Temporal and Functional Analysis

UNS International Master1 Lecture 4

Frédéric Mallet

Robert de Simone

DataFlow Process Networks

Conflict-freeness !

- Main objective (with several equivalent formulations):
  - Have concurrency and functional determinism
  - Be speed/latency-independent (for communications and computations),
  - and so reproducible (basically all executions are just different scheduling linearizations of the same concurrent partial-order trace)
  - Monotoneous (having more initial data/tokens cannot cause less behaviors)

Conflict example:

```
await
  case chan_u? do P end
||
  case chan_v? do Q end
```

zoology

- **Marked Graphs** (aka Event Graphs)
  - "pure data flow", unit data size
- **Synchronous DataFlow** (SDF), Weighted MGs
  - (constant) sizes of data taken into account
- **CycloStatic DataFlow** (CS DF)
  - Cyclically changing data sizes
- **Boolean DataFlow** (BDF)
  - Conditional switches for control
- **KRG**s (K-periodically Routed marked Graphs, our local effort)
  - BDF with ultimately cyclostatic switching conditions
- **Kahn Process Networks**
  - Sequential programs inside the nodes

Marked Graphs

- Computation nodes, communication arcs (channels)
- Execution/firing: consumes one data on each input channel, produces one on each output one. By default takes one unit time
- Initial state/marking: number of data/token in each channel
- Bounded channels can be modeled (with back arcs)

- Definitions:
  - Latency of a graph cycle (or a path): its length
  - Latency of the graph (if cyclic): max length amongst its cycles

  - (structural) Throughput of a cycle: Σ tokens / Latency
  - Throughput of a (cyclic) graph: min throughputs of its cycles

Marked Graphs: results

- *Trivia*: number of tokens in a cycle remains unchanged
- **MG live** : iff at least one token in each cycle initially

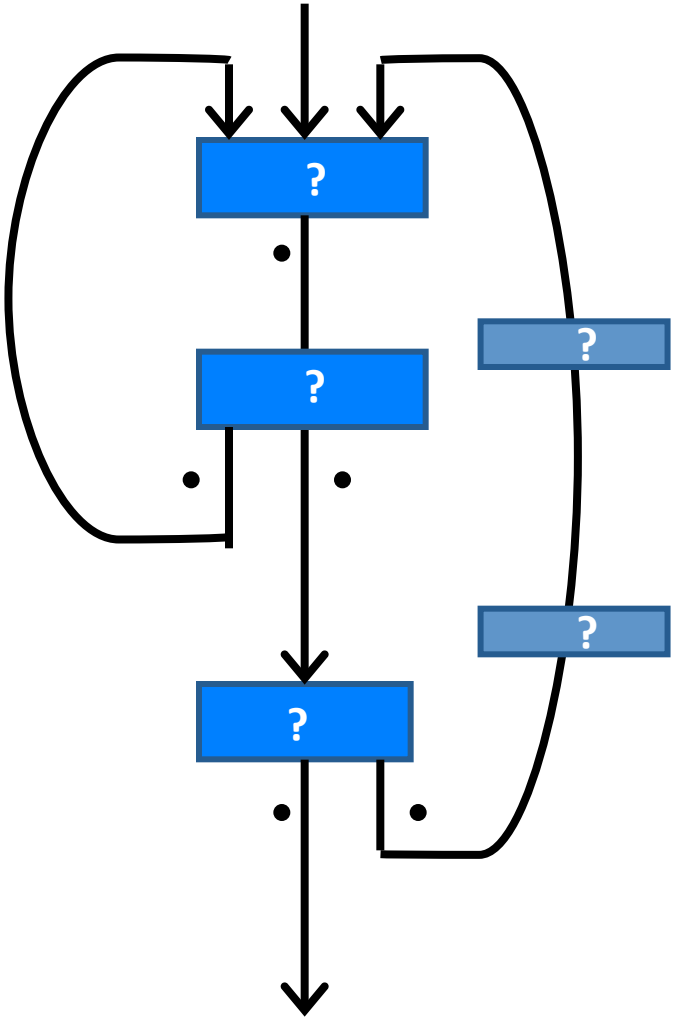
Commoner/Holt/Pnueli 1970

- Strongly connected **MG safe**: always ***adapt to general case ?***

- *Harder*: Strongly connected MG can in practice reach its throughput (and a k-periodic regime, where each node fires according to a regular pattern after initial phase) Carlier/Chretienne 1976

Pb: minimize channel dimensions

Static Scheduling (k-periodic)



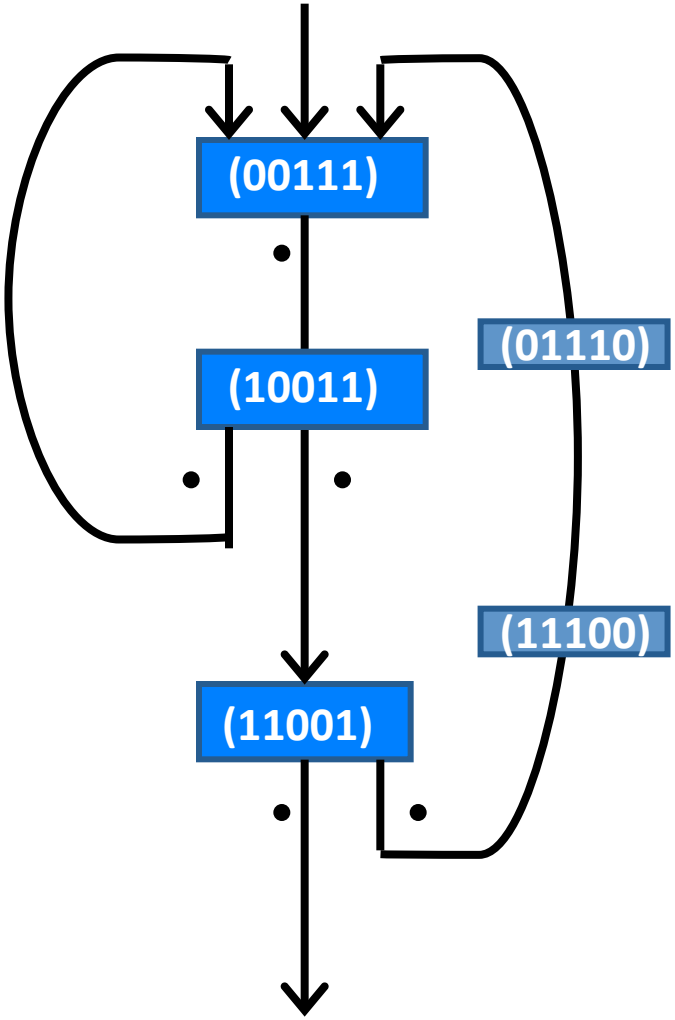
- ASAP firing policy leads to ultimately k-periodic schedules:
  [CarlierChretienne, CohenBaccelliQuadrat et al]

each nodes fires according to a pattern $\mathbf{u.v^{\omega}}$, with $\mathbf{u,v} \in \mathbf{\{0,1\}^*}$

$\mathbf{p} = \mathbf{|v|}$: period
$\mathbf{k} = \mathbf{|v|_1}$: periodicity
$\mathbf{k/p}$: throughput

- All nodes have same throughput, inherited from slowest cycle (here 3/5)
- But all tokens in a cycle may accumulate in the same place/channel !

Static Scheduling (k-periodic)



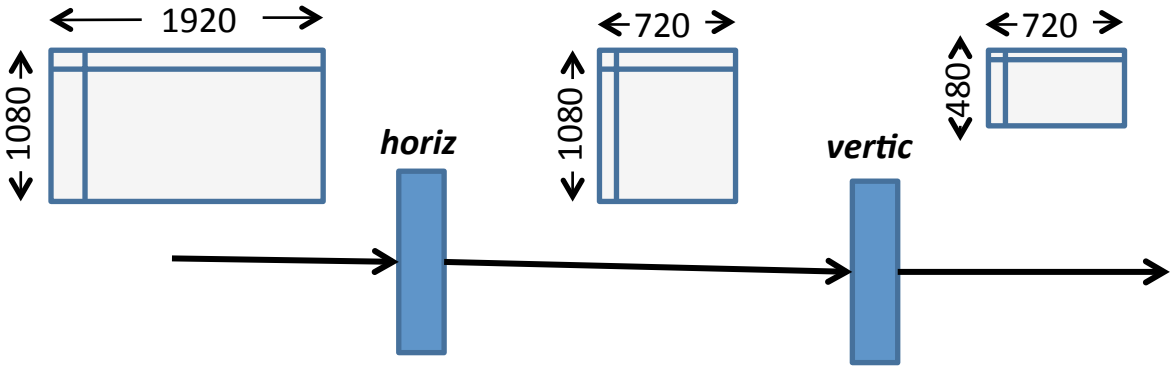
Synchronous DataFlow/ Weighted Marked Graphs

- Now constant number of tokens produced and consumed (not always unitary)

(MGs sometimes called "uniform SDF", better call them unitary)

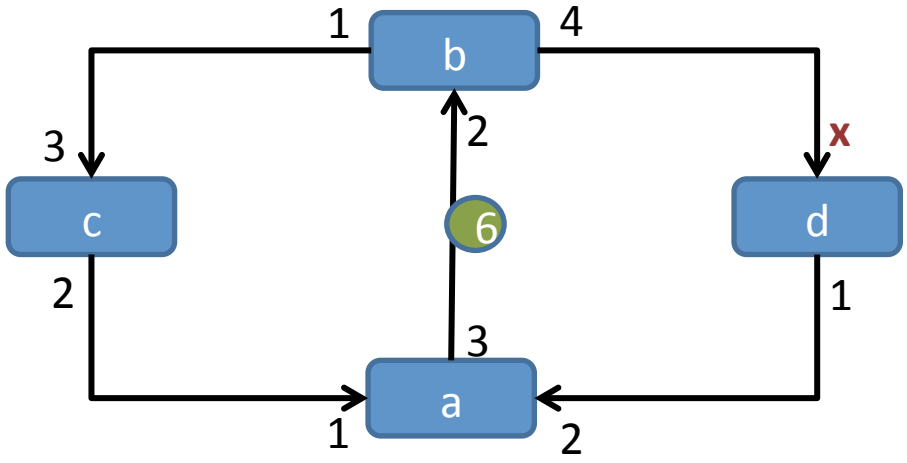
- New issue: will production always match consumption ("ecological" balance) ?
- What about safety ? Liveness ?

splitting rate example: downscaler



- Ratio 1920/720 = 8:3 1080/480 = 9:4

SDF example *(from S. Edwards)*



(we note also a the number of executions of a)

must obey

3a = 2b

4b = **x**d

2a = d

3c = b

2c = a

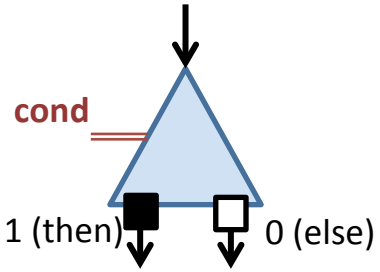
solutions ?

When balanced, a system can be normalized so that each node has uniform rates in all its channel ends

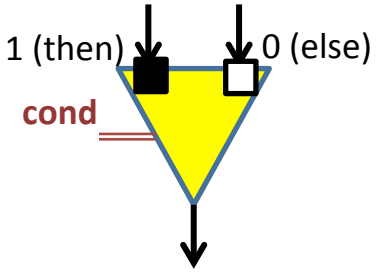
Liveness ? Safety ?

Boolean DataFlow

- Adds boolean stream of switching condition values !
  - Source unspecified, so ratio of 0/1 in conds unknown…



select/demux



merge/mux

CycloStatic DataFlow

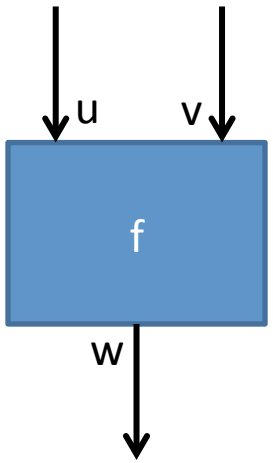
- Now consumption/production rates change along cyclic schemes (should rather be ultimately regular, with initial part)
- When “averaged” on periods, can be expanded to SDF. ***How ?***

Kahn Process Networks

- Networks of sequential processes, linked by channels
  - Operations **read(chan)** *blocking*, **write(chan)** *not blocking*
  - No operation **peek(chan)** *consult but do not consume data*
  - Individual sequential processes allowed any classical syntax with internal_data-dependent if-then-else, but often finite control automaton (loop programs)

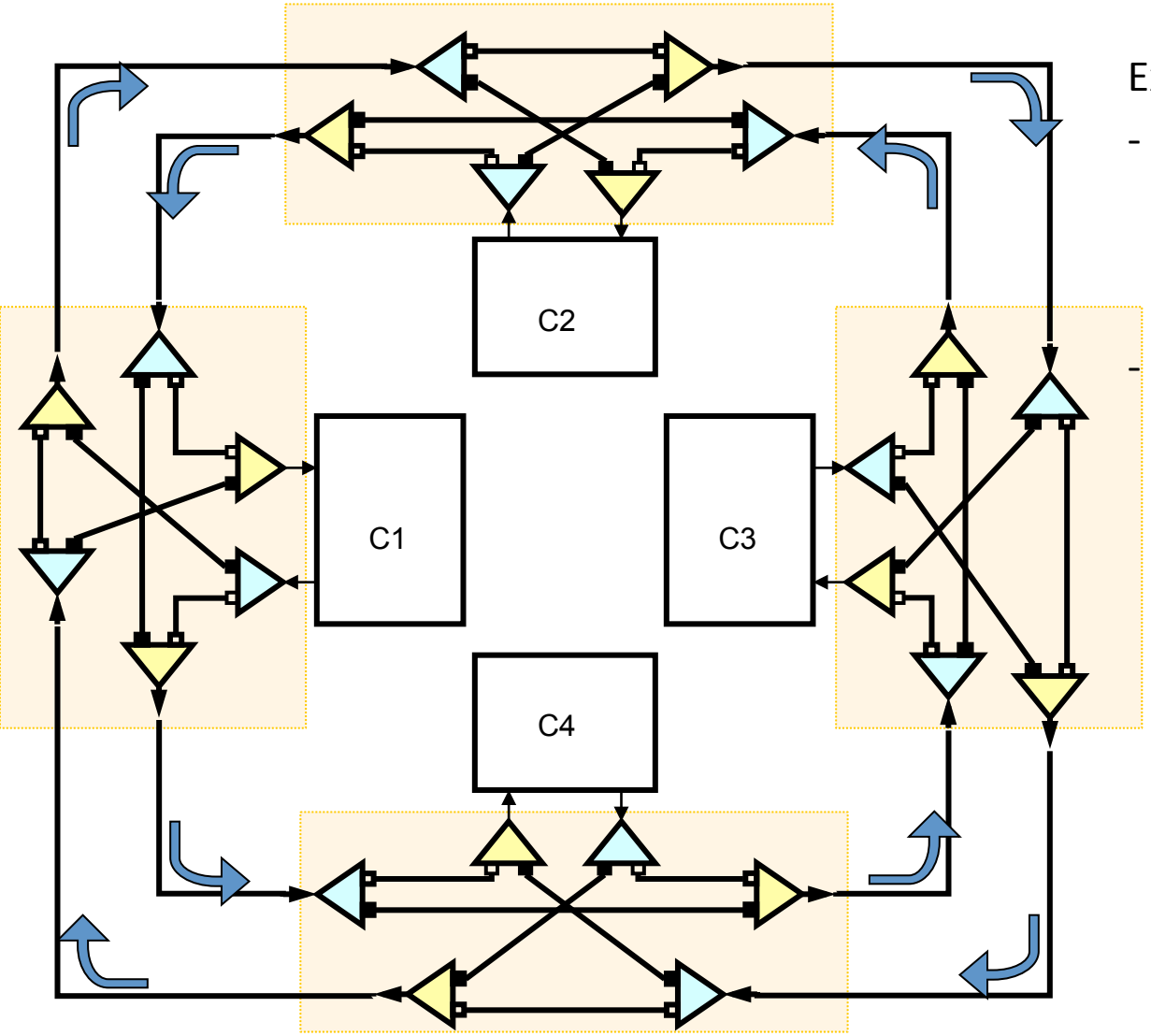
  → Conflict-freeness (here called monotony, as behaviors may depend on internal decisions)

```
process f(in_chan int u, int
v;                          out_chan int w)
{
  int i; bool b = true;
  for (;;) {
    i = b ? read(u) : read(w);
    printf("%i\n", i);
    write(i, w);
    b = !b;
  }
}
```



Alternates passing values from u then v to w

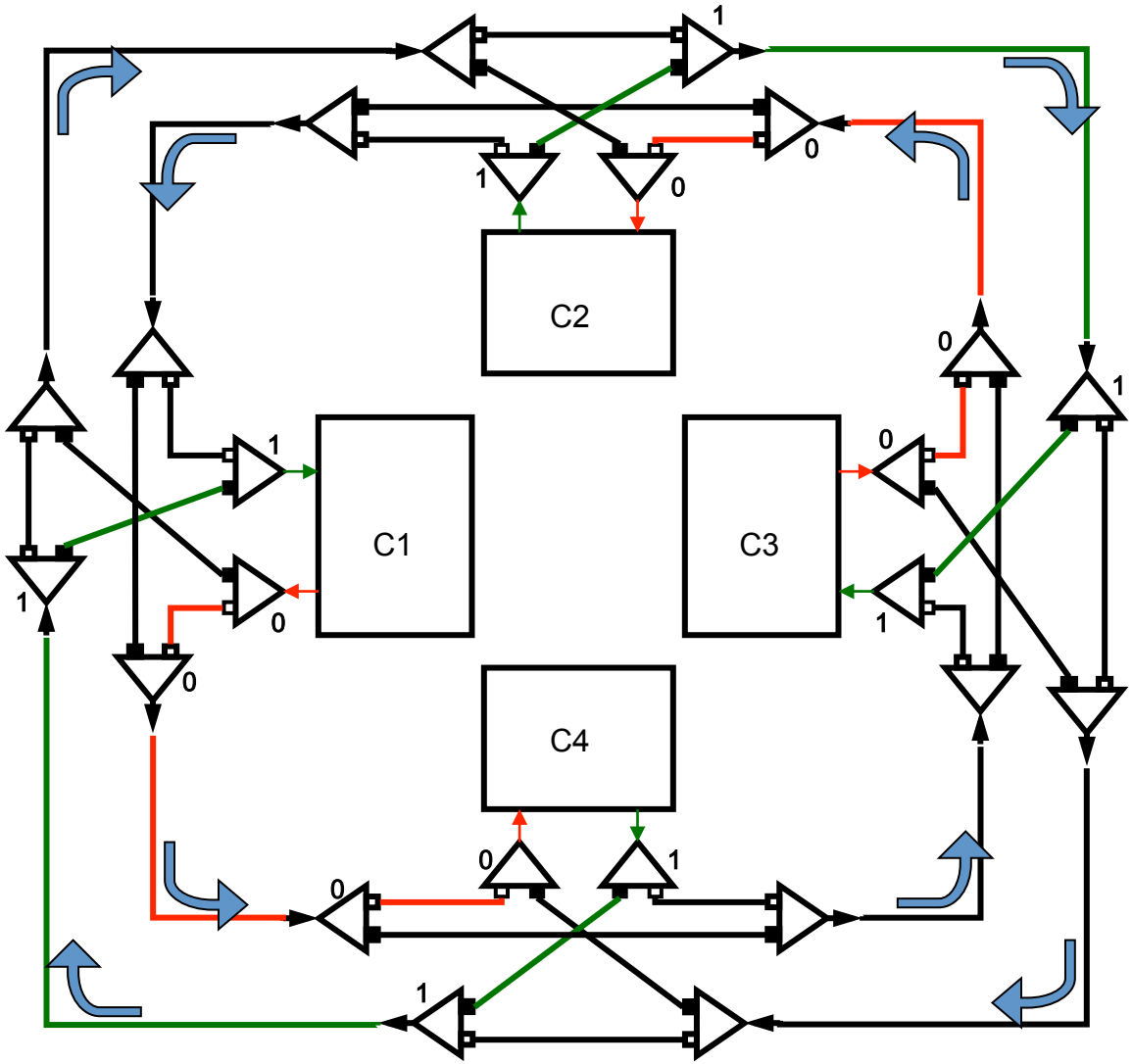
K-periodically Routed marked Graphs



Exercice:

- try to assign switches to build full-duplex connections between C2 and C3, and C1 and C4
- Same between C1 and C3, C2 and C4

Answer 1



Answer 2