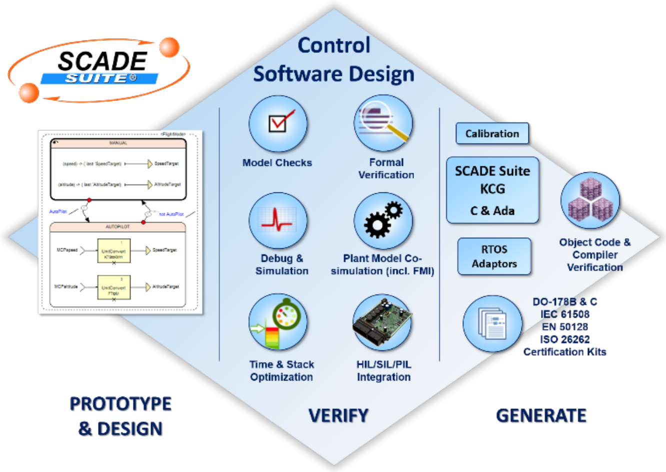
Signal: an original positioning in the landscape of synchronous languages

Albert Benveniste and Thierry Gautier

June 2018

Lustre
dataflow functional languages

Lustre, Lucid Synchrone, Scade, (Zélus)



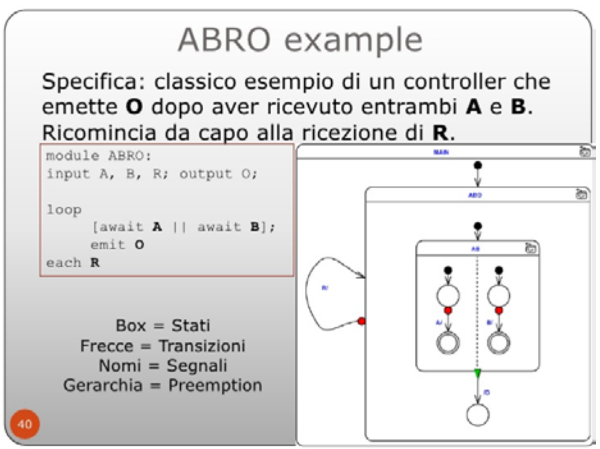
- Streams (seq. of values)
- Dataflow composition à la Kahn: functional

- Simple
- No delay-free loop
- Higher order: dynamicity

- (Clocks as types)

Esterel
imperative languages

Esterel, SyncCharts, SCL/SCCharts, ReactiveML, the web



- variables and values, await, emit, ||, preemption
- Difficulty: combining || and immediate control passing
- Reaction as a fixpoint problem: 0/**1**/several solutions

Signal
equation based language

Open systems and architecture modeling:

- Synchronization
- Clocks as 1st class citizens

A program can have 1000's of clocks $\Rightarrow$ clocks must be synthesized, not verified

- (clocks as types in Lustre $\Rightarrow$ "condact" used in Scade)

- Clock equations + Dataflow expressions
- Nondeterminism (but controlled)
- Open systems: stuttering invariance

  (a system has always the provision to sleep while its environment acts)

- Difficulty: Clocks $\leftrightarrow$ Data

Contents

The Signal vintage watch

An example of Signal program and its compilation

Intuitive pseudo-code


```
X := pre(X)-1
     reset IN every pre(X)≤0
```

Input **IN** returns **X** (mmmmhhh??)

IN is provided only when used

An example of Signal program and its compilation

```
(   X := IN default ZX-1          stream funct
|  ZX := X$1 init 0               stream funct
|  IN ^= when (ZX ≤ 0) )          clock eqn
```

Signal code

Intuitive pseudo-code

```
X := pre(X)-1
      reset IN every pre(X)≤0
```

Input **IN** returns **X** (mmmmhhh??)

IN is provided only when used

An example of Signal program and its compilation

```
(   X := IN default ZX-1      stream funct
| ZX := X$1 init 0            stream funct
| IN ^= when (ZX ≤ 0) )       clock eqn
```

IN	2			3				5	
ZX	0	2	1	0	3	2	1	0	5
X	2	1	0	3	2	1	0	5	4

Input **IN** returns **X** (mmmmhhh??)

IN is provided only when used

An example of Signal program and its compilation

```
(   X := IN default ZX-1       stream funct
|  ZX := X$1 init 0            stream funct
|  IN ^= when (ZX ≤ 0) )       clock eqn
```

IN	2			3				5	
ZX	0	2	1	0	3	2	1	0	5
X	2	1	0	3	2	1	0	5	4

IN is schizophrenic: its value is an input of the program but its clock (instants of presence) is not

Signal

```
X := f(U,V)
```

X	f(u,v)	•	•	•	f(u,v)	•	•	•	
U	u1	•	•	•	u2	•	•	•	
V	v1	•	•	•	v2	•	•	•	

```
X := Y$1 init X0
```

X	x0	•	•	y1	•	•	•	y2	
Y	y1	•	•	y2	•	•	•	y3	

• : absence (stuttering invariance)

Signal

`X := f(U,V)`

X	f(u,v)	•	•	•	f(u,v)	•	•	•	
U	u1	•	•	•	u2	•	•	•	
V	v1	•	•	•	v2	•	•	•	

`X := Y$1 init X0`

X	x0	•	•	y1	•	•	•	y2	
Y	y1	•	•	y2	•	•	•	y3	

`X := U default V`

X	u1	•	•	•	v2	•	u2	•	
U	u1	•	•	•	•	•	u2	•	
V	v1	•	•	•	v2	•	•	•	

`X := Y when B`

X	y	•	•	•	yk	•	•	•	
Y	y1				yk				
B	True				True				

`X := f(U,V)`

X	f(u,v)	•	•	•	f(u,v)	•	•	•	
U	u1	•	•	•	u2	•	•	•	
V	v1	•	•	•	v2	•	•	•	

`X := Y$1 init X0`

X	x0	•	•	y1	•	•	•	y2	
Y	y1	•	•	y2	•	•	•	y3	

`X := U default V`

X	u1	•	•	•	v2	•	u2	•	
U	u1	•	•	•	•	•	u2	•	
V	v1	•	•	•	v2	•	•	•	

`X := Y when B`

X	y	•	•	•	yk	•	•	•	
Y	y1				yk				
B	True				True				

`K ^= H` equality of clocks: a constraint

An example of Signal program and its compilation

```
(   X := IN default ZX-1      stream func
|  ZX := X$1 init 0           stream func
|   B := (ZX ≤ 0)             stream func
|  IN ^= (when B)             clock eqn
|   H ^= B ^= X ^= ZX )       clock eqn
```

```
[B]: when B
```

Expanded as

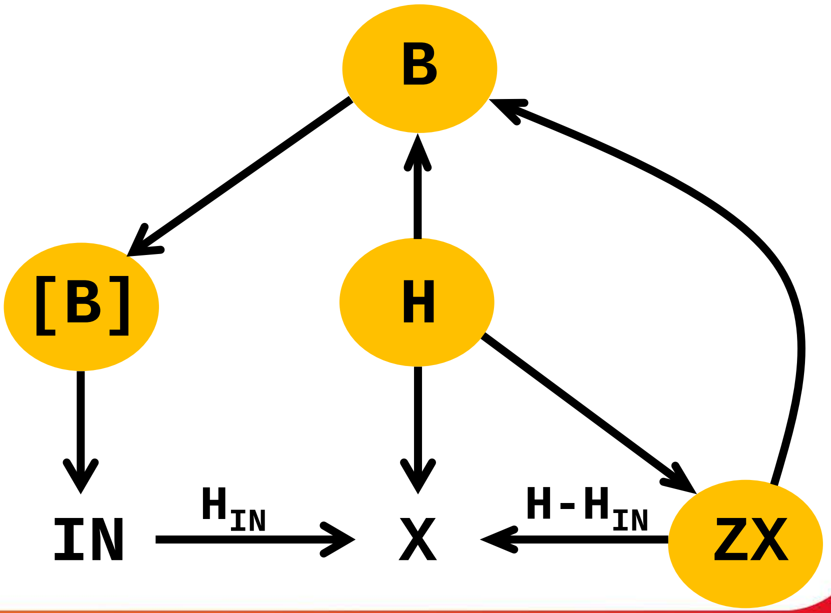
```
(   X := IN default ZX-1
|  ZX := X$1 init 0
|  IN ^= when (ZX ≤ 0) )
```

An example of Signal program and its compilation

`(  X := IN default ZX-1`	stream func
`| ZX := X$1 init 0`	stream func
`|  B := (ZX ≤ 0)`	stream func
`| IN ^= (when B)`	clock eqn
`|  H ^= B ^= X ^= ZX )`	clock eqn

```
[B]: when B
```

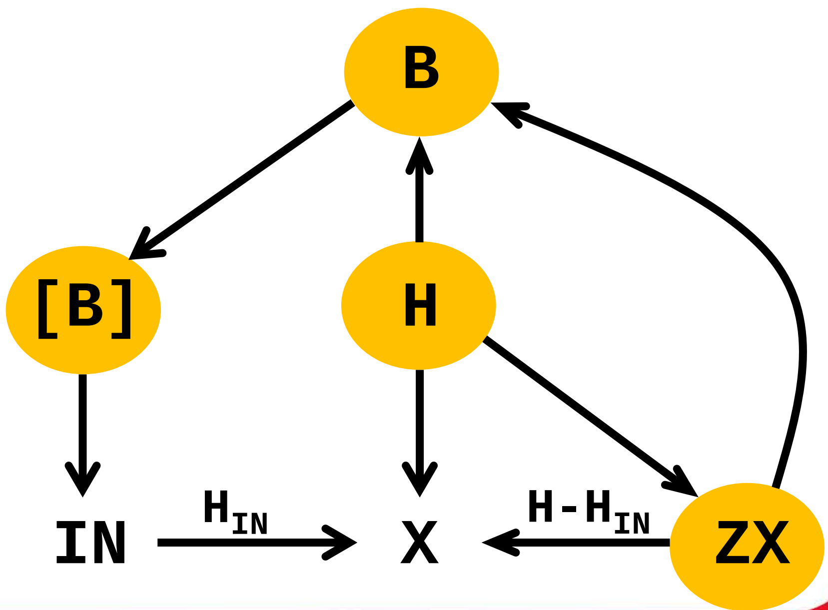
```
(  X := IN default ZX-1
| ZX := X$1 init 0
| IN ^= when (ZX ≤ 0) )
```



An example of Signal program and its compilation

```
(  X := IN default ZX-1        stream func
| ZX := X$1 init 0             stream func
|  B := (ZX ≤ 0)               stream func
| IN ^= (when B)               clock eqn
|  H ^= B ^= X ^= ZX )         clock eqn
```

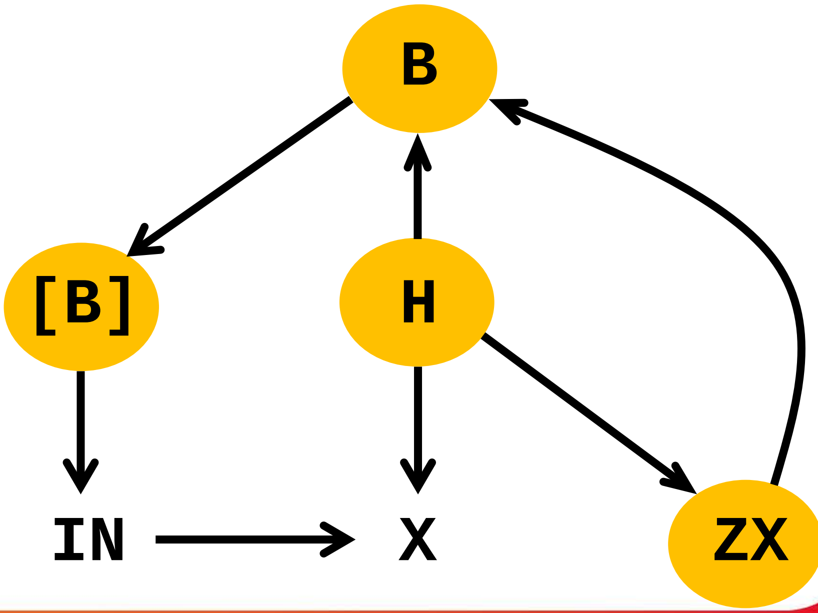
```
[B]: when B
case B true
case B false
```



An example of Signal program and its compilation

```
(  X := IN default ZX-1      stream func
| ZX := X$1 init 0           stream func
|  B := (ZX ≤ 0)             stream func
| IN ^= (when B)             clock eqn
|  H ^= B ^= X ^= ZX )       clock eqn
```

```
[B]: when B
case B true
```

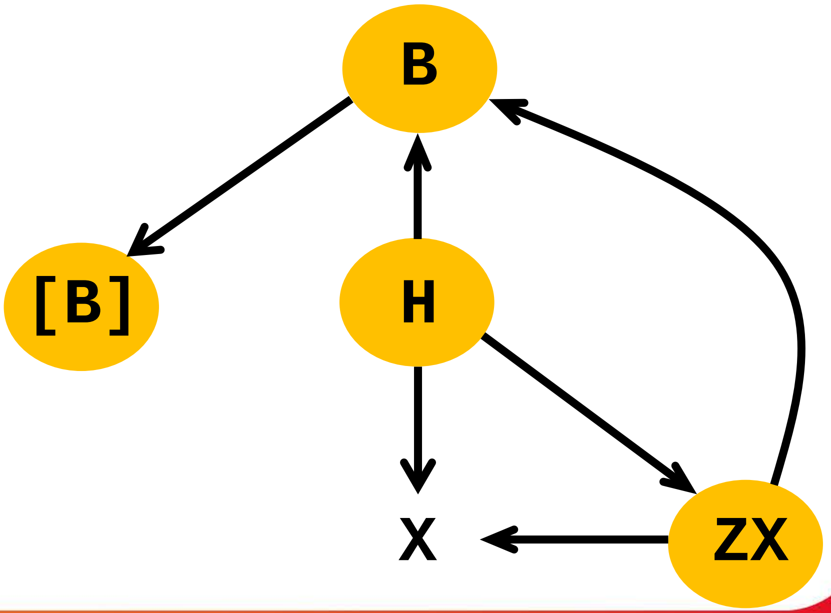


An example of Signal program and its compilation

```
(  X := IN default ZX-1         stream func
| ZX := X$1 init 0              stream func
|  B := (ZX ≤ 0)                stream func
| IN ^= (when B)                clock eqn
|  H ^= B ^= X ^= ZX )          clock eqn
```

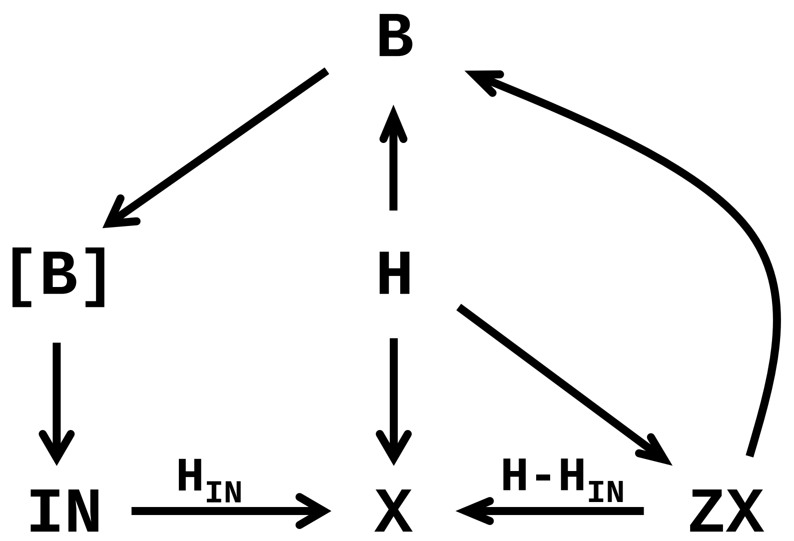
```
[B]: when B

case B false
```



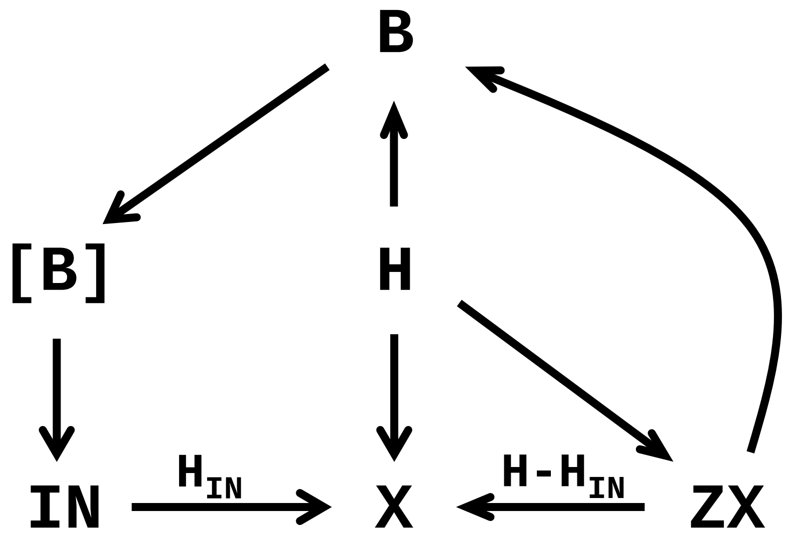
An example of Signal program and its compilation

```
(  X := IN default ZX-1
| ZX := X$1 init 0
|  B := (ZX ≤ 0)
| IN ^= (when B)
|  H ^= B ^= X ^= ZX )
```



An example of Signal program and its compilation

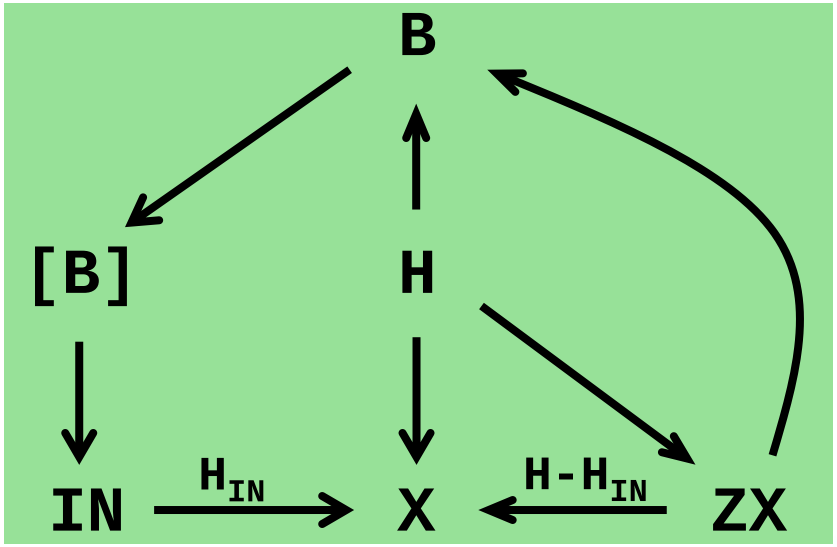
```
(   X := IN default ZX-1
 | ZX := X$1 init 0
 |  B := (ZX ≤ 0)
 | IN ^= (when B)
 |  H ^= B ^= X ^= ZX )
```



An example of Signal program and its compilation

```
(
  ( H ^= B ^= X ^= ZX
  | IN ^= (when B) )
|
  ( X  ←  H
  | ZX ←  H
  | B  ← (H,ZX)
  | (when B) ← B
  | IN ← (when B)
  | (X ← IN) when B
  | (X ← ZX) when not B )
|
  ( B := (ZX ≤ 0)
  | ZX := X$1 init 0
  | (X := IN) when B
  | (X := ZX-1) when not B)
)
```

```
(  X := IN default ZX-1
| ZX := X$1 init 0
|  B := (ZX ≤ 0)
| IN ^= (when B)
|  H ^= B ^= X ^= ZX )
```



An example of Signal program and its compilation

```
(
  ( H ^= B ^= X ^= ZX
  | IN ^= (when B) )
|
  ( X  ←  H
  | ZX  ←  H
  | B  ← (H,ZX)
  | (when B) ←  B
  | IN  ← (when B)
  | (X  ← IN) when B
  | (X  ← ZX) when not B )
|
  ( B := (ZX ≤ 0)
  | ZX := X$1 init 0
  | (X := IN) when B
  | (X := ZX-1) when not B)
)
```

Clock equations

Causality constraints

Computation actions

An example of Signal program and its compilation

```
(
  (  H ^= B ^= X ^= ZX
  | IN ^= (when B) )
|
  (  X  ←  H
  | ZX  ←  H
  |  B  ← (H,ZX)
  | (when B) ←  B
  | IN  ← (when B)
  | (X  ← IN) when B
  | (X  ← ZX) when not B )
|
  (  B := (ZX ≤ 0)
  | ZX := X$1 init 0
  | (X := IN) when B
  | (X := ZX-1) when not B)
)
```

```
(  X := IN default ZX-1
| ZX := X$1 init 0
| IN ^= when (ZX ≤ 0) )
```

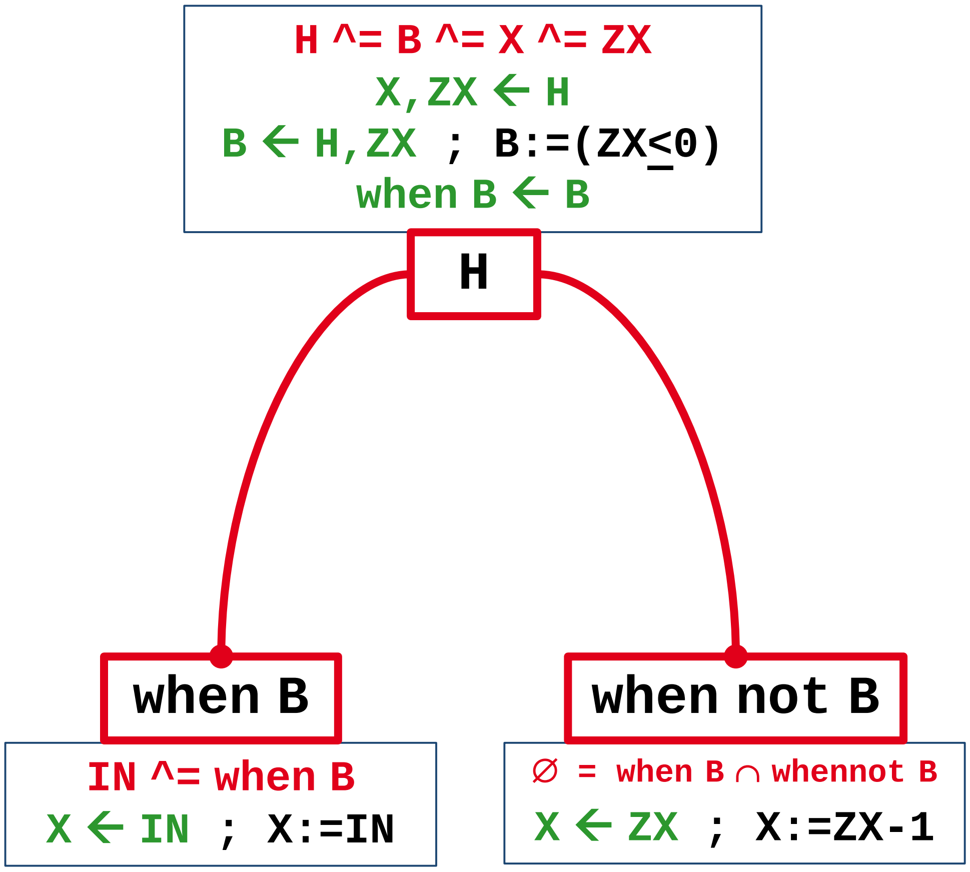
Signal compilation
is by
program rewriting

The clock and causality calculus

Intuition

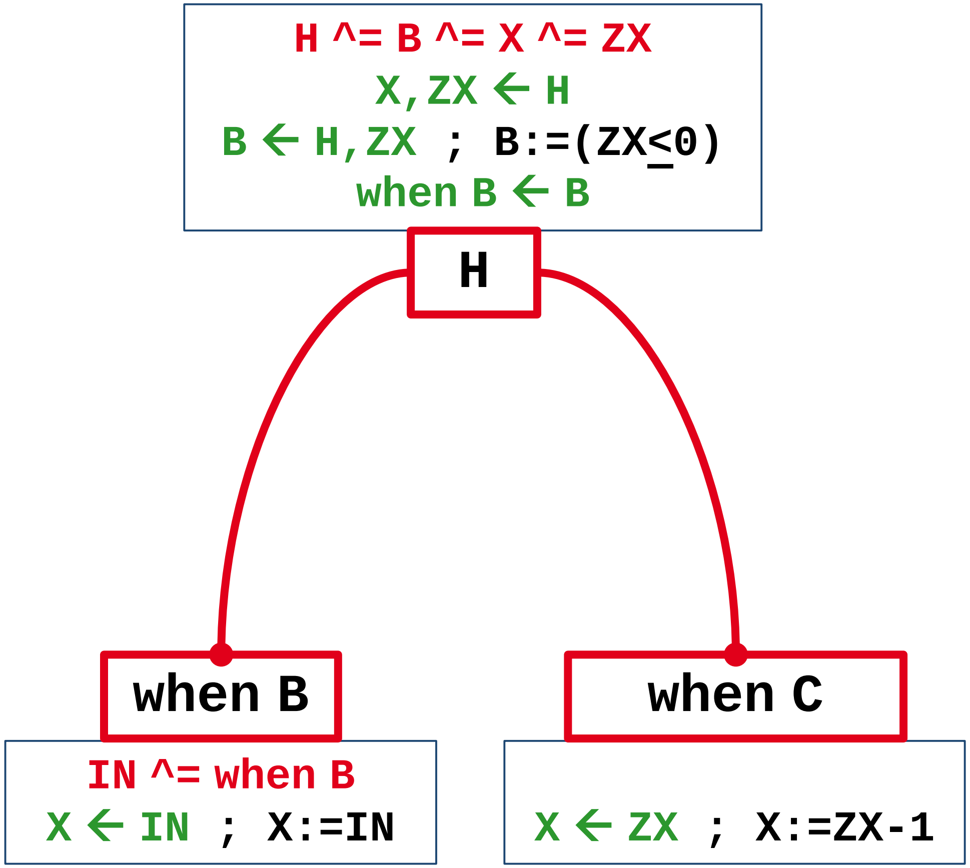
Clock and causality calculus

```
(
  ( H ^= B ^= X ^= ZX
  | IN ^= (when B) )
|
  ( X ← H
  | ZX ← H
  | B ← H,ZX
  | (when B)← B
  | IN ← (when B)
  | (X ← IN) when B
  | (X ← ZX) when not B )
|
  ( B := (ZX ≤ 0)
  | ZX := X$1 init 0
  | (X := IN) when B
  | (X := ZX-1) when not B)
)
```



Clock and causality calculus

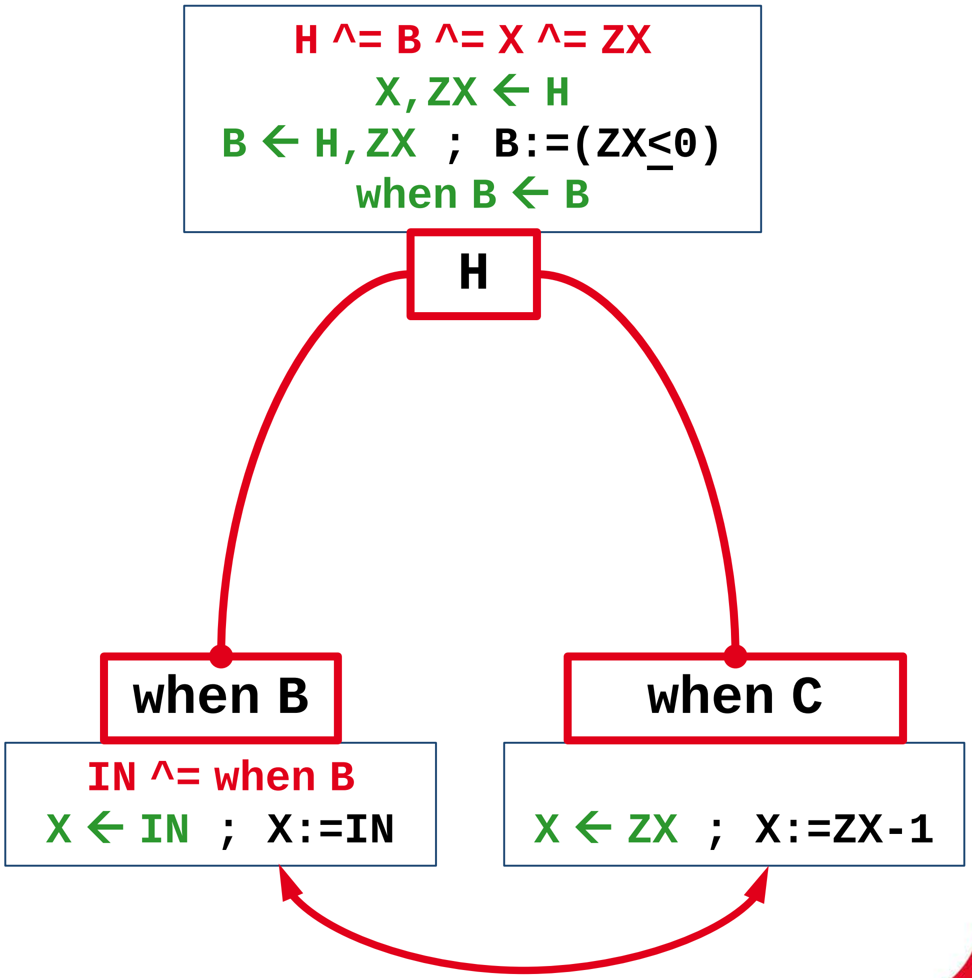
```
(
   (  H ^= B ^= X ^= ZX
   | IN ^= (when B) )
|
   (  X ← H
   | ZX ← H
   |  B,C ← H,ZX
   | (when B)← B
   | IN ← (when B)
   | (X ← IN) when B
   | (X ← ZX) when C )
|
   (  B := (ZX ≤ 0); C:=…
   | ZX := X$1 init 0
   | (X := IN) when B
   | (X := ZX-1) when C )
)
```



Clock and causality calculus

To ensure the absence of race condition, a proof obligation is added to the clock calculus:

$\emptyset$ ^= when B $\cap$ when C

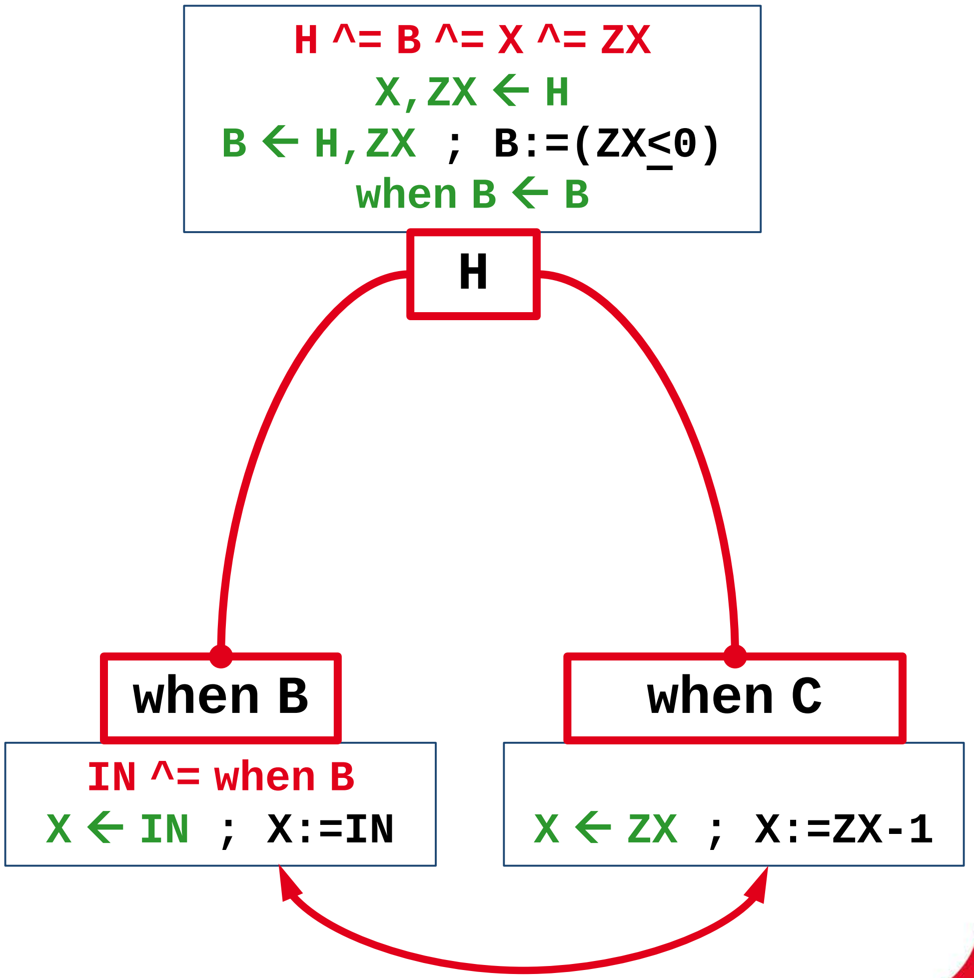


Clock and causality calculus

In general, clock equations originate from:

- the code itself
- race conditions: have them with ∅ clock
- causality circuits: have them with ∅ clock

We need to prove that the clock system is satisfiable and we must represent all solutions of it



The clock equations

Clock equations originate from:

- the code itself
- race conditions: have them with ∅ clock
- causality circuits: have them with ∅ clock

Wanted: a **clock hierarchy,** leading to code with nested ifs

Clocks and clock equations

1. `∅ (nil)`; no "`top`"
2. `H ^= K`
3. `H ^∧ K, H ^∨ K`
4. `H ^- K` (**`not K`** by abuse)
5. `when pred(X,Y,…)`

The clock equations

For the classes 1—4 of eqns a near-Boolean calculus applies:

- the only difference is that no top exists

Class 5 is special:
`when pred(X,Y,…)`
is a predicate that cannot be rewritten in a different form
(**`X,Y,…`** uncontrolled)

Clocks and clock equations

1. **`∅ (nil)`**; no **`“top”`**
2. **`H ^= K`**
3. **`H ^∧ K, H ^∨ K`**
4. **`H ^- K`** (**`not K`** by abuse)
5. **`when pred(X,Y,…)`**

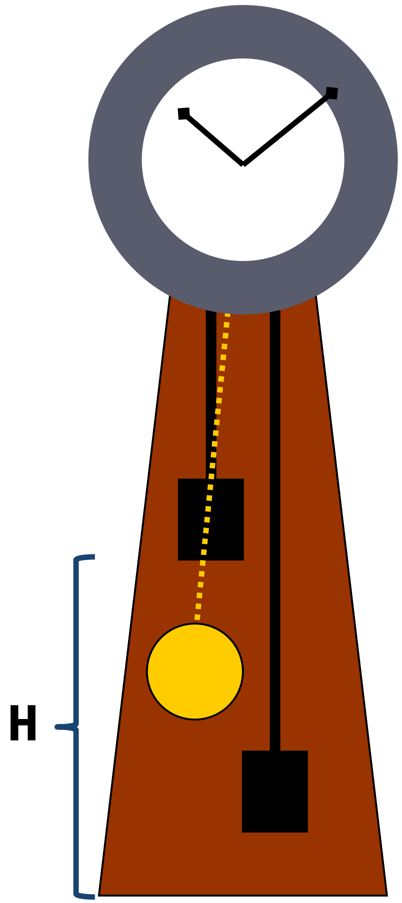
Beyond the causality calculus

Upgrading Signal to Signal+ supporting data constraints

Albert Benveniste and Thierry Gautier

June 2018

The venerable Signal+ clock




```
(  next T – T = -k * (next H - H)
| (next H = H – v) when not (H ≤ 0)
| (next H = IN) when (H ≤ 0)
)
```

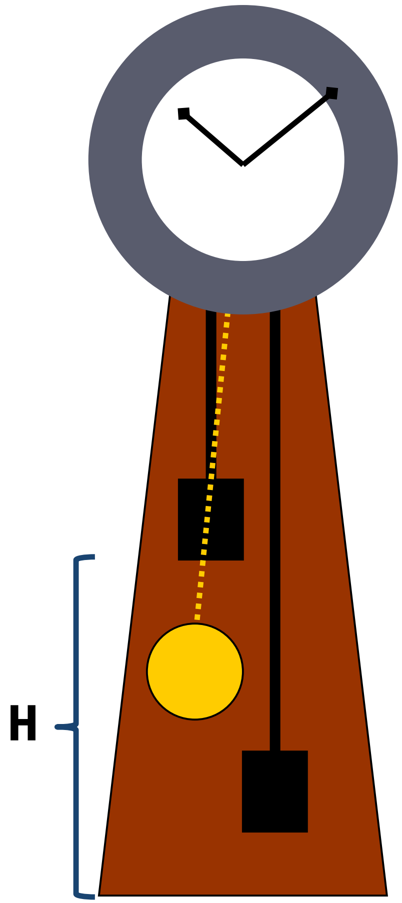
T: time

H: height of the main weight

IN: reset value for **H**

Statements: guarded equations

The venerable Signal+ clock

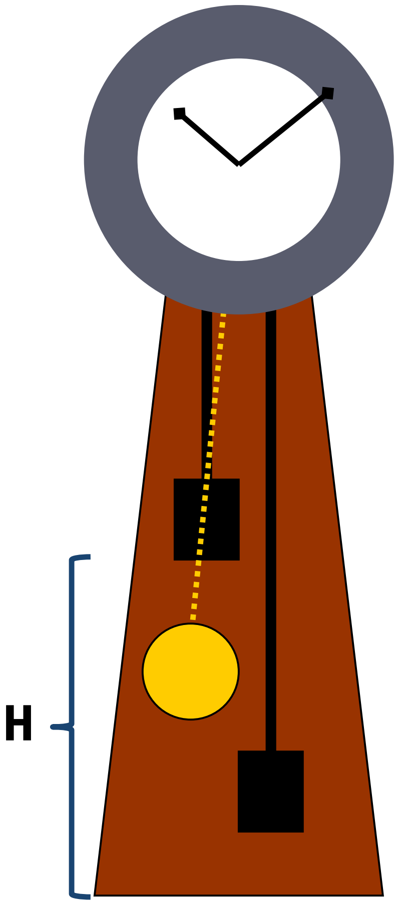



```
(   next T – T = -k * (next H - H)
| (next H = H – v) when not (H ≤ 0)
| (next H = IN) when (H ≤ 0)
)
```

Guarded equations

```
(   E1
|   E2 when not (H ≤ 0)
|   E3 when (H ≤ 0)
)
```

The venerable Signal+ clock




```
(   next T – T = -k * (next H - H)
| (next H = H – v) when not (H ≤ 0)
| (next H = IN) when (H ≤ 0)
)
```

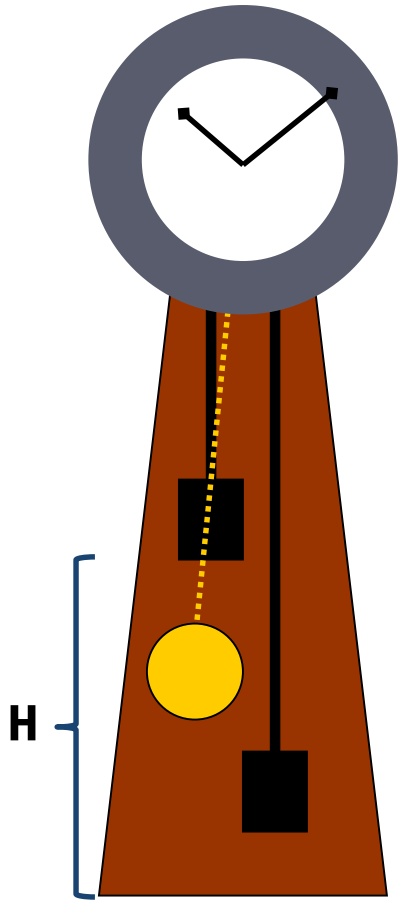
Guarded equations

```
(   E1
|   E2 when not (H ≤ 0)
|   E3 when (H ≤ 0)
)
```

Incidence graph (bi-partite, non directed)

```
(   E1 ↔ next T, next H
| (E2 ↔ next H) when not (H ≤ 0)
| (E3 ↔ next H, IN) when (H ≤ 0)
)
```

The venerable Signal+ clock




```
(   next T – T = -k * (next H - H)
| (next H = H – v) when not (H ≤ 0)
| (next H = IN) when (H ≤ 0)
)
```

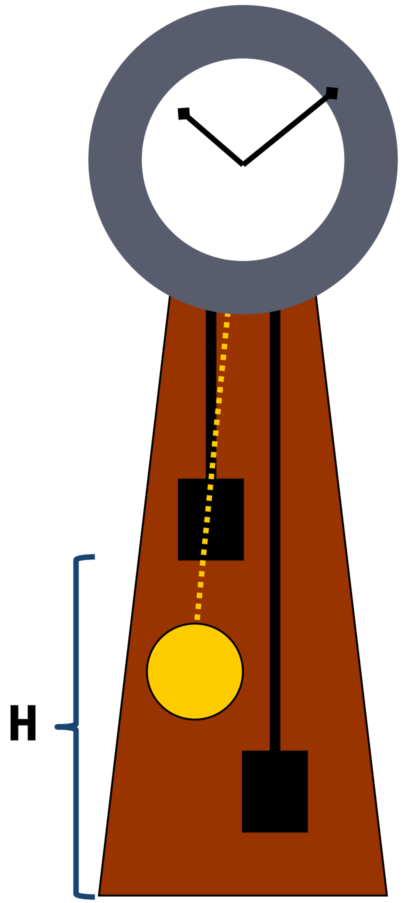
Guarded equations

```
(   E1
|   E2 when not (H ≤ 0)
|   E3 when (H ≤ 0)
)
```

Finding a guarded matching

```
(   E1 ↔ next T, next H
| (E2 ↔ next H) when not (H ≤ 0)
| (E3 ↔ next H, IN) when (H ≤ 0)
)
```

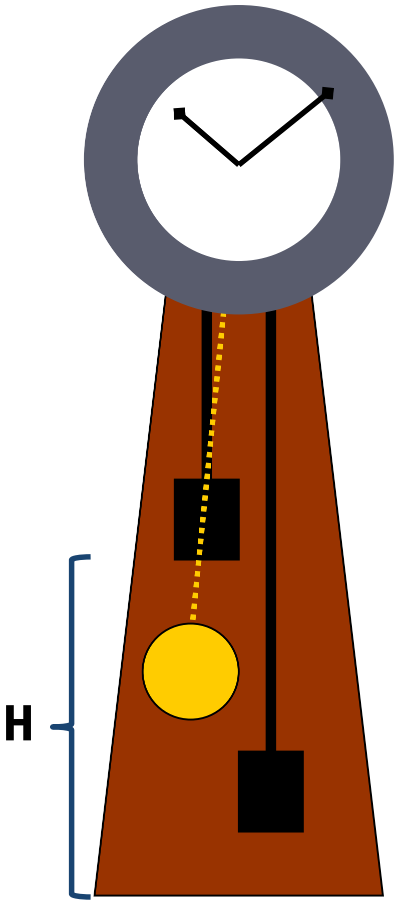
The venerable Signal+ clock



Finding a guarded matching

```
(   E1 ↔ next T, next H
| (E2 ↔ next H) when not (H ≤ 0)
| (E3 ↔ next H, IN) when (H ≤ 0)
)
```

The venerable Signal+ clock



Finding a guarded matching

```
(   E1 ↔ next T, next H
| (E2 ↔ next H) when not (H ≤ 0)
| (E3 ↔ next H, IN) when (H ≤ 0)
)
```

Yields again a scheduling

```
(   next H → E1 → next T
| (E2 → next H) when not (H ≤ 0)
| (IN → E3 → next H) when (H ≤ 0)
)
```

The rules we applied

We assumed a solver handling algebraic equations:

- ❑ **Solving system of eqns $C(x, y, z, \dots) = 0$ for $x, y, z \dots$ "scalar" variables (no tuples, no vectors)**
- ❑ **Equations possess a notion of "dimension":**
  - **if equation C=0 is itself scalar and x occurs in C, then the solver can, generically, use eqn $C = 0$ for determining x, given values for other variables**
  - **pair variables with equations defining them: $C \leftrightarrow x$**

Typical example: $x, y, z \in R$ and $C(x, y, z, \dots) = 0$ smooth

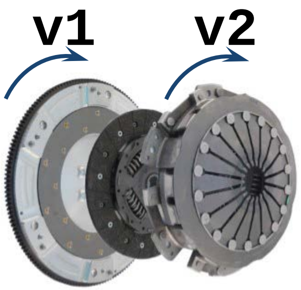
This looks like an easy generalization

HHHmmmm??????? Too easy??????

Synchronous *specification* languages are much more difficult (but also more powerful) than synchronous languages

Example of a clutch

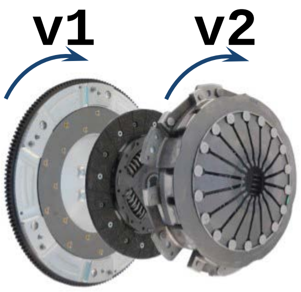
The clutch



Clutch released

```
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0)
 | (torque2 = 0)                                    )
```

The clutch



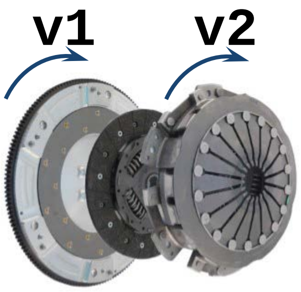
Clutch engaged

```
( next v1 = f(v1,torque1)
| next v2 = f(v2,torque2) )

|
 (
 | (v1 = v2)
 | (torque1 + torque2 = 0)                )
```

The clutch

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0) when not Engaged
 | (torque2 = 0) when not Engaged )
|
 (
 | (v1 = v2) when Engaged
 | (torque1 + torque2 = 0) when Engaged )
)
```

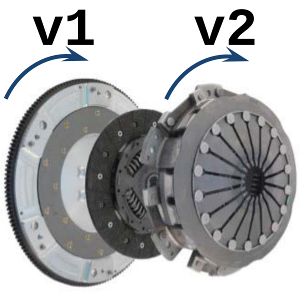


Clutch

At each reaction, the following must be evaluated from current states & inputs: `torque1, torque2, next v1, next v2`

The clutch

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0) when not Engaged
 | (torque2 = 0) when not Engaged )
|
 (
 | (v1 = v2) when Engaged
 | (torque1 + torque2 = 0) when Engaged )
)
```



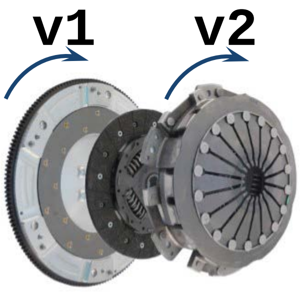
Clutch

Two problems:

- **`v1 = v2`** constrains the memories
- Engaged mode : 4 variables but only 3 equations

The clutch

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0) when not Engaged
 | (torque2 = 0) when not Engaged )
|
 ( (next v1 = next v2) when Engaged
 | (v1 = v2) when Engaged
 | (torque1 + torque2 = 0) when Engaged )
)
```



Clutch

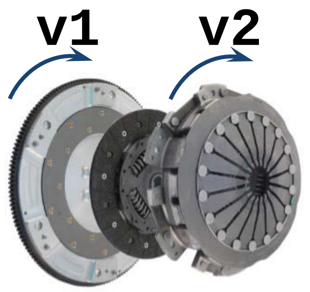
Case clutch engaged at previous reaction:

adding the blue eqn is legitimate and gives the missing equation

(*index reduction*)

The clutch

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0) when not Engaged
 | (torque2 = 0) when not Engaged )
|
 ( (next v1 = next v2) when Engaged
 | (v1 = v2) when Engaged
 | (torque1 + torque2 = 0) when Engaged )
)
```



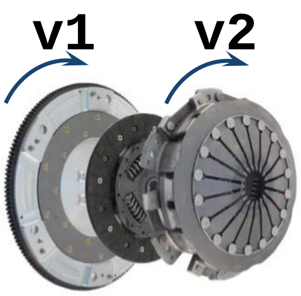
Clutch

Case clutch *not* engaged at previous reaction:

adding the blue eqn is legitimate and gives the missing equation

the green eqn is falsified

The clutch



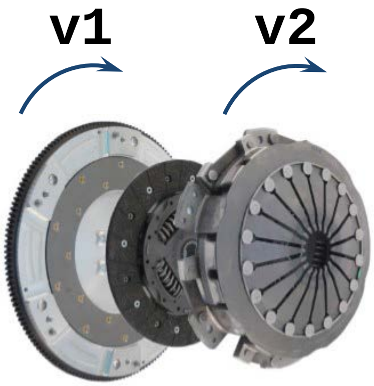
Clutch

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0) when not Engaged
 | (torque2 = 0) when not Engaged )
|
 ( (next v1 = next v2) when Engaged
 |
 | (torque1 + torque2 = 0) when Engaged )
)
```

Case clutch *not* engaged at previous reaction:

adding the blue eqn is legitimate and gives the missing equation

the green eqn is falsified: we remove it



The final code for the clutch

engaging

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (next v1 = next v2)
 | (torque1 + torque2 = 0) )
)
```

released

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (torque1 = 0)
 | (torque2 = 0) )
)
```

engaged

```
(
 ( next v1 = f(v1,torque1)
 | next v2 = f(v2,torque2) )
|
 ( (next v1 = next v2)
 | (v1 = v2)
 | (torque1 + torque2 = 0) )
)
```

Conclusion

- At our big fights Signal was deemed complex and cryptic; looking backwards, it appears simpler
- Clocks-and-causalities emerge as a very powerful framework, which can be the seed for much more…

- Synchronous Languages were developed on strong ideological bases; it even turned to true radicalization
- So many of these ideas are more and more fertile and so many areas need them desperately…

Thanks to Nicolas and remember Paulo…

INVENTEURS DU MONDE NUMÉRIQUE