TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

From clock driven to Data-driven models

Y. Bai and K. Schneider and N. Bhardwaj and B. Katti and T. Shazadi

Formal Methods and Models for Co-design

(MEMOCODE'14)

Motivation

- Synchronous systems:
  - bound to a clock – i.e. clock dependent → **clock constraint**
  - expected to do 0 time computation but NOT
  - they're time consuming – read - compute – write - takes time → **timing constraint**
- These two constraints are not good for distributed systems

Outline

- Quartz – the synchronous language
- Synchronous systems
  - Clock and timing constraint
  - Example – Seq ITE
- Desynchronization
  - Asynchronous systems
  - Box values □
  - Example – Seq ITE
- Endochrony
  - Endochronous systems
  - No clock yet determinstc
  - Symbolic representation
  - Proof: Examples – Seq ITE is endo!
- Verification and SAT

Core Idea

- **independent of clock** yet **deterministic**
- **transformations** on the synchronous systems
- **verify** whether or not it is **Endochronous !!**
- For verification, we have reduced the problem to SAT!
- **Quartz** – ***clocked guarded actions*** we describe the system such that it is reduced to a ***boolean expression*** whose SATisfiability check defines the presence of **Endochrony!**

Quartz – The programming Language

- asynchronous parallel execution of threads
- explicit implementation of non determinism
- *delayed data assignments* (next(λ) = T)
- *guarded actions*
- boolean expression → SAT solvability!

```
module site(clocked bool ?x1,?x2,?x3,!y) {
   loop{
      if(clk(x1) & clk(x2) & x1) {
         y = (x2,true);
      }
      if(clk(x1) & clk(x3) & !x1) {
         y = (x3,true);
      }
      pause;
   }
}
```

Seq ITE program in Quartz

```
clk(x1)& clk(x2)& x1 -> y=(x2, true)
clk(x1)& clk(x3)& !x1 -> y=(x3, true)
```

Clocked guarded actions

Synchronous systems

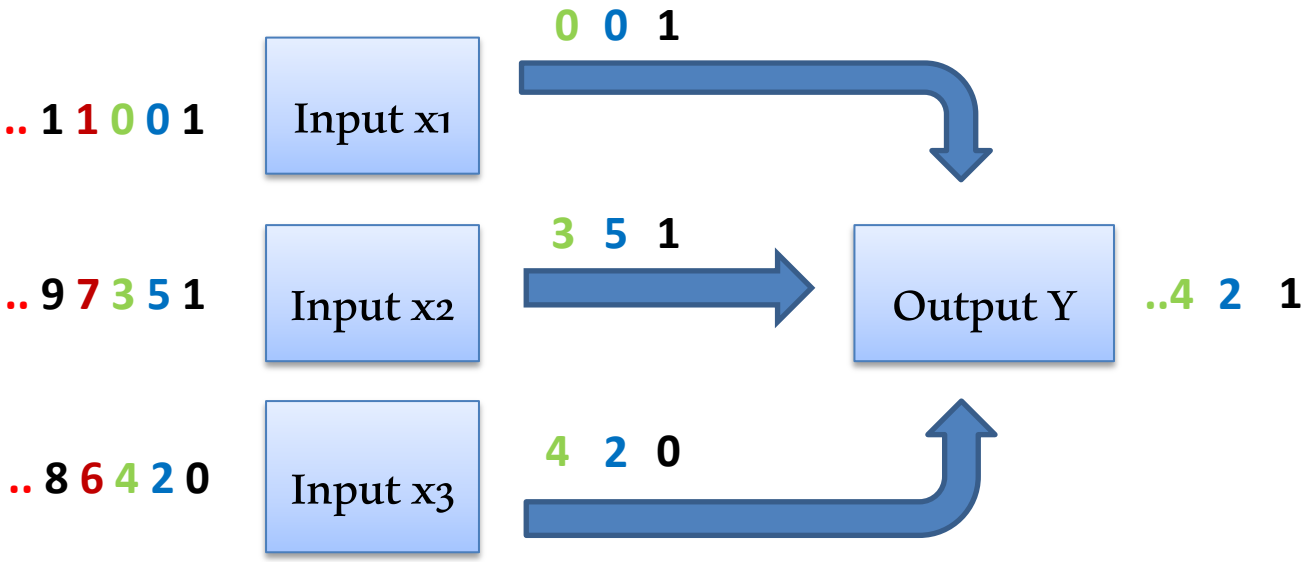
- Clock driven
  - +ve → **Deterministic!**
  - -ve –**> dependent & inefficient**
- Slow clock
- Solution: *Elastic systems* → the communication wires are replaced by buffers
  - +ve → **improves worst case execution time**
  - **-**ve → **still clock driven**

Synchronous systems

- Example Seq ITE

x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	$(c :: C)$	$[b]$
$(0 :: A)$	$(b :: B)$	$(c :: C)$	$[c]$

$\xi(x_1)$:	1	0	0	1	1	...
$\xi(x_2)$:	1	3	5	7	9	...
$\xi(x_3)$:	0	2	4	6	8	...
$\xi(y)$:	1	2	4	7	9	...



Introducing ⊡

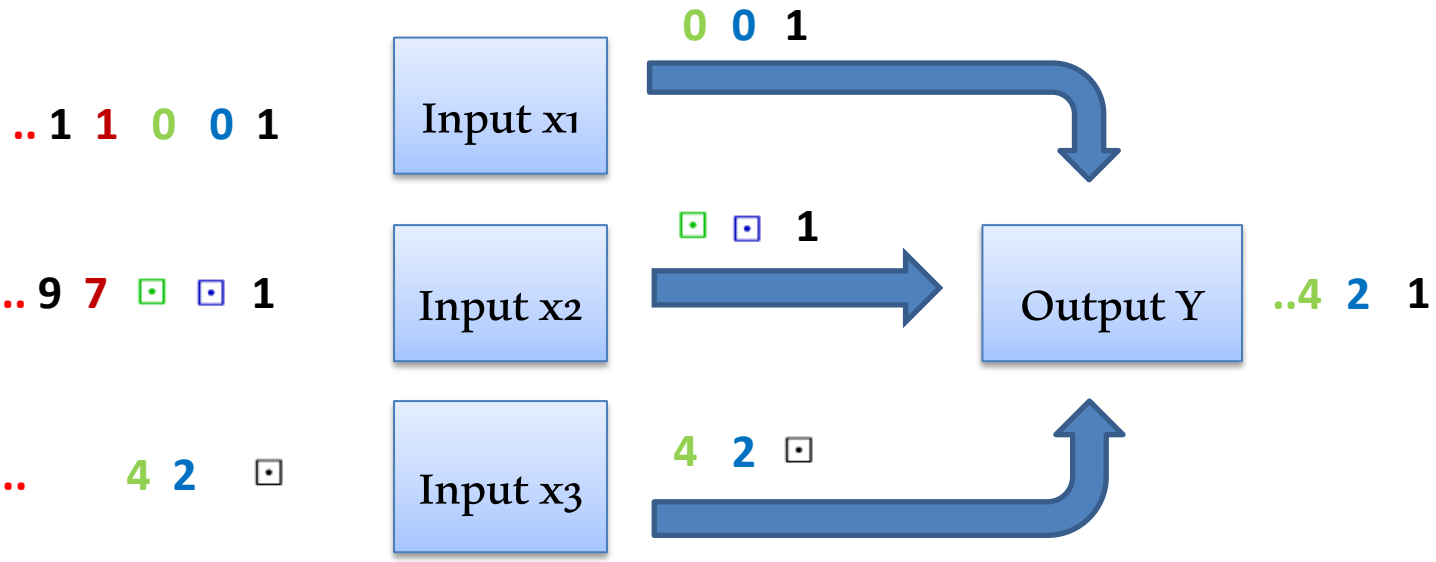
- In Seq ITE there are inputs that consume values even when they're not needed → waste of energy and time!
- Replace these values by ⊡
- ⊡ are the values which are not required for computation but for ***alignment of input*** streams
- System is ***still working on a clock***!

Introducing ⊡

- Seq ITE – aligned input stream BUT with ⊡

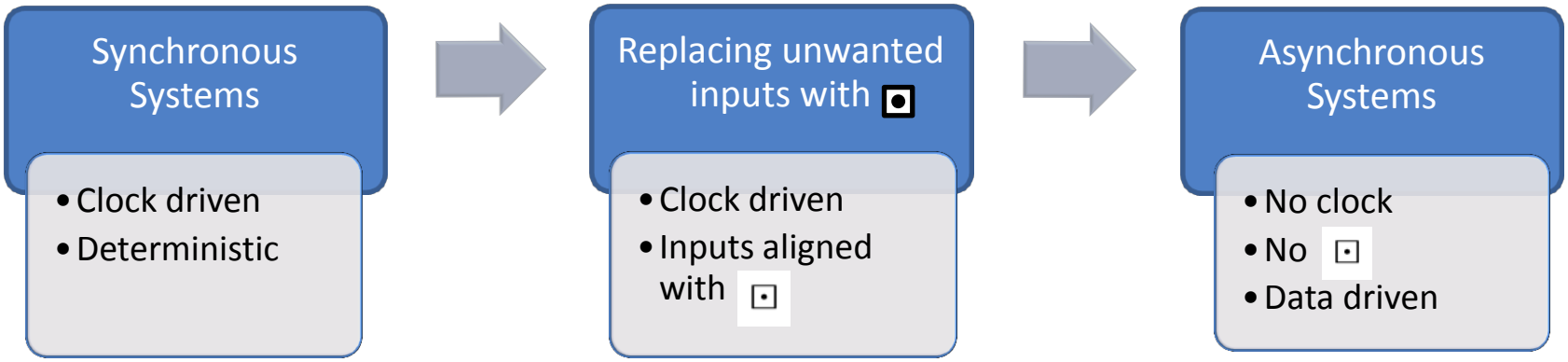
x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	$(\boxdot :: C)$	$[b]$
$(0 :: A)$	$(\boxdot :: B)$	$(c :: C)$	$[c]$

$\xi(x_1)$:	1	0	0	1	1	…
$\xi(x_2)$:	1	⊡	⊡	7	9	…
$\xi(x_3)$:	⊡	2	4	⊡	⊡	…
$\xi(y)$:	1	2	4	7	9	…



Desynchronization

- Finally, we remove ⊡
- Streams are desynchronized
- Nodes now have to ***resynchronize*** their inputs

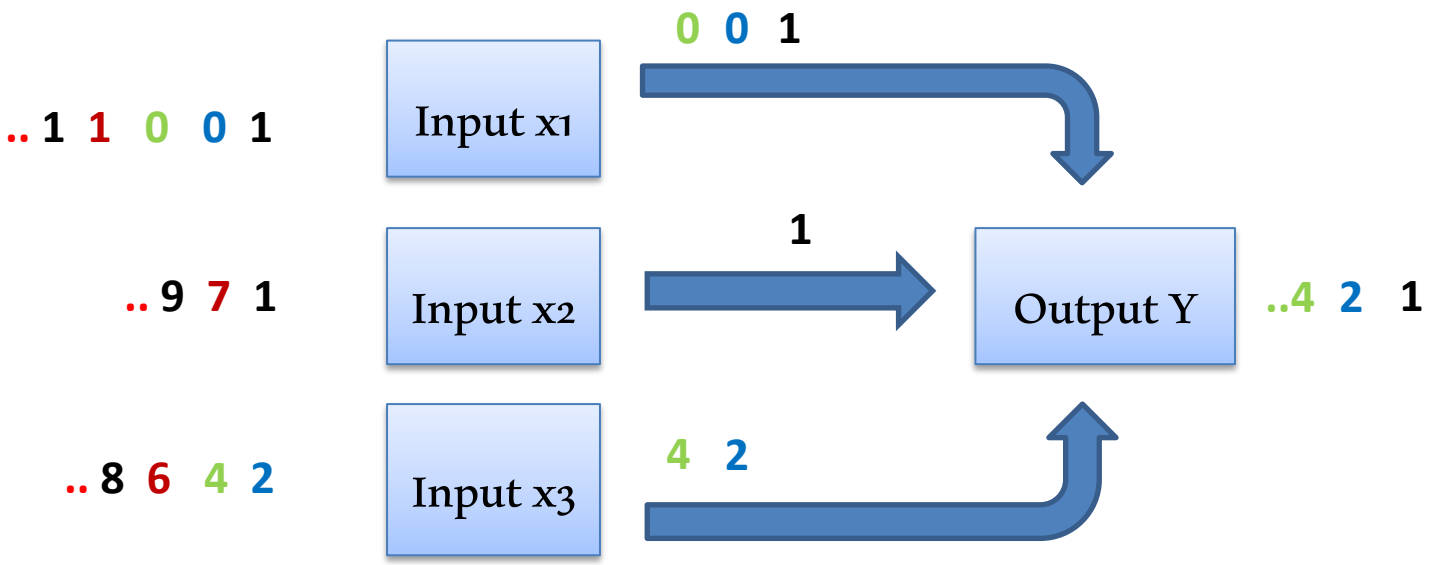


Desynchronization

- Seq ITE – no more ⊡

x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	C	$[b]$
$(0 :: A)$	B	$(c :: C)$	$[c]$

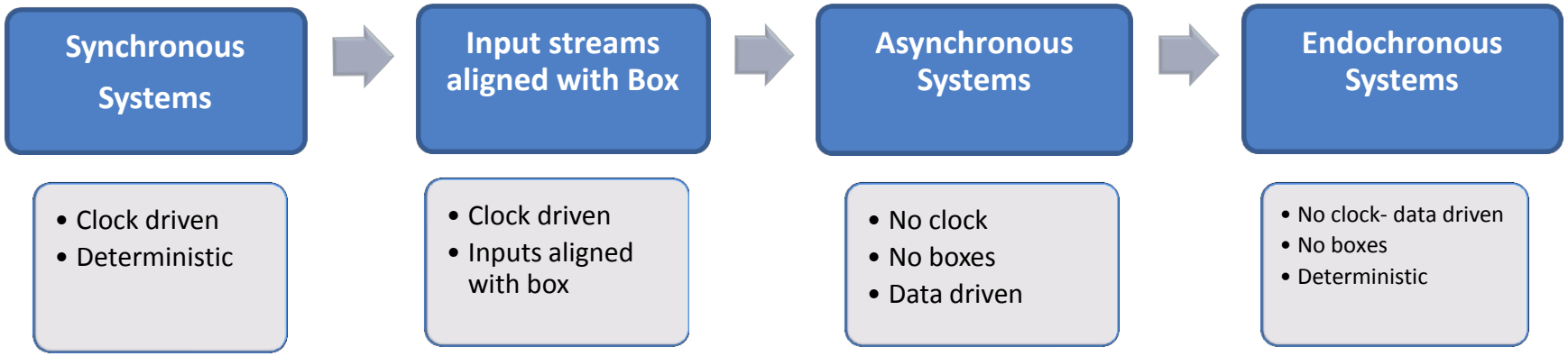
$\xi(x_1)$:	1	0	0	1	1	...
$\xi(x_2)$:	1	7	9	...		
$\xi(x_3)$:	2	4	...			
$\xi(y)$:	1	2	4	7	9	...



Endochronous systems

- If the asynchronous system obtained after desynchronization is able to:
  - Resynchronize its input stream
  - Generate same output as parent – synchronous – system
  - Exhibit *constructiveness*
  - Maintains determinism

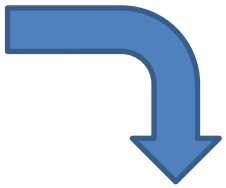
→ System possesses endochrony!



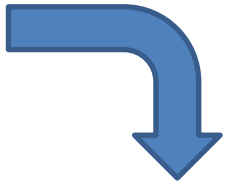
Endochronous systems

- But not all systems are endochronous!
- Example – Par ITE

x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	$(c :: C)$	$[b]$
$(0 :: A)$	$(b :: B)$	$(c :: C)$	$[c]$
$(a :: A)$	$(b :: B)$	$(b :: C)$	$[b]$



x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	$(\boxdot :: C)$	$[b]$
$(0 :: A)$	$(\boxdot :: B)$	$(c :: C)$	$[c]$
$(\boxdot :: A)$	$(b :: B)$	$(b :: C)$	$[b]$



x_1	x_2	x_3	y
$(1 :: A)$	$(b :: B)$	C	$[b]$
$(0 :: A)$	B	$(c :: C)$	$[c]$
A	$(b :: B)$	$(b :: C)$	$[b]$

Endochronous systems

For input stream – **0 0 1 1 1 0…**

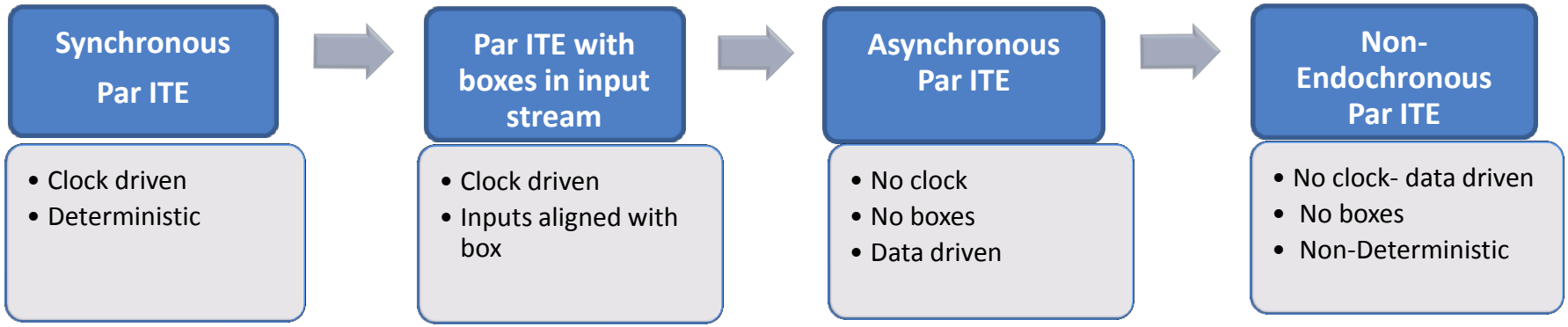
Output stream – 1 2 1 2 4 3 … ≠ Output stream – 1 2 3 2 4 6 5 …

$\varrho(x_1)$:	0	0	1	1	1	0	…
$\varrho(x_2)$:	1	2	4	6	8	10	…
$\varrho(x_3)$:	1	2	3	5	7	9	…
$\varrho(y)$:	1	2	1	2	4	3	…

$\varrho(x_1)$:	0	0	1	1	1	0	…
$\varrho(x_2)$:	1	2	4	6	8	10	…
$\varrho(x_3)$:	1	2	3	5	7	9	…
$\varrho(y)$:	1	2	3	2	4	6	5 …

- Not *Latency-insensitive*
- Not even a function
- Not deterministic

Endochronous systems

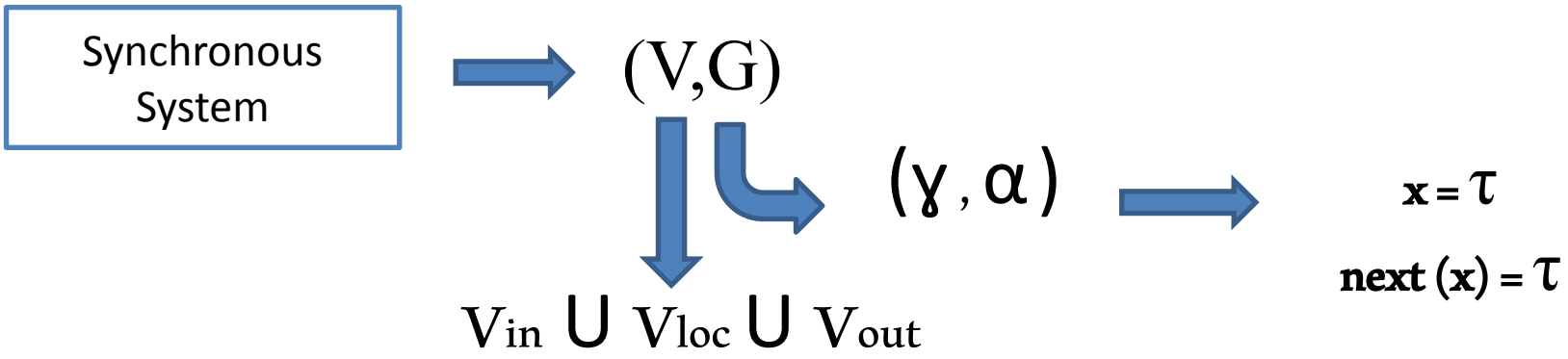


→ **Not all systems** are Endochronous

→ **Some need clock** for deterministic output!

Verification

- **Step 1:** Describing synchronous systems



- For every x, clock **clk(x)** - such that for all points of time **clk(x)** = true iff $x \neq \boxdot$

```
if(clk(x1) & clk(x2) & x1)  ⟶  clk(x1)& clk(x2)& x1 –> y=(x2, true)
```

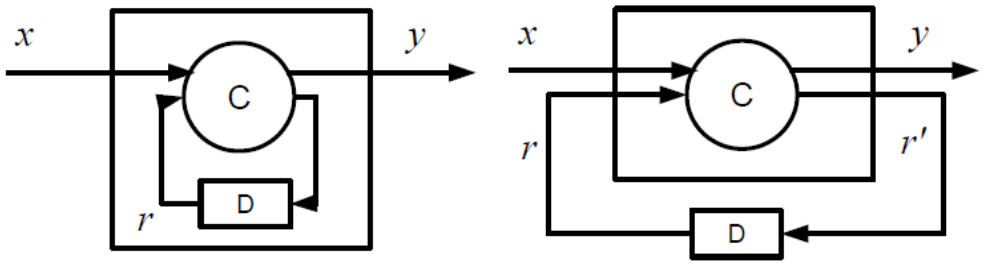
Verification

- Transition relation of the system:

$$\mathcal{R} :\Leftrightarrow \left(\bigwedge_{x\in\mathcal{V}_{loc}\cup\mathcal{V}_{out}}(\mathcal{T}_x \wedge \mathcal{C}_x) \wedge \bigvee_{x\in\mathcal{V}_{loc}\cup\mathcal{V}_{out}} \mathcal{B}_x\right)$$

$$\mathcal{R}_{\|} :\Leftrightarrow (\mathcal{R} \wedge \mathsf{next}(\mathcal{R})).$$

- **Step 2:** Achieving stateless components with $V_{loc} = \{\}$
  - next(x) = τ **=>** x' = τ



Verification

- **Step 3:** ⊡ are no longer communicated
  - **Clock** is a ***schedule instruction***
  - No computation is lost
  - Transition relation of the system:

$$\mathcal{R}_{\mathsf{buf}} :\Leftrightarrow \mathcal{R}_{\|} \wedge \mathsf{BC}$$

$$\mathsf{BC} :\Leftrightarrow \forall x \in \mathcal{V}_{in}.\ (\neg \mathsf{clk}(x) \rightarrow \mathsf{next}(x) = x)$$

- **Step 4:** Data Driven components
  - Transition relation of the system:

$$\mathcal{R}_{\mathsf{DPN}} :\Leftrightarrow \exists \mathsf{clk}(\vec{x}), \mathsf{clk}(\vec{x'}).\mathcal{R}_{\mathsf{buf}} :\Leftrightarrow \exists \mathsf{clk}(\vec{x}), \mathsf{clk}(\vec{x'}), .\mathcal{R}_{\|} \wedge \mathsf{BC}$$

Verification

- Final ***stateless component*** with R_{buf} :

$$\forall s_1, s_2 \in \mathcal{R}_{\mathsf{buf}}.\ \forall x \in \mathcal{V}_{in}.$$
$$s_1(x) = s_2(x) \rightarrow s_1(\mathsf{clk}(x)) = s_2(\mathsf{clk}(x))$$

  - Quantify over states
  - SAT checks the validity – generates number of clauses!
  - BDD based solver – records the size of the BDDs!
  - If BDD = true node, system is **Endochronous!**

Thank you!