ClockSystem: Embedding Time in Smalltalk

Ciprian Teodorov

ENSTA Bretagne, Brest, France

Overview

- Context & Motivations
- Logical Time Formalism
- ClockSystem: Logical Time in Smalltalk
- Example
- Conclusion & Perspectives

Context & Motivation

- **Parallel** platforms available (multi-core, GPU, www)
- More and more Parallel & Distributed apps

- General-purpose languages have **constructs** for *expressing* **concurrency** and *exploiting* **parallelism**

- Difficulties for reasoning about concurrency:
  - Low-level, implementation specific
  - Lack of formal semantics

Logical Time and Synchronous languages

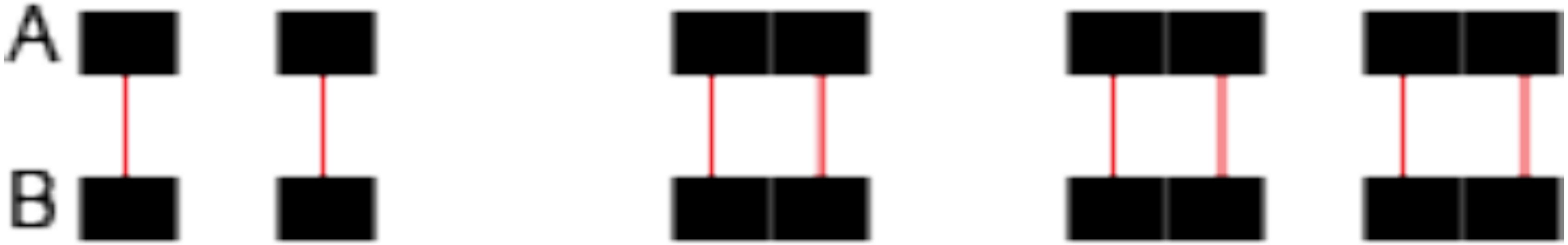
- *Logical Time* (Leslie Lamport '78)
  - Abstracts "physical" time as a **partial order of events**
  - **Multi-form**, the event need not be time related
- Enables to describe, manipulate and analyze interactions, communications, synchronizations between processes.
- Used in hardware, embedded and distributed systems
  - Signal, Lustre, Esterel, **CCSL**

Clock Constraint Specification Language (CCSL)

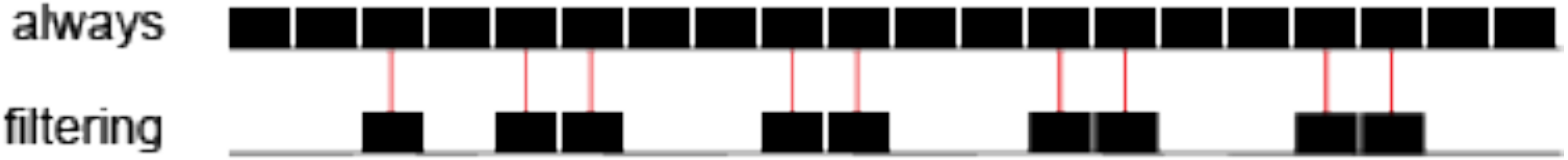
- Part of the OMG Marte UML2 profile
- Formally expresses timed behaviors
  - Relations: *precedence*, *coincidence*, *exclusion* …
  - Expressions: *intersection*, *union*, *filtering* …
- Usages:
  - specifying concurrency semantics
  - expressing timing requirements

CCSL primitives: Examples

A = B

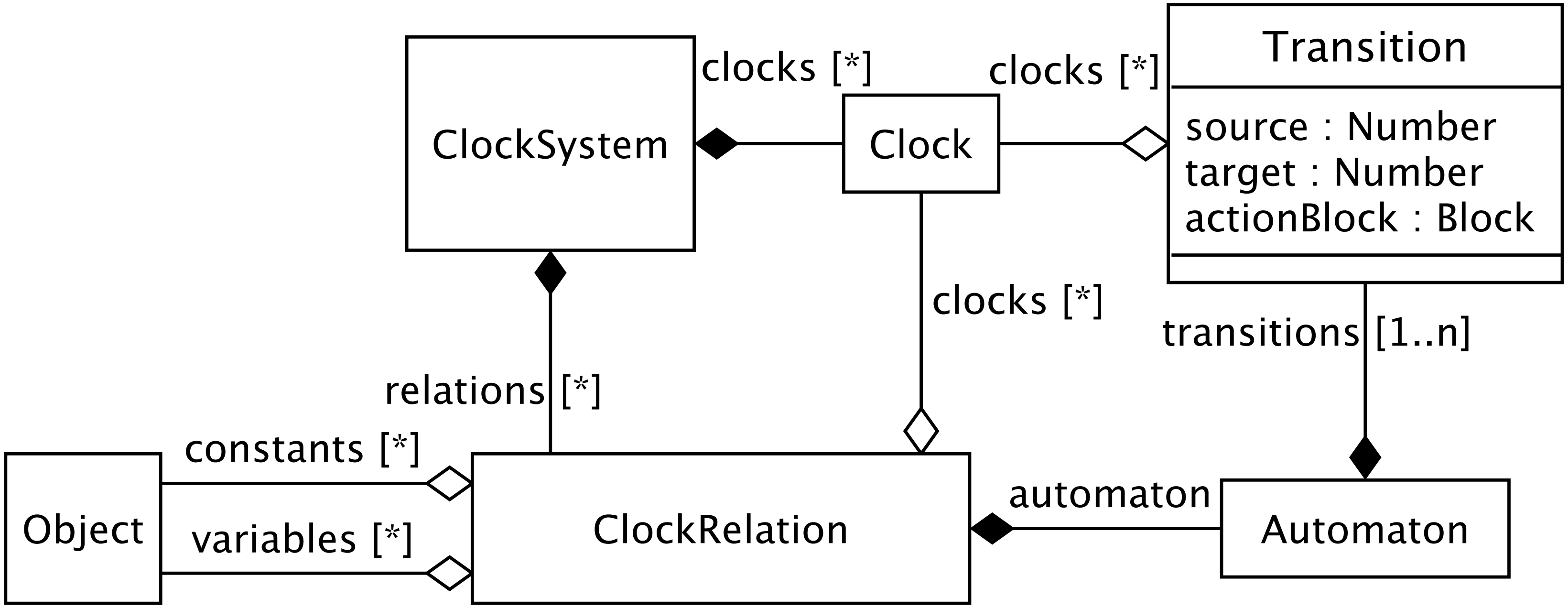


filtering := always ▼ $001(0110)^{\omega}$

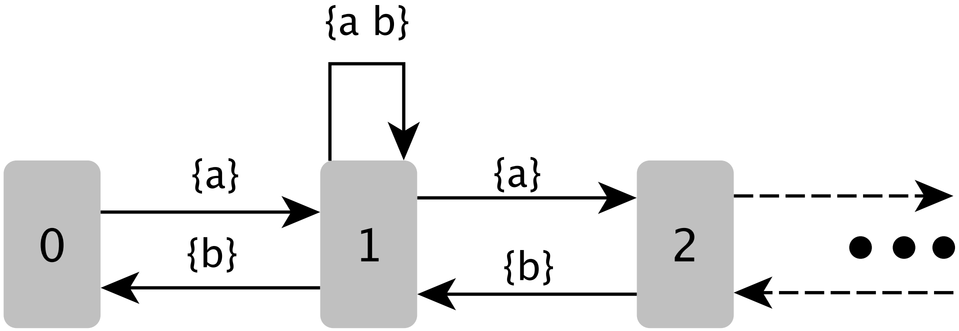


ClockSystem

- Logical Time embedded in Smalltalk
- Automata interpretation of CCSL primitives

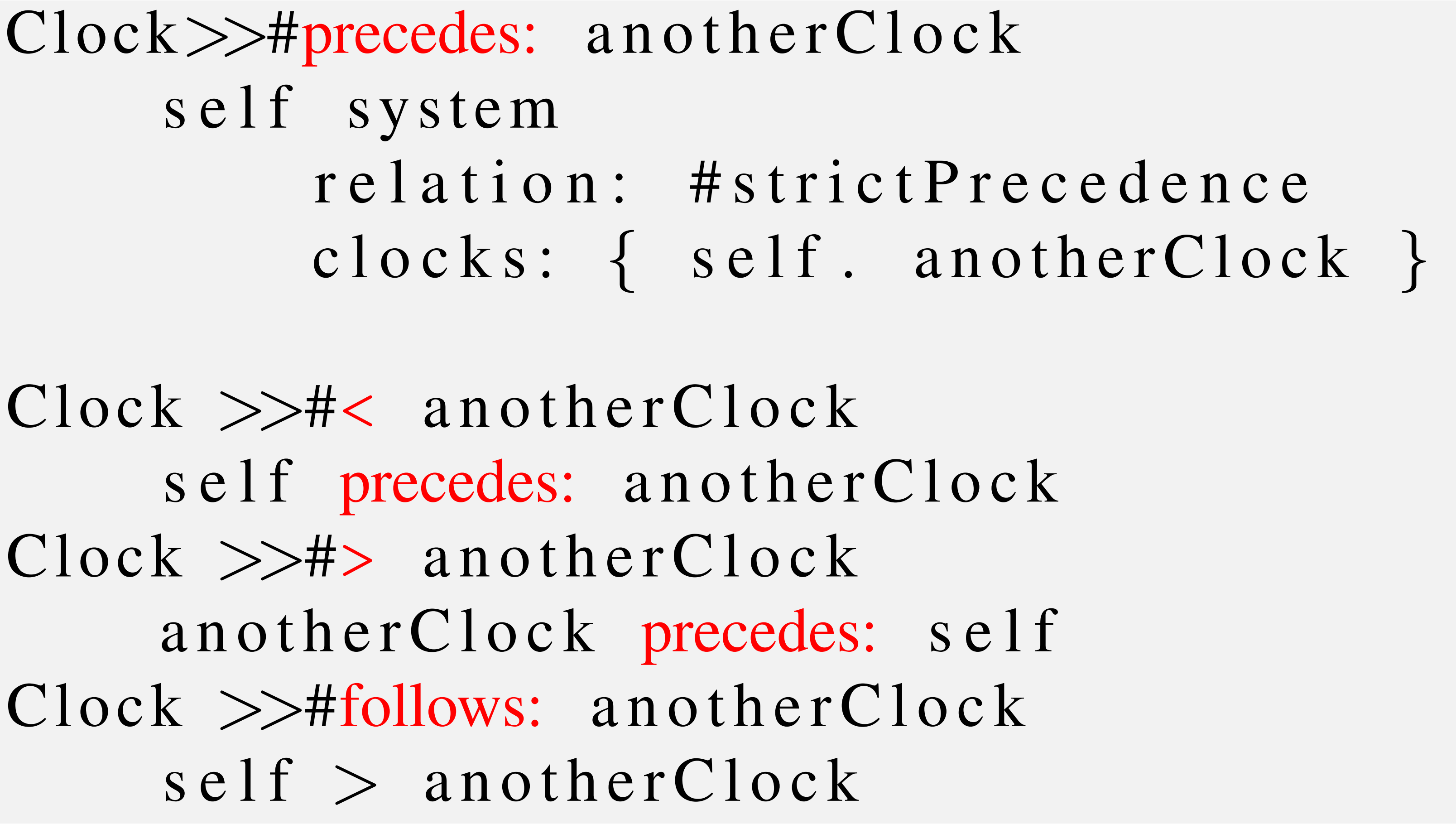


relDSL for primitives: StrictPrecedence (<)



```
KernelLibrary>>#strictPrecedence
    ^ [ :s :a :b |
    ”unbounded strict precedence”
    s = 0
        ifTrue:  [ {
            s -> (s + 1) when: {a} } ]
        ifFalse: [ {
            s -> s when: {a. b}.
            s -> (s + 1) when: {a}.
            s -> (s - 1) when: {b} } ] ]
```

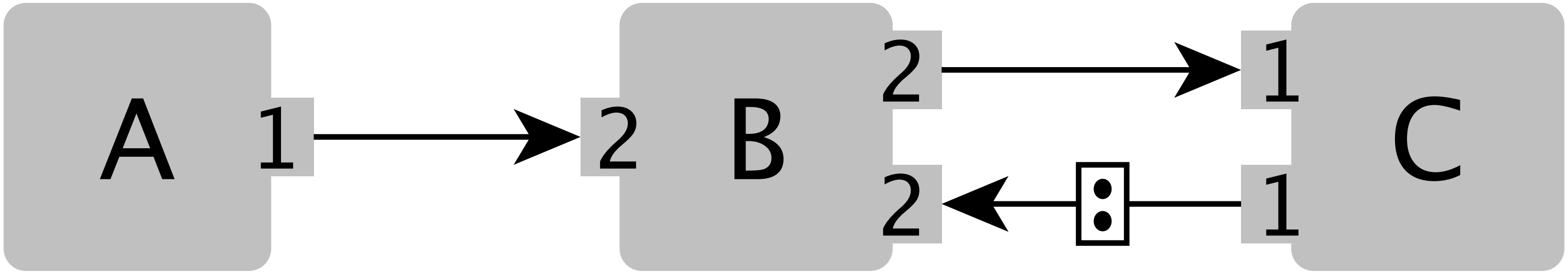
Constraints instantiation



```
Clock>>#precedes: anotherClock
    self system
        relation: #strictPrecedence
        clocks: { self. anotherClock }

Clock >>#< anotherClock
    self precedes: anotherClock
Clock >>#> anotherClock
    anotherClock precedes: self
Clock >>#follows: anotherClock
    self > anotherClock
```

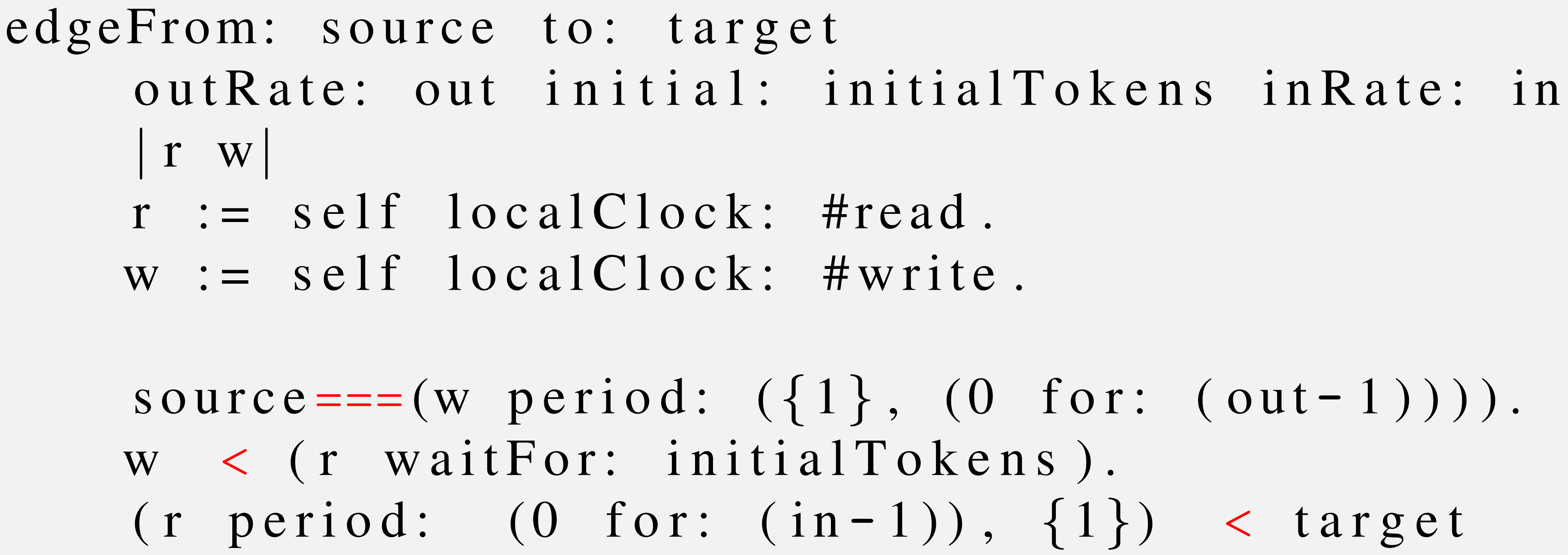
Synchronous Data Flow (SDF) Example



SDF Constraints: CCSL

```
def edge(clock source, clock target,
        int out, int initialTokens, int in) $\triangleq$
        clock read
        clock write
        $source = (write \blacktriangledown .(1.0^{out-1})^{\omega})$
        $\wedge\ write \prec \text{read} \; \$ \; \text{initialTokens}$
        $\wedge\ (read \blacktriangledown .(0^{in-1}.1)^{\omega}) \prec target$
```

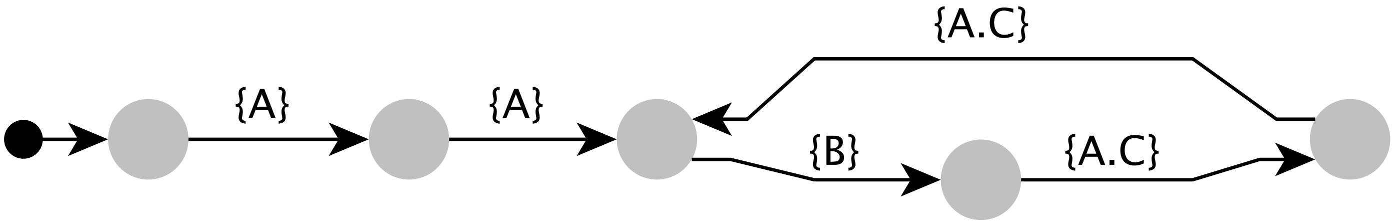
SDF Constraints: ClockSystem



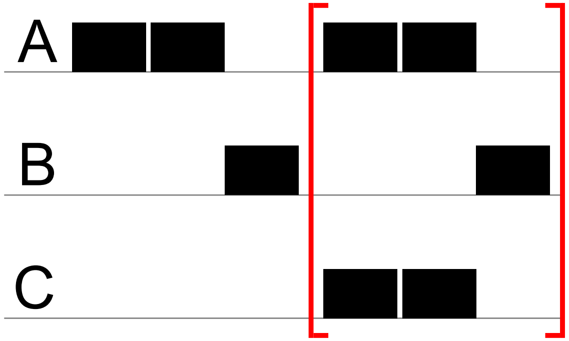
```
edgeFrom: source to: target
    outRate: out initial: initialTokens inRate: in
    |r w|
    r := self localClock: #read.
    w := self localClock: #write.

    source===(w period: ({1}, (0 for: (out-1)))).
    w  < (r waitFor: initialTokens).
    (r period:  (0 for: (in-1)), {1})  < target
```

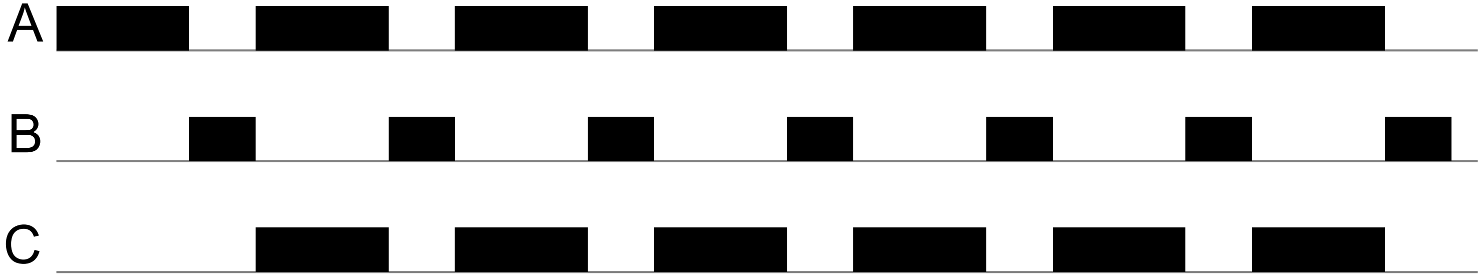
Simulation



(a) Periodic Trace Automaton

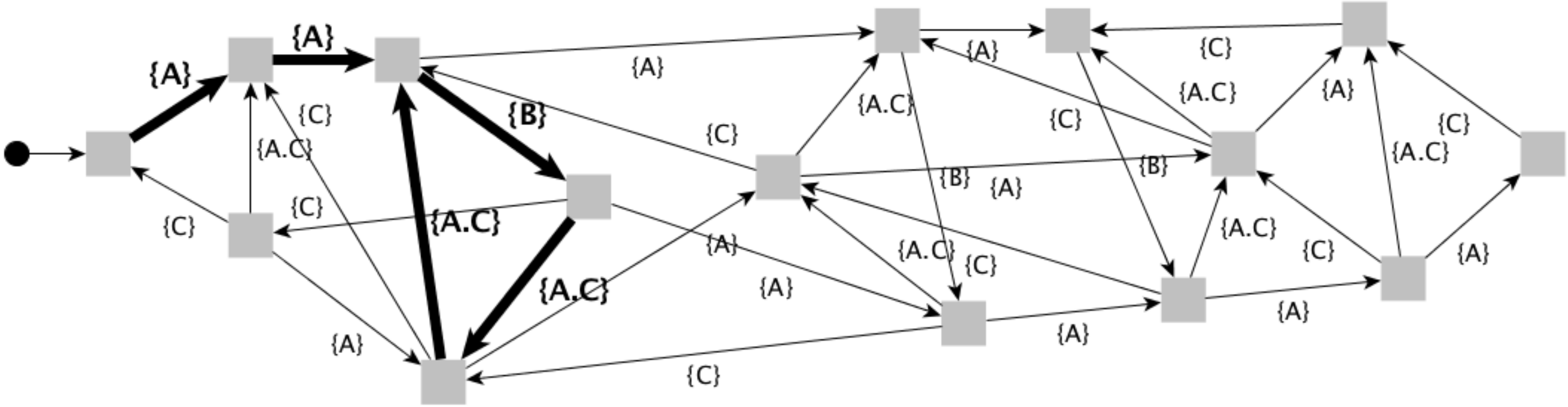


(b) Periodic Waveform

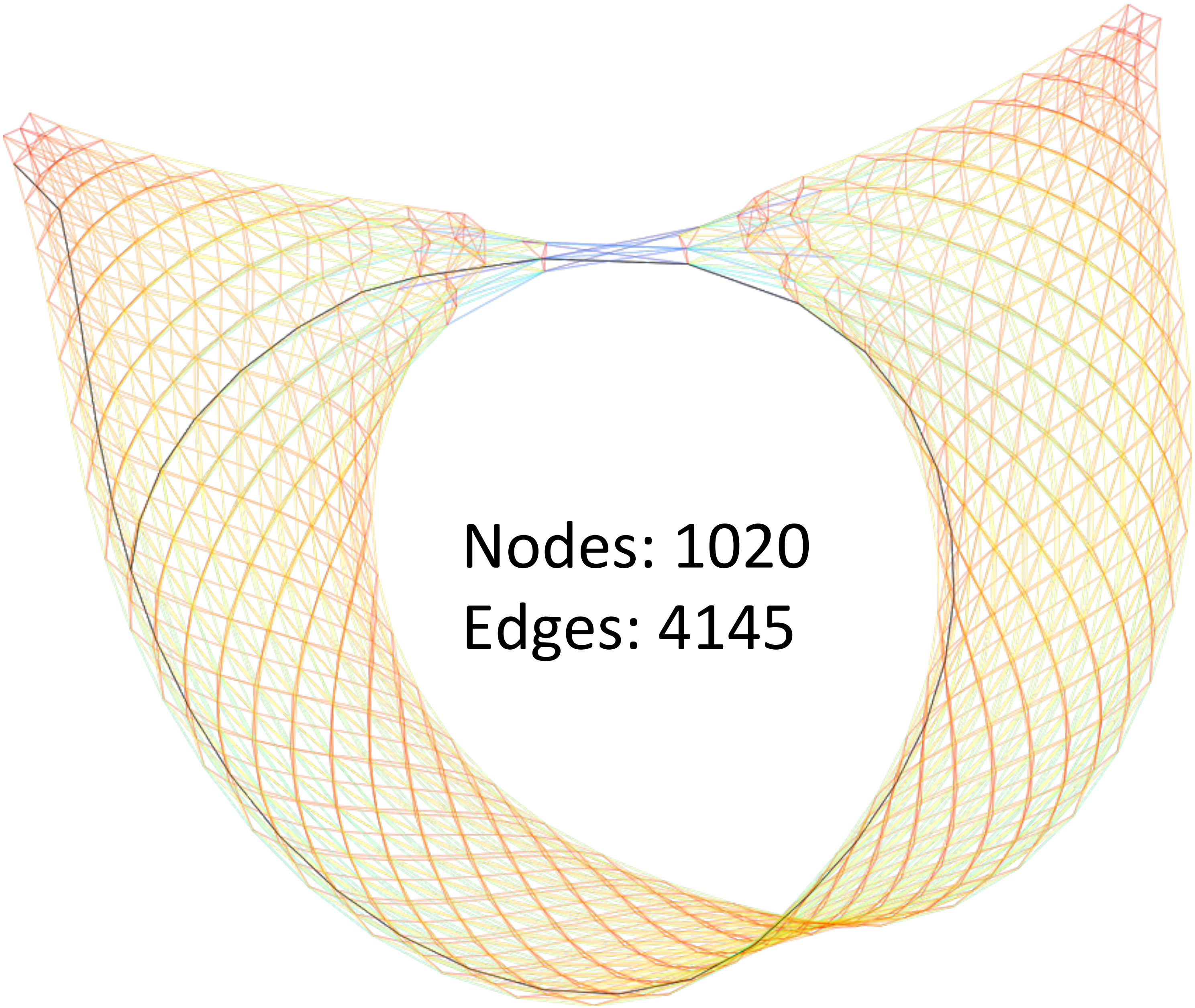


(c) Trace interpretation (21 steps)

Exhaustive Execution Analysis



Passive Acoustic Monitoring Application



Nodes: 1020

Edges: 4145

Conclusion

- Embedding of Logical Time in Pharo Smalltalk
- Extensible automaton-based formal kernel

- Flexible DSL through message-synonyms

- Usage Scenarios
  - Trace interpretation
  - model-checking
  - DSE
  - testing & monitoring

Future Work

testing & monitoring concurrent Smalltalk apps by intercepting reflectively generated events (like *var access*, method *activations*, etc)

- Support for dense-time representation
- Mechanisms for dynamically evolving systems
- Study the connection between ClockSystem constraints and state-space decomposition in model-checking context