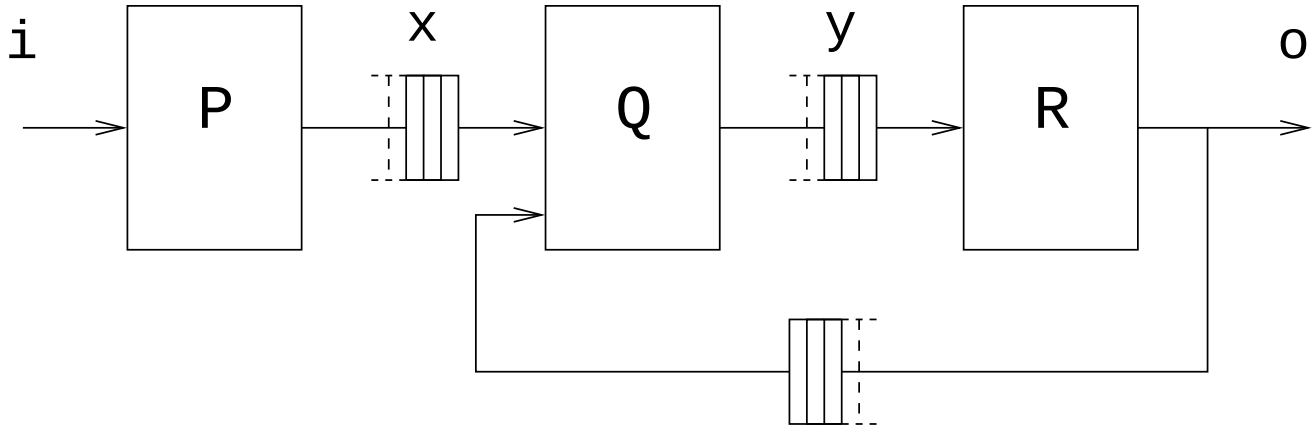
Clock Typing of n-Synchronous Programs

Louis Mandel Florence Plateau Marc Pouzet

Laboratoire de Recherche en Informatique
Université Paris-Sud 11
INRIA Saclay – Ile-de-France

DCC – 20-21/03/2010

Kahn Process Networks [Gilles Kahn, 1974]



Network of processes

- concurrent execution
- communication through buffers of sufficient size

If processes are deterministic then the network is deterministic

Programming Kahn Process Networks

Problem: computation of sufficient buffer sizes

- risk of data loss, of deadlock
- sometimes, need of infinite buffers

Goal:

- rejection of infinite buffers
- automatic sizing of buffers

Related work:

- Synchronous Data Flow and variants [Lee *et al.*] [Buck]
- scheduling [Carlier, Chretienne] [Baccelli, Cohen, Quadrat]
- Network Calculus [Cruz], Real-time Calculus [Thiele *et al.*]

Dataflow Synchronous Model

Programming Kahn networks without buffers:

- programming languages: Lustre, Signal, Lucid Synchrone
- instantaneous consumption of produced data
- strong guaranties: bounded memory, absence of deadlocks

But: communication without buffers sometimes too restrictive
(e.g. multimedia applications)

n-Synchronous Model: Programming Kahn Networks with Bounded Memory

- accept to store data in buffers

Automatic methods at compile time to:

- reject networks needing infinite memory
- compute activation paces of computations nodes
- compute sufficient buffers sizes

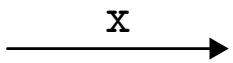
More flexibility with the same guaranties

Overview

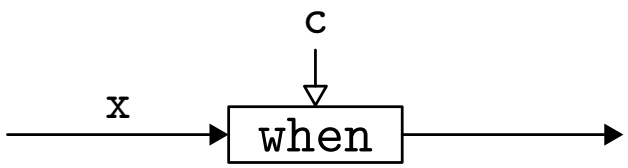
1. Lucy-n: a n-Synchronous Extension of Lustre
2. Clock Typing
3. Application to Latency Insensitive Design

Lucy-n: a n-Synchronous Extension of Lustre

A Dataflow Synchronous Kernel



flow	values	clock
x	5 7 3 6 2 8 1 ...	1111111...



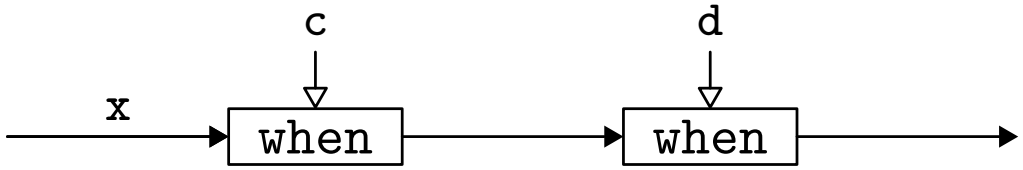
flow	values								clock
x	5	7	3	6	2	8	1	...	1111111...
c	1	0	1	0	1	0	1	...	
x when c	5		3		2		1	...	1010101...

$clock(\texttt{x when c}) = clock(\texttt{x})\ on\ \texttt{c}$

on operator :

$$0.w_1\ on\ w_2 \stackrel{def}{=} 0.(w_1\ on\ w_2)$$
$$1.w_1\ on\ 1.w_2 \stackrel{def}{=} 1.(w_1\ on\ w_2)$$
$$1.w_1\ on\ 0.w_2 \stackrel{def}{=} 0.(w_1\ on\ w_2)$$

A Dataflow Synchronous Kernel



flow	values	clock
x	5 7 3 6 2 8 1 ...	1111111...
c	1 0 1 0 1 0 1 ...	
x when c	5 3 2 1 ...	1010101...
d	1 0 1 1 ...	
(x when c) when d	5 2 1 ...	1000101...

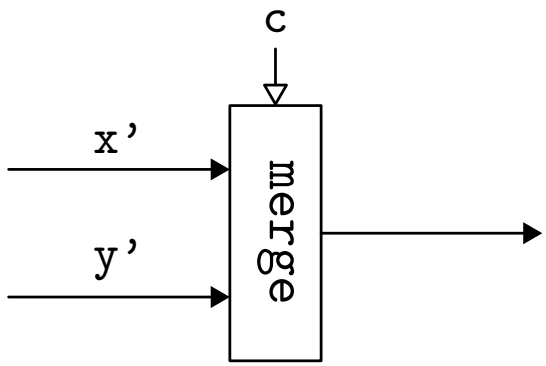
clock((x when c) when d) = *clock*(x when c) *on* d

on operator :

$$0.w_1 \ \textit{on}\ w_2 \stackrel{\textit{def}}{=} 0.(w_1 \ \textit{on}\ w_2)$$

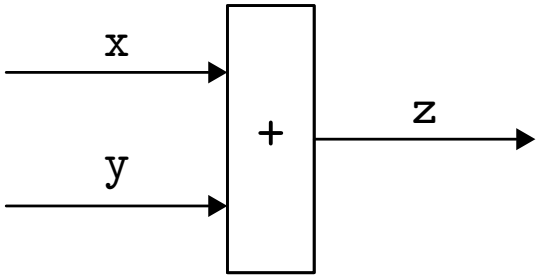
$$1.w_1 \ \textit{on}\ 1.w_2 \stackrel{\textit{def}}{=} 1.(w_1 \ \textit{on}\ w_2)$$

$$1.w_1 \ \textit{on}\ 0.w_2 \stackrel{\textit{def}}{=} 0.(w_1 \ \textit{on}\ w_2)$$



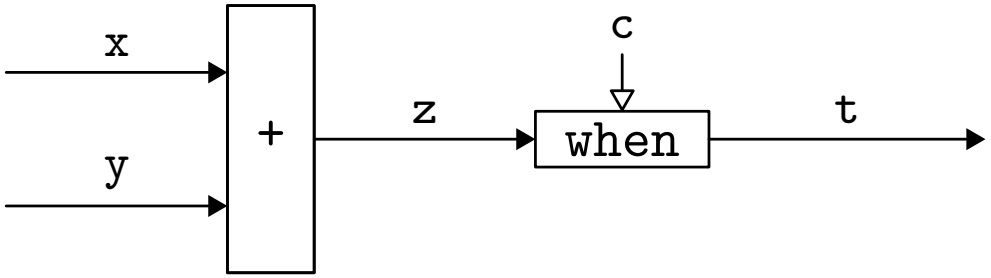
flow	value								clock
x'	5		3		2		1	...	1010101...
y'		2		5		1		...	0101010...
c	1	0	1	0	1	0	1	...	
merge c x' y'	5	2	3	5	2	1	1	...	1111111...

A Dataflow Synchronous Kernel



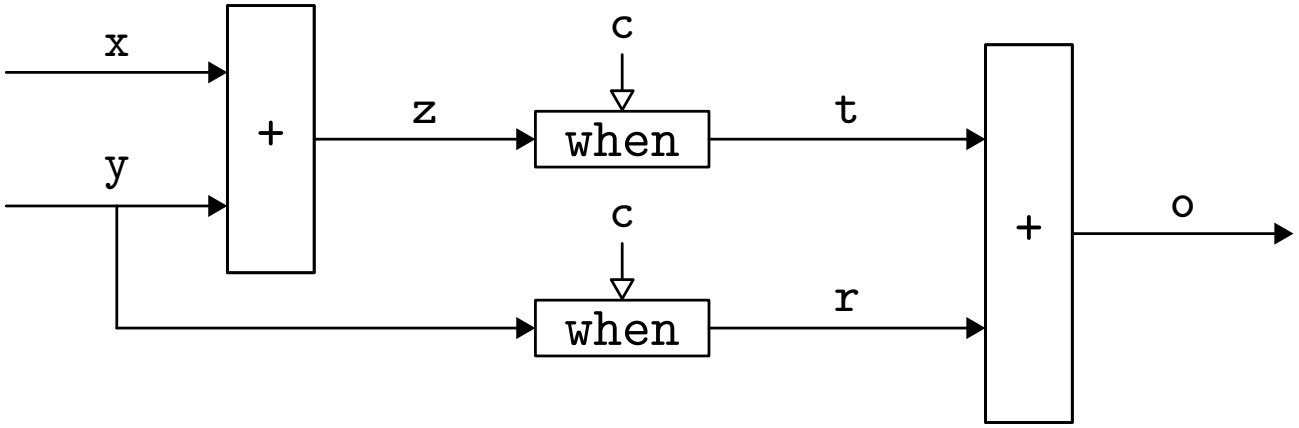
flow	values								clock
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...

A Dataflow Synchronous Kernel



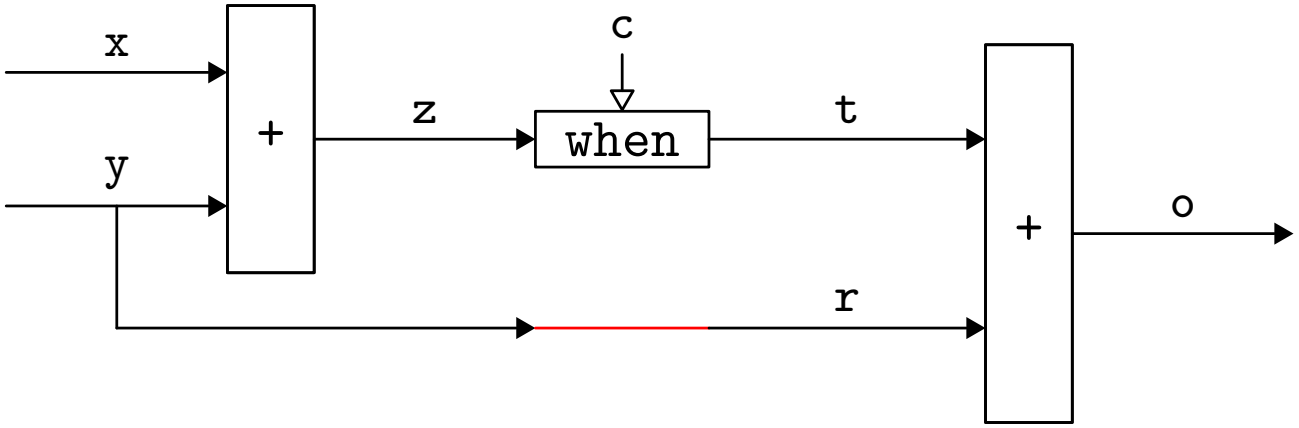
flow	values								clock
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
t = z when c	8		4		6		8	...	101010...

A Dataflow Synchronous Kernel



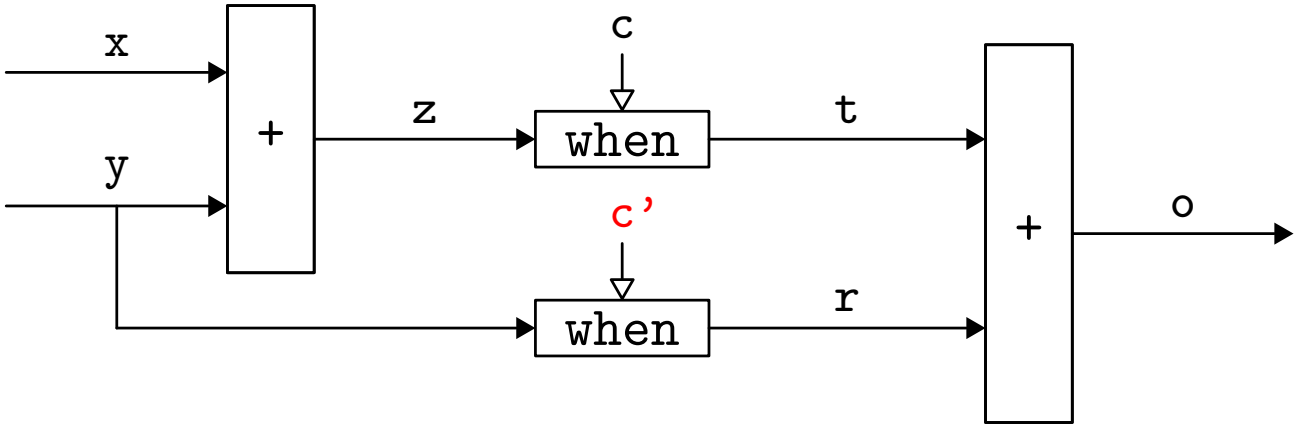
flow	values								clock
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
t = z when c	8		4		6		8	...	101010...
r = y when c	3		1		4		7	...	101010...
o = t + r	11		5		10		15	...	101010...

A Dataflow Synchronous Kernel



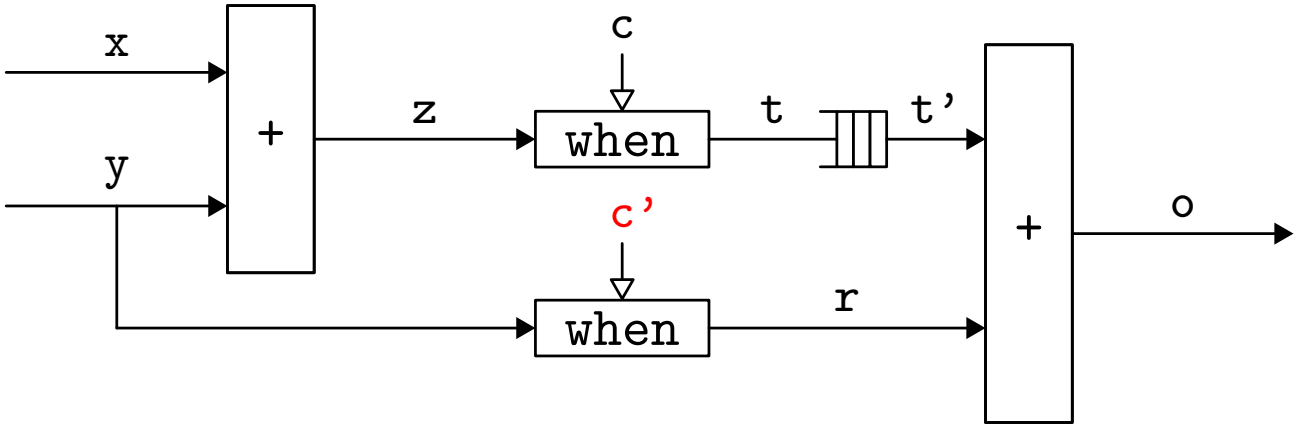
flow	values								clock
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
t = z when c	8		4		6		8	...	101010...
r = y	3	2	1	5	4	1	7	...	111111...
o = t + r	rejected								

A Dataflow Synchronous Kernel



flow	values								clock
x	5	7	3	6	2	8	1	...	111111...
y	3	2	1	5	4	1	7	...	111111...
z = x + y	8	9	4	11	6	9	8	...	111111...
c	1	0	1	0	1	0	1	...	
t = z when c	8		4		6		8	...	101010...
r = y when c'		2		5		1		...	010101...
o = t + r	rejected								

n-Synchronous Extension: Bufferization Operator

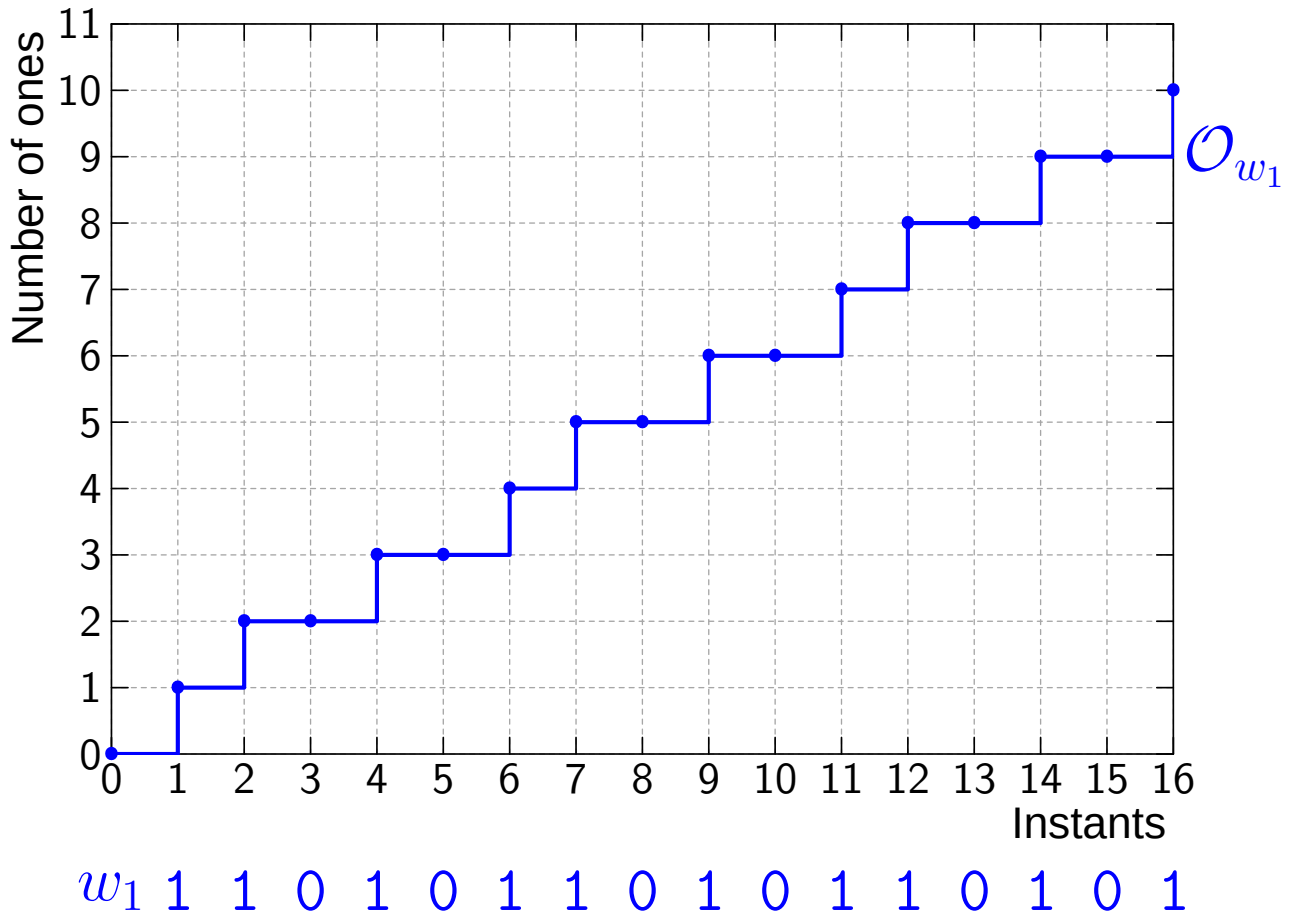


flow	values	clock
t = z when c	8 4 6 8 ...	101010...
t' = buffer(t)	8 4 6 ...	010101...
r = y when c'	2 5 1 ...	010101...
o = t' + r	10 9 7 ...	010101...

- adaptability relation $\Rightarrow$ communication through a bounded buffer
- example : 101010... $<:$ 010101...

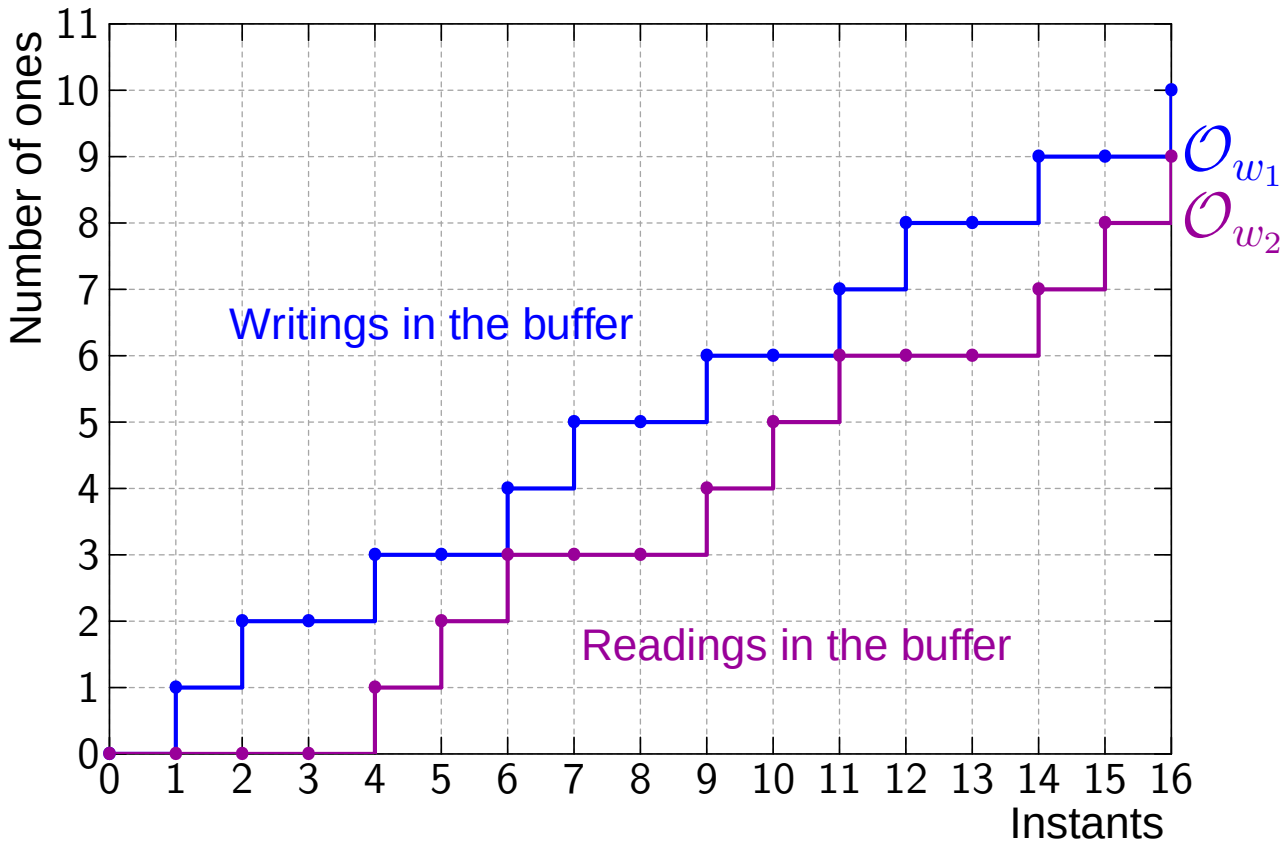
Adaptability Relation

Cumulative Function



$\mathcal{O}_{w_1}$: cumulative function of the word w_1

Adaptability Relation



buffer size $size(w_1, w_2) = \max_{i \in \mathbb{N}}(\mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i))$

adaptability $w_1 <: w_2 \overset{def}{\Leftrightarrow} \exists n \in \mathbb{N}, \forall i,\ 0 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq n$

Ultimately Periodic Clocks

Example : $0(00111) = 0\,00111\,00111\,...$

Verification of relations on clocks

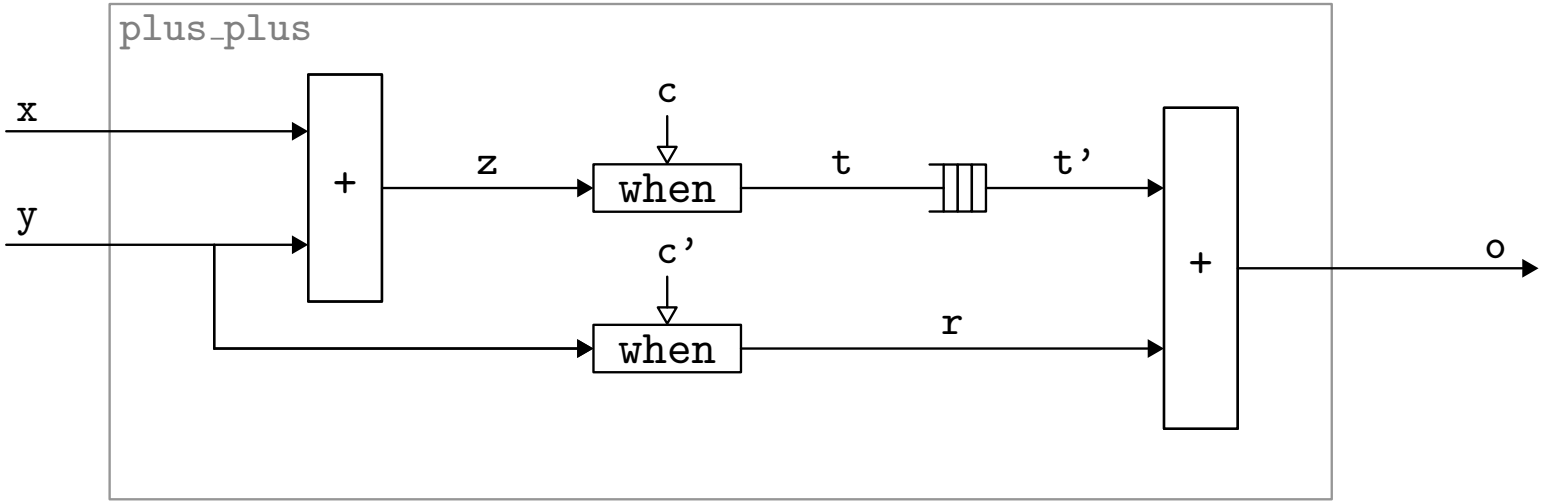
- equality test: $0(00111\ 00111) = 0\ 00(111\ 00)$
- adaptability test: $(11010) <: 0(00111)$

Clock expressions

- computation of on : $0(00111)\ on\ (101) = 0(00101)$

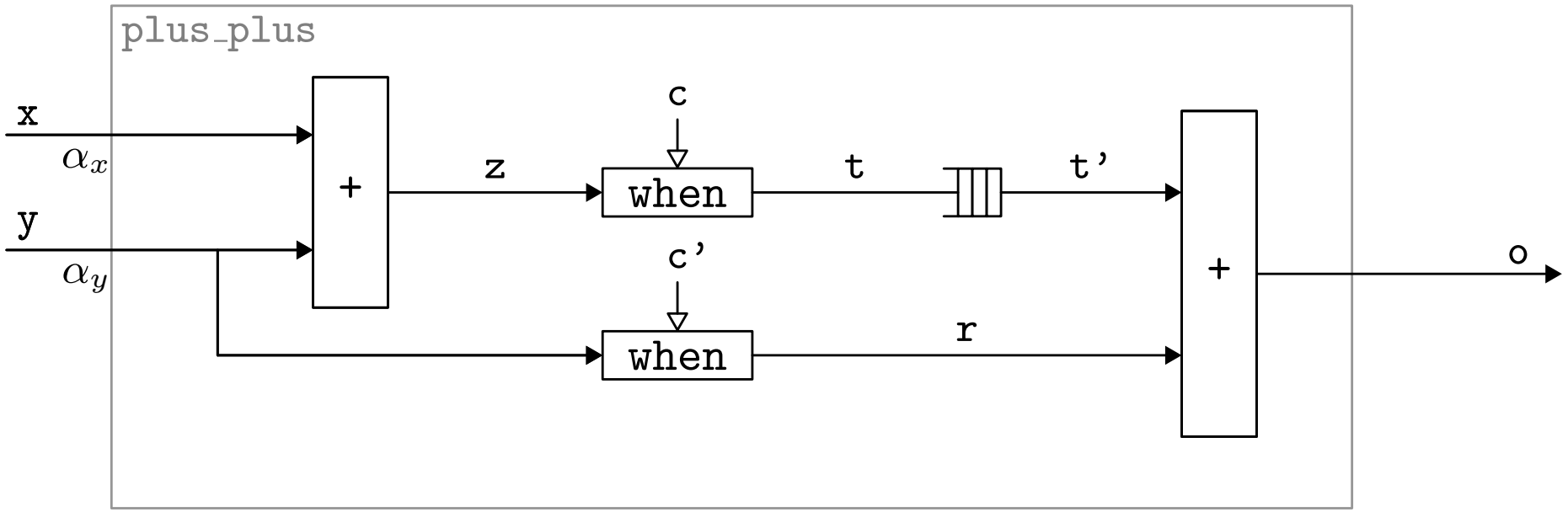
Clock Typing

Typing

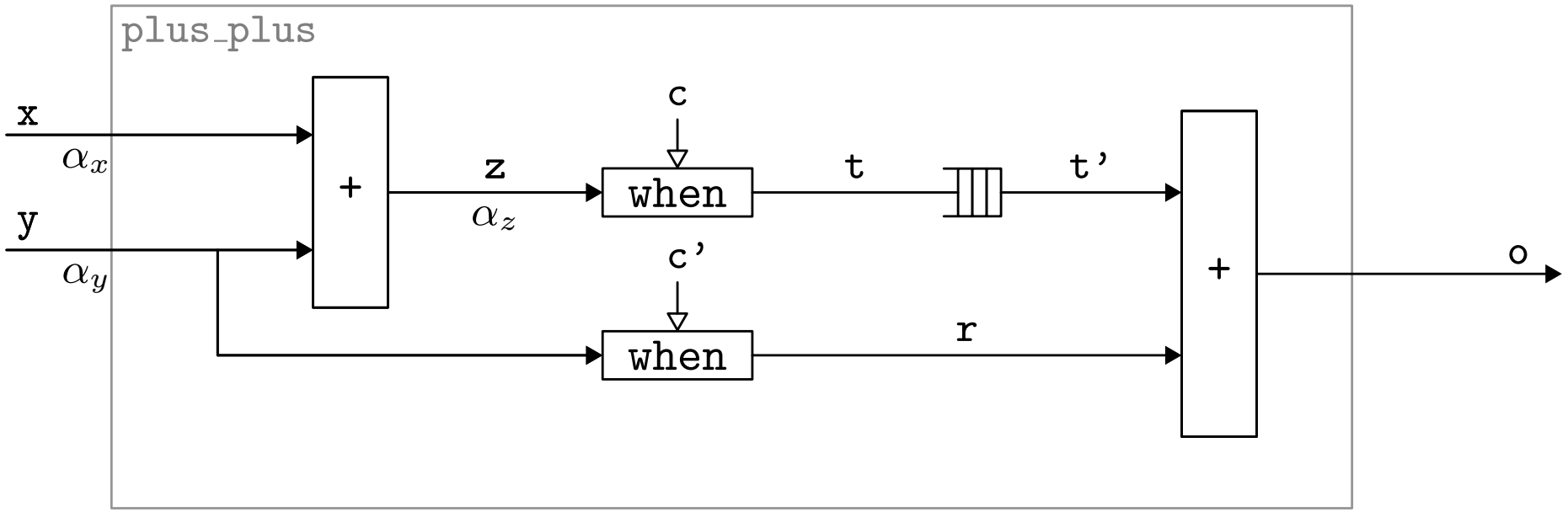



```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when c
7    and t'= buffer(t)
8    and r = y when c'
9    and o = t' + r
```

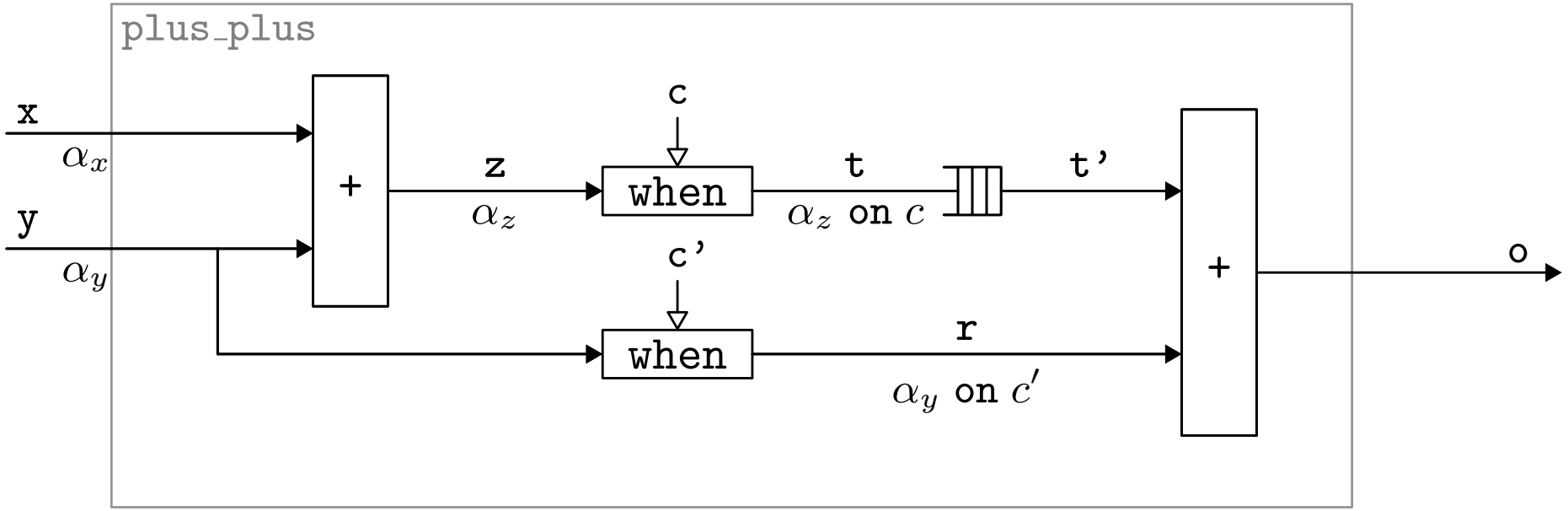
```
val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on c'
Buffer line 7, characters 10-19: size = 1
```



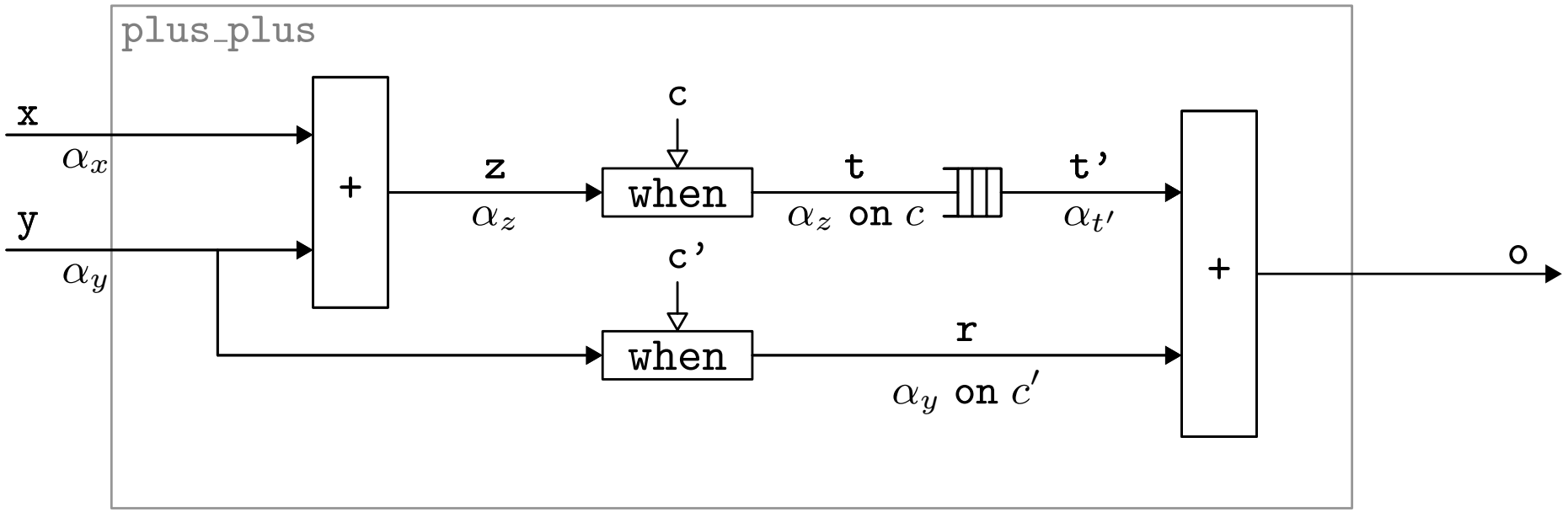
$$C = \left\{ \begin{array}{l} \alpha_x = \alpha_y \\ \\ \\ \end{array} \right\}$$



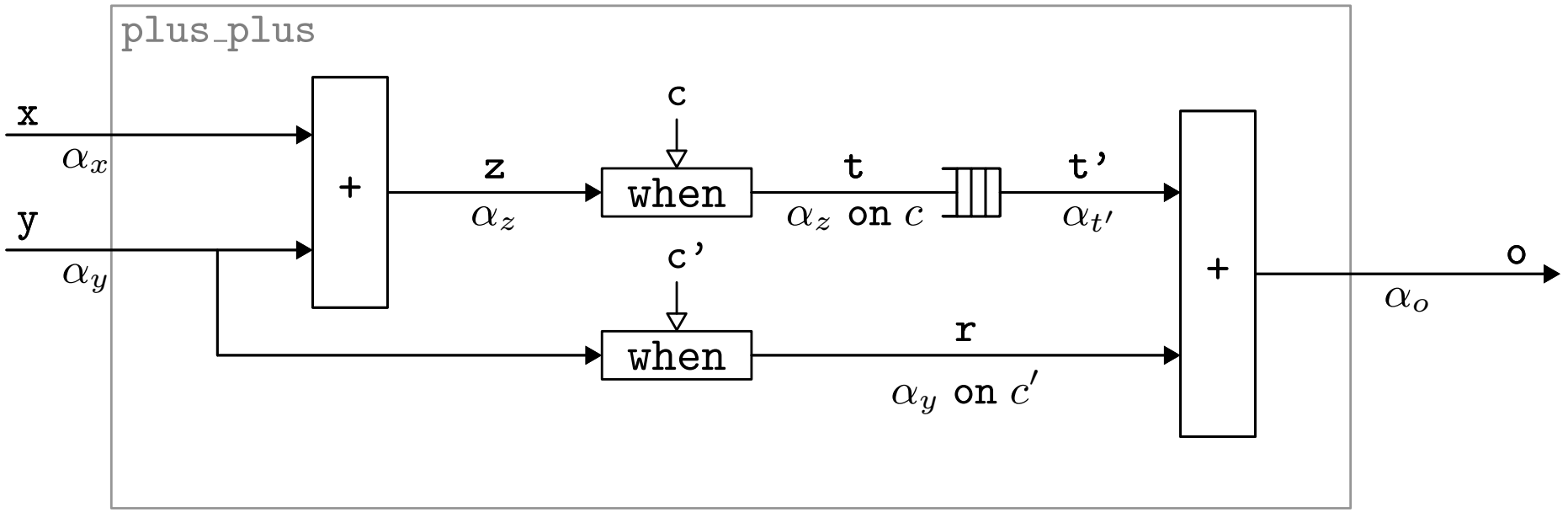
$$C = \left\{ \begin{array}{l} \alpha_x = \alpha_y = \alpha_z \end{array} \right\}$$



$$C = \left\{ \begin{array}{l} \alpha_x = \alpha_y = \alpha_z \\ \\ \\ \end{array} \right\}$$



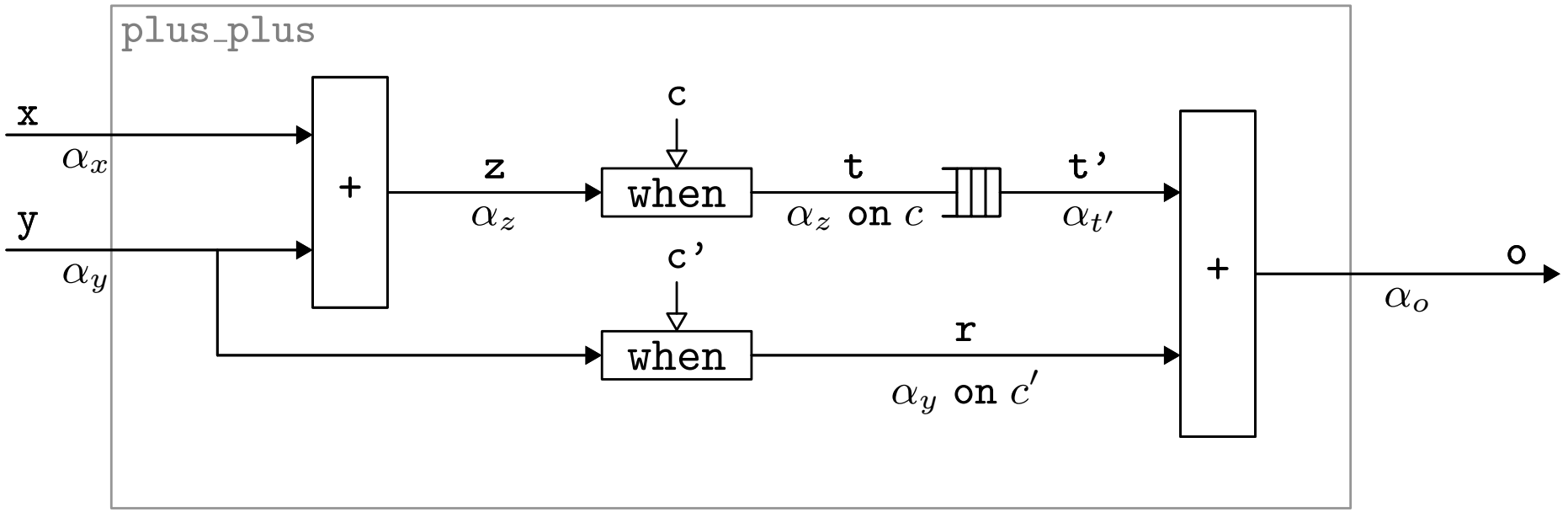
$$C = \left\{ \begin{array}{l} \alpha_x = \alpha_y = \alpha_z \\ \alpha_z \text{ on } c <: \alpha_{t'} \\ \\ \end{array} \right\}$$



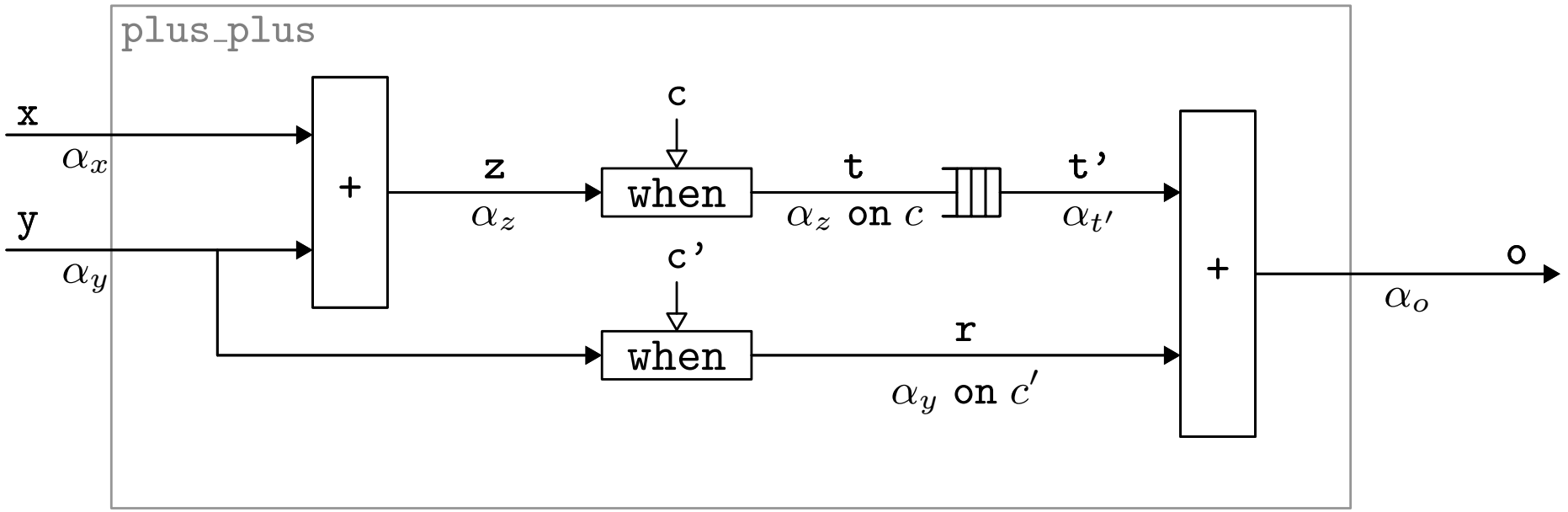
$$C = \left\{ \begin{array}{l} \alpha_x = \alpha_y = \alpha_z \\ \alpha_z \text{ on } c <: \alpha_{t'} \\ \alpha_y \text{ on } c' = \alpha_{t'} = \alpha_o \end{array} \right\}$$

Clock Calculus

Example



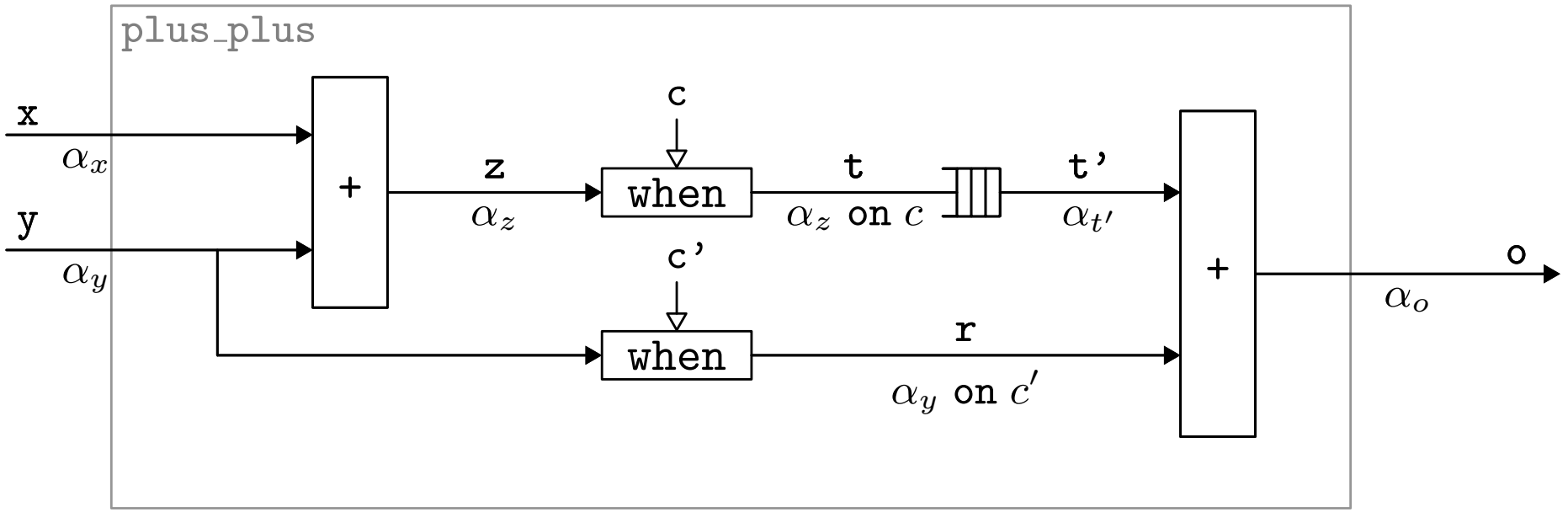
$$(\alpha_x \times \alpha_y) \rightarrow \alpha_o \quad \text{such that} \quad C = \left\{ \begin{array}{l} \alpha_x = \alpha_y = \alpha_z \\ \alpha_z \text{ on } c <: \alpha_{t'} \\ \alpha_y \text{ on } c' = \alpha_{t'} = \alpha_o \end{array} \right\}$$



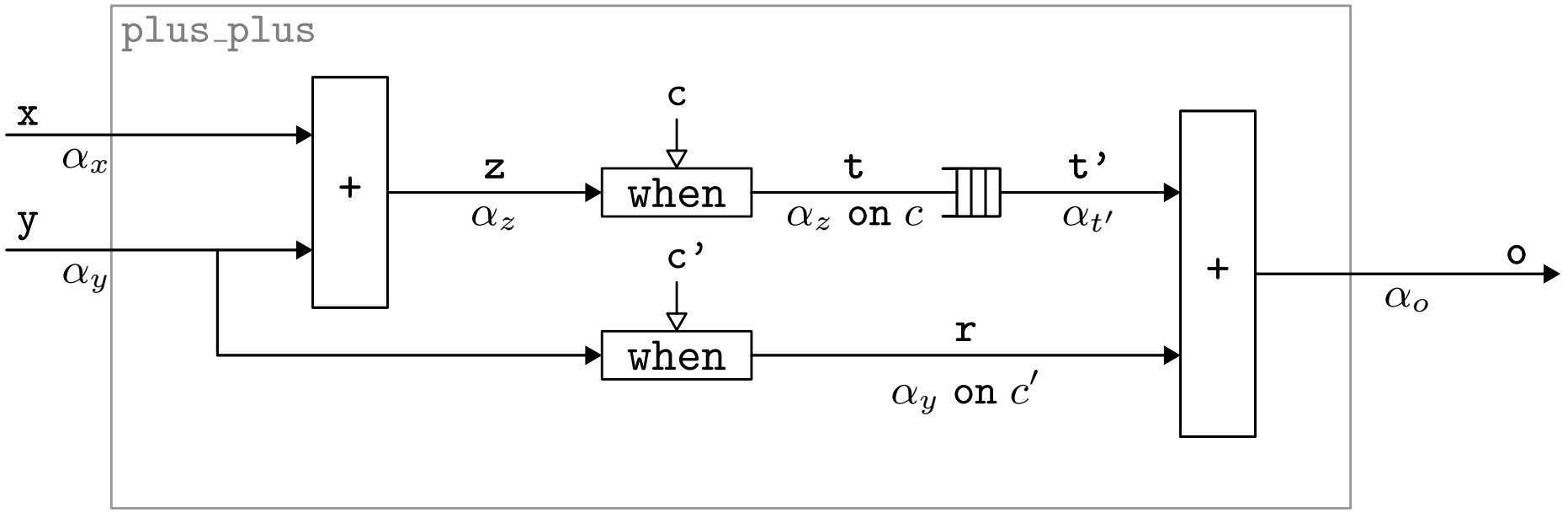
$$(\alpha \times \alpha) \rightarrow \alpha_o \quad \text{such that} \quad C = \left\{ \begin{array}{lll} \alpha & = \alpha & = \alpha \\ \alpha \text{ on } c & <: & \alpha_{t'} \\ \alpha \text{ on } c' & = \alpha_{t'} & = \alpha_o \end{array} \right\}$$

Clock Calculus

Example



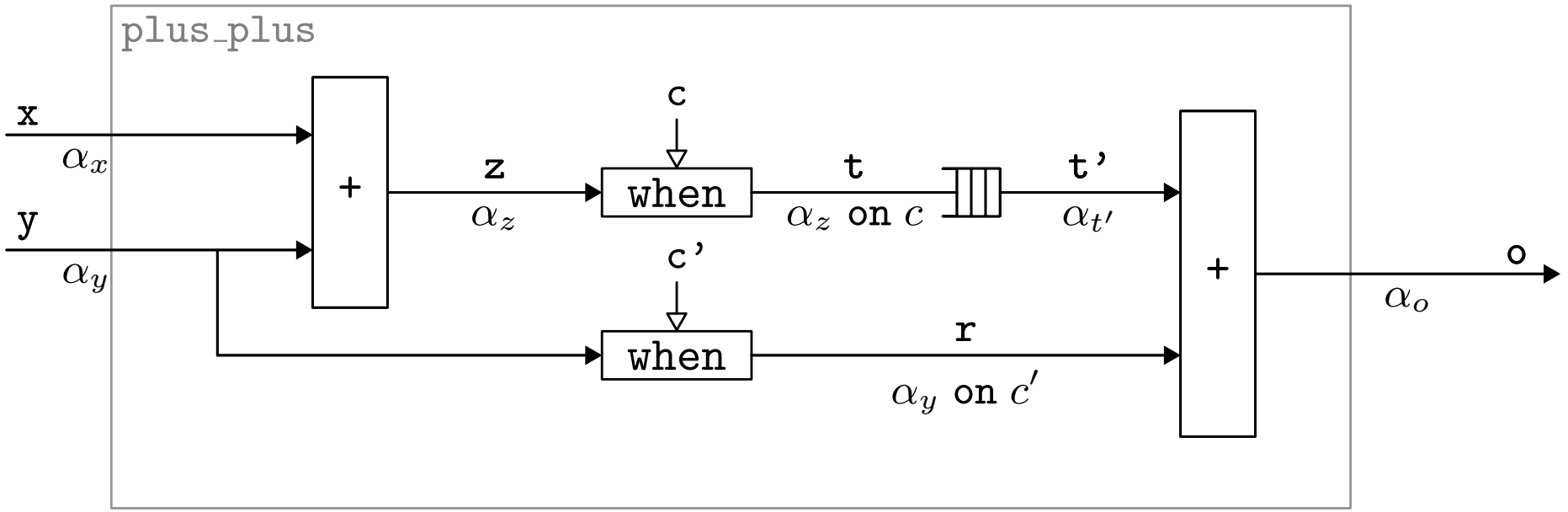
$$(\alpha \times \alpha) \rightarrow \alpha_o \quad \text{such that} \quad C = \left\{ \begin{array}{l} \alpha \text{ on } c <: \alpha_{t'} \\ \alpha \text{ on } c' = \alpha_{t'} = \alpha_o \end{array} \right\}$$



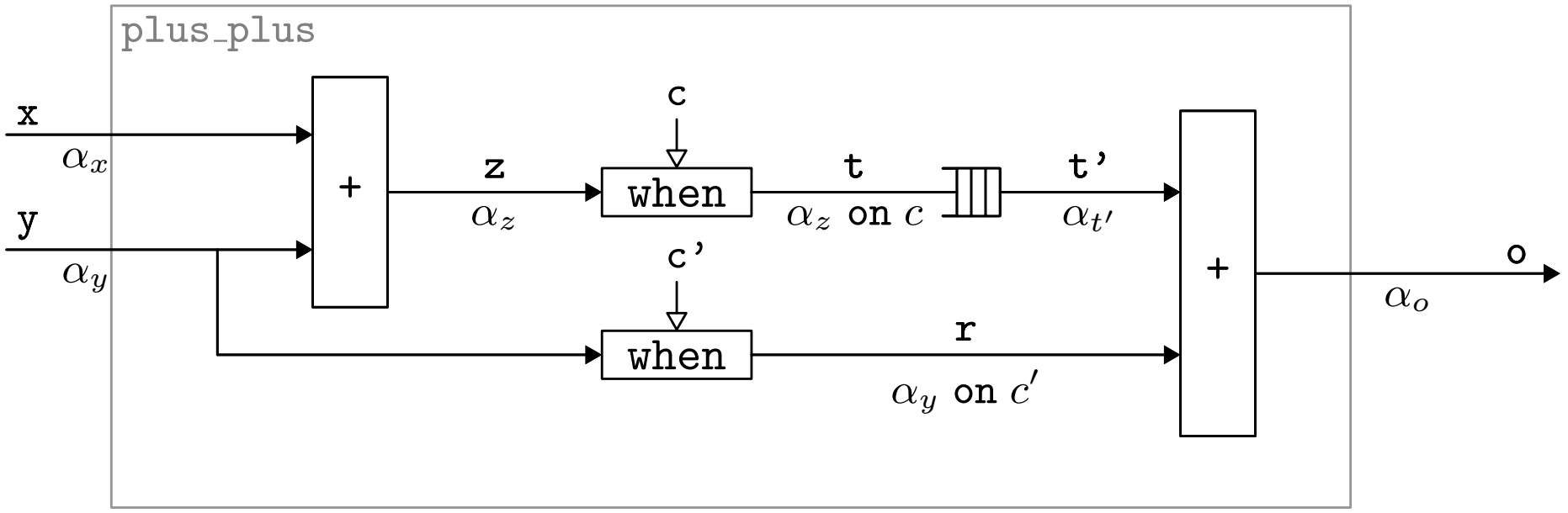
$$(\alpha \times \alpha) \rightarrow \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \begin{array}{l} \alpha \text{ on } c \ <: \ \alpha \text{ on } c' \\ \alpha \text{ on } c' \ = \ \alpha \text{ on } c' \ = \ \alpha \text{ on } c' \end{array} \right\}$$

Clock Calculus

Example



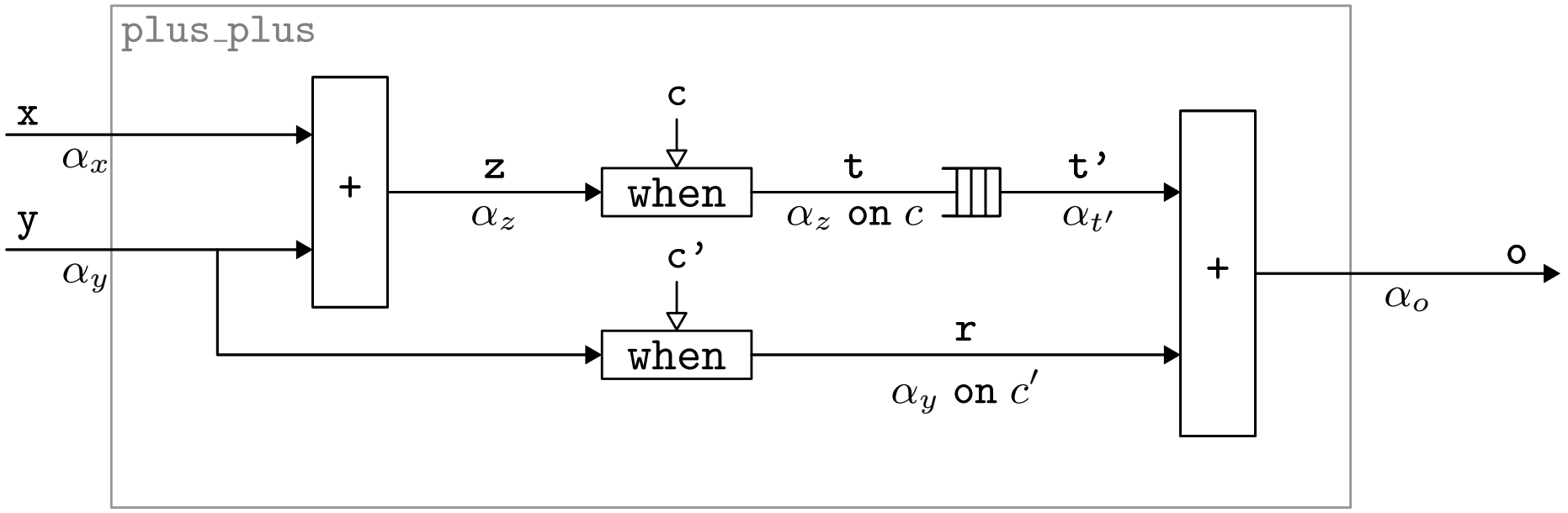
$$(\alpha \times \alpha) \rightarrow \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \alpha \text{ on } c \ <: \ \alpha \text{ on } c' \right\}$$



$$(\alpha \times \alpha) \to \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \begin{array}{l} \alpha \text{ on } c \ <: \ \alpha \text{ on } c' \\ \\ \end{array} \right\}$$

Subtyping constraint solving

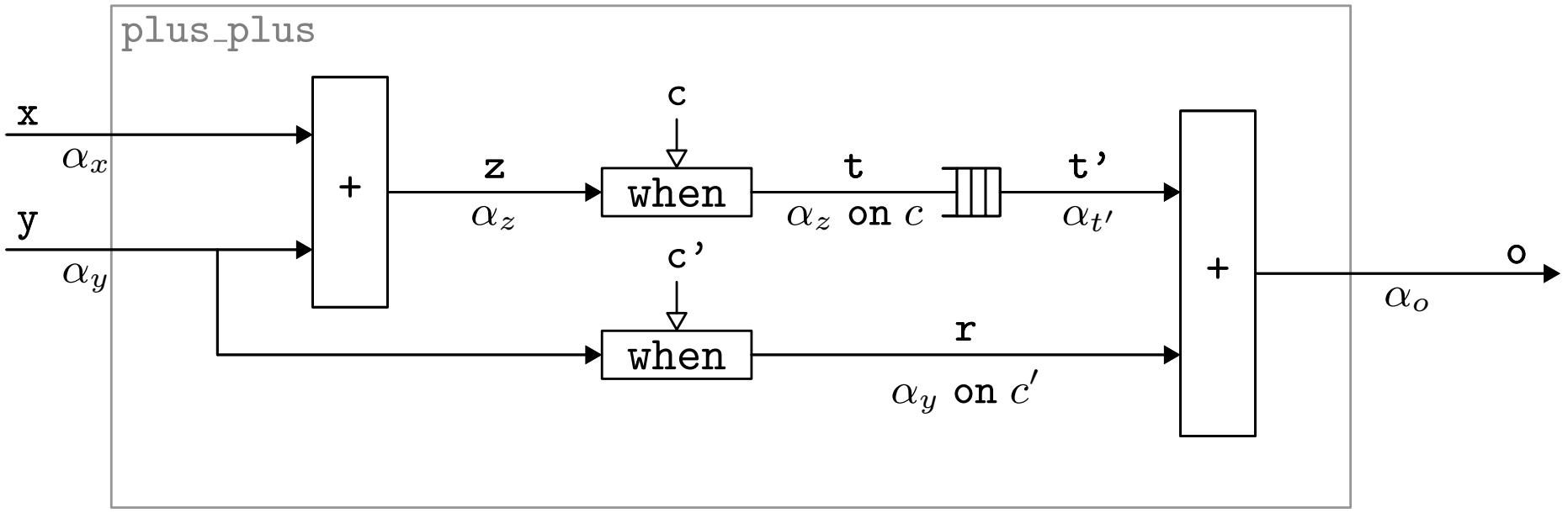
- simple case: α on w_1 <: α on w_2 $\Leftrightarrow$ w_1 <: w_2



$$(\alpha \times \alpha) \to \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \qquad c \ <: \qquad c' \qquad \right\}$$

Subtyping constraint solving

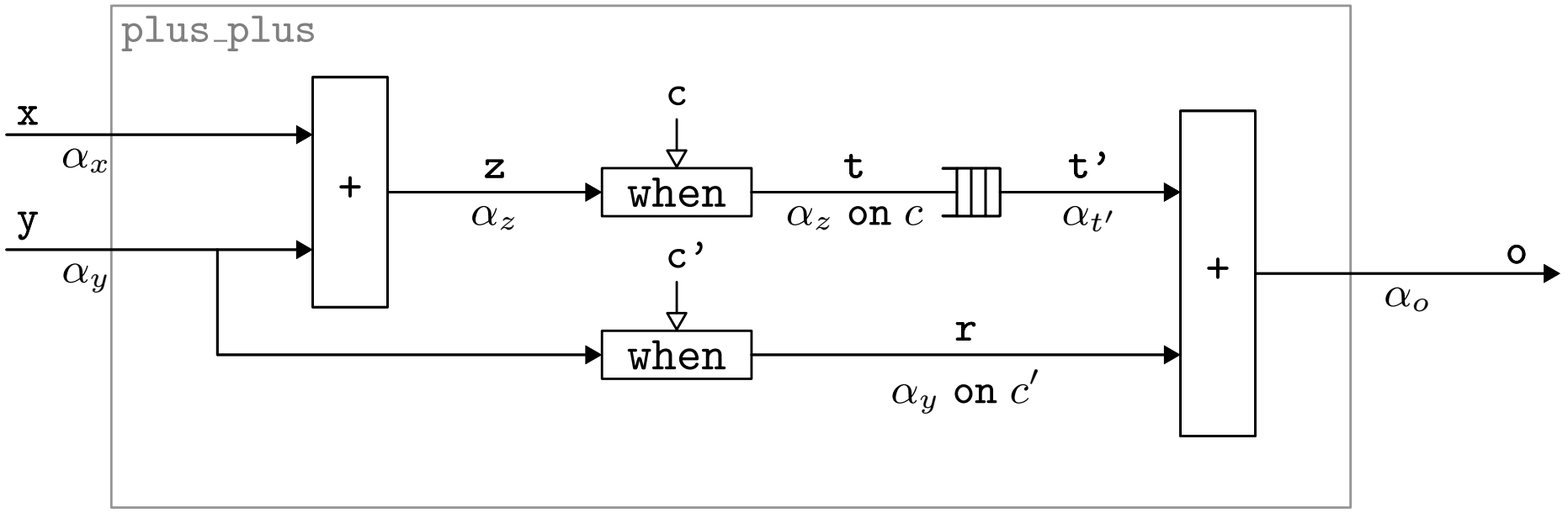
- simple case: α on w_1 <: α on w_2 $\Leftrightarrow$ $w_1 <: w_2$



$$(\alpha \times \alpha) \to \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \begin{array}{ccc} (10) & <: & (01) \end{array} \right\}$$

Subtyping constraint solving

- simple case: $\alpha \text{ on } w_1 \ <: \ \alpha \text{ on } w_2 \quad \Leftrightarrow \quad w_1 <: w_2$



$$(\alpha \times \alpha) \rightarrow \alpha \text{ on } c' \quad \text{such that} \quad C = \left\{ \qquad \right\}$$

Subtyping constraint solving

- simple case: $\alpha \text{ on } w_1 \ <: \ \alpha \text{ on } w_2 \quad \Leftrightarrow \quad w_1 <: w_2$

Paces Inference

Example of a more complex system:

$$\left\{ \begin{array}{lcl} \alpha_1 \texttt{ on (011)} & \texttt{<:} & \alpha_2 \texttt{ on (10)} \\ \alpha_1 \texttt{ on (1)} & \texttt{<:} & \alpha_3 \texttt{ on (01)} \\ \alpha_3 \texttt{ on (010)} & \texttt{<:} & \alpha_2 \texttt{ on (01)} \end{array} \right\}$$

1. Instantiation of type variables

$$\alpha_1 \leftarrow \alpha \texttt{ on } c_1, \quad \alpha_2 \leftarrow \alpha \texttt{ on } c_2, \quad \alpha_3 \leftarrow \alpha \texttt{ on } c_3$$

$$\left\{ \begin{array}{lcl} \alpha \texttt{ on } c_1 \textit{ on (011)} & \texttt{<:} & \alpha \texttt{ on } c_2 \textit{ on (10)} \\ \alpha \texttt{ on } c_1 \textit{ on (1)} & \texttt{<:} & \alpha \texttt{ on } c_3 \textit{ on (01)} \\ \alpha \texttt{ on } c_3 \textit{ on (010)} & \texttt{<:} & \alpha \texttt{ on } c_2 \textit{ on (01)} \end{array} \right\}$$

Paces Inference

Example of a more complex system:

$$\left\{\begin{array}{lcl} \alpha_1 \texttt{ on (011)} & \texttt{<:} & \alpha_2 \texttt{ on (10)} \\ \alpha_1 \texttt{ on (1)} & \texttt{<:} & \alpha_3 \texttt{ on (01)} \\ \alpha_3 \texttt{ on (010)} & \texttt{<:} & \alpha_2 \texttt{ on (01)} \end{array}\right\}$$

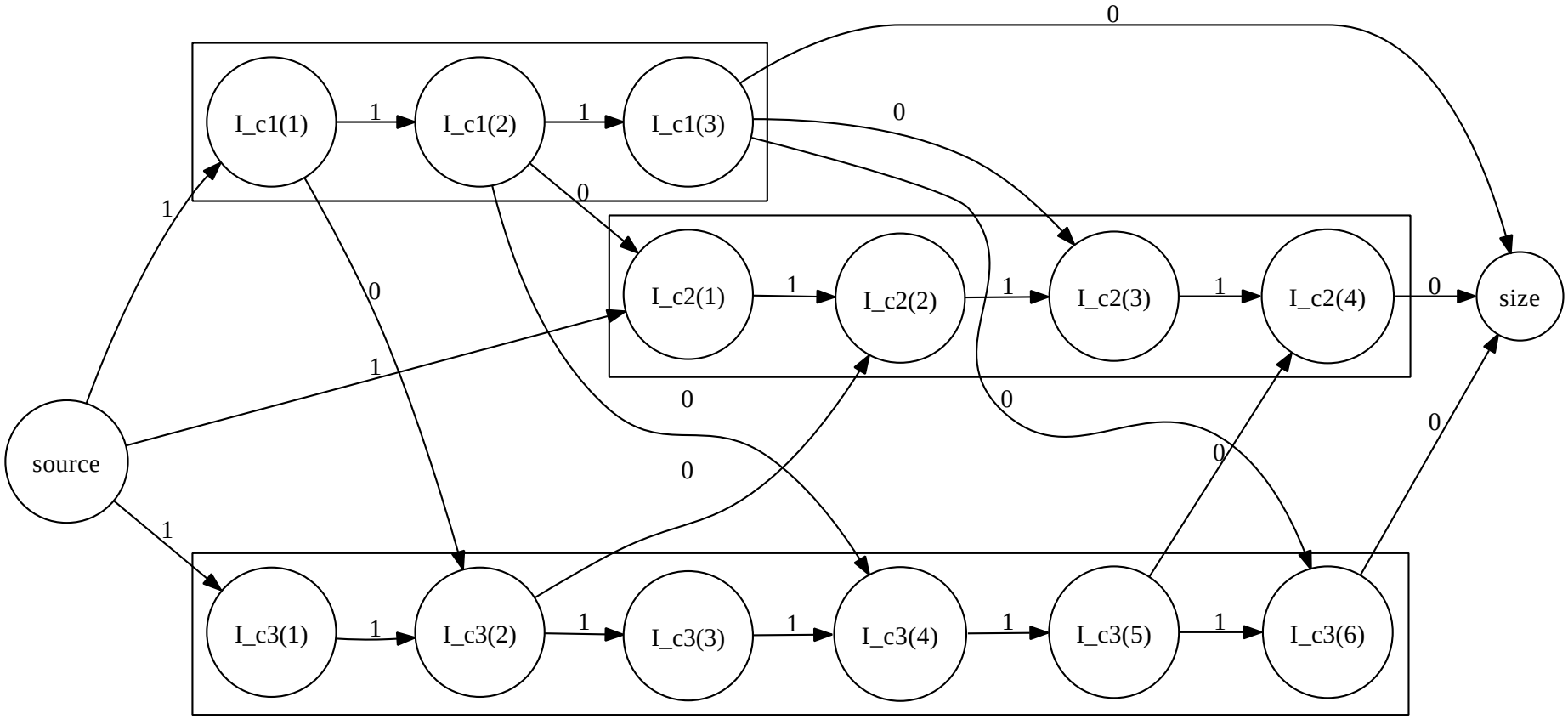
1. Instantiation of type variables

$$\alpha_1 \leftarrow \alpha \texttt{ on } c_1, \quad \alpha_2 \leftarrow \alpha \texttt{ on } c_2, \quad \alpha_3 \leftarrow \alpha \texttt{ on } c_3$$

$$\left\{\begin{array}{lcl} c_1 \texttt{ on (011)} & <: & c_2 \texttt{ on (10)} \\ c_1 \texttt{ on (1)} & <: & c_3 \texttt{ on (01)} \\ c_3 \texttt{ on (010)} & <: & c_2 \texttt{ on (01)} \end{array}\right\}$$

Paces Inference

2. Transformation into linear inequations on indexes of 1s in clock variables c_i



3. Resolution using standard techniques (e.g. Glpk)

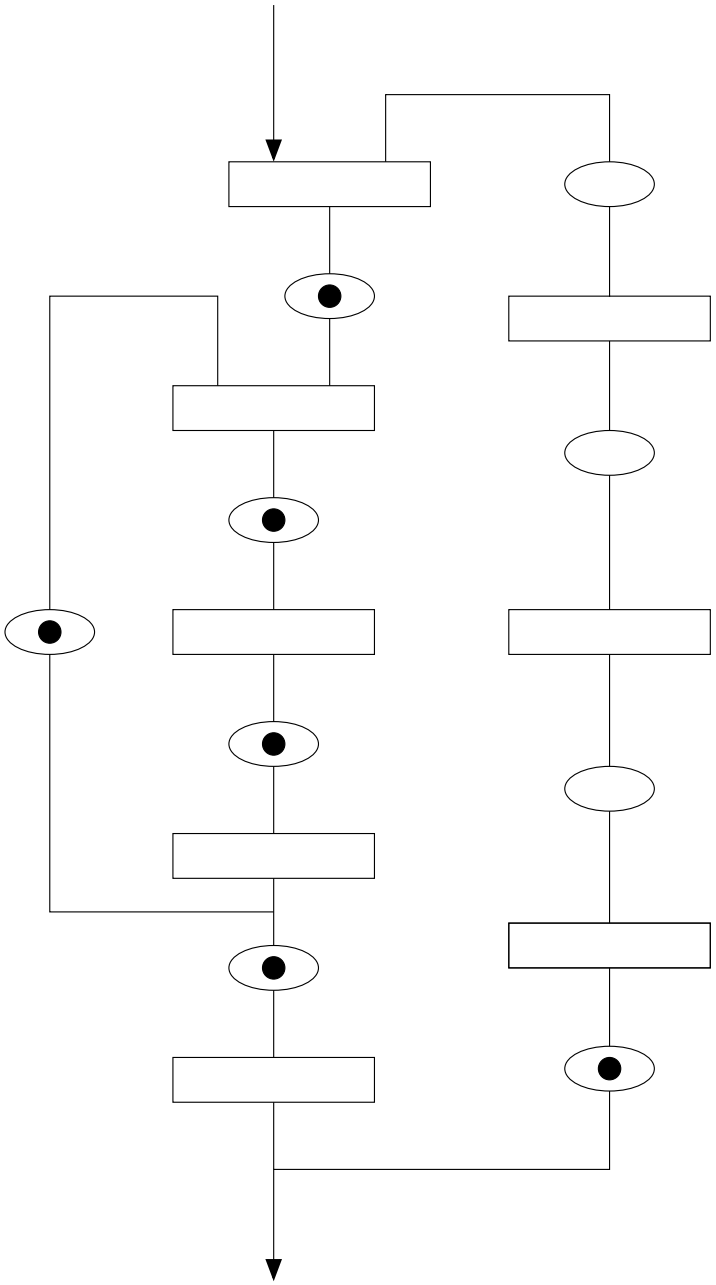
Latency Insensitive Design

Latency Insensitive Design [CMSV01]

Method used to design synchronous circuits that tolerate data transfer latency

- design synchronous IPs and interconnect them
- add relay stations and buffers on the wires
- analysing the sum of delays and initial values on each cycle indicates whether the system is alive [KCKO08]
- elastic circuits dynamic schedule [KCKO06]: every wire is transformed into a channel carrying data and control bits
- k-periodic static schedule [BdSM07]: maximize rate and minimize storage elements by insertion of fractional registers and computation of an explicit schedule

Jean-Vivien Millo Example [Mil08]



Strict Buffers

Example:

```
let node f x =
  strict_buffer (x when (100))
```

val f :: forall 'a. 'a -> 'a on (010)

LID encoding:

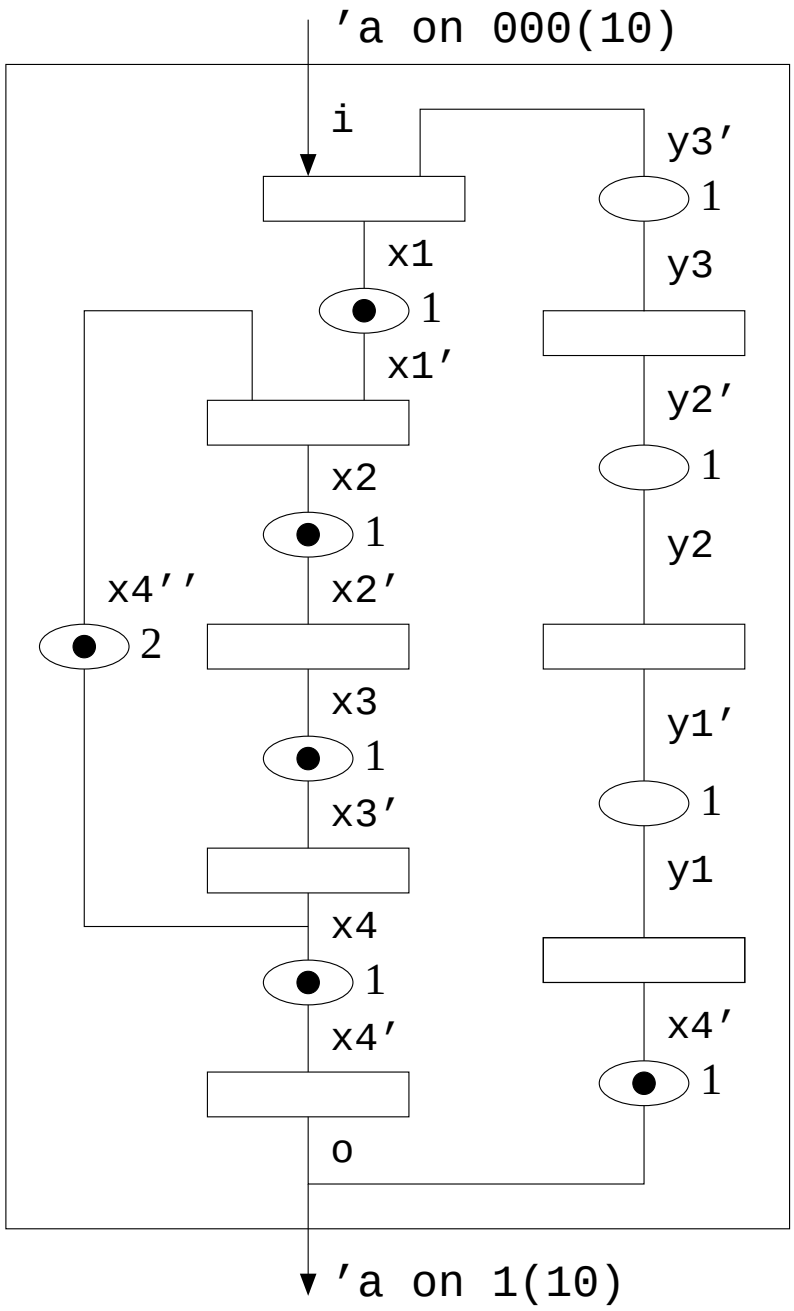
- uninitialised relay station:

  ```
  x' = strict_buffer x
  ```

- initialised relay station:

  ```
  x' = (merge 1(0) init (strict_buffer x))
  ```

Jean-Vivien Millo Example [Mil08]




```
let node f i = o where
  rec x1 = ip2(i, y3')
  and x1' = (merge 1(0) 42 (strict_buffer x1))
  and x2 = ip2(x4'', x1')
  and x2' = (merge 1(0) 42 (strict_buffer x2))
  and x3 = ip1(x2')
  and x3' = (merge 1(0) 42 (strict_buffer x3))
  and x4 = ip1(x3')
  and x4' = (merge 1(0) 42 (strict_buffer x4))
  and x4'' = (merge 1(0) 42 (strict_buffer x4))
  and o = ip1(x4')
  and o' = (merge 1(0) 42 (strict_buffer o))
  and y1 = ip1(o')
  and y1' = strict_buffer y1
  and y2 = ip1(y1')
  and y2' = strict_buffer y2
  and y3 = ip1(y2')
  and y3' = strict_buffer y3
```

Contribution of our method about scheduling LID (temporary conclusion)

Static schedules computed automatically

- some existing semi-automatic techniques find better schedules [Mil08]
- other automatic techniques do not find better schedules [Bou07]

Modularity

- statically scheduled IPs can be composed in a latency insensitive way

Conclusion and Future Work

Conclusion

Summary

- n-synchronous model:
  more flexible composition of nodes without loss of guaranties
- two clocks languages studied: periodic clocks and abstract clocks [MPP10]
- algorithms implemented in Lucy-n
- first experiments about static scheduling in Latency Insensitive Design

Future Work

- find better schedulings
- more efficient resolution algorithms
- code generation

References

[BdSM07] Julien Boucaron, Robert de Simone, and Jean-Vivien Millo. Formal methods for scheduling of latency-insensitive designs. *EURASIP Journal on Embedded Systems*, Issue 1(ISSN:1687-3955):8 – 8, January 2007.

[Bou07] Julien Boucaron. *Modélisation formelle de systèmes Insensibles à la Latence et ordonnancement*. PhD thesis, Université de Nice Sophia-Antipolis, 2007.

[CMSV01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[KCKO06] S. Krstic, J. Cortadella, M. Kishinevsky, and J. O'Leary. Synchronous elastic networks. In *Proceedings of the Formal Methods in Computer Aided Design*, 2006.

[KCKO08] Sava Krstic, Jordi Cortadella, Mike Kishinevsky, and John O'Leary. Defining elastic circuits with negative delays. In *Designing Correct Circuits (DCC 2008)*, Budapest, Hungary, March 2008.

[Mil08] Jean-Vivien Millo. *Ordonnancements périodiques dans les réseaux de processus : Application à la conception insensible aux latences*. PhD thesis, Université de Nice-Sophia Antipolis, Décembre 2008.

[MPP10] Louis Mandel, Florence Plateau, and Marc Pouzet. Lucy-n: a n-synchronous extension of Lustre. In *Tenth International Conference on Mathematics of Program Construction (MPC 2010)*, Québec, Canada, June 2010.