Lazy parallel composition of infinite transition systems

Yuliia Romenska & Frédéric Mallet

Université Nice Sophia Antipolis

EPC Aoste

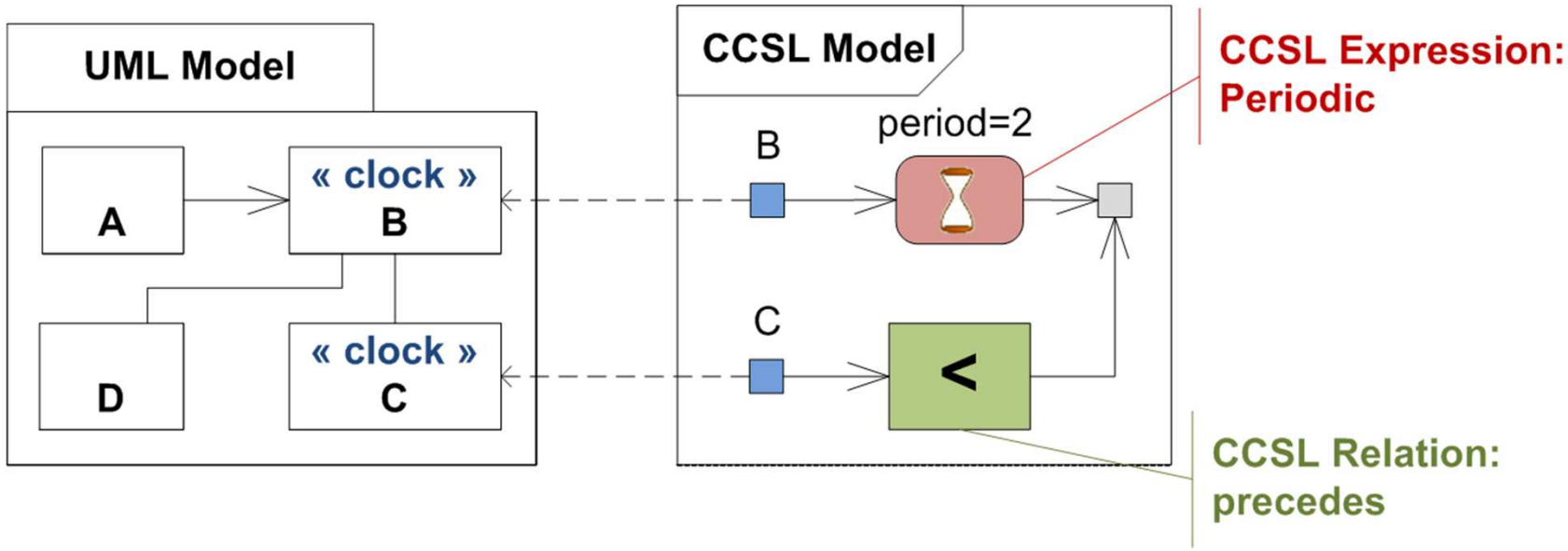
UML Profile for MARTE

- Model-Driven design of **real-time reactive embedded systems,**
  - Modeling for analysis (static or dynamic) and synthesis
- Reconcile
  - Some **formal models** (concurrency theory)
  - with some **engineering models** (MDE)
  - For different **application domains**
    - Avionics, Automotive, Systems-on-chip
- **OMG UML Marte** (**M**odeling and **A**nalysis of **R**eal-**T**ime and **E**mbedded systems)
  - Supersedes **S**chedulability **P**erformance & **T**ime
  - Two chapters in MARTE foundations: *Time*, *Allocation*
  - Convergence with SysML : Systems Engineering

MARTE/CCSL

- **C**lock **C**onstraint **S**pecification **L**anguage
  - Syntax to make explicit the semantics of models (of computations and communications)
    - Causal and temporal relationships
  - Clocks become first-class citizens (not hidden behind signals)

- Logical clocks (Lamport, synchronous languages)
  - Clock = (infinite) sequence of instants <=> events
  - Instant = Event occurrence



MARTE/CCSL

- **C**lock **C**onstraint **S**pecification **L**anguage
  - Syntax to make explicit the semantics of models (of computations and communications)
    - Causal and temporal relationships
  - Clocks become first-class citizens (not hidden behind signals)

- Logical clocks (Lamport, synchronous languages)
  - Clock = (infinite) sequence of instants
  - Instant = Event occurrence

  <=> events

- **Asynchronous**/causal relationships
  - a causes b **or** b depends on a
    - $\forall i \in \mathbb{N}^*, a[i] \leq b[i]$
  - a precedes b
    - $\forall i \in \mathbb{N}^*, a[i] < b[i]$

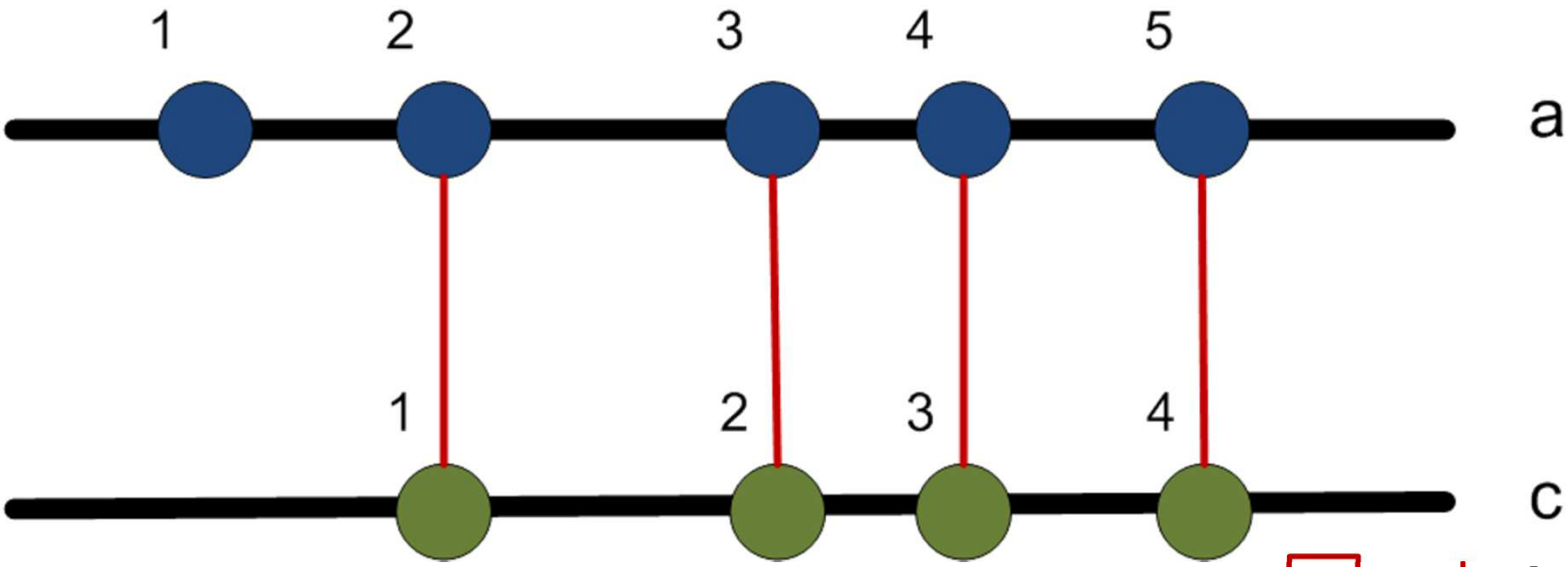
- **Synchronous** relationships
  - a isSynchronousWith b
    - $\forall i \in \mathbb{N}^*, a[i] \equiv b[i]$
  - a isSubclockOf b
    - $\forall i \in \mathbb{N}^*, a[i] \equiv b[\mathbf{f(i)}]$

Example (pure delay): a = b $ n

$\forall i \in \mathbb{N}^*, a[i] \equiv b[i + n]$

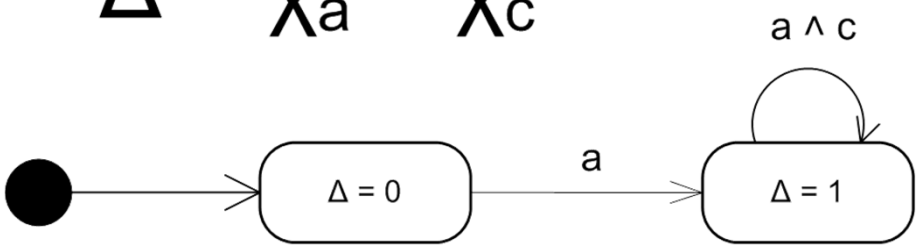
Clock relations – Binary delay

- Infinitely many coincidence relations



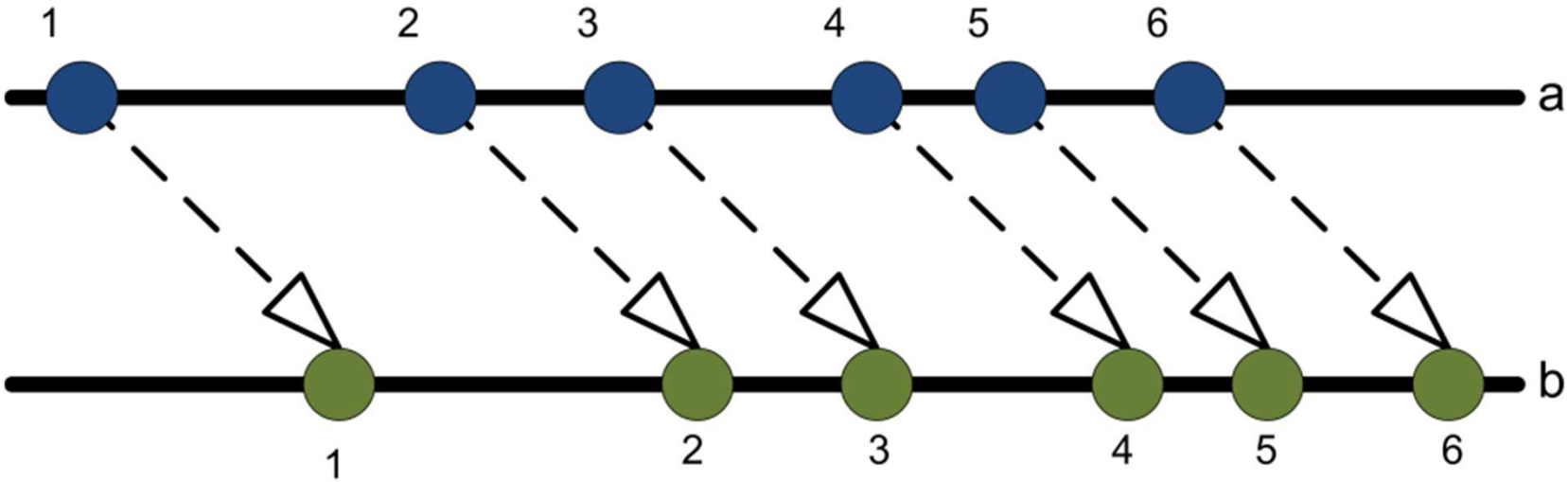
$c \boxed{=} a \$ 1$

$$\Delta = X_a - X_c$$



Clock relations – Precedence

- Infinitely many precedence relations



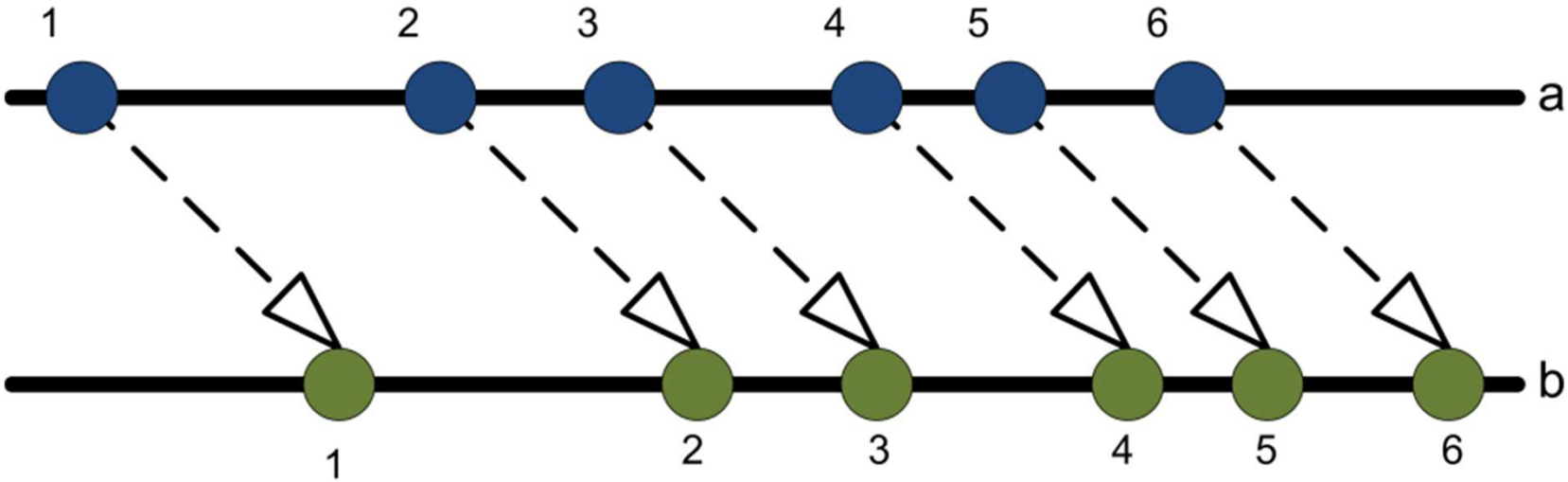
a $\prec$ b

a **precedes** b

Not Bounded !
Problem for model-checking

Clock relations – Precedence

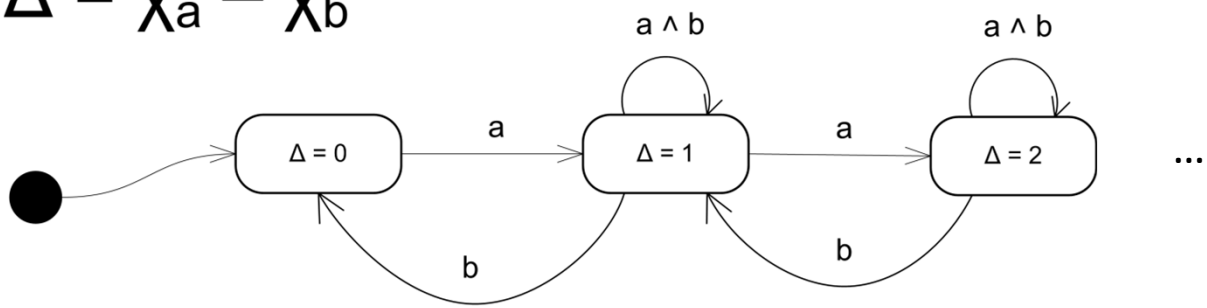
- Infinitely many precedence relations



a ⊲ b

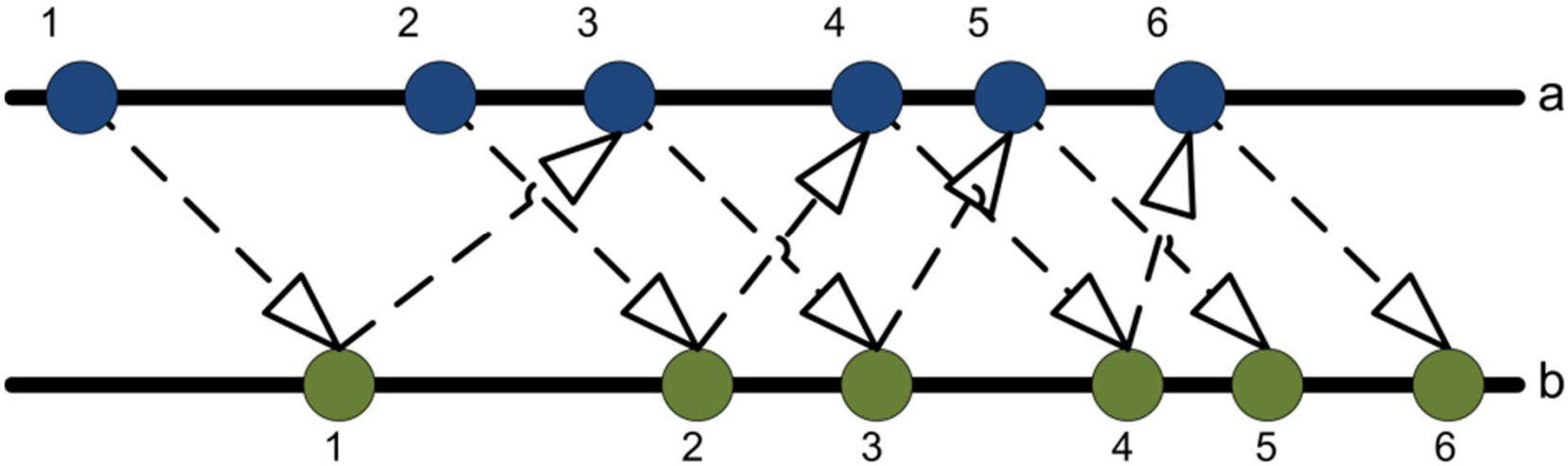
a **precedes** b

$\Delta = X_a - X_b$



Clock relations – bounded-precedence

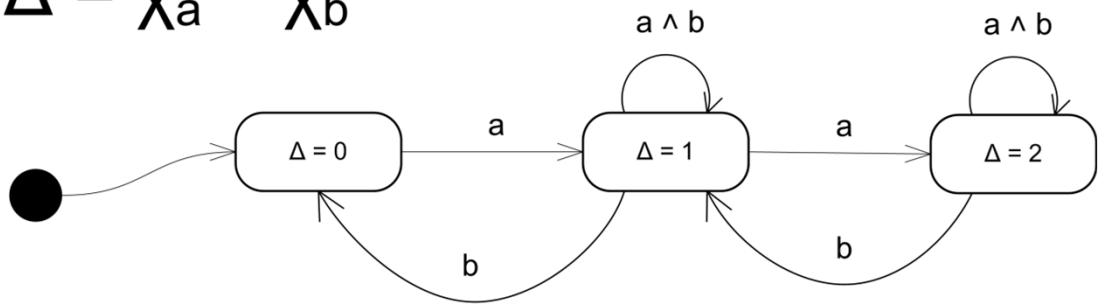
- Infinitely many precedence relations



$a \prec_2 b$

a **2**-**precedes** b

$\Delta = X_a - X_b$



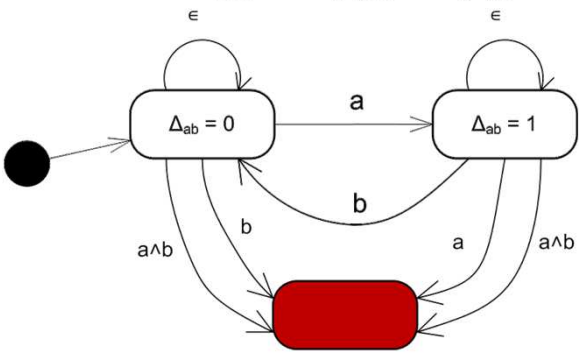
Parallel composition
Finite automata

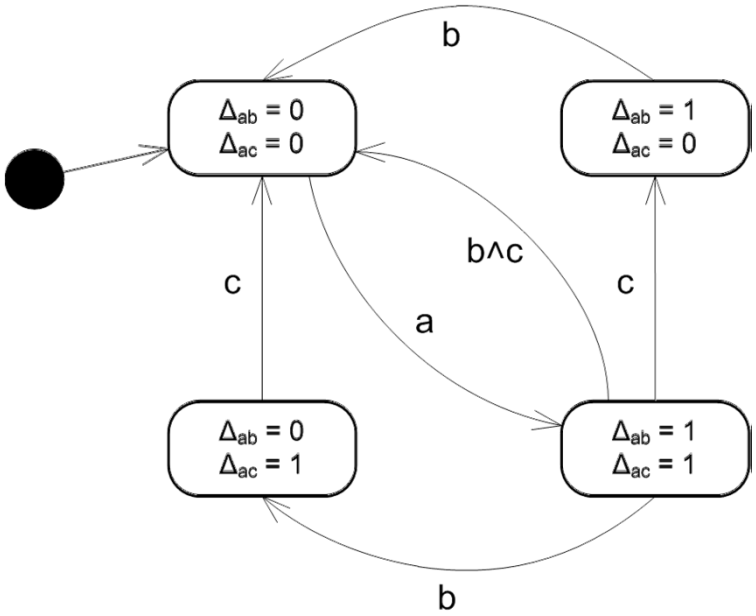
- **Synchronized product** of automata

$a \prec_1 b$

a **1**-**precedes** b

$\Delta_{ab} = X_a - X_b$



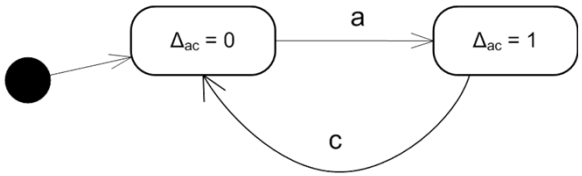


||

$a \prec_1 c$

a **1**-**precedes** c

$\Delta_{ac} = X_a - X_c$



Parallel composition
Infinite automata

- **Synchronized product** of automata

a ≺ b

a **precedes** b

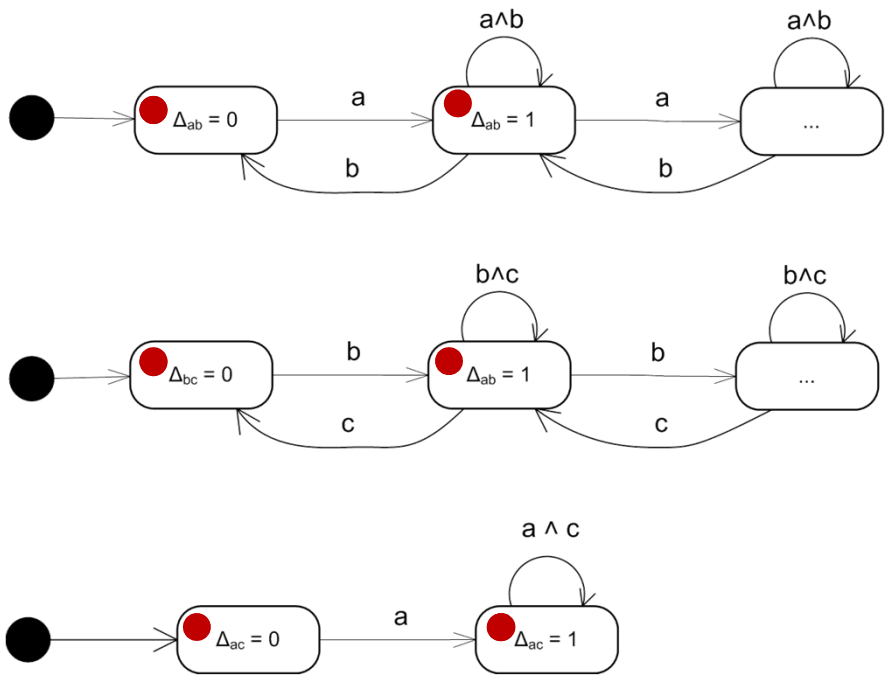
||

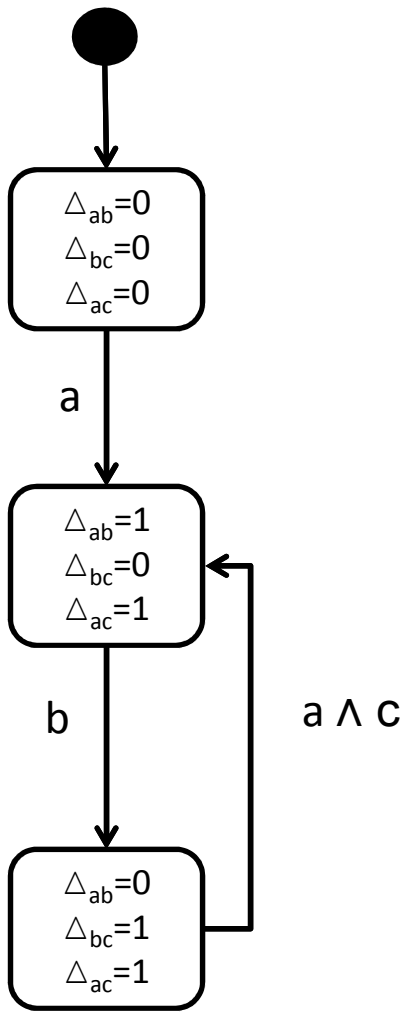
b ≺ c

b **precedes** c

||

c = a $ 1





Problem statement

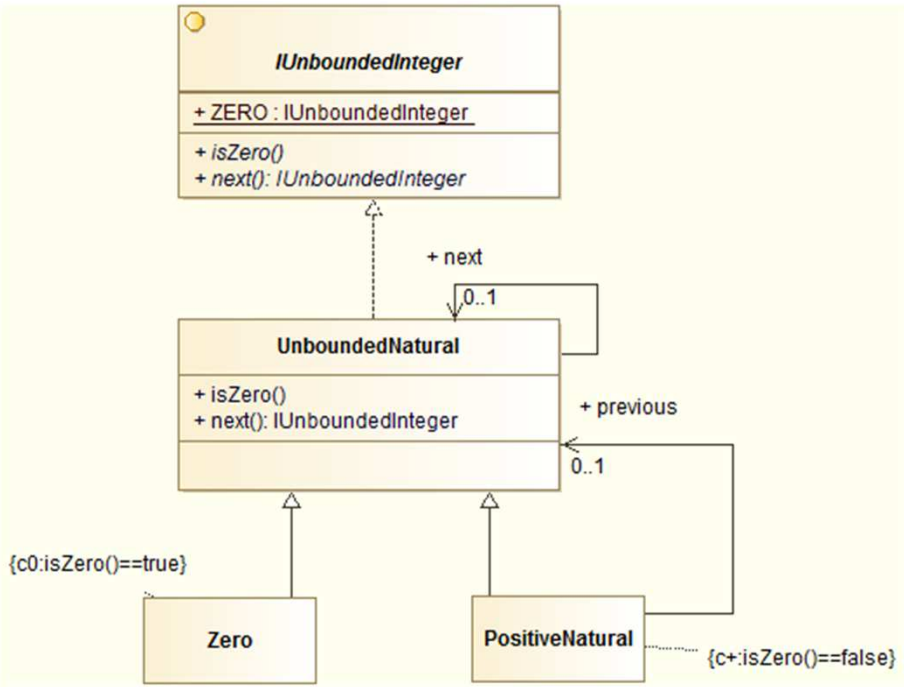
- Use CCSL as a **specification language**
  - Generation of Esterel observers **[LCTES'09]**
    - Modular generation => Bounded-subset of CCSL
- Verify properties on CCSL specifications through **model-checking**
  - Transformation into Büchi Automata **[Time'11]**
    - Bounded-subset of CCSL operators
- Controller synthesis (Signal/Sigali) **[Memocode'11]**

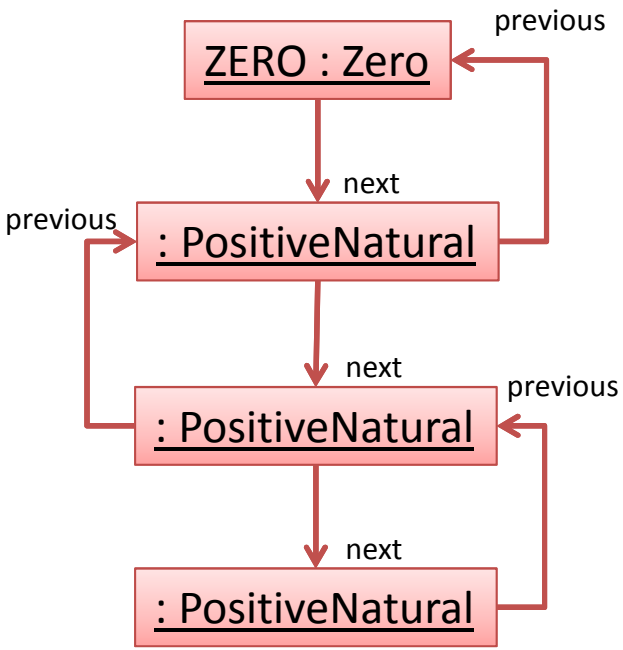
➢ How to deal with an infinite structure?

- Perform model-checking
- => Lazy evaluation

Solution

- Lazy evaluation: Encode Infinite Transition Systems
  - int => Unbounded integers
  - Intentional infinite structure: not in extension



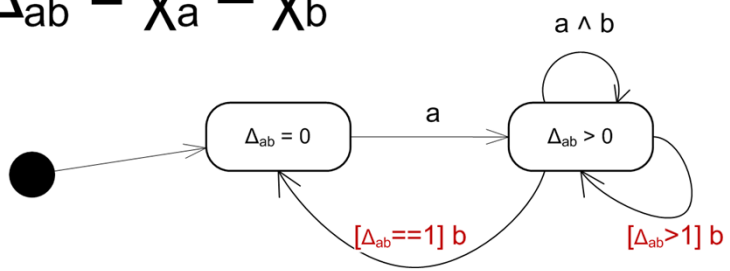


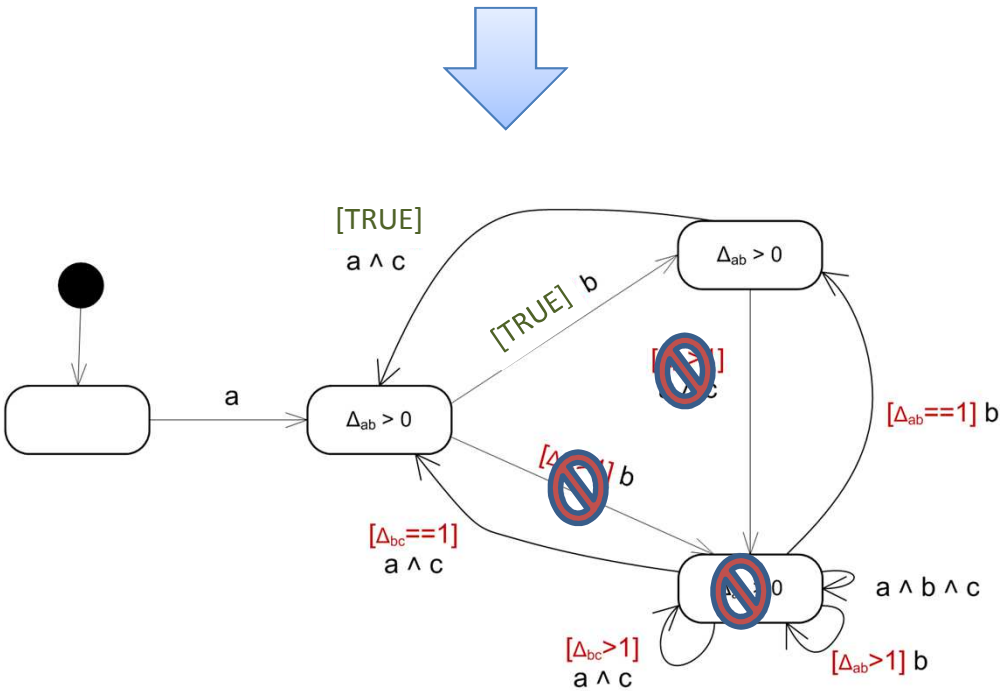
Other solution

- Extended Finite State Machine
  - Pushback automata

a<b || c=a$1 || b<c

$\Delta_{ab} = X_a - X_b$





Integer Linear Programming
With infinite integers

[ESWeek - RSP'2012]

Time Complexity

- Number of states |S|
  - No maximal bound (may be infinite)
- Number of transitions
  - For each state, need to consider all the combinations for all the clocks
    - Each clock can: tick, stall, be dead
    - N clocks => 3^N combinations => 3|C|*|S|
- Actual case
  - Much better than worst case
  - No precise criterion
  - Cycles in precedence or subclocking seem to reduce drastically the complexity

Conclusion and future work

- State-based semantics for CCSL
  - Allows for the code generation
  - Does not assume bounded operators as in previous works
- Synchronized Product (vs. RSP 2012)
  - Does not finish if the product is infinite
    - Need a way to detect the boundness
    - Problem similar to safety in process networks
  - Exponential in number of clocks when finite
    - Need a way to split the specification into pieces
    - Avoid building the whole state space