An Experiment With Lustre and Real-Time Calculus

Introduction du cours

Matthieu Moy

Verimag
Grenoble
France

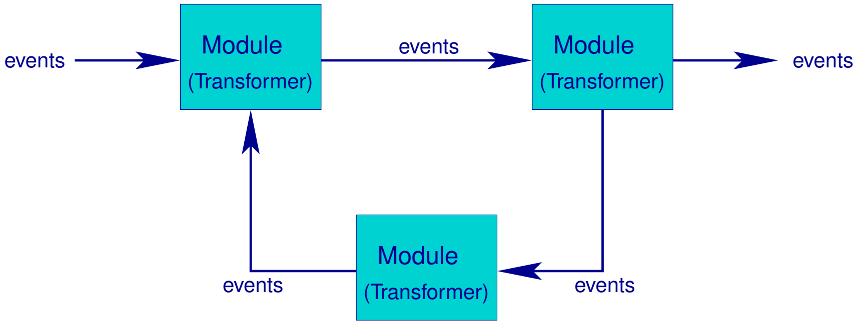
December 1, 2008

Summary

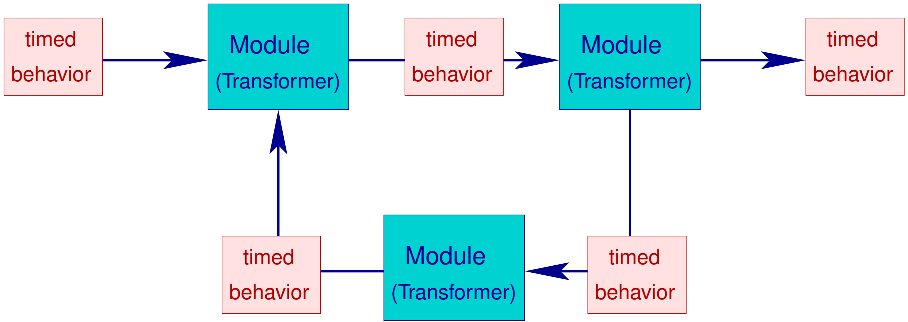
Motivations

- The goal: performance analysis
  - Timing
  - Energy (?)
- The tools: Formal methods
  - Will it scale?
- The context:
  - Background in simulation, synchronous systems
  - ... trying to work with performance models
- Who:
  - Verimag, “synchronous team”
  - ETHZ, Lothar Thiele and his team
  - (Combest project)

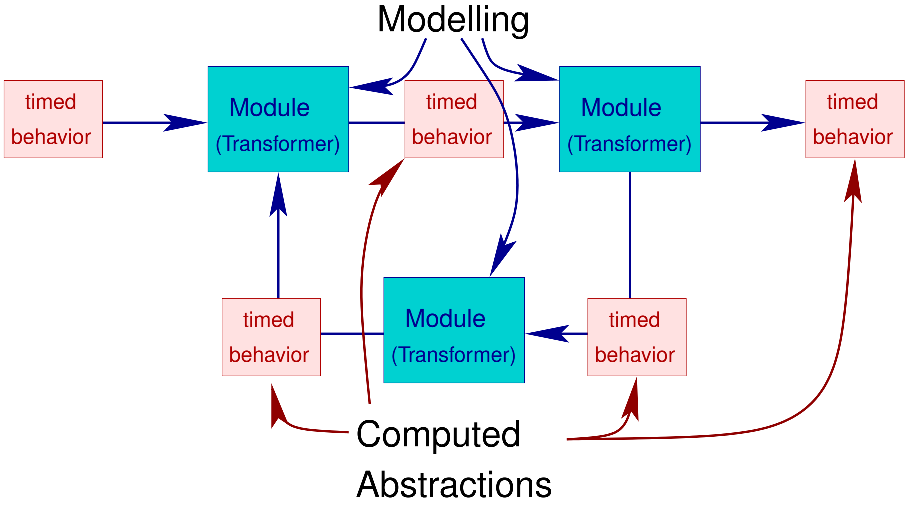
Modular Performance Analysis (MPA): The Big Picture



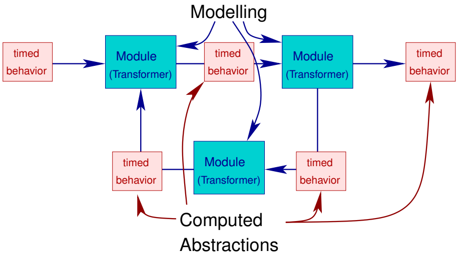
Modular Performance Analysis (MPA): The Big Picture



Modular Performance Analysis (MPA): The Big Picture

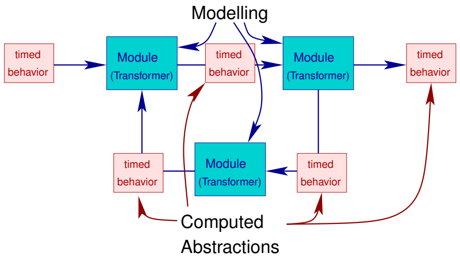


Modular Performance Analysis: Content



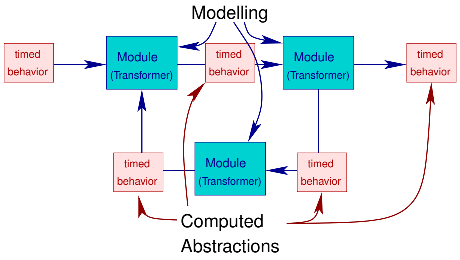
- What can “timed behavior” be?
  - Number of events per time unit?
  - Bounds for number of events?

Modular Performance Analysis: Content



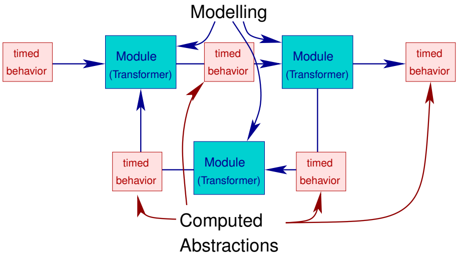
- What can “timed behavior” be?
  - Number of events per time unit?
  - Bounds for number of events?
  - MPA uses “arrival curves”.

Modular Performance Analysis: Content



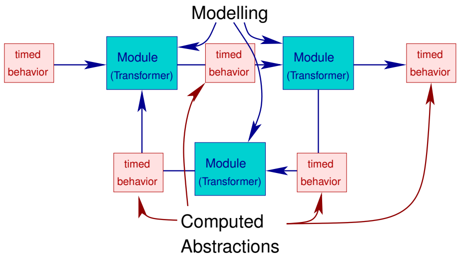
- What can "Modules" be?
  - FIFO + processing element?
  - "Service curve"

Modular Performance Analysis: Content



- What can “Modules” be?
  - FIFO + processing element?
  - “Service curve”
  - Can also be a “program”

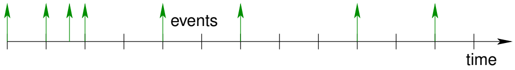
The Question...



Can we put Lustre in the modules?

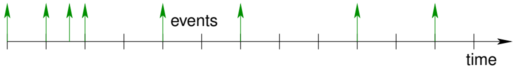
Summary

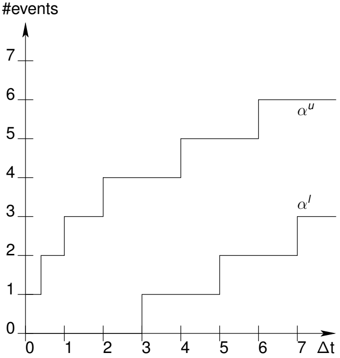
Arrival Curves



- $\alpha^u(t)$: max number of events in any window of size t.
- $\alpha^l(t)$: min number of events in any window of size t.

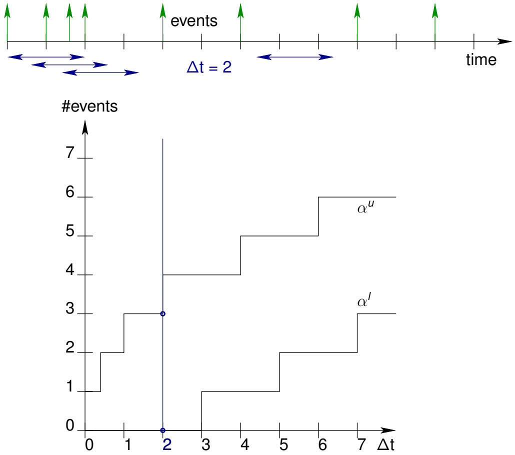
Arrival Curves





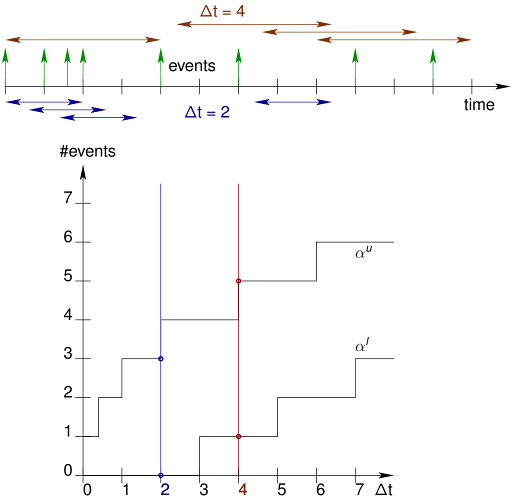
- $\alpha^u(t)$: max number of events in any window of size t.
- $\alpha^l(t)$: min number of events in any window of size t.

Arrival Curves



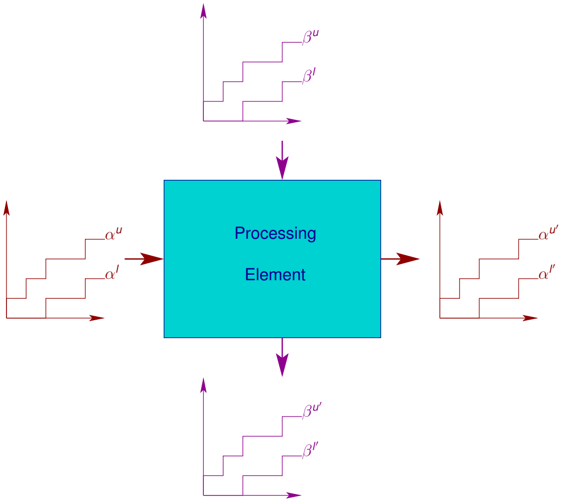
- $\alpha^u(t)$: max number of events in any window of size t.
- $\alpha^l(t)$: min number of events in any window of size t.

Arrival Curves

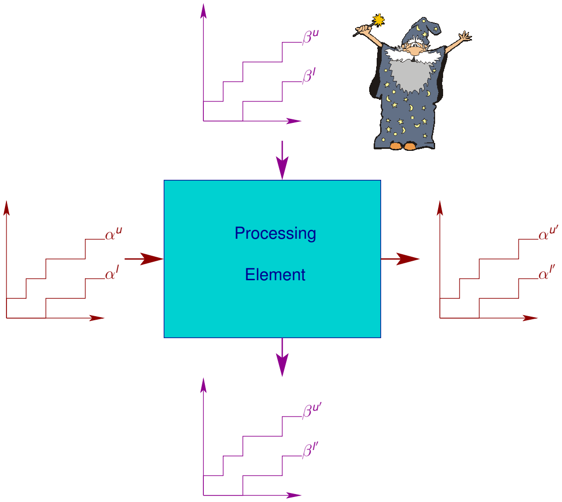


- $\alpha^u(t)$: max number of events in any window of size t.
- $\alpha^l(t)$: min number of events in any window of size t.

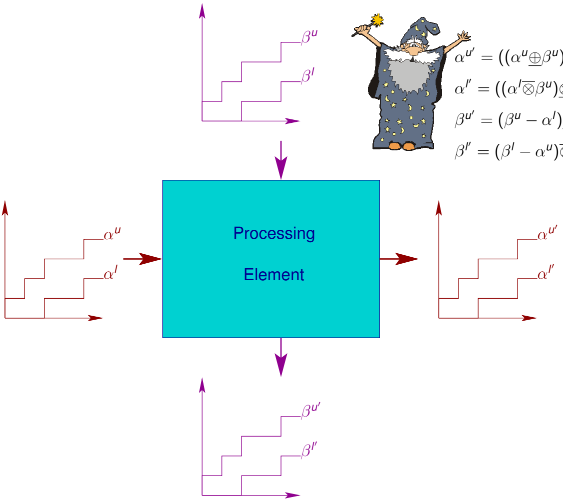
Service Curves



Service Curves



Service Curves



$$\alpha^{u'} = ((\alpha^u \underline{\oplus} \beta^u) \overline{\oplus} \beta^l) \wedge \beta^u$$

$$\alpha^{l'} = ((\alpha^l \overline{\otimes} \beta^u) \underline{\otimes} \beta^l) \wedge \beta^l$$

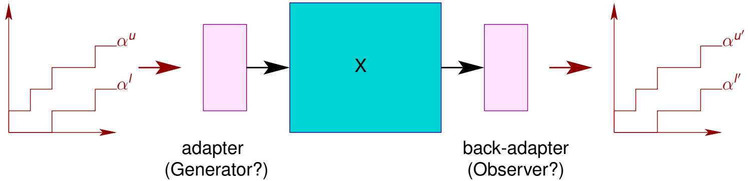
$$\beta^{u'} = (\beta^u - \alpha^l) \underline{\otimes} 0$$

$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\otimes} 0$$

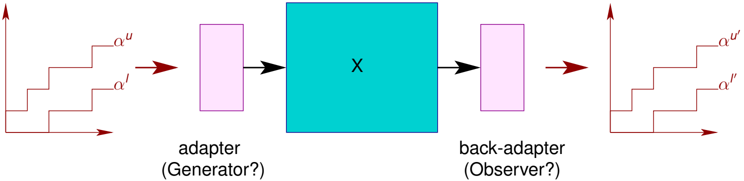
RTC: pros and cons

- Nice things with RTC
  - Can model: event flows, simple scheduling policies
  - Scales up nicely
  - Library of common behaviors available
  - Exact hard bounds
- Less nice things with RTC
  - Cannot model: state-based behavior, arbitrary scheduling policies.
  - Hardly models behavior not in the library ("Hire another Ph.D" approach).

Allowing more complex behaviors in MPA

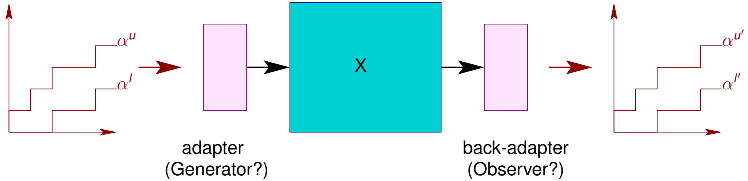


Allowing more complex behaviors in MPA



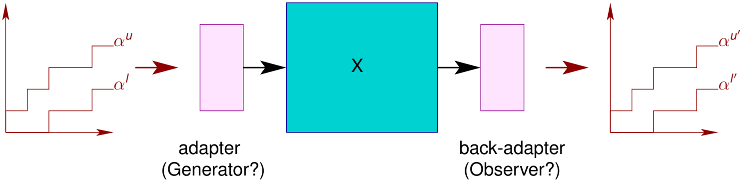
- X = Arbitrary program ⇒ testing (ETHZ)

Allowing more complex behaviors in MPA



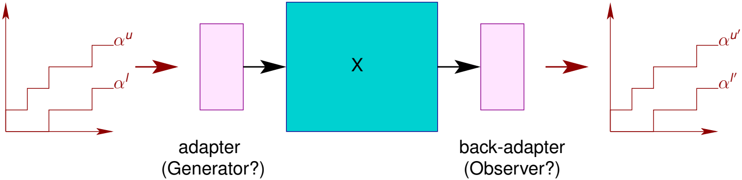
- X = Arbitrary program $\Rightarrow$ testing (ETHZ)
- X = Timed automata $\Rightarrow$ model-checking (Y. Liu in Verimag, K. Lampka in ETHZ, CATS tool by Uppsala).

Allowing more complex behaviors in MPA



- X = Arbitrary program ⇒ testing (ETHZ)
- X = Timed automata ⇒ model-checking (Y. Liu in Verimag, K. Lampka in ETHZ, CATS tool by Uppsala).
- X = Lustre ⇒ why we're here now.

Allowing more complex behaviors in MPA



- Adapted for systems where the complex behavior is local
- Scales nicely if the complex behavior “islands” are small enough.
- But: loss of information on the way back to RTC!

Summary

Reminder about Lustre

- data-flow, synchronous language

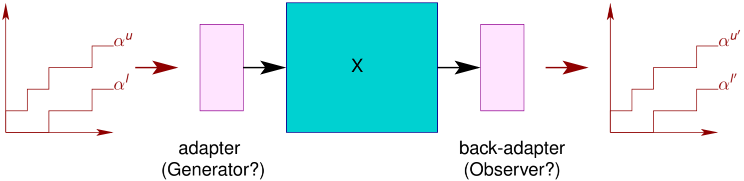
```
node counter(x: bool) returns (y: int)
let
  y = 0 -> (if x then pre(y) + 1 else pre(y));
tel
```

pre(y) value of y at the previous instant

x -> y x at the first clock tick, y otherwise.

Summary

Allowing more complex behaviors in MPA



Lustre in MPA: Why

- First exercise to understand MPA
- Use of a real programming language to program the behavior.
- Use of abstract interpretation tools (may scale better that timed-automata model-checking).

Lustre in MPA: The approach

- The question:

  Given a stream of events conforming to α^u, α^l, what is the best provable curve $\alpha^{u'}, \alpha^{l'}$ that the output stream conforms with?

Lustre in MPA: The approach

- The question:

  Given a stream of events conforming to α^u, α^l, what is the best provable curve $\alpha^{u'}, \alpha^{l'}$ that the output stream conforms with?

- Natural approach: Generate a stream conforming to α^u, α^l, and discover the invariant on the output.

Lustre in MPA: The approach

- The question:

  Given a stream of events conforming to α^u, α^l, what is the best provable curve $\alpha^{u'}, \alpha^{l'}$ that the output stream conforms with?
- Natural approach: Generate a stream conforming to α^u, α^l, and discover the invariant on the output.
- Simpler approach: Given an arbitrary input stream, find the best $\alpha^{u'}, \alpha^{l'}$ such that we can prove

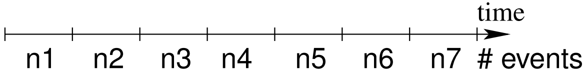
$$(\text{input} \models \alpha^u, \alpha^l) \quad \Rightarrow \quad (\text{output} \models \alpha^{u'}, \alpha^{l'})$$

  - We find the curve using a binary search, point by point,
  - We need only observers.

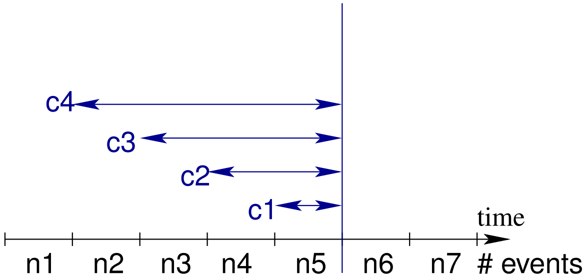
Lustre in MPA: How

- Limitations:
  - Discrete time
  - Discrete event
  - Finite arrival curves
- $\Rightarrow$ arrival curves are merely arrays of integers.

RTC Observer in Lustre: the idea

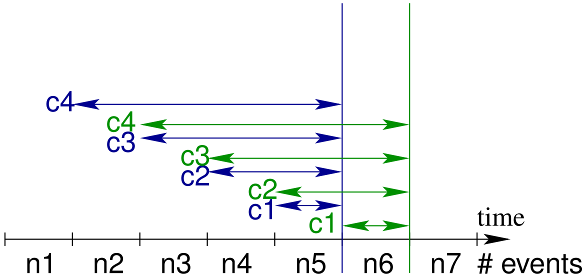


RTC Observer in Lustre: the idea



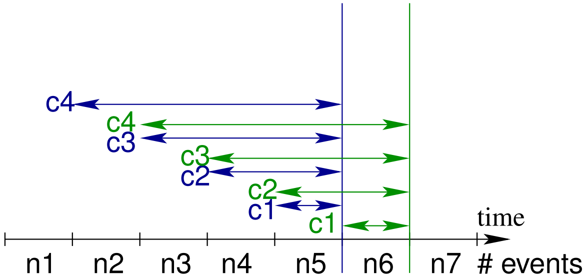
- Key ideas:
  - At time t, check time windows $[t - \Delta, t]$.

RTC Observer in Lustre: the idea



- Key ideas:
  - At time t, check time windows $[t - \Delta, t]$.
  - At time $t + 1$, reuse counters for time t.

RTC Observer in Lustre: the idea



c2 = c1 +n6

c3 = c2 +n6

- Key ideas:
  - At time t, check time windows $[t - \Delta, t]$.
  - At time $t + 1$, reuse counters for time t.

RTC Observer in Lustre: the code

```
-- deterministic observer, with 3 counters
-- (one for each size of interval)
node AC_det (i: int) returns (OK: bool);
  count1, count2, count3: int;
let
  count1 = i;
  count2 = i->(pre(count1) + i);
  count3 = i->(pre(count2) + i);
  OK =  m1 <= count1 and count1 <= M1
    and m2 <= count2 and count2 <= M2
    and m3 <= count3 and count3 <= M3
    and (true -> pre(OK)); -- never be true again
                           -- after being false once.
tel
```

(modulo uninteresting details)

RTC and Lustre: the Main Node

```
node main(in_seq: int)
returns (ok: bool)
var
  out_seq: int;
  in_ok: bool;
  out_ok: bool
let
  ok = out_ok or (not in_ok);
  out_ok = output_observer(out_seq);
  out_seq = transformer(in_seq);
  in_ok = input_observer(in_seq);
tel
```

(modulo uninteresting details)

Writing the module in Lustre: Example

```
-- simplest transformer ever:
-- process everything immediately!
node trivial_transformer (in_seq: int)
                  returns (out_seq: int)
let
  out_seq = in_seq;
tel
```

Writing the module in Lustre: Example (2)

```
-- shaper: process as fast as possible, but no
-- faster than max_speed events per ticks.
-- Accumulate other events in a buffer.
node queue_transformer (in_seq: int; max_speed: int)
returns (out_seq: int)
var
  backlog: int; work: int; empty_queue: bool;
let
  -- things to do at the current instant (new + past)
  work = in_seq -> (in_seq + pre(backlog));

  -- whether we'll empty the queue at the current instant.
  empty_queue = (work <= max_speed);

  out_seq = if (empty_queue) then work else max_speed;
  backlog = if (empty_queue) then 0 else work - out_seq;
tel
```

Causality Issue

- We wanted:

$$(\text{input} \models \alpha^{u}, \alpha^{l}) \quad \Rightarrow \quad (\text{output} \models \alpha^{u'}, \alpha^{l'})$$

- We wrote (Lustre):

```
ok = out_ok or (not in_ok);
```

Causality Issue

- We wanted:

$$(\text{input} \models \alpha^u, \alpha^l) \quad \Rightarrow \quad (\text{output} \models \alpha^{u'}, \alpha^{l'})$$

- We wrote (Lustre):

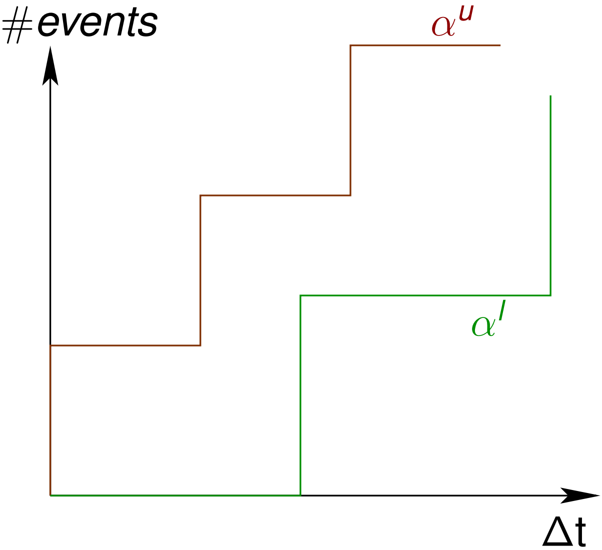
```
ok = out_ok or (not in_ok);
```

- Not equivalent in general:

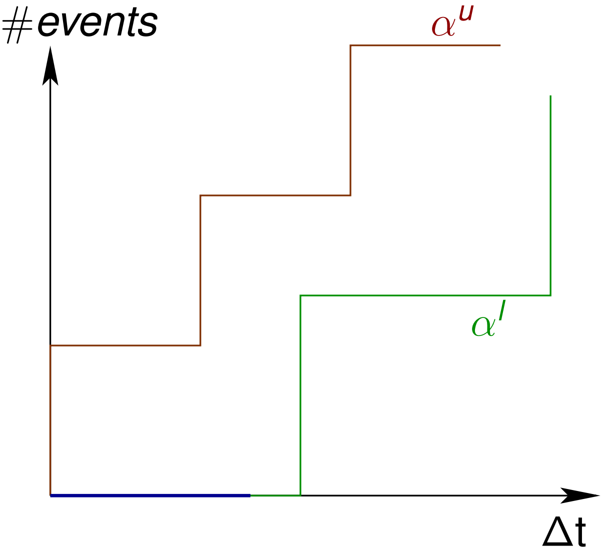
$$\text{always(in\_ok)} \Rightarrow \text{always(out\_ok)} \quad \neq \quad \text{always(in\_ok} \Rightarrow \text{out\_ok)}$$

- Condition on in_ok must be causal
  i.e. any execution verifying in_ok can be continued indefinitely.

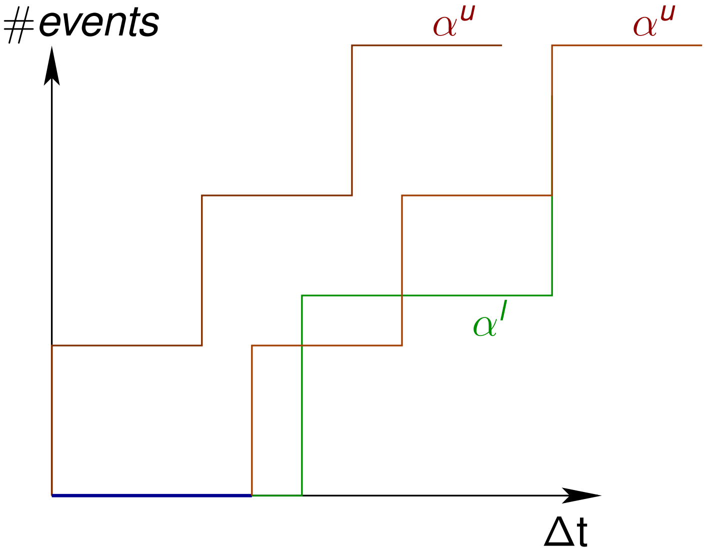
Causality problem: Forbidden Regions



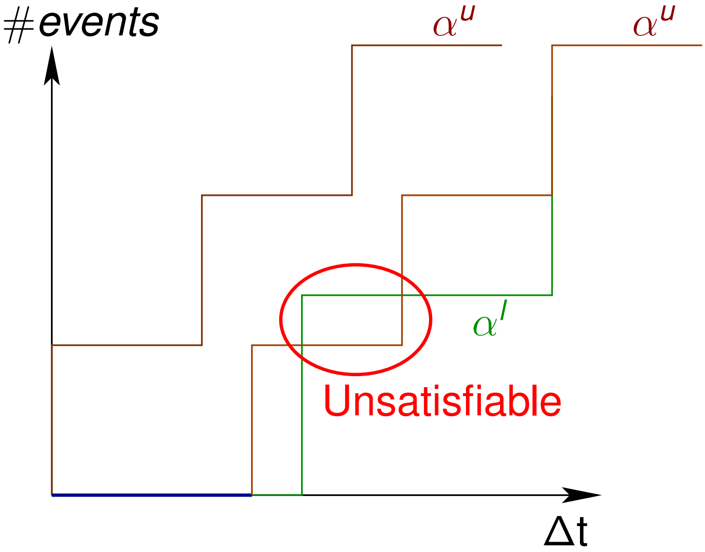
Causality problem: Forbidden Regions



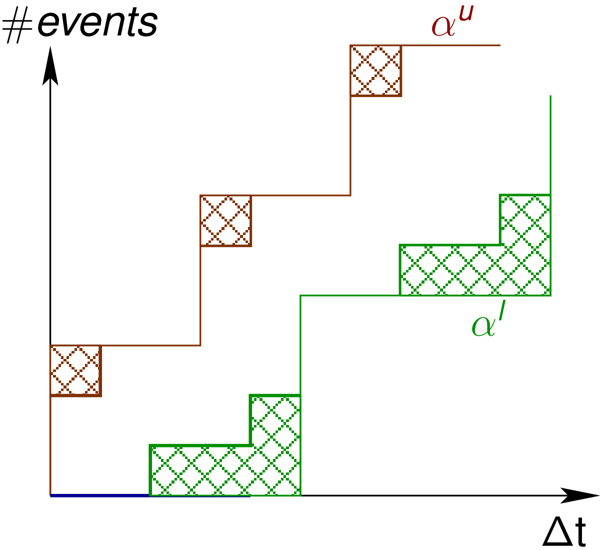
Causality problem: Forbidden Regions



Causality problem: Forbidden Regions



Causality problem: Forbidden Regions



Causality problem: our solution

- For a given α^u, α^l, one can compute another α^{u*}, α^{l*} which is causal, and accepts the same behaviors.

```
while (not fixed point)
   remove unreachable regions
   make the curve sub-additive
end while;
```

- α^{u*}, α^{l*} is tighter that α^u, α^l
- α^{u*}, α^{l*} is indeed the tightest possible pair of arrival curves.

Summary

Summary: the ac2lus toolbox

- Works with discrete-time, discrete-event, finite arrival curves.
- Can generate deterministic observer in Lustre.
- Curve normalization to make the curves causal before generating the observer.
- Compute the output curve with a binary search, using nbac.
- A few simple transformers implemented.

Remaining Issues

- Analysis still slow and limited
- Loss of information when computing output arrival curves
- Binary search can probably be replaced with invariant discovery.

Future Works

- Try tools other than nbac (aspic?)
- Try variations of the approach
  - Generators instead of observer
  - Non-deterministic observer
  - Specific generator/observers for classes of curves
- Performance/precision comparison with other approaches (timed automata, pure RTC, ...).