

Developing temporal abstract domains that prove the temporal specifications of reactive systems

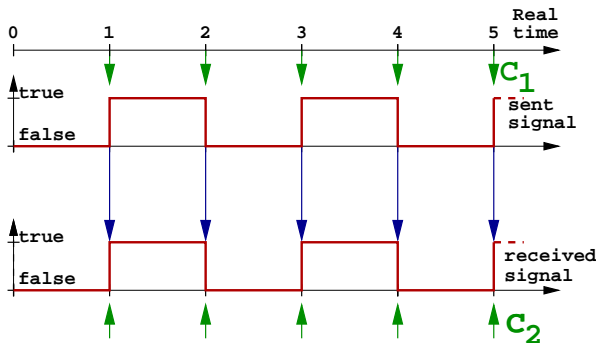
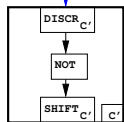
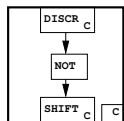
Julien Bertrane
julien@sei.ecnu.edu.cn
bertrane@di.ens.fr

ECNU - ENS

September 23rd, 2009

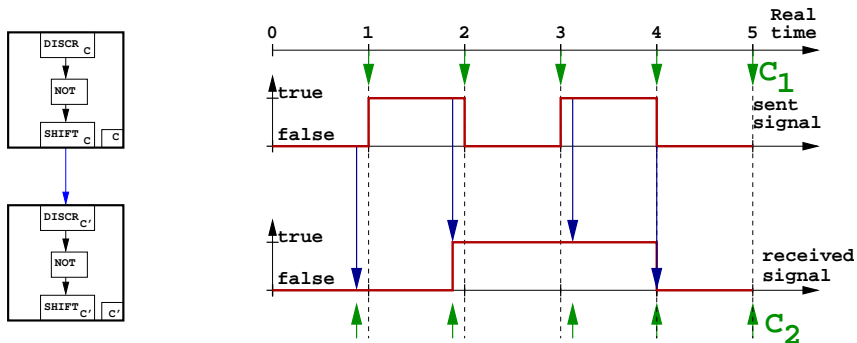
Introduction

Constraints from the studied cases



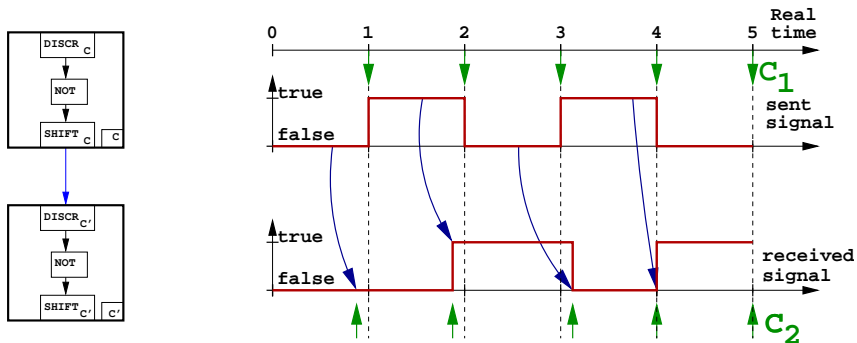
- Some unavoidable yet not so often modeled hardware imperfections :

Constraints from the studied cases



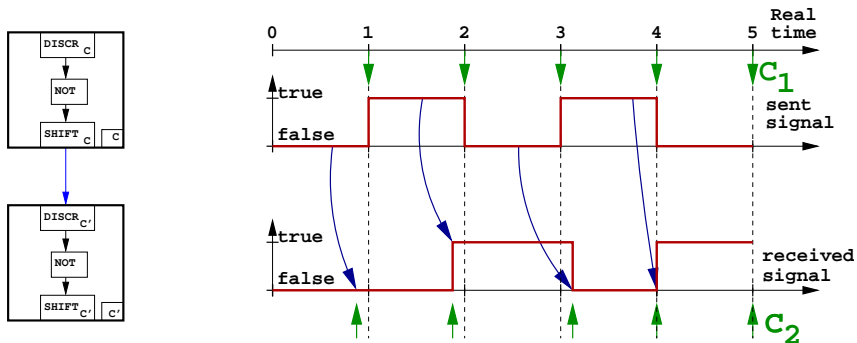
- Some unavoidable yet not so often modeled hardware imperfections :
 - The clocks of the control units may **de-synchronize**

Constraints from the studied cases



- Some unavoidable yet not so often modeled hardware imperfections :
 - ▶ The clocks of the control units may **de-synchronize**
 - ▶ The communications **cannot always** be considered **instantaneous**

Constraints from the studied cases

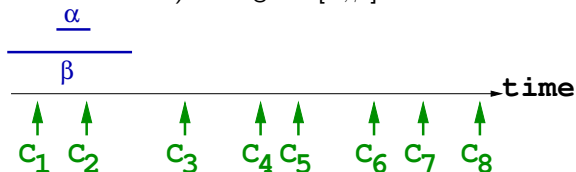


- Some unavoidable yet not so often modeled hardware imperfections :
 - ▶ The clocks of the control units may **de-synchronize**
 - ▶ The communications **cannot always** be considered **instantaneous**
 - ▶ The communication delays **cannot always** be considered **constant**

Hypotheses for the semantics

- **Imperfect-synchrony** :

- ▶ De-synchronization : the length of each clock *cycle* (period between two clock **ticks**) belongs to $[\alpha, \beta]$.

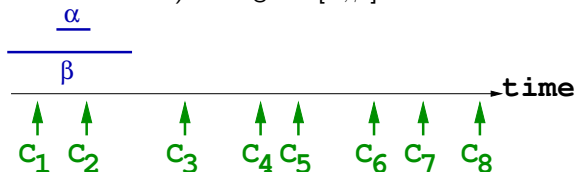


- ▶ A clock c satisfies $[\alpha, \beta]$ iff $c_{n+1} - c_n \in [\alpha, \beta]$

Hypotheses for the semantics

- **Imperfect-synchrony** :

- ▶ De-synchronization : the length of each clock *cycle* (period between two clock **ticks**) belongs to $[\alpha, \beta]$.



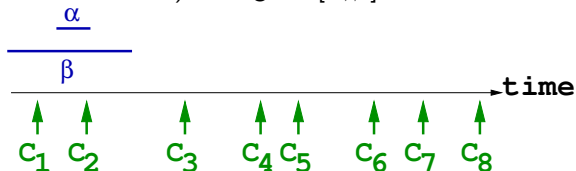
- ▶ A clock c satisfies $[\alpha, \beta]$ iff $c_{n+1} - c_n \in [\alpha, \beta]$

- **Serial** transmissions between imperfectly synchronous systems

Hypotheses for the semantics

- **Imperfect-synchrony** :

- ▶ De-synchronization : the length of each clock *cycle* (period between two clock **ticks**) belongs to $[\alpha, \beta]$.



- ▶ A clock c satisfies $[\alpha, \beta]$ iff $c_{n+1} - c_n \in [\alpha, \beta]$

- **Serial** transmissions between imperfectly synchronous systems
- **Blackboard** at each unit inputs

Verifying specifications

- **Safety** specifications
 - ▶ for any behavior s , at any moment t , $s(t) \neq true$

Verifying specifications

- **Safety** specifications

- ▶ for any behavior s , at any moment t , $s(t) \neq true$

- **Temporal** specifications

- ▶ for any behavior s , there is no time t such that :

for any $t' \in [t, t + \alpha]$, $s(t') = true$

Verifying specifications

- **Safety** specifications

- ▶ for any behavior s , at any moment t , $s(t) \neq true$

- **Temporal** specifications

- ▶ for any behavior s , there is no time t such that :

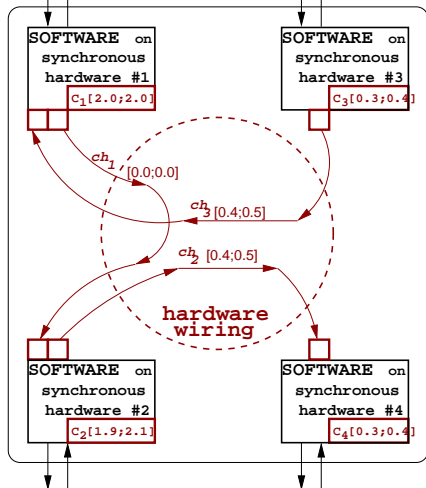
for any $t' \in [t, t + \alpha]$, $s(t') = true$

- **Quantitative** specifications

- ▶ The **outputs** of 2 redundant systems remain **equal at least 75% of the time** during any time interval of width at least δ .

Typical system to verify : quantifying the hardware imperfections

HARDWARE (environment, sensors, actuators)

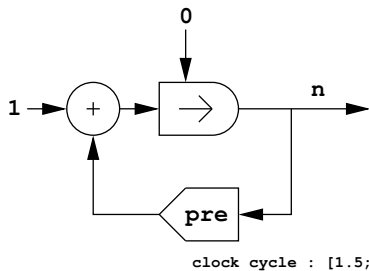


HARDWARE (environment, sensors, actuators)

Semantics

Which semantics ?

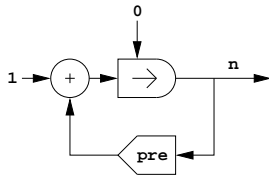
Synchronous programs can be compiled to C



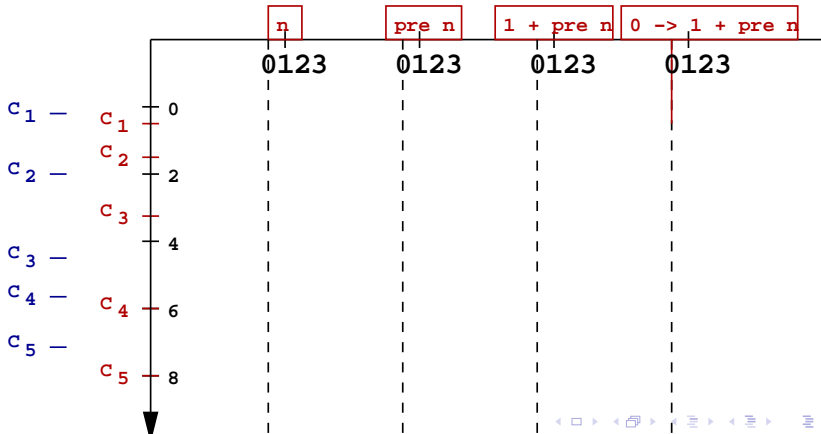
```
while true{
  switch (global_state->current_state){
  case 0:
    global_state->n = 0;
    global_state->current_state = 1;
    break;

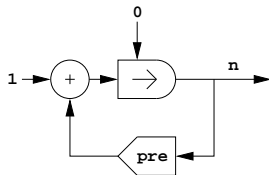
  case 1:
    global_state->n = (global_state->n)+ 1;
    global_state->current_state = 1;
    break;}}
```

- In C, there is no notion of verifiable guarantee on execution-time (it depends on hardware, optimizations...)

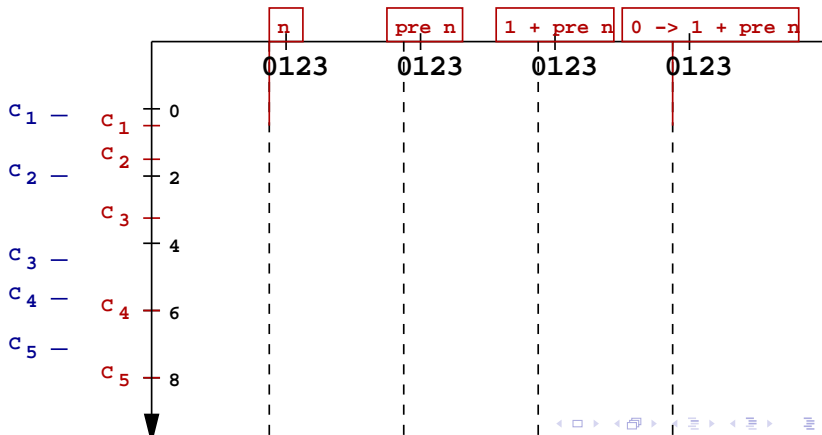


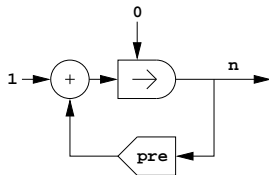
clock cycle : 1.5-2.5 ms



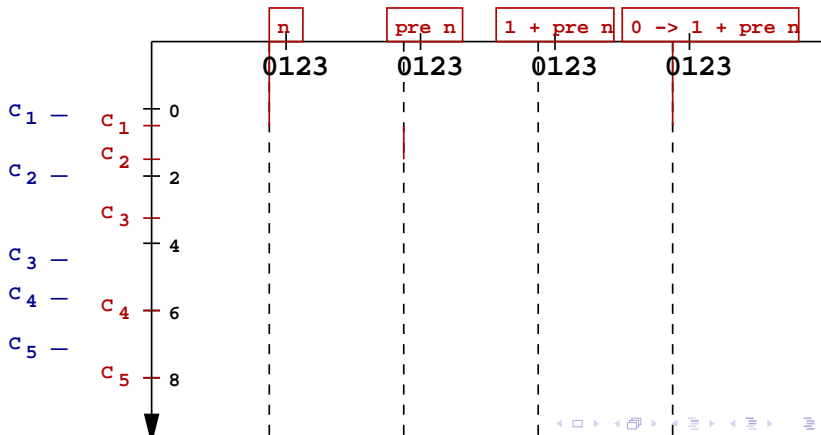


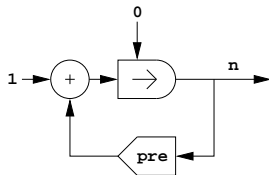
clock cycle : 1.5-2.5 ms



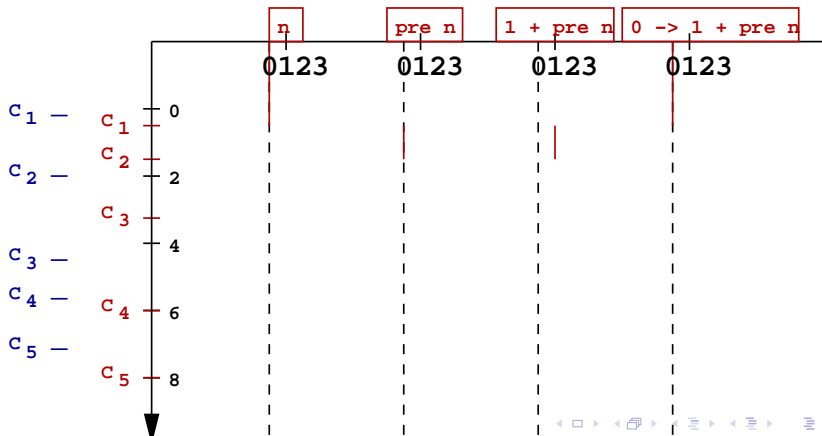


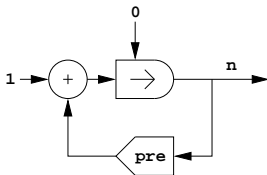
clock cycle : 1.5-2.5 ms



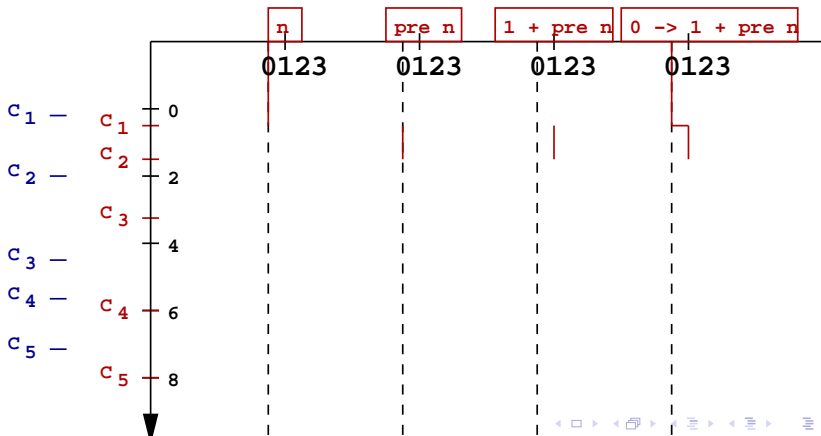


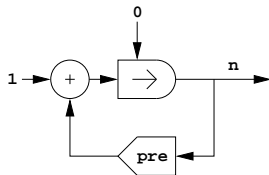
clock cycle : 1.5-2.5 ms



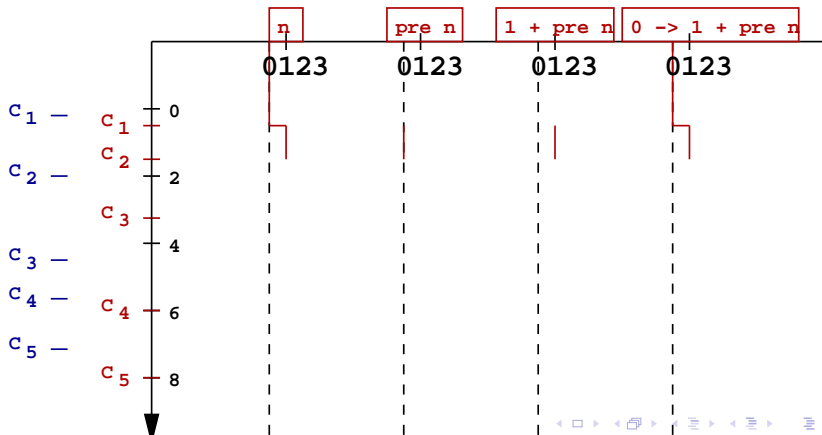


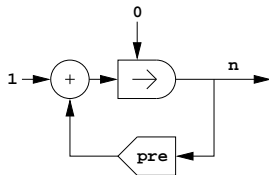
clock cycle : 1.5-2.5 ms



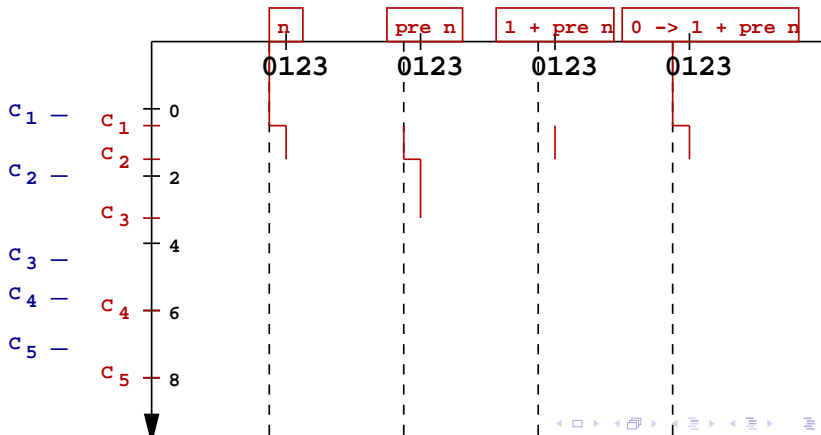


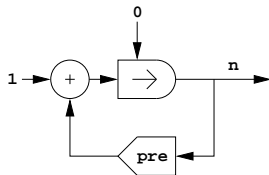
clock cycle : 1.5-2.5 ms



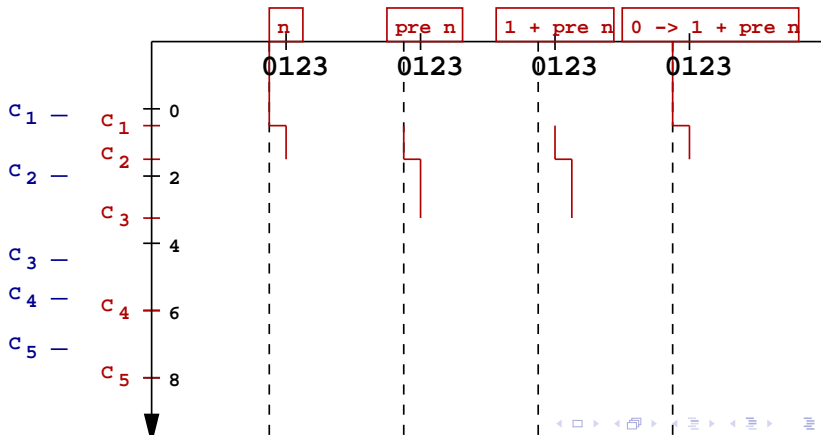


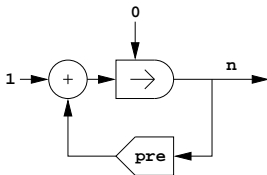
clock cycle : 1.5-2.5 ms



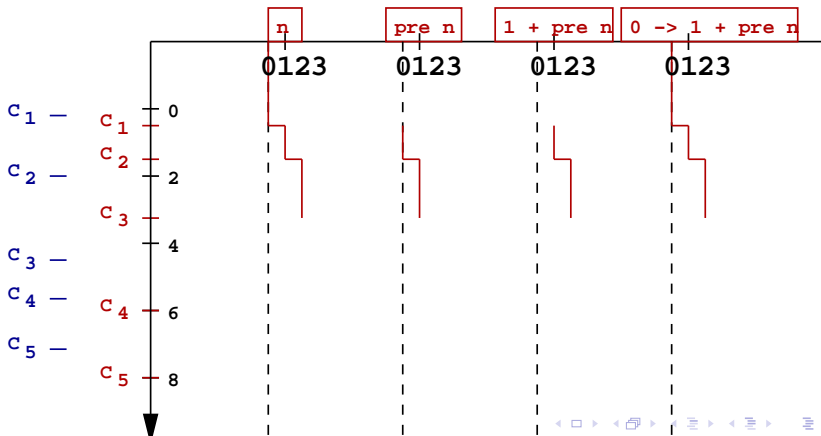


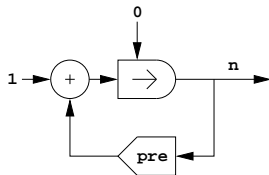
clock cycle : 1.5-2.5 ms



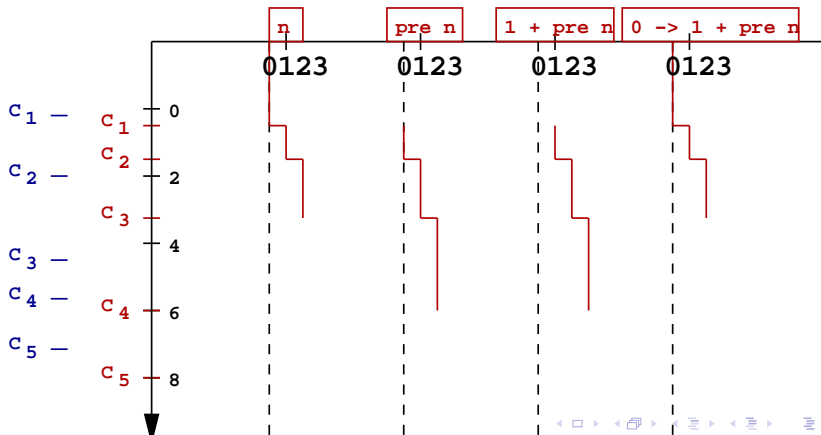


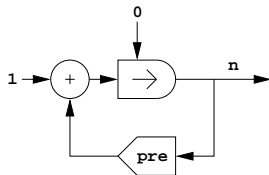
clock cycle : 1.5-2.5 ms



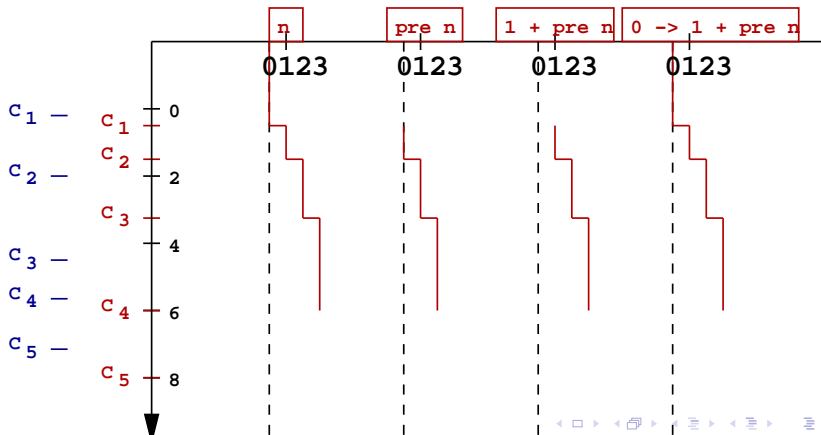


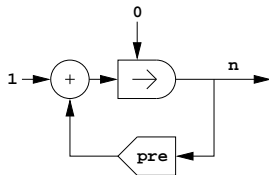
clock cycle : 1.5-2.5 ms



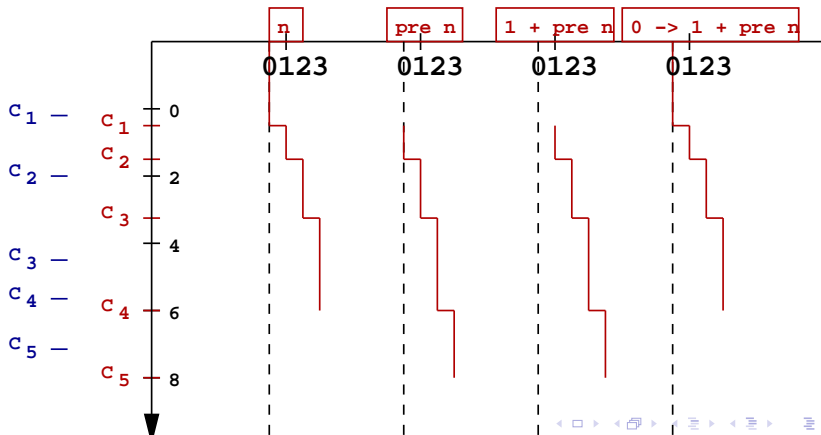


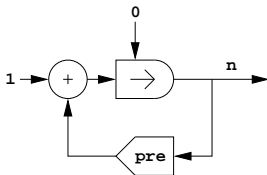
clock cycle : 1.5-2.5 ms



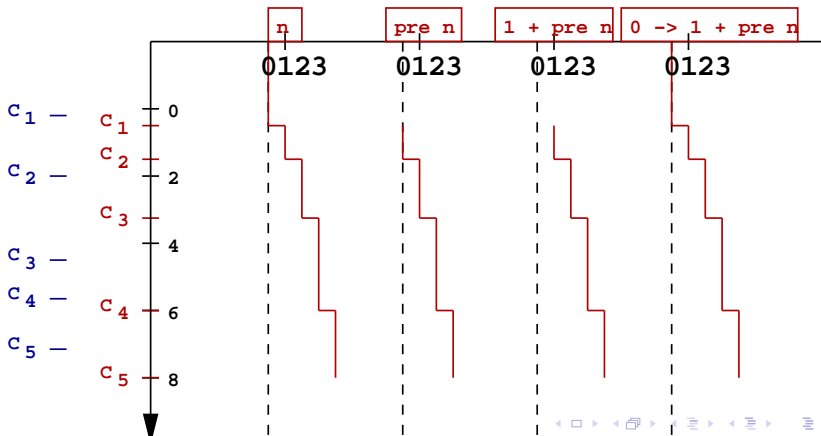


clock cycle : 1.5-2.5 ms

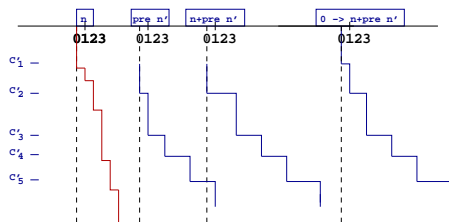
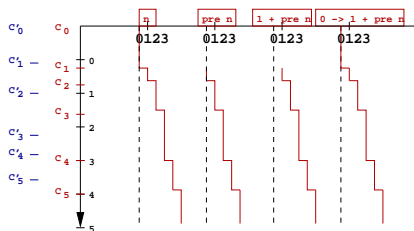
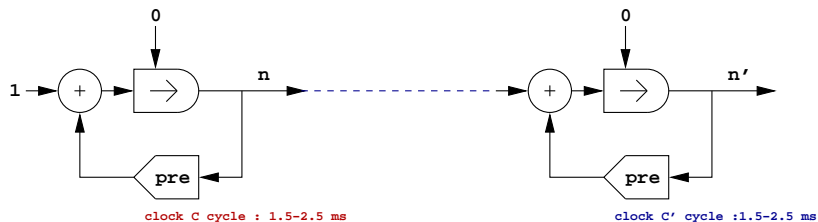




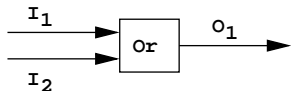
clock cycle : 1.5-2.5 ms



Toward a non-standard semantics ?

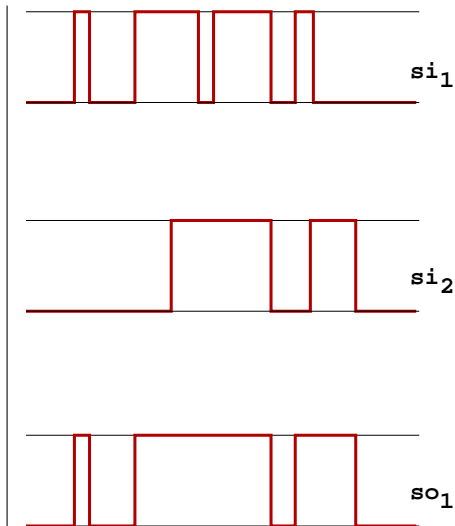


Semantics of non-temporal operators

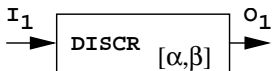


$$so_1(t) = \begin{cases} \bullet \text{ true} \\ \text{if } si_1(t) = \text{true} \\ \text{or } si_2(t) = \text{true} \\ \bullet \text{ false} \text{ else} \end{cases}$$

$$so_1 \triangleq \Psi_{OR}(si_1, si_2)$$



Semantics of temporal operators

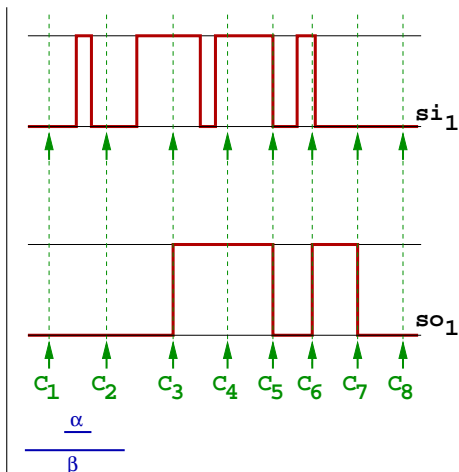


There exists a clock C of
parameter $[\alpha, \beta]$

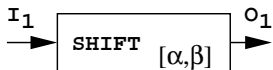
i.e. : $c_{n+1} - c_n \in [\alpha, \beta]$ such that

$$so_1(t) = \begin{cases} \bullet \text{ false} & \text{if } t < c(0) \\ \bullet \text{ } si_1(c_n) & \text{if } t \in [c_n, c_{n+1}) \end{cases}$$

$$so_1 \triangleq \Psi_{DISCR_{[\alpha, \beta]}}(si_1)$$



Semantics of temporal operators

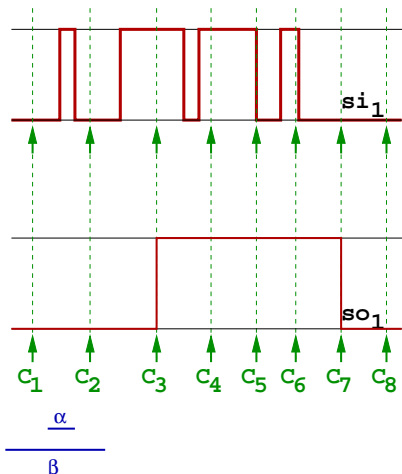


There exists a clock C of parameter $[\alpha, \beta]$

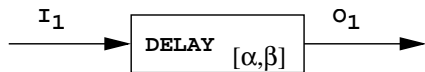
i.e. : $c_{n+1} - c_n \in [\alpha, \beta]$ such that

$$so_1(t) = \begin{cases} \bullet \text{ false} & \text{if } t < c(0) \\ \bullet \lim_{\substack{t \rightarrow c_n \\ t < c_n}} si_1(t) & \text{if } t \in [c_n, c_{n+1}) \end{cases}$$

$$so_1 \triangleq \Psi_{\text{SHIFT}_{[\alpha, \beta]}}(si_1)$$



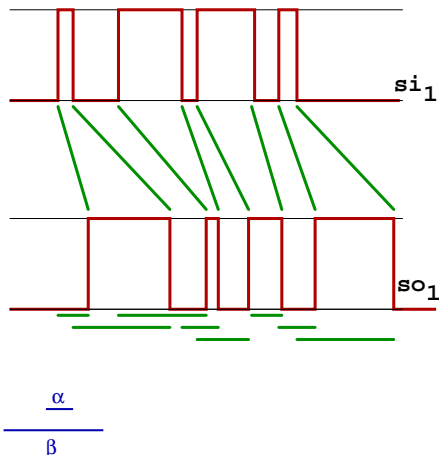
Semantics of temporal operators



There exists a function δ
 increasing bijection of $\mathbb{R} \rightarrow \mathbb{R}$
 of parameter $[\alpha, \beta]$ i.e. :
 $\forall t \in \mathbb{R}, \delta(t) - t \in [\alpha, \beta]$ such that

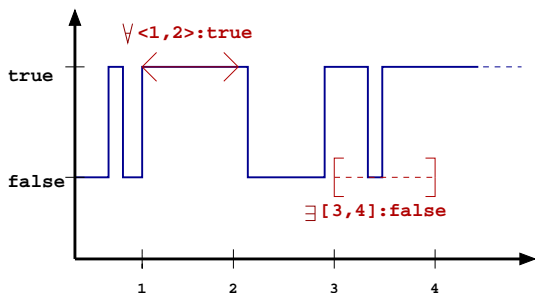
$$so_1(\delta(t)) = si_1(t)$$

$$so_1 \triangleq \Psi_{\text{DELAY}_{[\alpha, \beta]}}(si_1)$$



Temporal abstract domains

1st abstract domain : constraints



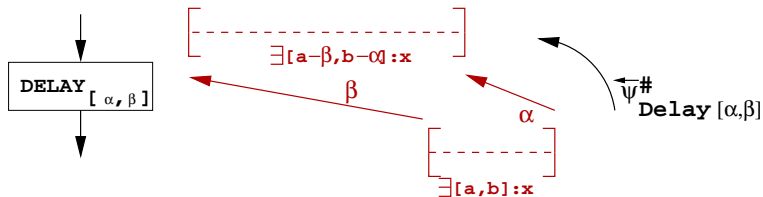
- the constraint $\exists[a; b] : x$ asserts that the signals take the value x **at least once** pendant $[a; b]$.
- the constraint $\forall \langle a; b \rangle : x$ asserts that the signals take the value x **during any interval** $[a; b]$.

1st abstract domain : constraints

- Allows the expression of numerous **temporal properties**
- the signal s takes the value *true* during more than 3 seconds without the alarm a being raised less than 1 second later :

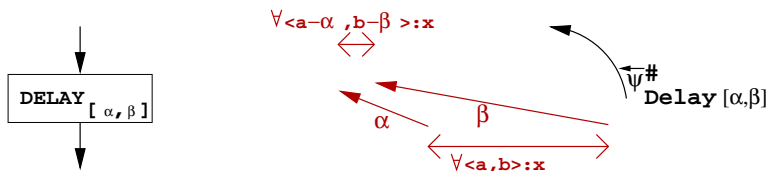
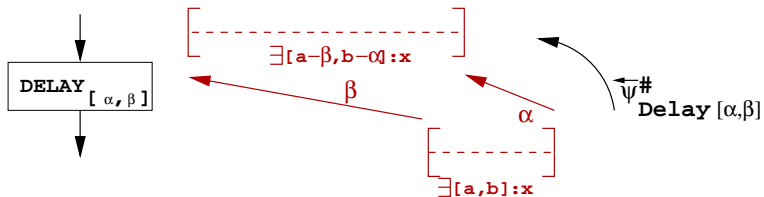
$$s : \forall \langle \delta; \delta + 3 \rangle : \text{True} \wedge a : \forall \langle \delta + 3; \delta + 4 \rangle : \text{False}$$

Abstract Operators and Constraints : example



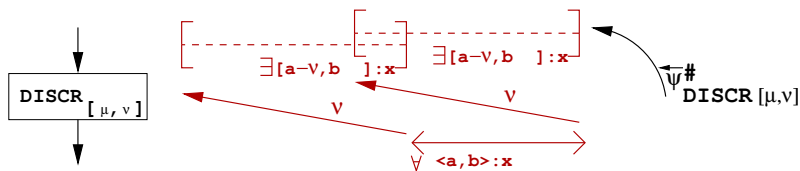
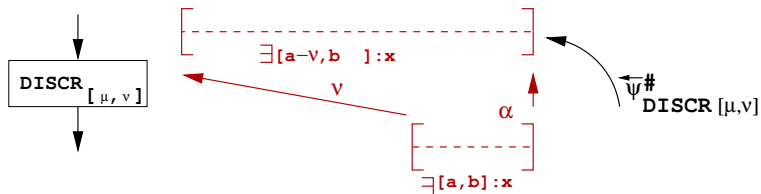
- $\overleftarrow{\Psi}^{\#}_{\text{DELAY}[\alpha, \beta]}(\exists[a; b] : x) \triangleq \exists[a - \beta; b - \alpha] : x$

Abstract Operators and Constraints : example



- $\overleftarrow{\Psi}^{\#}_{\text{DELAY}[\alpha, \beta]}(\exists[a; b] : x) \triangleq \exists[a - \beta; b - \alpha] : x$
- $\overleftarrow{\Psi}^{\#}_{\text{DELAY}[\alpha, \beta]}(\forall\langle a; b \rangle : x) \triangleq \forall\langle a - \alpha; b - \beta \rangle : x$

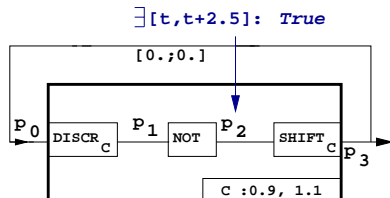
Abstract Operators and Constraints : example



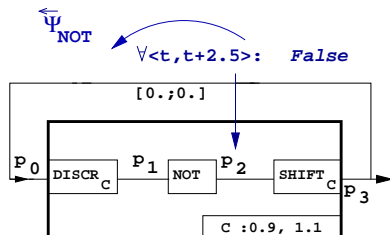
- $\overleftarrow{\Psi}^{\#}_{\text{DISCR}[\mu, \nu]}(\exists[a; b] : x) \triangleq \exists[a - \nu; b] : x$
- $\overleftarrow{\Psi}^{\#}_{\text{DISCR}[\mu, \nu]}(\forall \langle a; b \rangle : x) \triangleq \bigwedge_{u \in [a, b]} \exists[u - \nu; u] : x$

Analysis inside the abstract domain of **Constraints**

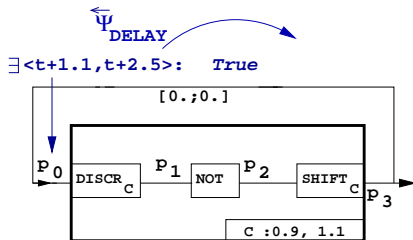
- Proving the following abstract property.



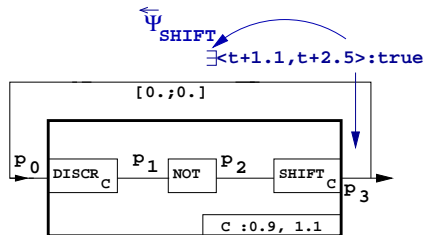
Analysis inside the abstract domain of **Constraints**



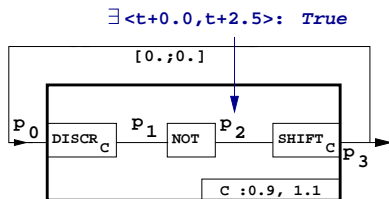
Analysis inside the abstract domain of **Constraints**



Analysis inside the abstract domain of **Constraints**



Analysis inside the abstract domain of **Constraints**



At point p_2 : 2 constraints have to be satisfied :

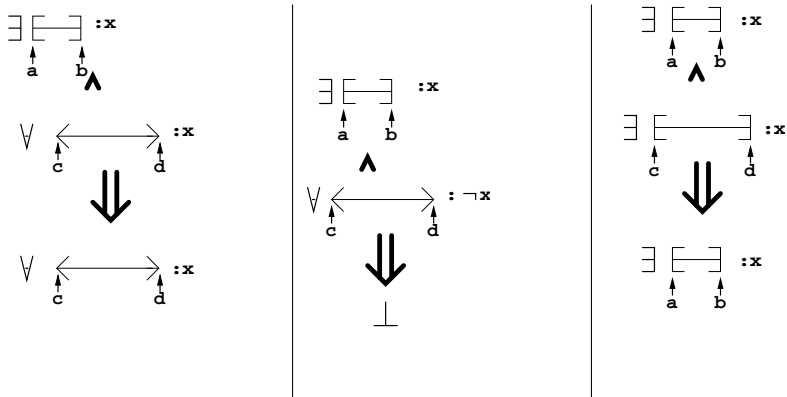
- $\forall \langle t; t + 2.5 \rangle : \text{False}$
- $\exists [t; t + 2.5] : \text{True}$

which is impossible and thus invalidates the initial hypothesis

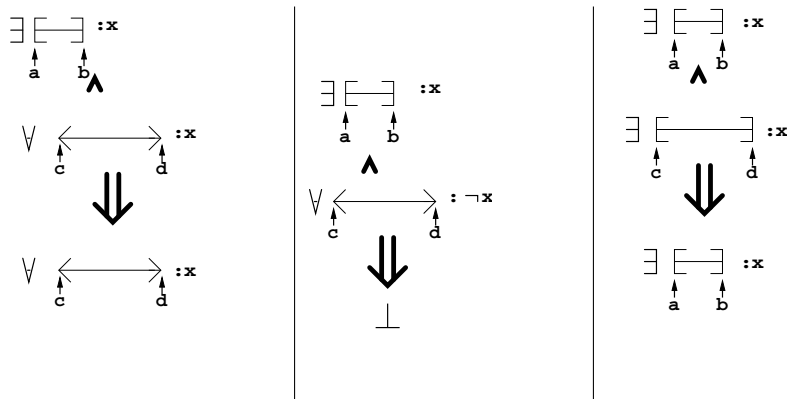
$\forall \langle t; t + 2.5 \rangle : \text{False}$.

- Thus $\exists [t; t + 2.5] : \text{True}$ is certified.

Operations on constraints : abstract conjunction



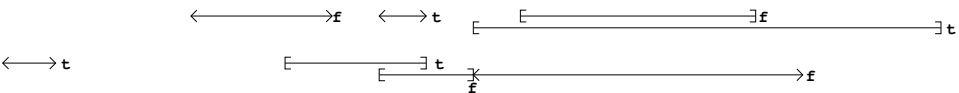
Operations on constraints : abstract conjunction



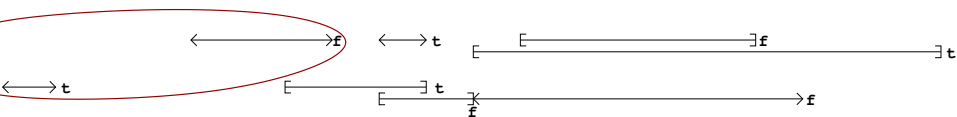
2 goals :

- An analysis that terminates! (without this $\subseteq^{\#} \emptyset^{\#}$ is not provable)
- A faster analysis, since the abstract conjunction is fast.

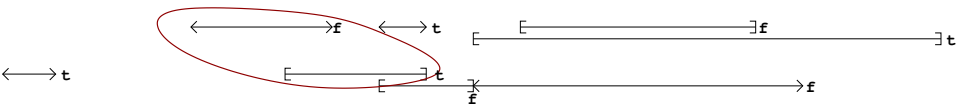
Optimized conjunction on Constraints



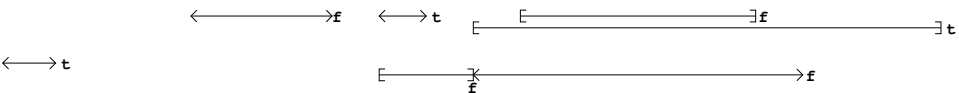
Optimized conjunction on Constraints



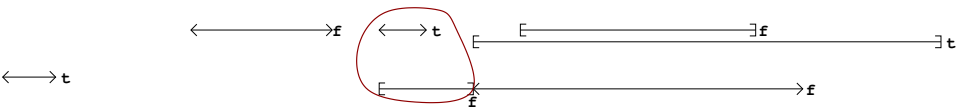
Optimized conjunction on Constraints



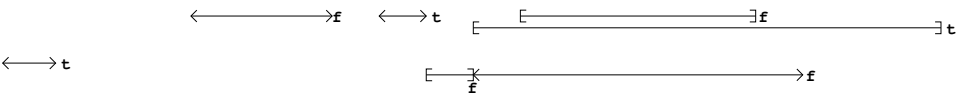
Optimized conjunction on Constraints



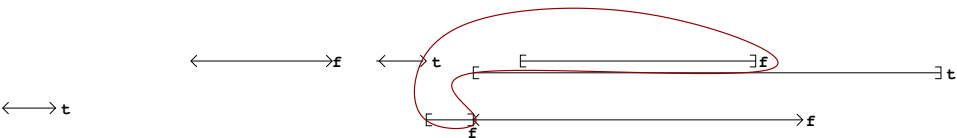
Optimized conjunction on Constraints



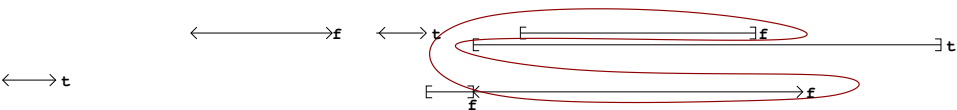
Optimized conjunction on Constraints



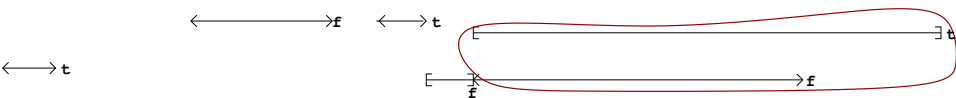
Optimized conjunction on Constraints



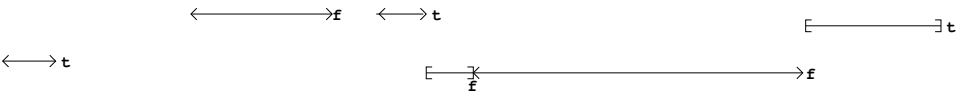
Optimized conjunction on Constraints



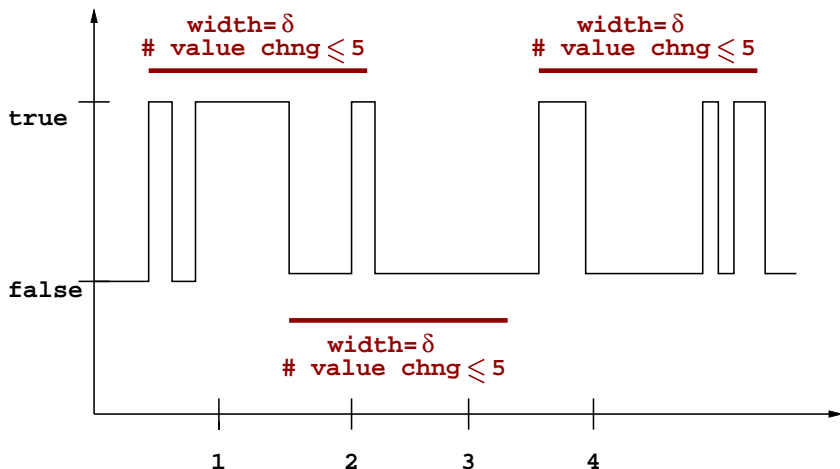
Optimized conjunction on Constraints



Optimized conjunction on Constraints

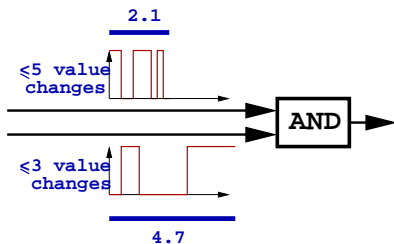


2nd Abstract Domain : Changes counting domain

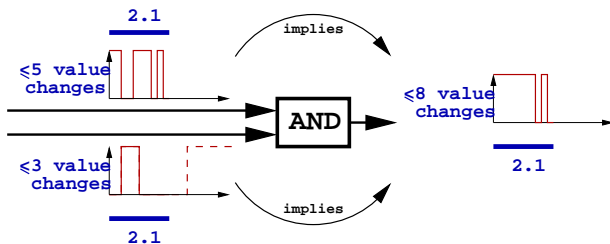


- Allows the expression of “stability” specifications .

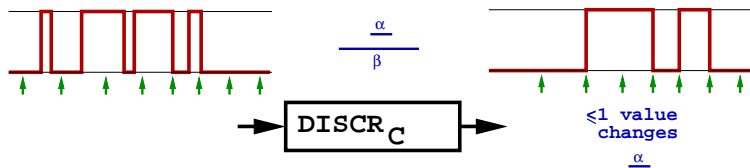
A non-temporal abstract operator



A non-temporal abstract operator

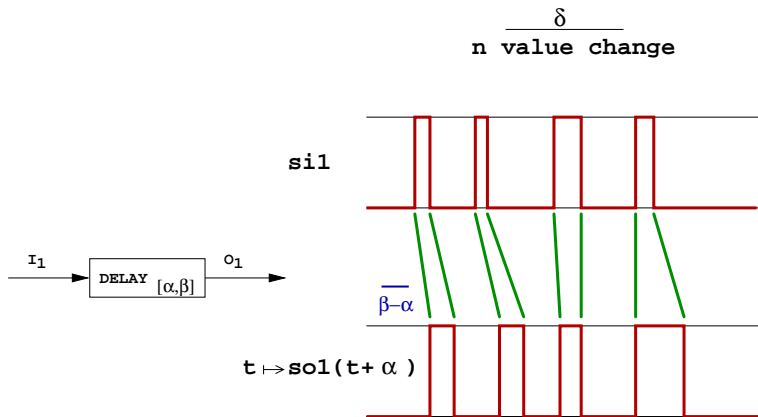


A temporal abstract operator



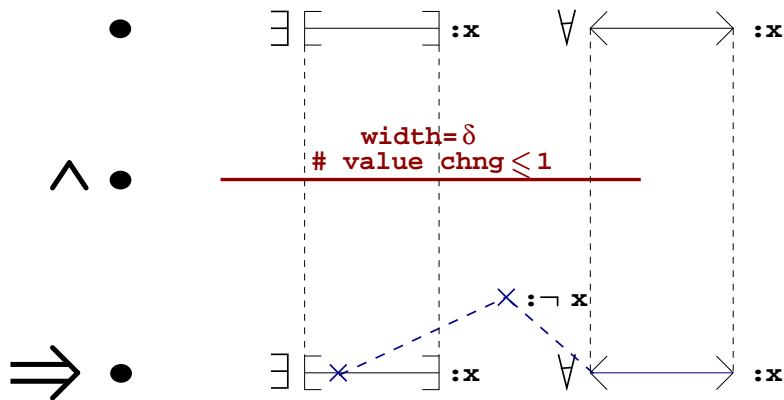
- $[\alpha, \beta]$ parameter of clock C
- $\forall A, \vec{\Psi}_{\text{DISCR}_{[\alpha, \beta]}}^\#(A) \triangleq (\leq 1, \alpha)$

A temporal abstract operator



$$\vec{\Psi}^\#(n, \delta)_{\mathcal{N}} \triangleq (n, \delta - \beta + \alpha)$$

Reduced Product Constraints - value changes counting



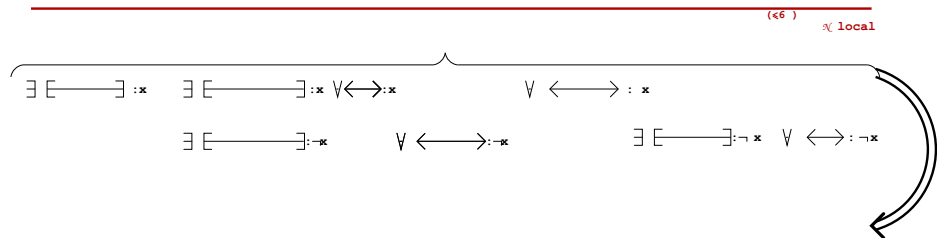
Reduced Product Constraints - value changes counting

$$\bullet \quad \exists [\text{---}] : \mathbf{x} \quad \forall \langle \text{---} \rangle : \mathbf{x}$$

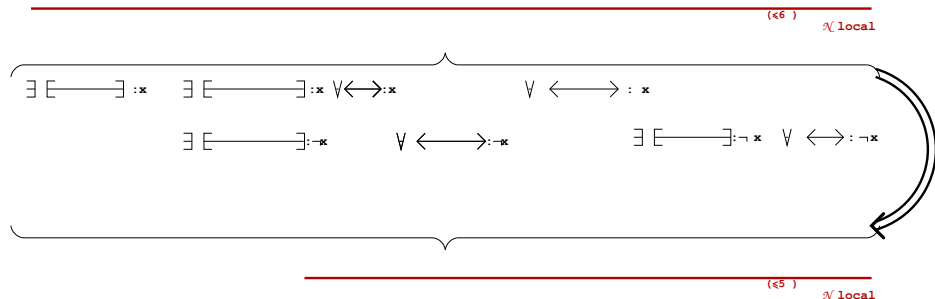
$$\wedge \bullet \quad \frac{\text{width} = \delta}{\# \text{ value chng} \leq 1}$$

$$\Rightarrow \bullet \quad \forall \langle \text{---} \rangle : \mathbf{x}$$

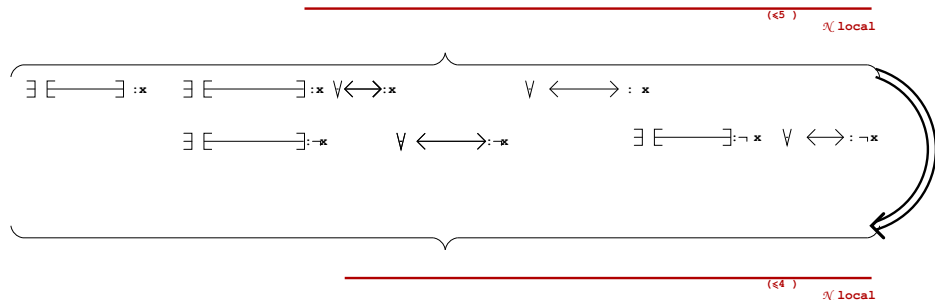
Linear Reduction : an example



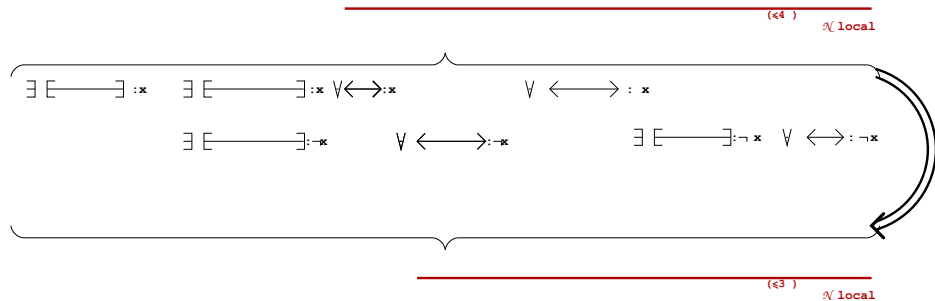
Linear Reduction : an example



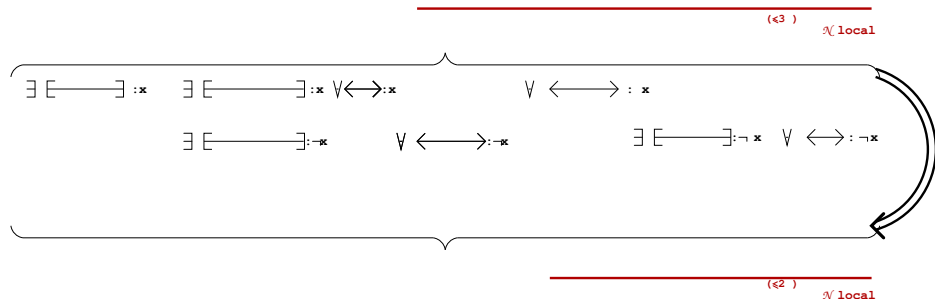
Linear Reduction : an example



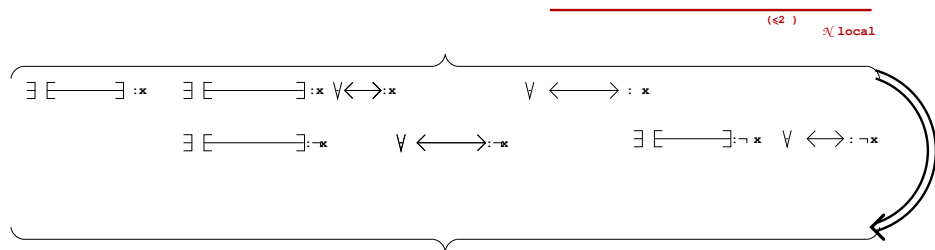
Linear Reduction : an example



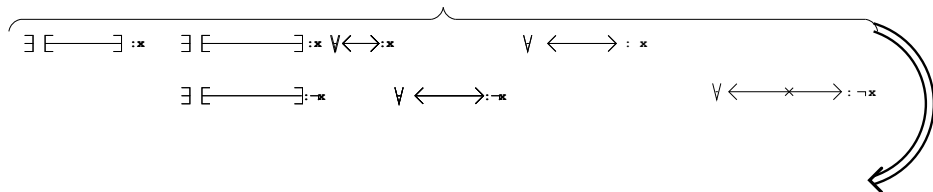
Linear Reduction : an example



Linear Reduction : an example

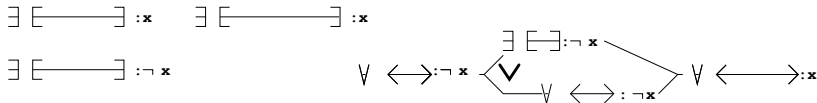


Linear Reduction : an example



Introducing generators

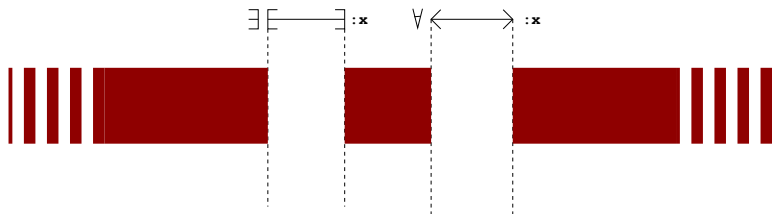
- New **systematic domain** generators (optimized temporal disjunction)



Common points of temporal domains

Temporal support

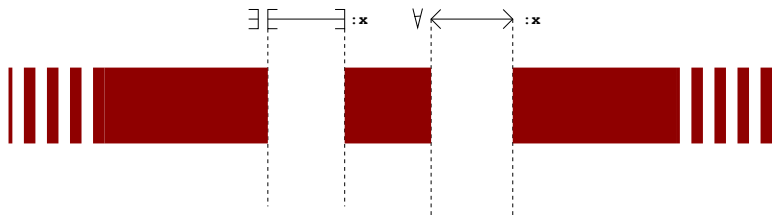
- Outside of **temporal support**, an abstract element has no influence.
- For example, for constraints :



Common points of temporal domains

Temporal support

- Outside of **temporal support**, an abstract element has no influence.
- For example, for constraints :



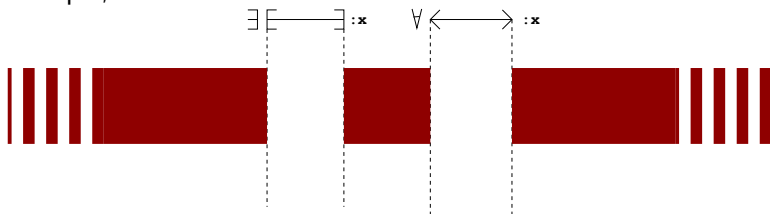
$$\text{right_border} : \begin{cases} \forall \langle a; b \rangle : x \mapsto b \\ \exists [a; b] : x \mapsto b \end{cases}$$

$$\text{left_border} : \begin{cases} \forall \langle a; b \rangle : x \mapsto a \\ \exists [a; b] : x \mapsto a \end{cases}$$

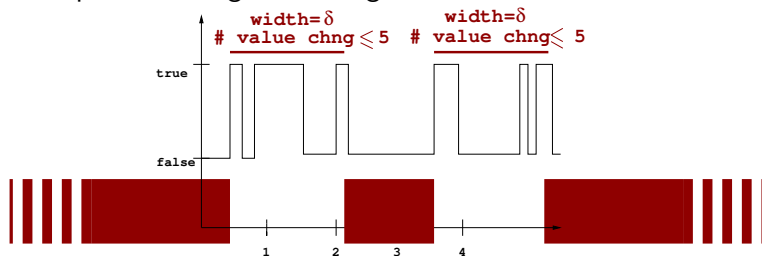
Common points of temporal domains

Temporal support

- For example, for constraints :



- For example, for changes countings :



Common points of temporal domains

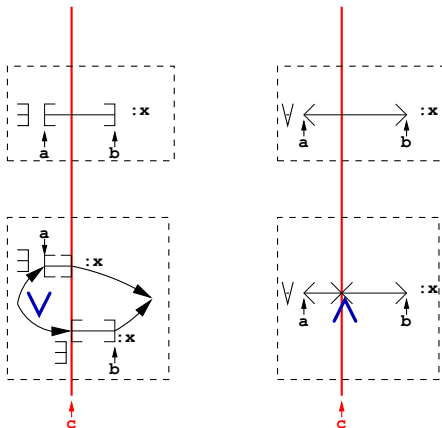
Temporal slicing

- the *slicing function* provides a weakened element fitting a temporal slice.

Common points of temporal domains

Temporal slicing

- the *slicing function* provides a weakened element fitting a temporal slice.



Common points of temporal domains

Temporal slicing

- the *slicing function* provides weakened element but fitting in temporal slices.

Common points of temporal domains

Temporal slicing

- the *slicing function* provides weakened element but fitting in temporal slices.
- The *slicing* of changes counting $(\leq 3, a \blacktriangleright, \blacktriangleleft b)_{\mathcal{N}}$ is

$$((\leq 3, a \blacktriangleright, \blacktriangleleft c)_{\mathcal{N}} \wedge (\leq 0, c \blacktriangleright, \blacktriangleleft b)_{\mathcal{N}})$$

$$\vee ((\leq 2, a \blacktriangleright, \blacktriangleleft c)_{\mathcal{N}} \wedge (\leq 1, c \blacktriangleright, \blacktriangleleft b)_{\mathcal{N}})$$

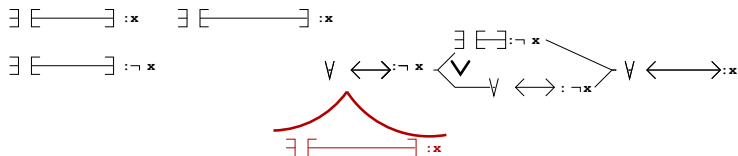
$$\vee ((\leq 1, a \blacktriangleright, \blacktriangleleft c)_{\mathcal{N}} \wedge (\leq 2, c \blacktriangleright, \blacktriangleleft b)_{\mathcal{N}})$$

$$\vee ((\leq 0, a \blacktriangleright, \blacktriangleleft c)_{\mathcal{N}} \wedge (\leq 3, c \blacktriangleright, \blacktriangleleft b)_{\mathcal{N}})$$

Common points of temporal domains

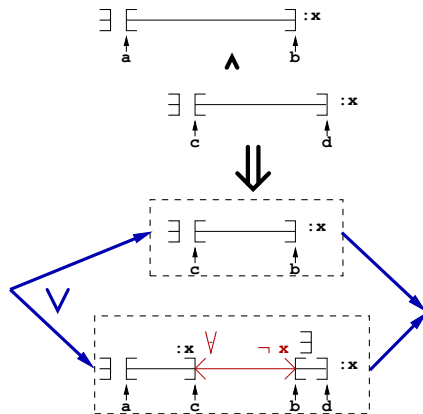
Temporal interaction

- The *temporal interaction* of two elements whose temporal support overlap is an over-approximation of the result of their conjunction .



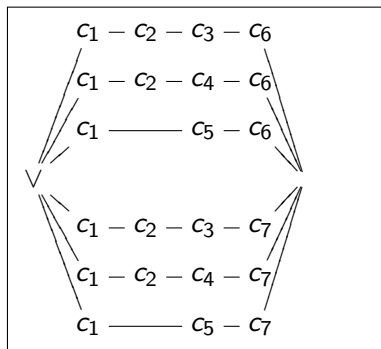
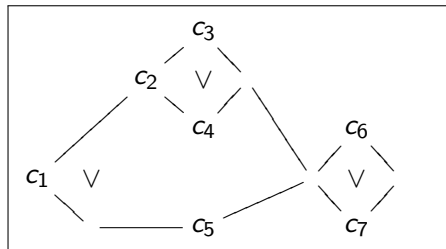
Common points of temporal domains

Temporal interaction



Common points of temporal domains

- Branching lists describe elements in a concise **disjunctive** way :

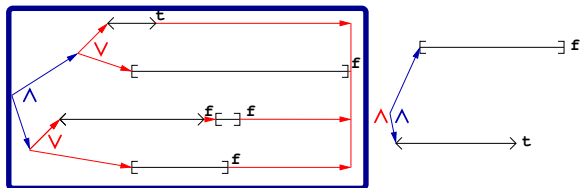
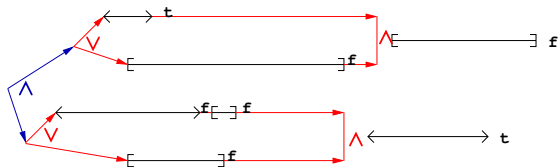


- Temporal policy :

- \triangleright If $l_1 = \begin{matrix} l_2 \\ \vee \\ l_3 \end{matrix} l_4$ then $\text{right_limit}(l_2) \leq \text{left_limit}(l_4)$
- \triangleright If $l_1 = l_2 :: l_3$, then $\text{right_limit}(l_2) \leq \text{left_limit}(l_3)$ except if existential and with same support and enforcing different values

Temporal domains : Conjunction

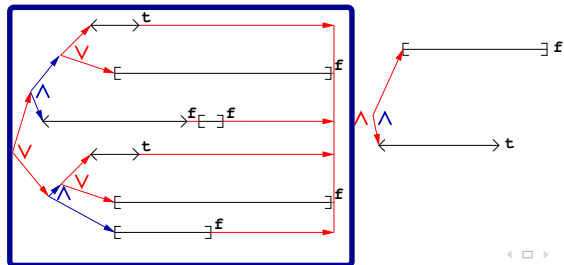
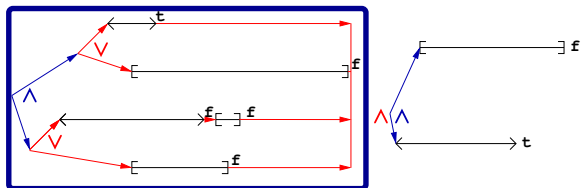
- How to perform the conjunction of two elements



Temporal slicing

Temporal domains : Conjunction

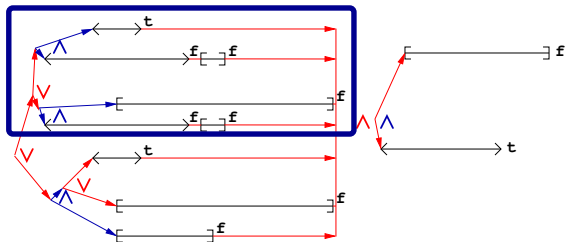
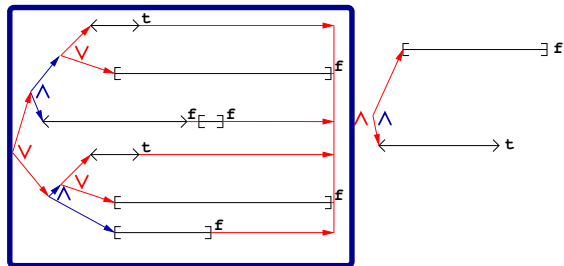
- How to perform the conjunction of two elements



Propagation

Temporal domains : Conjunction

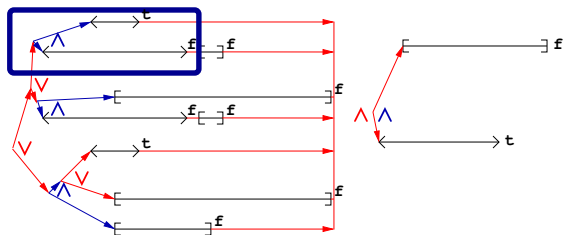
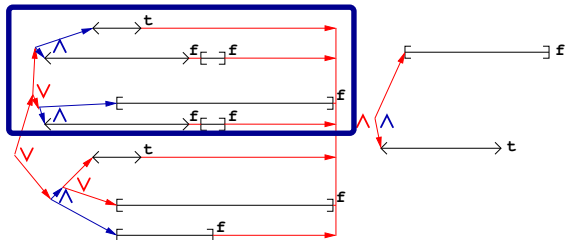
- How to perform the conjunction of two elements



Propagation

Temporal domains : Conjunction

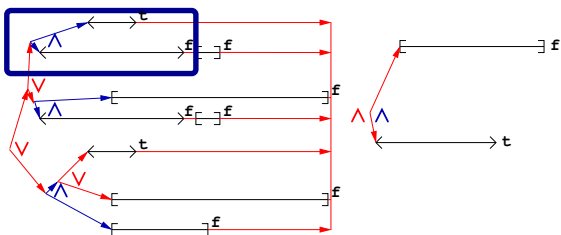
- How to perform the conjunction of two elements



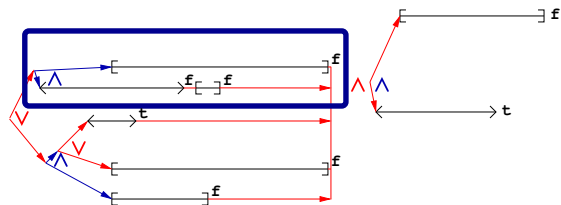
Temporal slicing

Temporal domains : Conjunction

- How to perform the conjunction of two elements

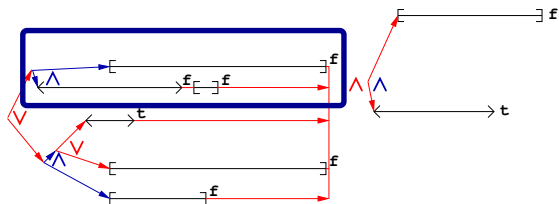


Interaction (\perp)

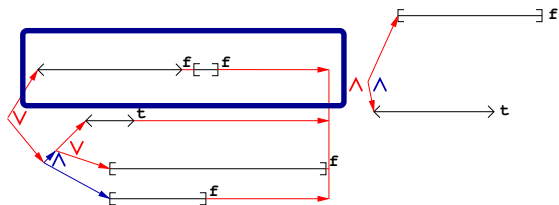


Temporal domains : Conjunction

- How to perform the conjunction of two elements

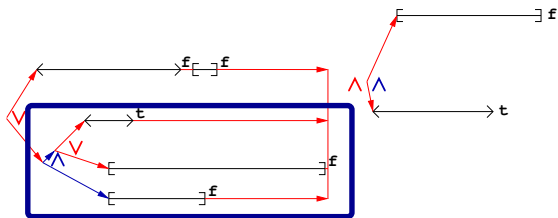


Interaction

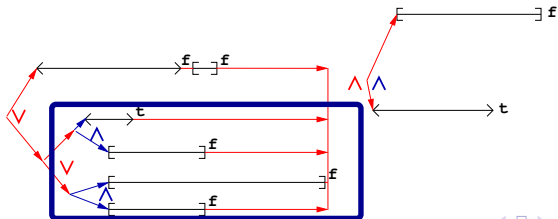


Temporal domains : Conjunction

- How to perform the conjunction of two elements

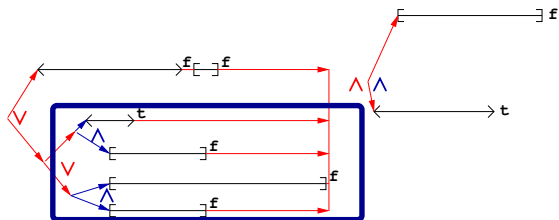


Propagation

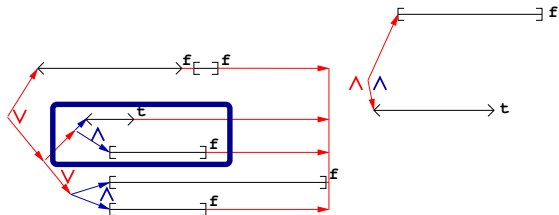


Temporal domains : Conjunction

- How to perform the conjunction of two elements

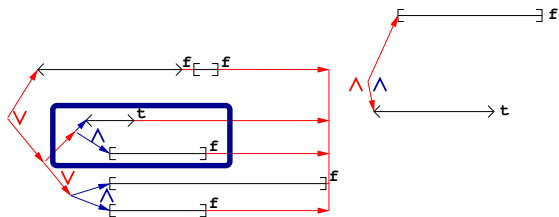


Temporal slicing

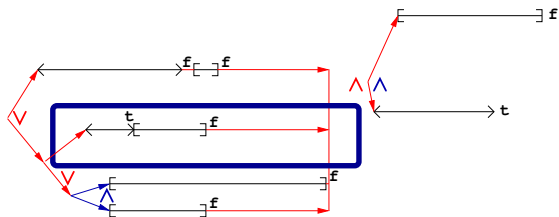


Temporal domains : Conjunction

- How to perform the conjunction of two elements

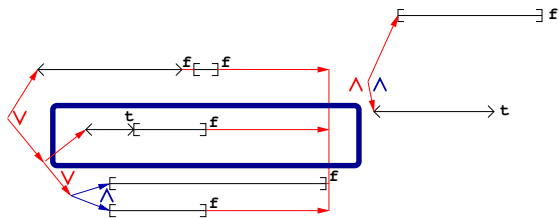


Interaction

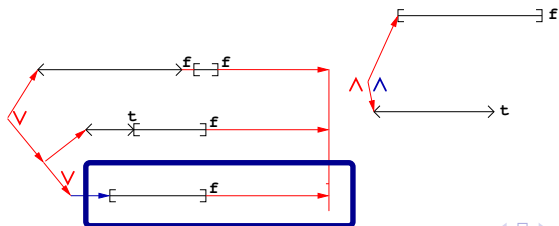


Temporal domains : Conjunction

- How to perform the conjunction of two elements

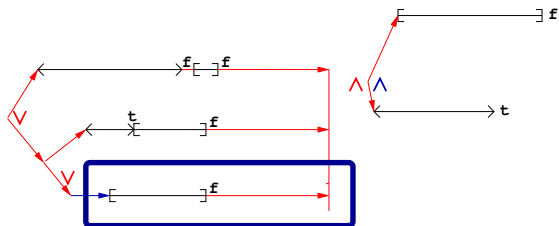


Interaction

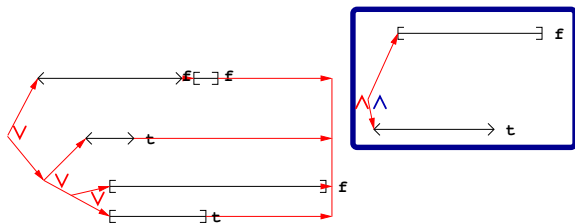


Temporal domains : Conjunction

- How to perform the conjunction of two elements

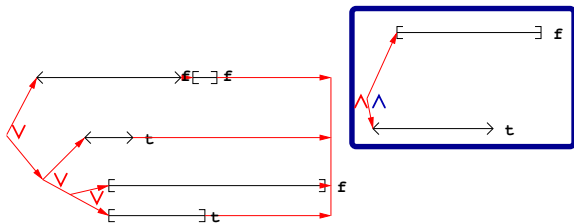


Temporal slicing

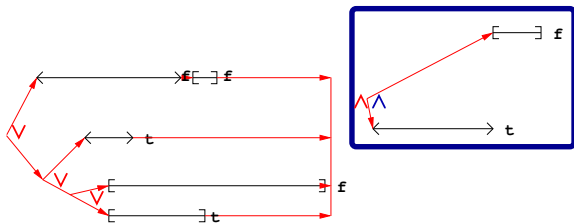


Temporal domains : Conjunction

- How to perform the conjunction of two elements

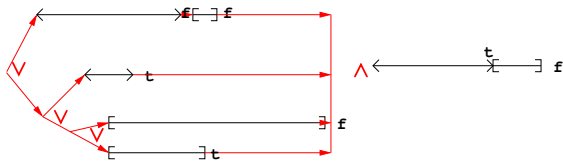
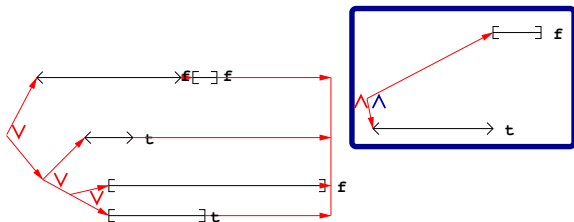


Interaction



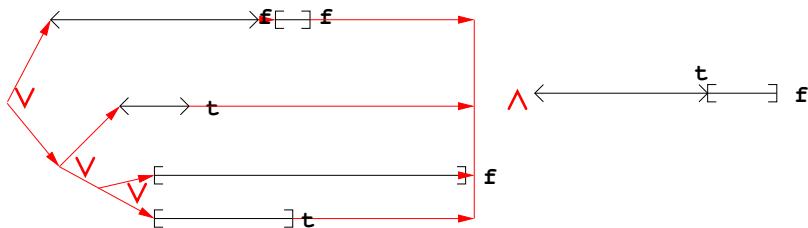
Temporal domains : Conjunction

- How to perform the conjunction of two elements

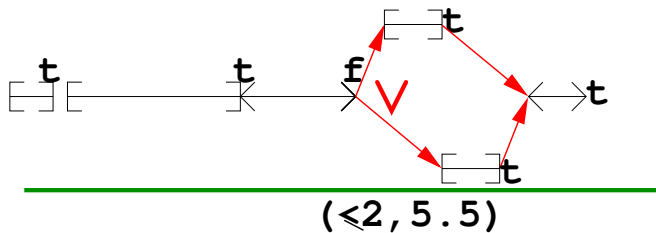


Temporal domains : Conjunction

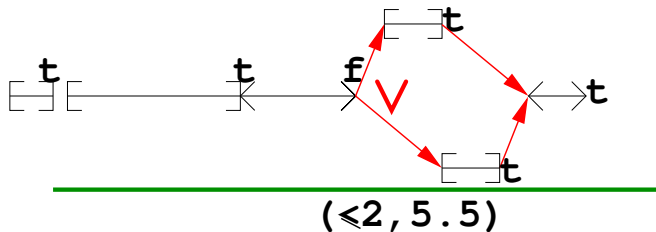
- How to perform the conjunction of two elements



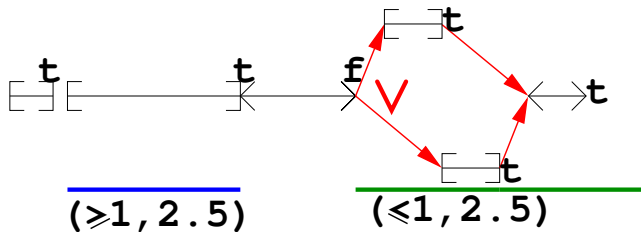
Disjunctive Reduced Product Constraints-value changes countings : an example



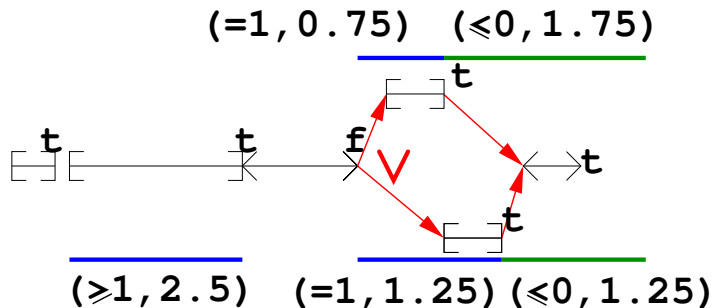
Disjunctive Reduced Product Constraints-value changes countings : an example



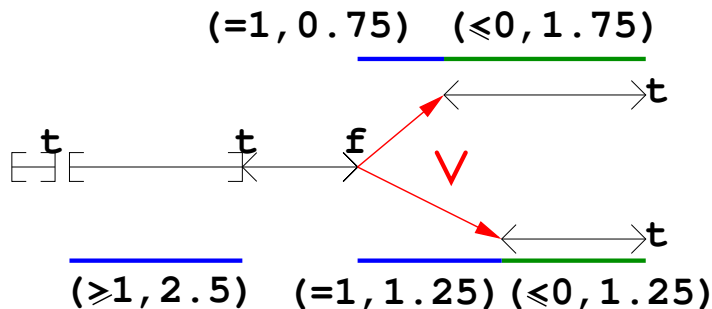
Disjunctive Reduced Product Constraints-value changes countings : an example



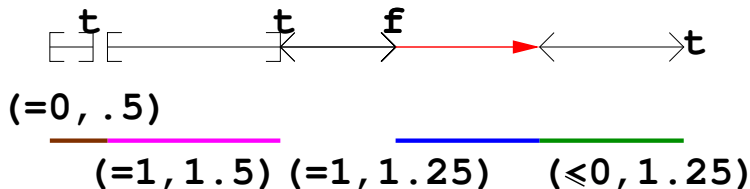
Disjunctive Reduced Product Constraints-value changes countings : an example



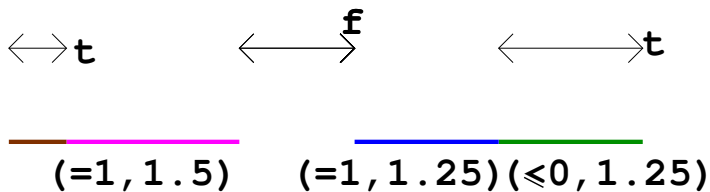
Disjunctive Reduced Product Constraints-value changes countings : an example



Disjunctive Reduced Product Constraints-value changes countings : an example



Disjunctive Reduced Product Constraints-value changes countings : an example



Conclusion

- Communicating synchronous systems have imperfect clock and non-instantaneous communication.
 - ▶ Proofs of correctness much more difficult and have to consider temporal aspects

- **Communicating synchronous systems** have imperfect clock and non-instantaneous communication.
 - ▶ **Proofs of correctness** much more difficult and have to consider temporal aspects
- We define a **continuous-time semantics**
 - ▶ The **continuous-time semantics** allows very fast and precise abstraction.
 - ▶ We need new **temporal abstract domains**.

- **Communicating synchronous systems** have imperfect clock and non-instantaneous communication.
 - ▶ **Proofs of correctness** much more difficult and have to consider temporal aspects
- We define a **continuous-time semantics**
 - ▶ The **continuous-time semantics** allows very fast and precise abstraction.
 - ▶ We need new **temporal abstract domains**.
- **Reduced product** can be performed fast if the time is taken into consideration.

- **Communicating synchronous systems** have imperfect clock and non-instantaneous communication.
 - ▶ **Proofs of correctness** much more difficult and have to consider temporal aspects
- We define a **continuous-time semantics**
 - ▶ The **continuous-time semantics** allows very fast and precise abstraction.
 - ▶ We need new **temporal abstract domains**.
- **Reduced product** can be performed fast if the time is taken into consideration.
- **Systematic domain** generators (like optimized temporal disjunction) should also be temporal.

- **Communicating synchronous systems** have imperfect clock and non-instantaneous communication.
 - ▶ **Proofs of correctness** much more difficult and have to consider temporal aspects
- We define a **continuous-time semantics**
 - ▶ The **continuous-time semantics** allows very fast and precise abstraction.
 - ▶ We need new **temporal abstract domains**.
- **Reduced product** can be performed fast if the time is taken into consideration.
- **Systematic domain** generators (like optimized temporal disjunction) should also be temporal.
- Temporal domains should not be too expressive : **trade-of** between precision and speed is achieved by defining a temporal policy.

Questions ?

Slides : www.di.ens.fr/~bertrane/

Contact : julien@sei.ecnu.edu.cn

Contact (France) : bertrane@di.ens.fr