Compilation of Unbounded CCSL operators

Frédéric Mallet

Université Nice Sophia Antipolis

EPC Aoste

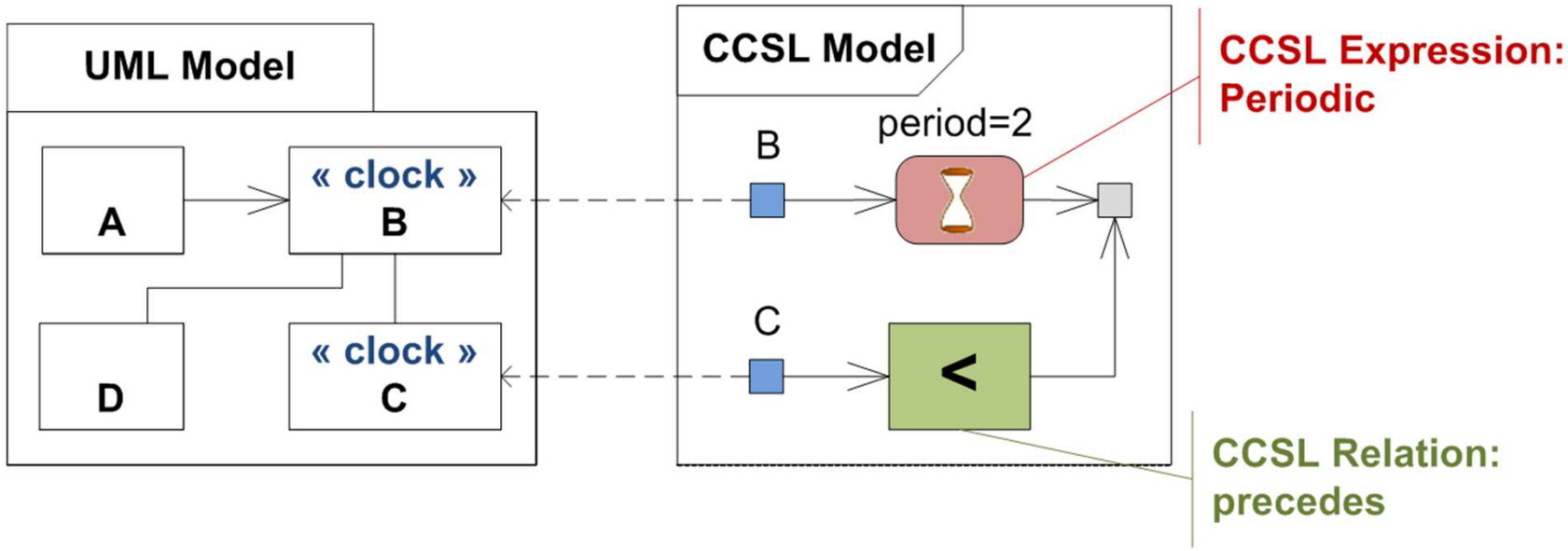
UML Profile for MARTE

- 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
    - $\forall i \in \mathbb{N}^*$, $a[i] < b[i]$

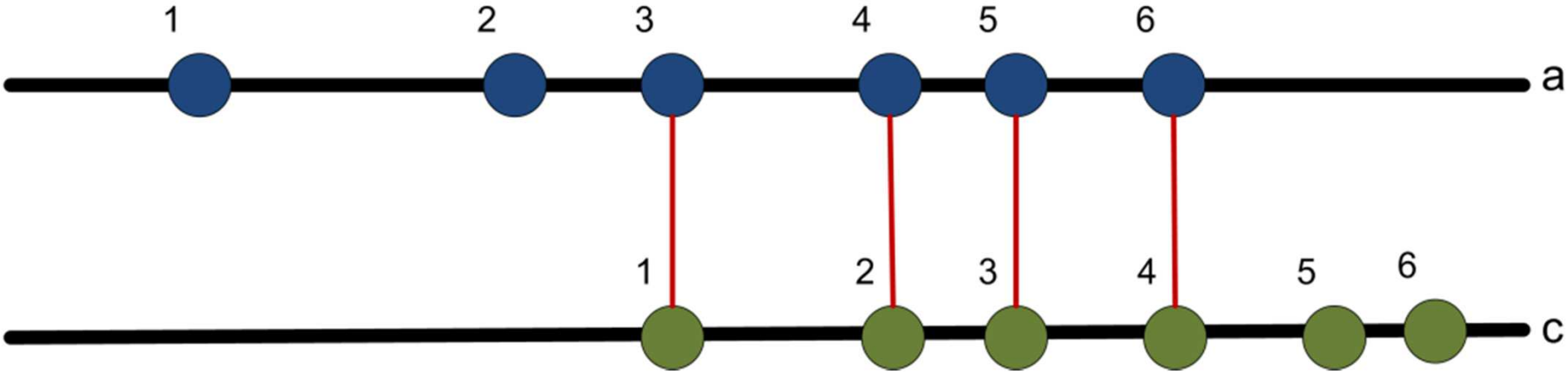
- **Synchronous** relationships
  - a isSynchronousWith b
    - $\forall i \in \mathbb{N}^*$, $a[i] \equiv b[i]$
  - a isSubclockOf b
    - $\forall i \in \mathbb{N}^*$, $a[i] \equiv b[\mathbf{f(i)}]$

Example (pure delay): a = b $ n

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

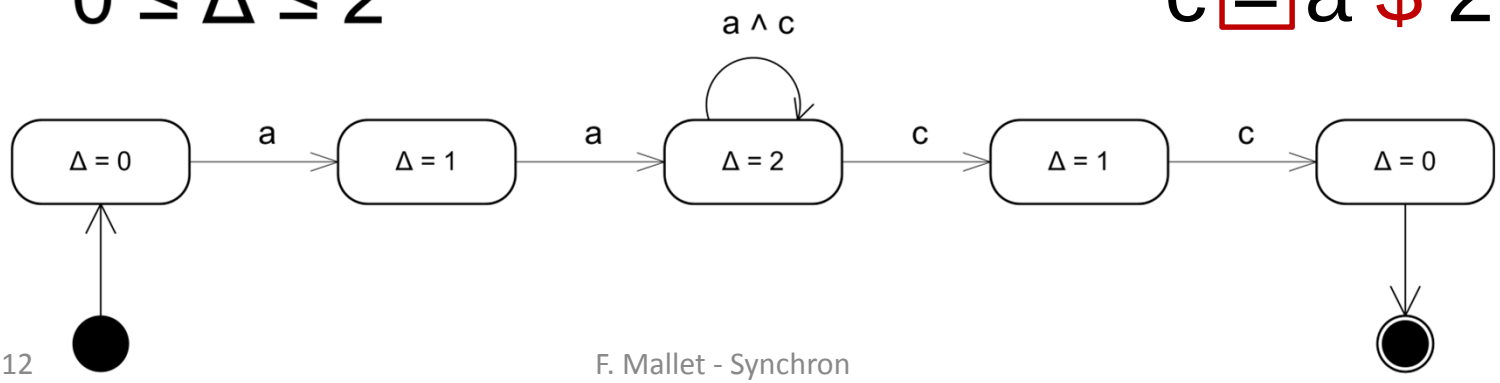
Clock relations – Binary delay

- Infinitely many coincidence relations



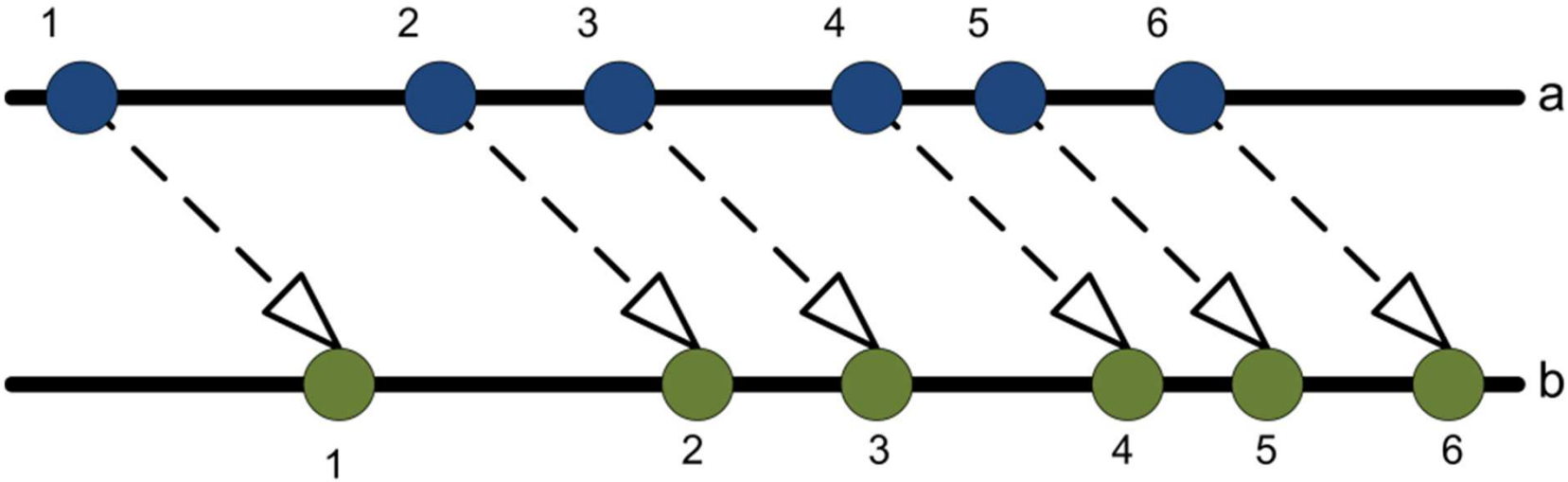
$$\Delta = X_a - X_c$$
$$0 \leq \Delta \leq 2$$

c $\boxed{=}$ a $ 2



Clock relations – Precedence

- Infinitely many precedence relations



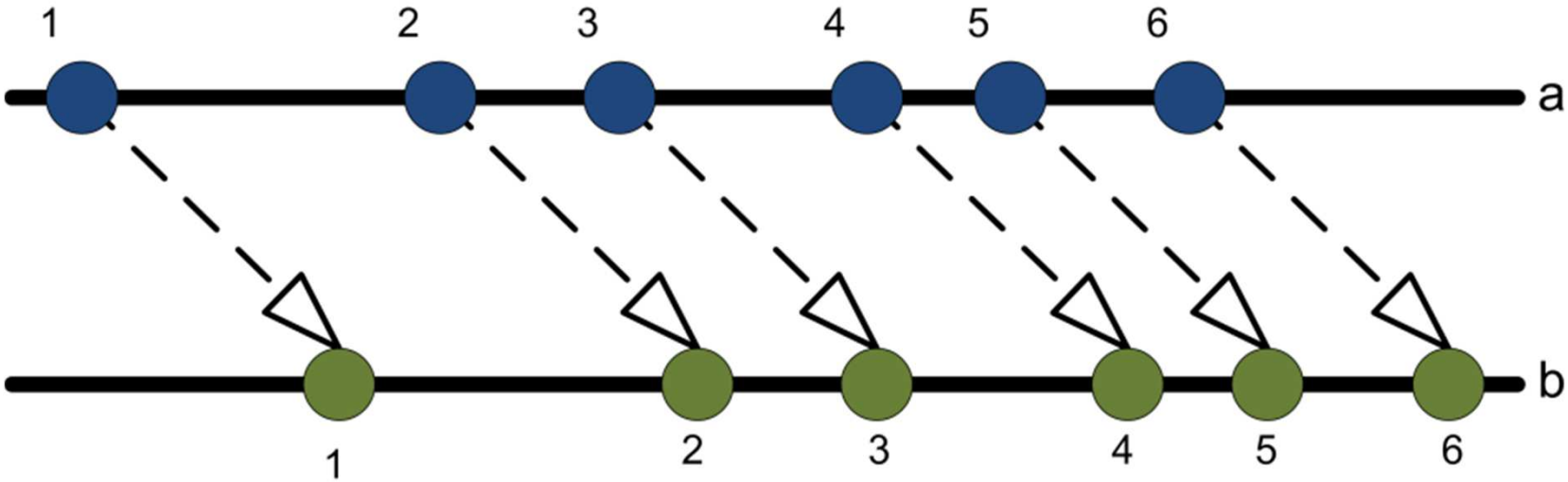
a ≺ b

a **precedes** b

Not Bounded !
Problem for model-checking

Clock relations – Precedence

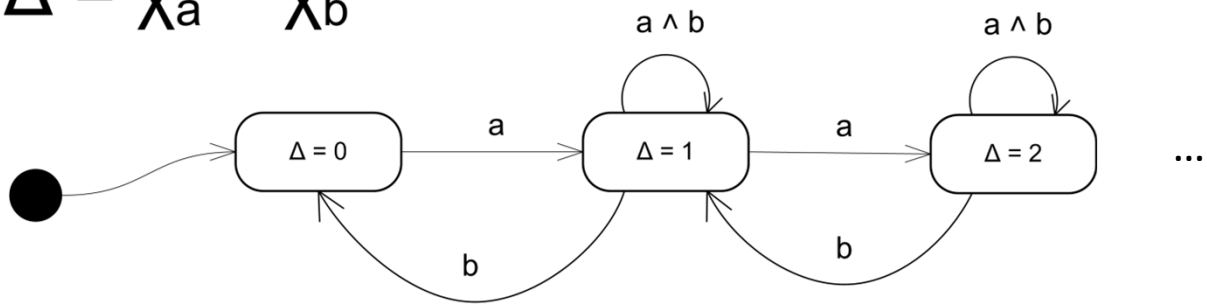
- Infinitely many precedence relations



a ≺ b

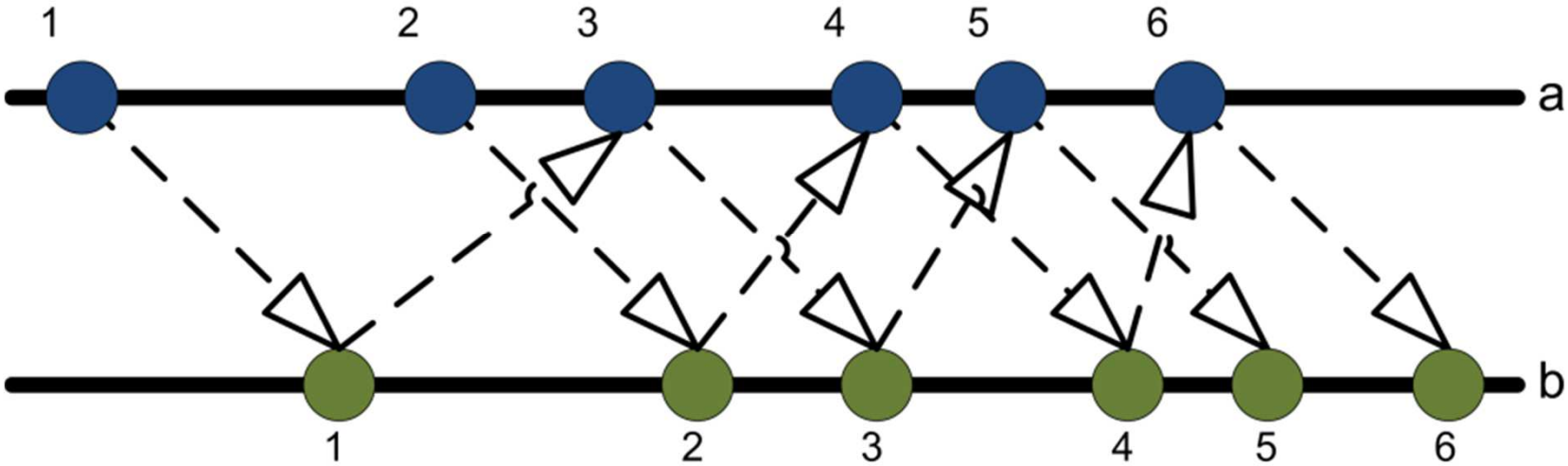
a **precedes** b

$\Delta = X_a - X_b$



Clock relations – bounded-precedence

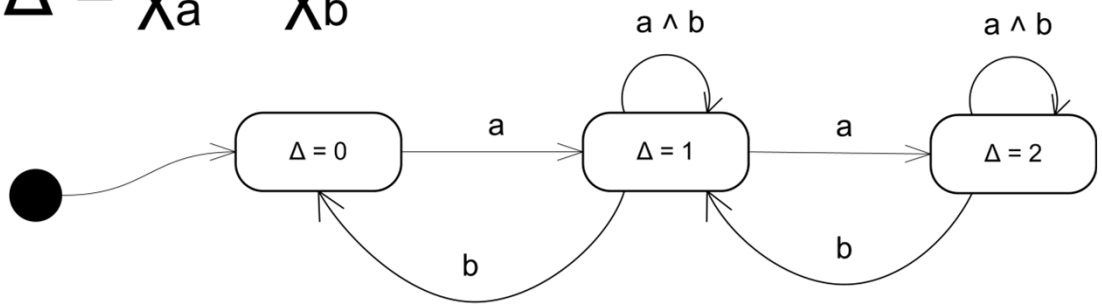
- Infinitely many precedence relations



a $\prec_2$ b

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

$\Delta = X_a - X_b$



Problem

- 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]**

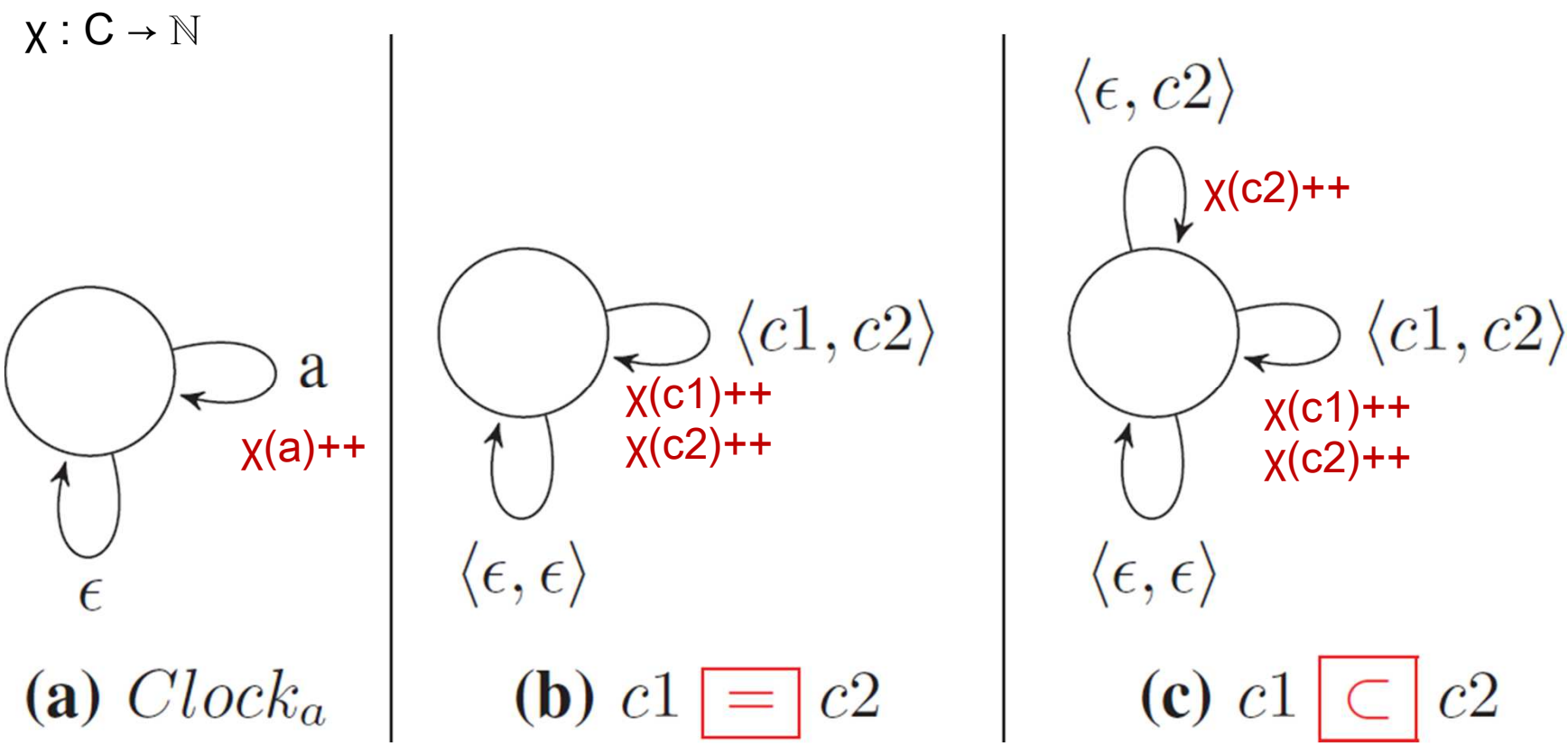
➢ Detect that the composition of unbounded operators is bounded ?

- Build the (symbolic) state representation
- Generate code

How ?

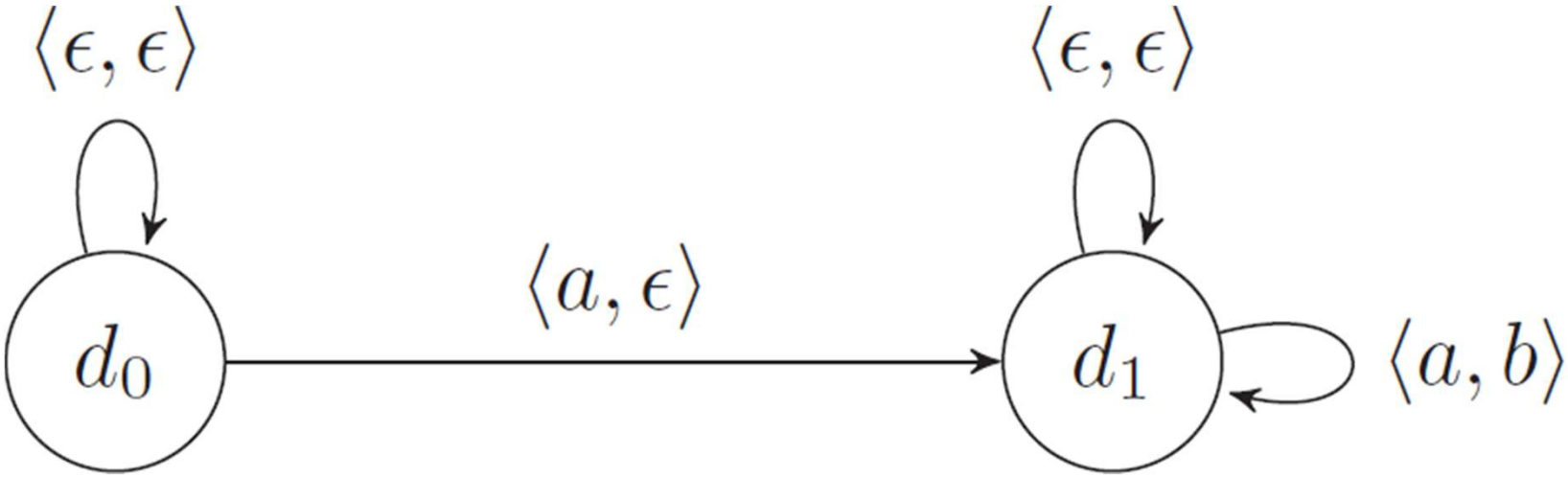
- Encode CCSL operators
  - Using *infinite*-state labeled transition systems
    - Code generation: Use (unbounded) integers to encode symbolically the unbounded number of states
    - Control: through lazy programming
      - no guarantee to terminate if the composition is actually unbounded
  - Use synchronization vectors to capture the coincidence-based relations

Stateless synchronous CCSL operators



Stateful synchronous CCSL operator

b = a $ 1



$X_a = 0$
$X_b = 0$

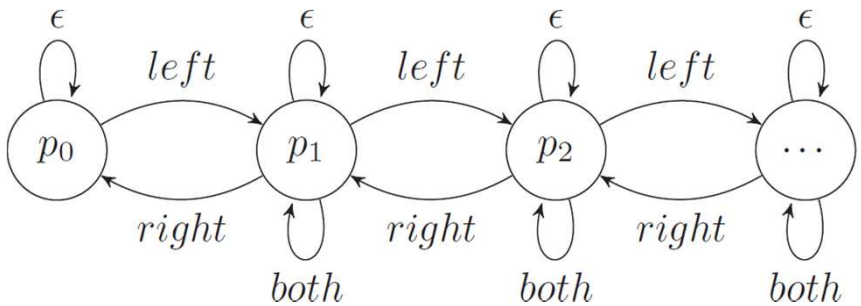
$X_a - X_b = 1$

Unbounded asynchronous operator

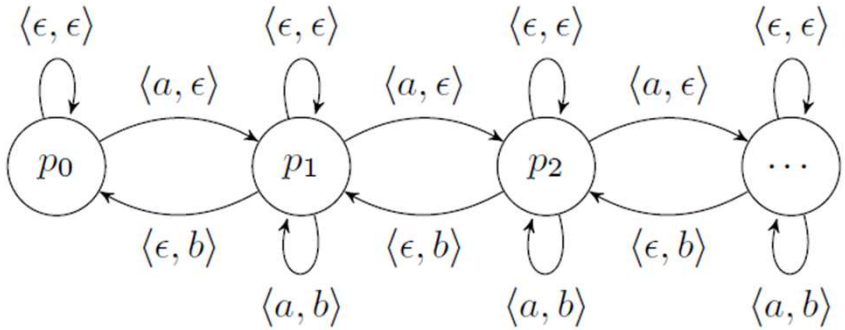
a ⊰ b

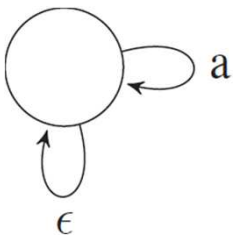
Infinite LTS: Precedes

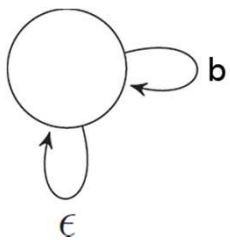
left ≺ right



Precedes || Clock$_a$ || Clock$_b$







Synchronization Vectors:

<left,a,ε>

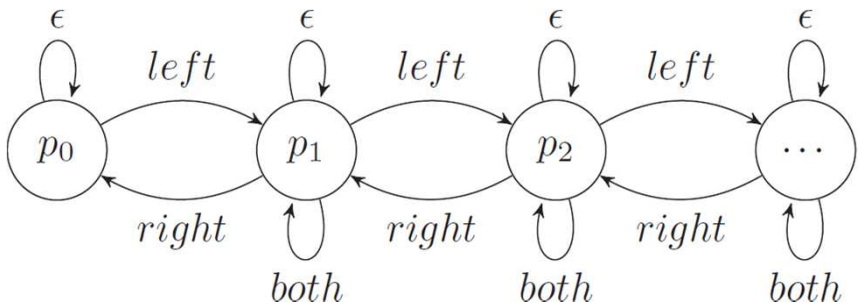
<right,ε,b>

<both,a,b>

<ε, ε, ε>

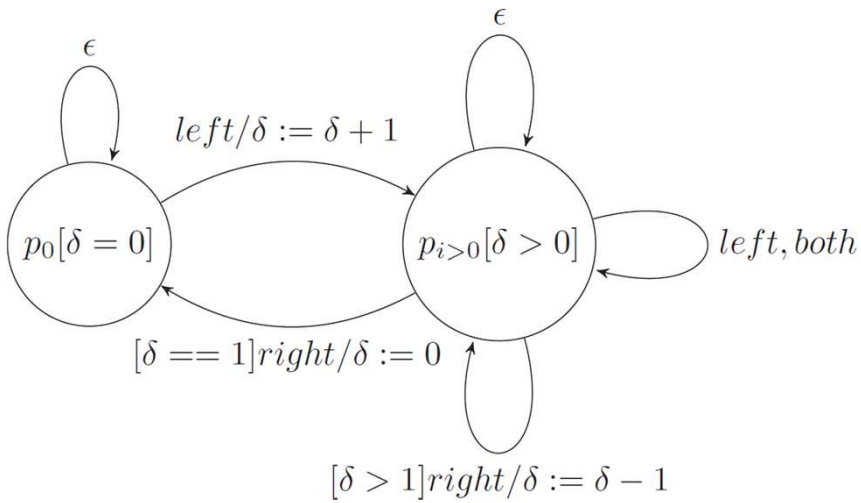
Unbounded asynchronous operator

Infinite LTS: left $\prec$ right



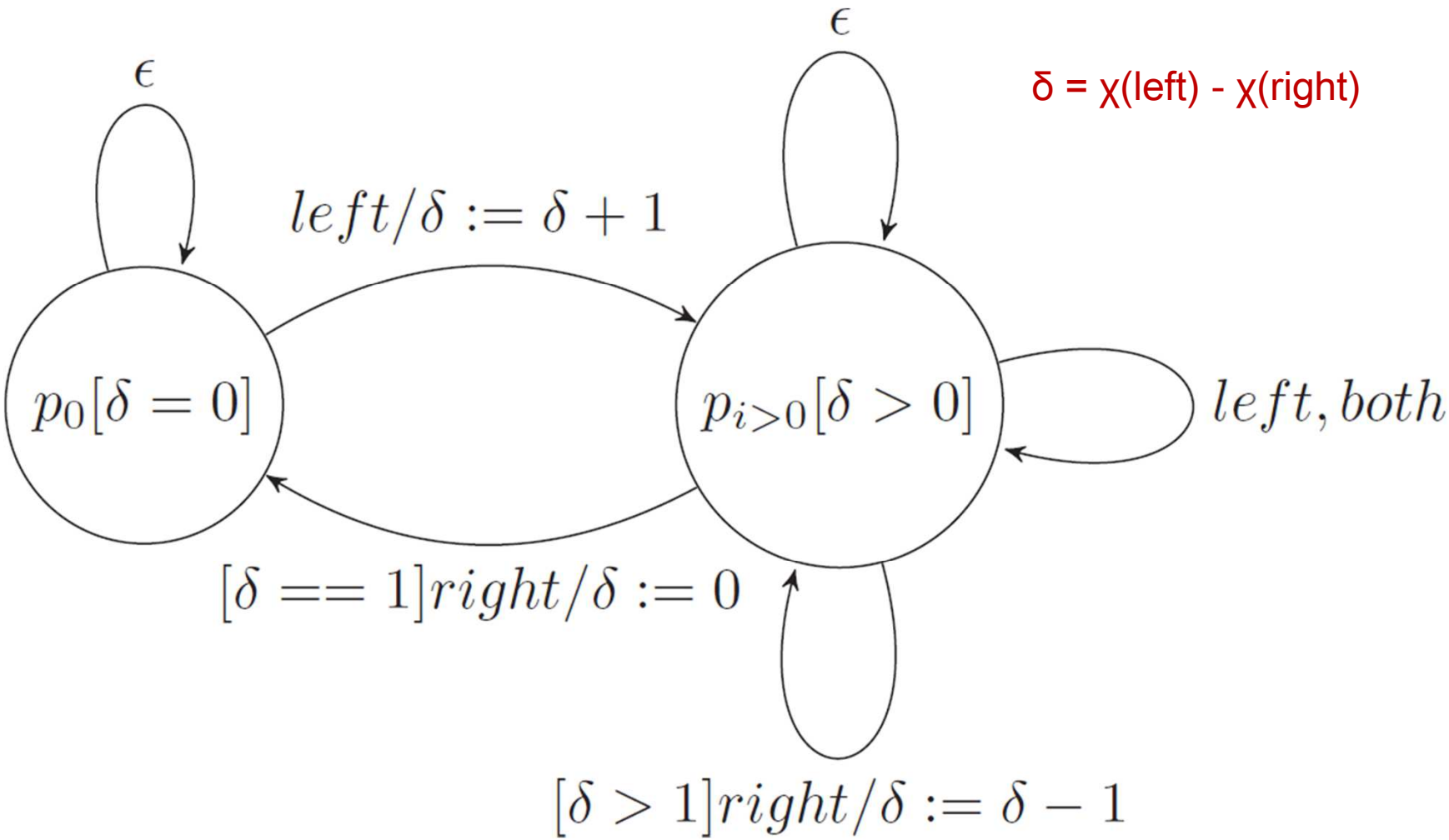
Folded with state variables: left $\prec$ right

δ = χ(left) - χ(right)



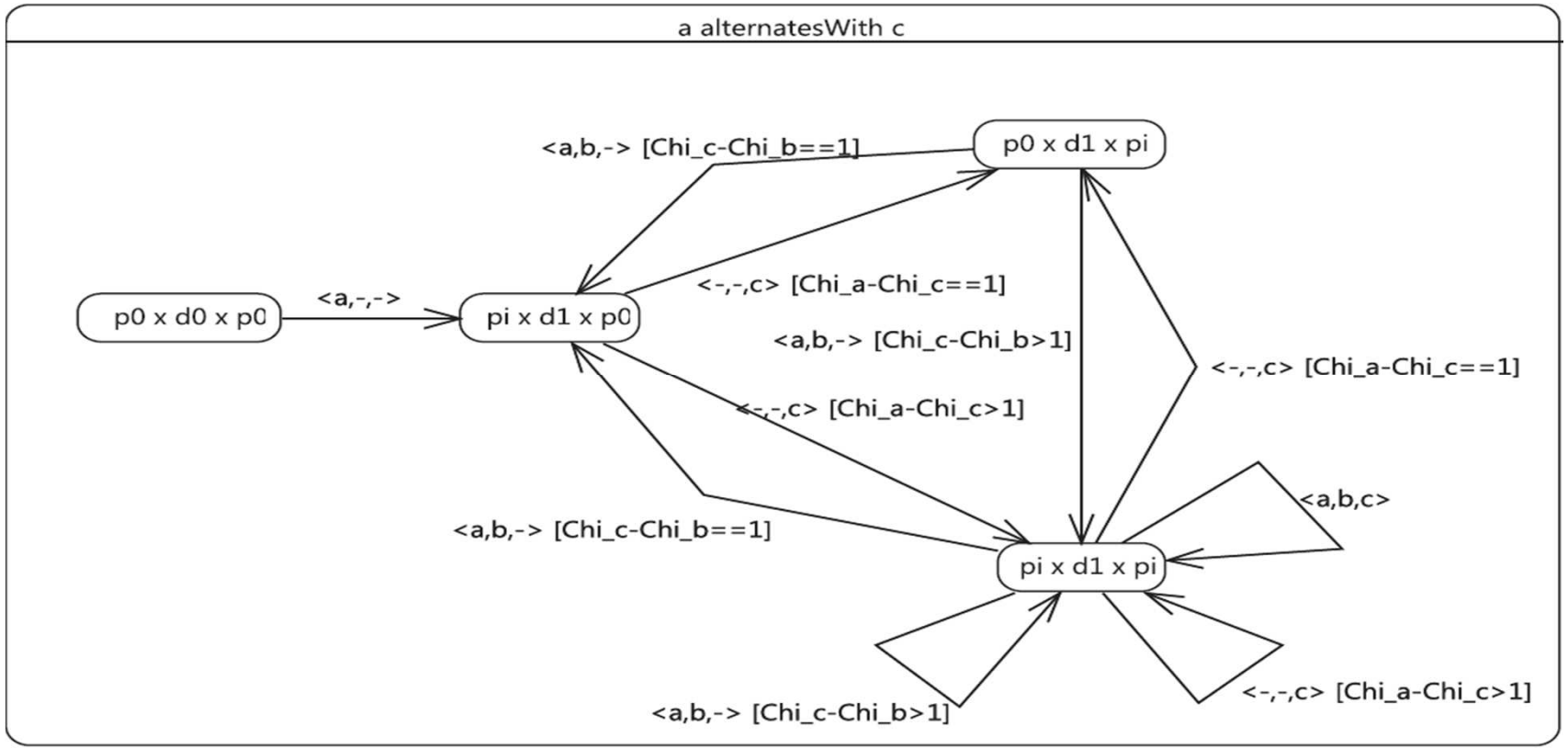
Folding infinite LTS

δ = χ(left) - χ(right)



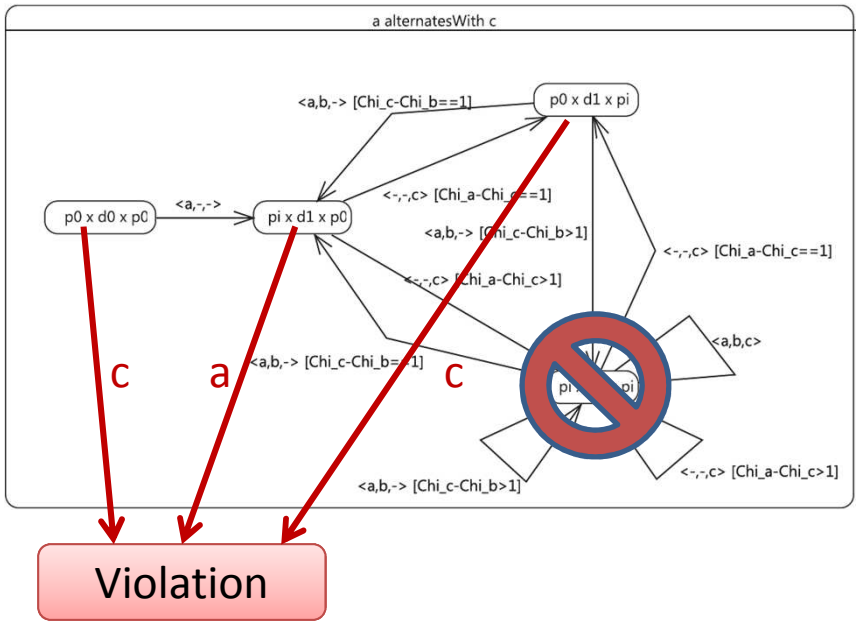
Example: alternation

- a ≺ c ^ b=a$1 ^ c ≺ b



Code generation

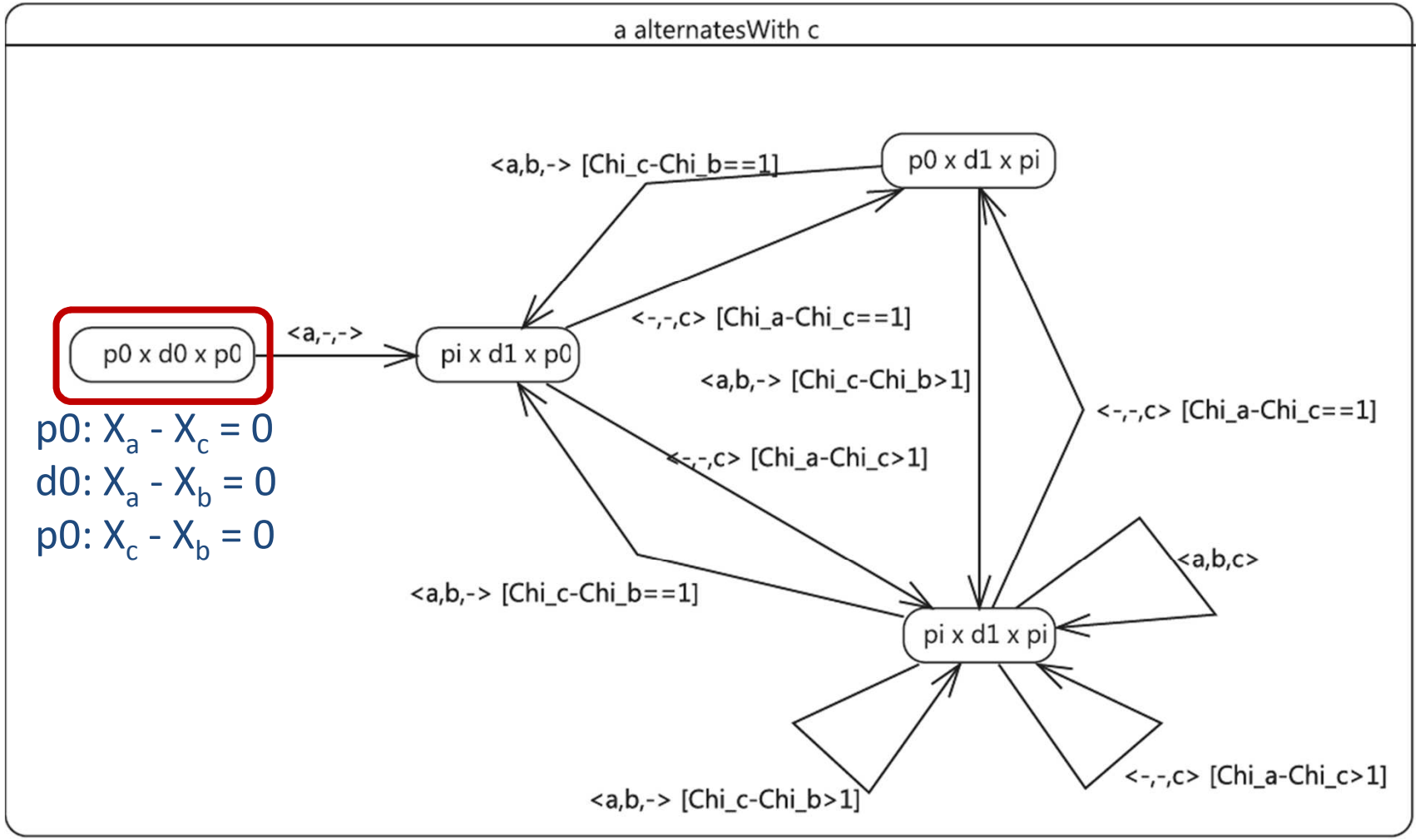
a ≺ c ^ b=a$1 ^ c ≺ b



Generation of Java code

```
class AlternatesComposed{
        static SynchronousTransitionSystem buildSts() {
                SynchronousTransitionSystem sts = new SynchronousTransitionSystem();
                sts.setName("a alternatesWith c");

                // events
                Event e0 = new Event(); e0.setName("a"); sts.addToEvents(e0);
                Event e1 = new Event(); e1.setName("c"); sts.addToEvents(e1);
                Event e2 = new Event(); e2.setName("b"); sts.addToEvents(e2);

                // states
                State s0 = new State(); s0.setName("p0xd0xp0"); s0.setTs(sts); sts.addToStates(s0);
                State s1 = new State(); s1.setName("pixd1xp0"); s1.setTs(sts); sts.addToStates(s1);
                State s2 = new State(); s2.setName("p0xd1xpi"); s2.setTs(sts); sts.addToStates(s2);

                // transitions
                {
                Transition t = new Transition(); t.setSource(s0); t.setTarget(s1);
                Trigger tr = new Trigger(); t.setTrigger(tr);
                tr.addToEvents(e0);
                s0.addToOutgoing(t);
                }
                {
                Transition t = new Transition(); t.setSource(s1); t.setTarget(s2);
                Trigger tr = new Trigger(); t.setTrigger(tr);
                tr.addToEvents(e1);
                s1.addToOutgoing(t);
                }
                {
                Transition t = new Transition(); t.setSource(s2); t.setTarget(s1);
                Trigger tr = new Trigger(); t.setTrigger(tr);
                tr.addToEvents(e0);
                tr.addToEvents(e2);
                s2.addToOutgoing(t);
                }
                sts.setInitial(s0);
                return sts;
        }

        public static void main(String[] args) {
                DTSRunner runner = DTSRunner.create(buildSts());
                runner.add(new DefaultSTSHandler());
                runner.fire(20);
        }
}
```

About invalid states

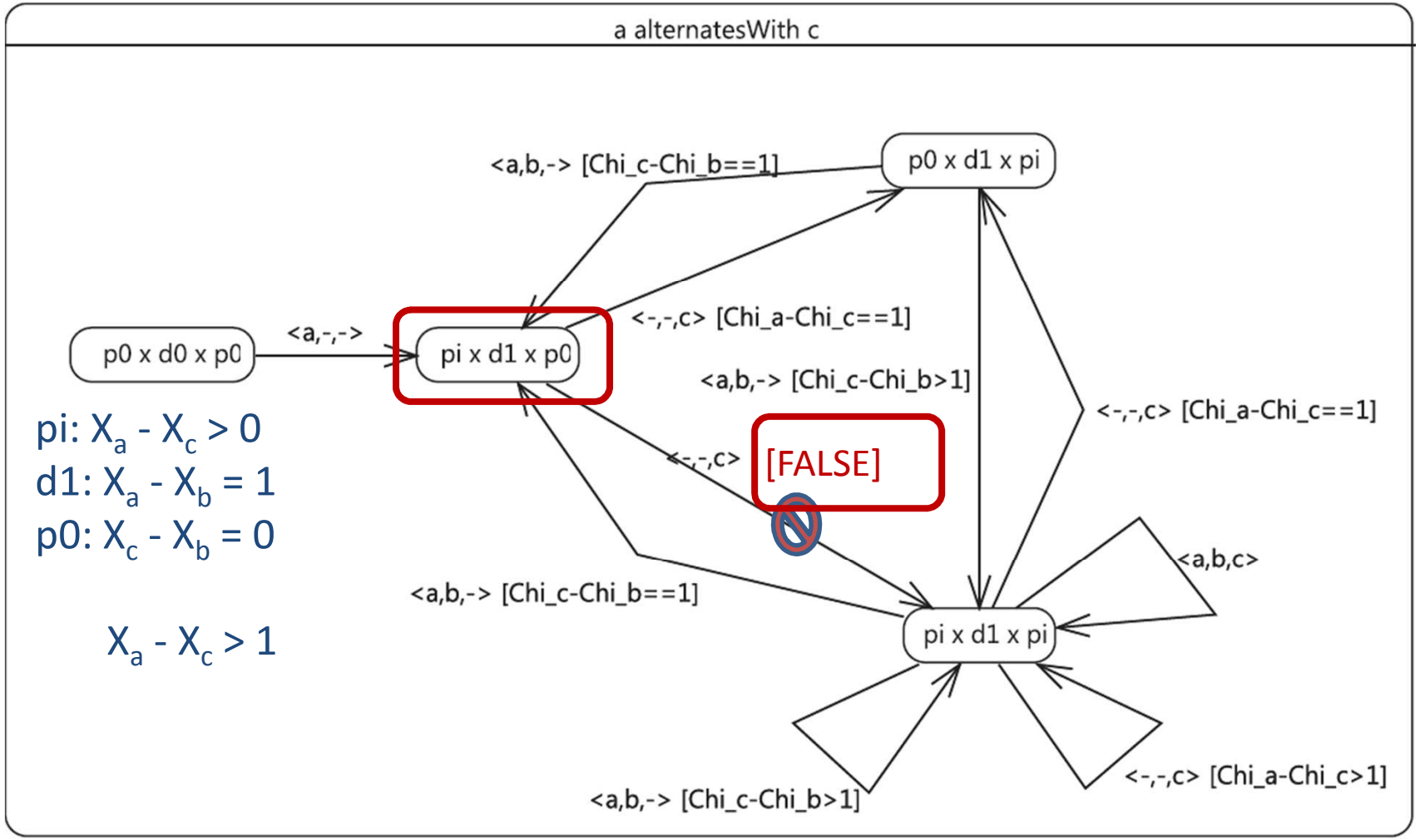


p0: $X_a - X_c = 0$

d0: $X_a - X_b = 0$

p0: $X_c - X_b = 0$

About invalid states



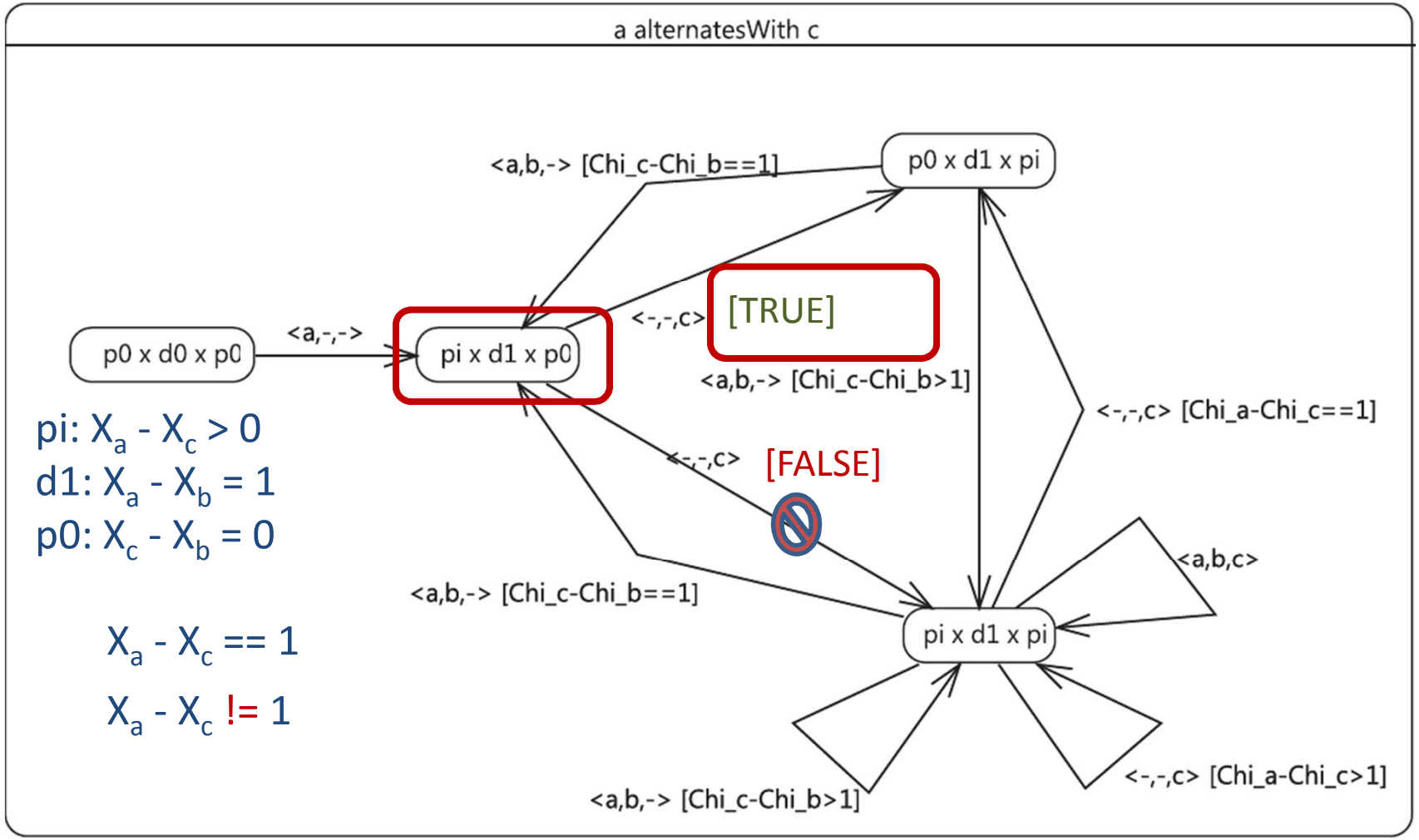
pi: $X_a - X_c > 0$

d1: $X_a - X_b = 1$

p0: $X_c - X_b = 0$

$X_a - X_c > 1$

About invalid states



pi: $X_a - X_c > 0$

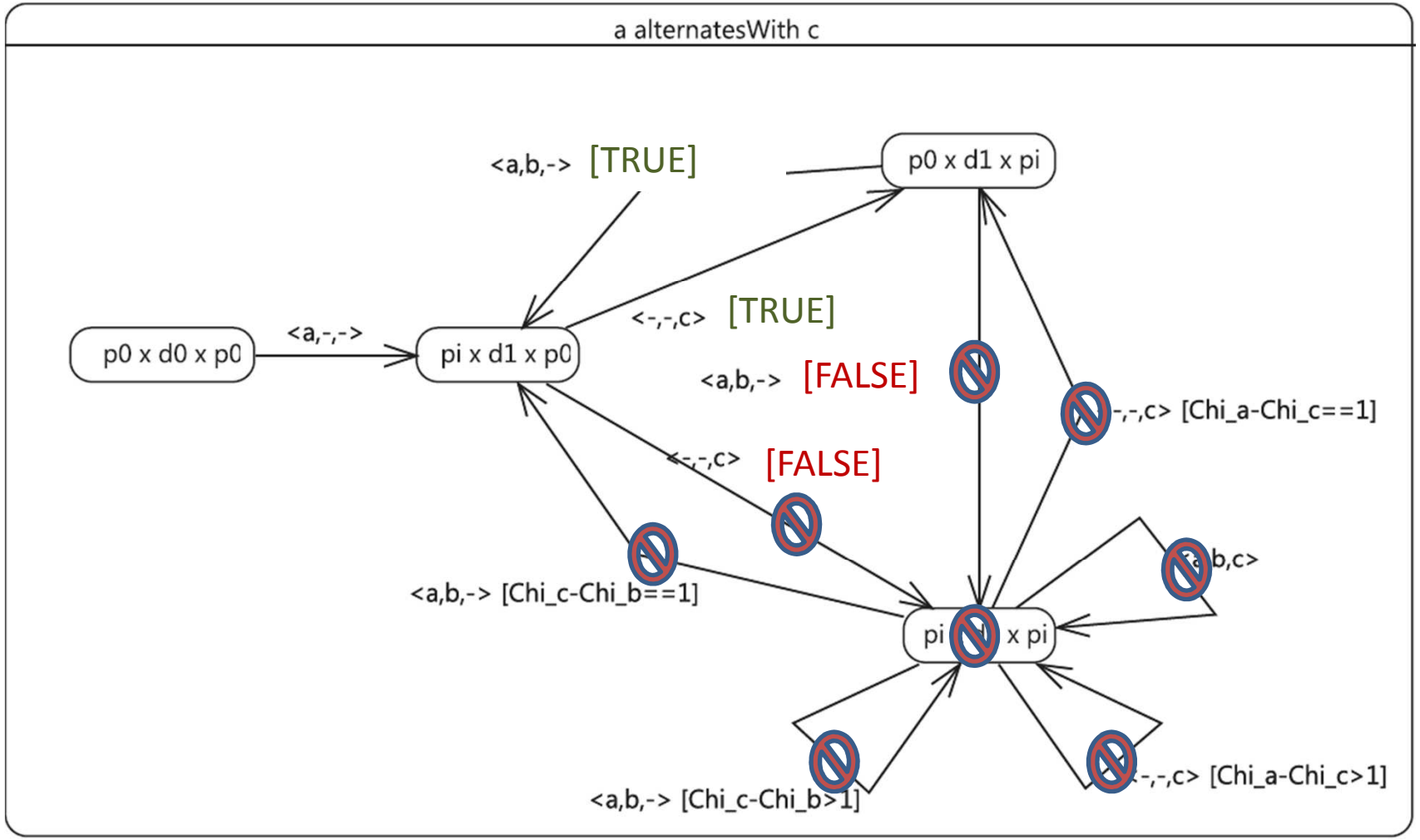
d1: $X_a - X_b = 1$

p0: $X_c - X_b = 0$

$X_a - X_c == 1$

$X_a - X_c$!= 1

About invalid states



Conclusion

- State-based semantics for CCSL
  - Allows for the code generation
  - Does not assume bounded operators as in previous works
- May be use to detect finite CCSL specifications
  - Integer Linear Programming
  - Needs ILP-tools working with infinite precision
    - Or other formalisms (counter systems, GAL)