TELECOM ParisTech

Institut Mines-Télécom

Parallel Asynchronous Program Analysis, an Example

Gérard Memmi

Agenda

- Few concepts
- An example
- Transition Systems
- Invariants
- Reachability Graph

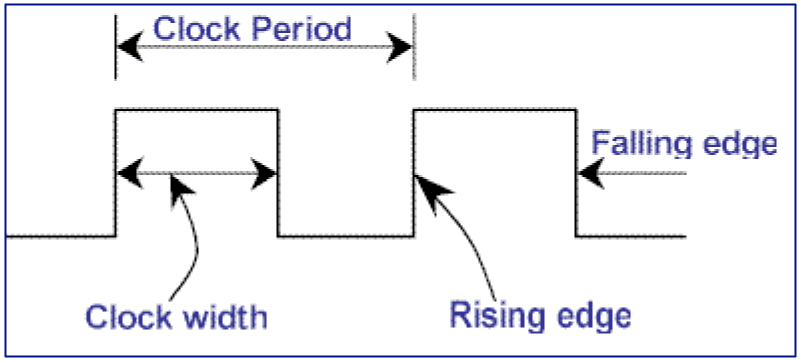
Few Concepts

First basic informal definitions

- Time, clock cycles, and clock domains
- Systems and Events
- Synchrony and asynchrony
- Processes and transitions

Notion of time = notion of clock

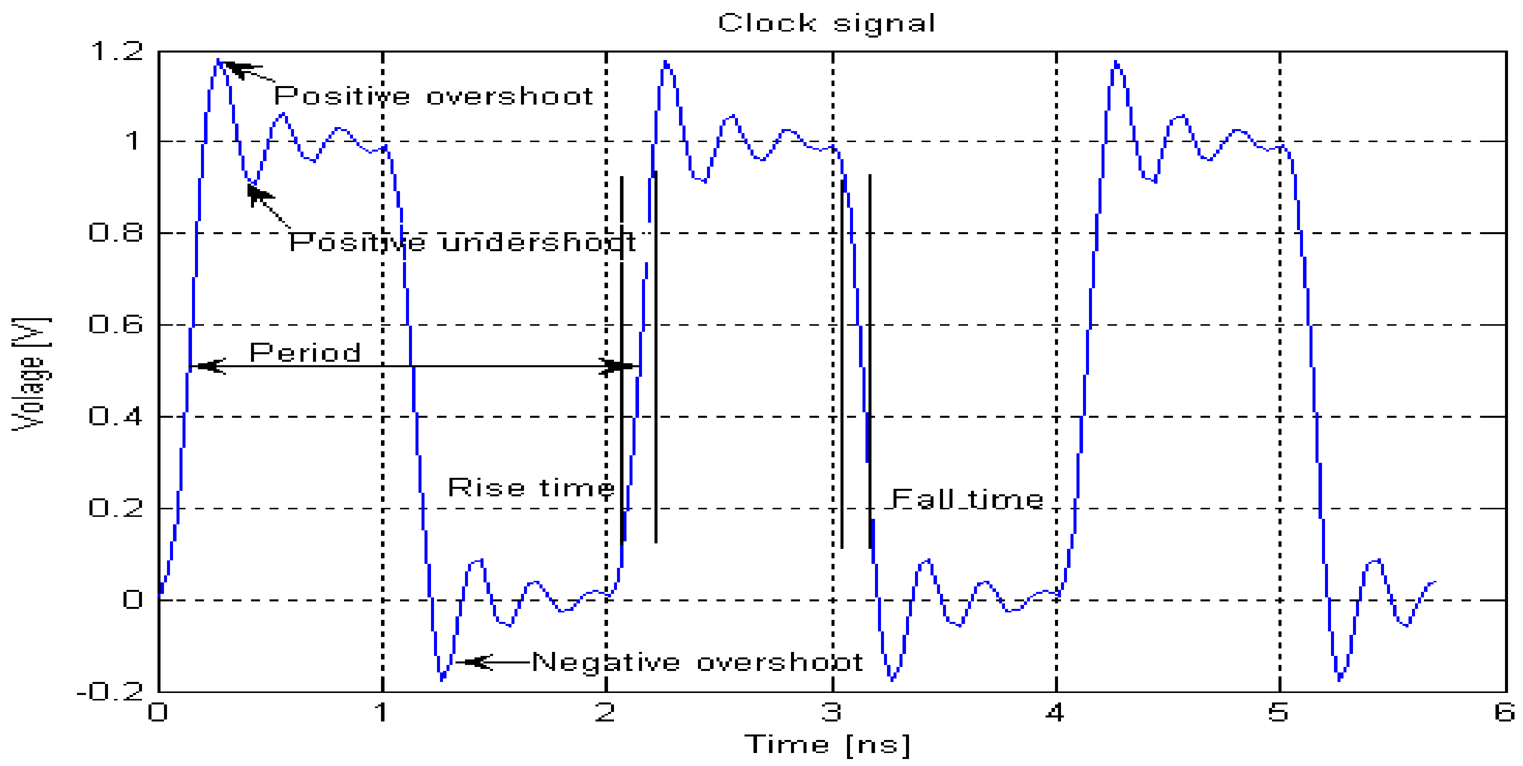
- In Computer Science, time is mostly a discrete variable cadenced by a clock



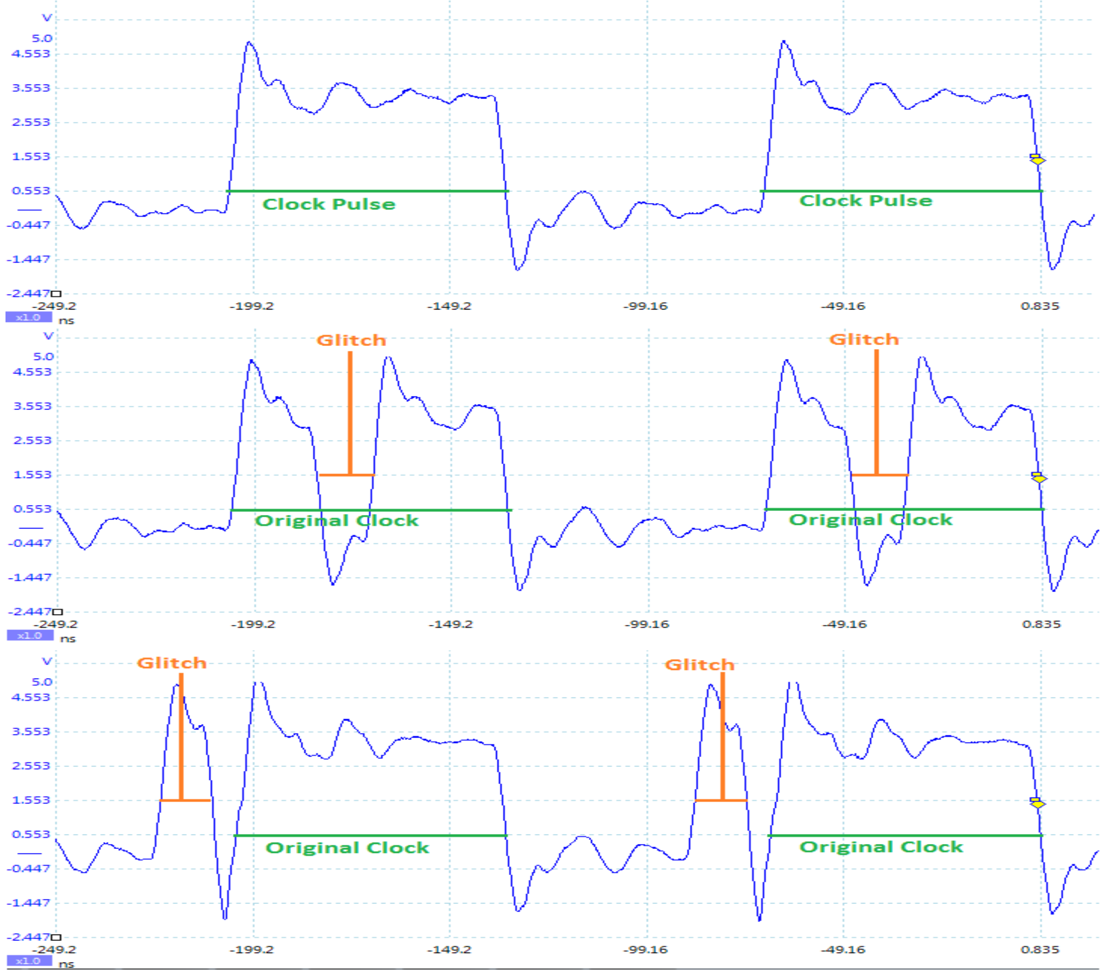
- An observer has his own clock that provides him with a corresponding level of accuracy
- A system has one or several clocks
- A clock is a special resource that drives events or transitions in a system or an associated part of a system (aka clock domain)

Clocks, clock cycles, and clock domains

- **In the real world a lot of ‘bad things’ can happen to clocks:**
  - poor transient period, a clock can **jitter**
  - Time difference between arriving clk edges at different places (**skew**)



Clocks also can have a signal glitch



Original Clock

7.37 MHz

Width = 10%
Offset = +15%

Clock XORd with Glitch

Width = 10%
Offset = -15%

Clock XORd with Glitch

- Conclusion : Model verification does **not** prevent system testing, however, it considerably brings confidence and speeds validation up

Synchrony and asynchrony

- Two events are simultaneous for an observer when they happen in the same ‘*moment*’ ie the same clk cycle of his own clock (the observer cannot differentiate the dates when the two events happen).
- A system is synchronous when simultaneous events have specified temporal constraints (via a central clock).
- Two events can run in parallel if they can be interleaved.
- When time (in term of duration) is not taken into account the system will be considered asynchronous. Parallelism can be optimized.

Systems and Events

- **A digital SYSTEM is using a set of resources evolving over time in order to execute a function or an activity. In general, it has inputs and outputs.**
  - A **state** is a temporal function that describes resource (including I/O) value evolution.
  - The outputs of a system ‘should be’ a function of its inputs and the system initial state (determinism).
- **An EVENT can consume or produce resources, it participates in changing the system state. According to a level of granularity of observation, it can be an activity, or associated with the beginning or the end of an activity**
- **An event has pre-conditions sometimes post-conditions. It is supposed to be instantaneous and indivisible for the system observer.**

Transitions and Processes

- **A TRANSITION is an indivisible system function that transforms one state into another one.**
  - A trajectory can be seen as a sequence of states.

- **A PROCESS is a sequence of possible events (or transitions) in order to achieve a piece of the system function. In general, a sequential process can be represented by an automaton.**
  - A process is distinguished because, it is going to be repeated or reused.
  - A trace is a sequence of transitions that represents the execution of a process.

System behavior

- A combination of trace and trajectory provide with the complete information to understand and analyze system behavior ie system executions.
- The set of all possible executions of is often represented by a graph : each node is a state, each edge a transition. This graph is sometimes called reachability graph: it supports proving whether a state is reachable or not from an initial state. Additional information can be attached to nodes or edges (probabilities, data types, time,…)
- A system property can be any statement over states and transitions, particularly a ‘temporal’ expression

An Example

Inspired from: “Specification and Validation of Sequential Processes Communicating by FIFO Channels.” R. Martin and G. Memmi 4th Int. Conf. on Software Engineering for Telecommunication Switching Systems, Warwick U. Conventry U.K, IEE pp 54-57, 1981.

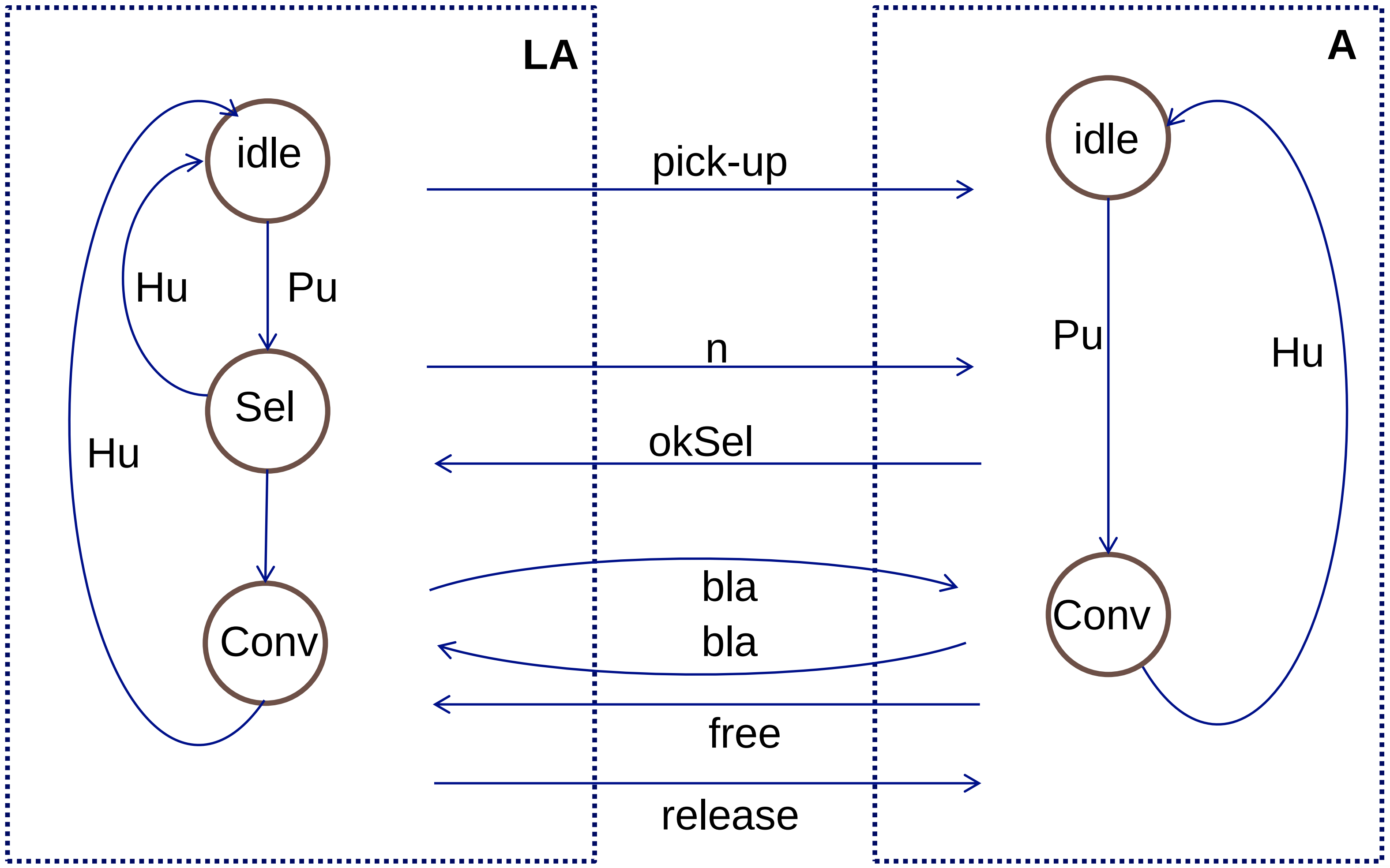
Two ‘subscribers’ called each other over a red line

- One subscriber (LA) picks up his phone and selects a second subscriber (A) via his phone number n, they have a conversation (bla), then go back to their respective occupations.
- Their phones start and finish in an ‘idle’ state
- Each subscriber can hang up (Hu) at any moment releasing the line, then pick up (Pu) again at any moment. Pu and Hu are signals sent by humans to their local device (phone)
- The switching system is not to be modeled : the communication between LA and A is modeled by {n, bla} the phone number and the conversation between A and LA; a set of control messages that transit through particular queues:

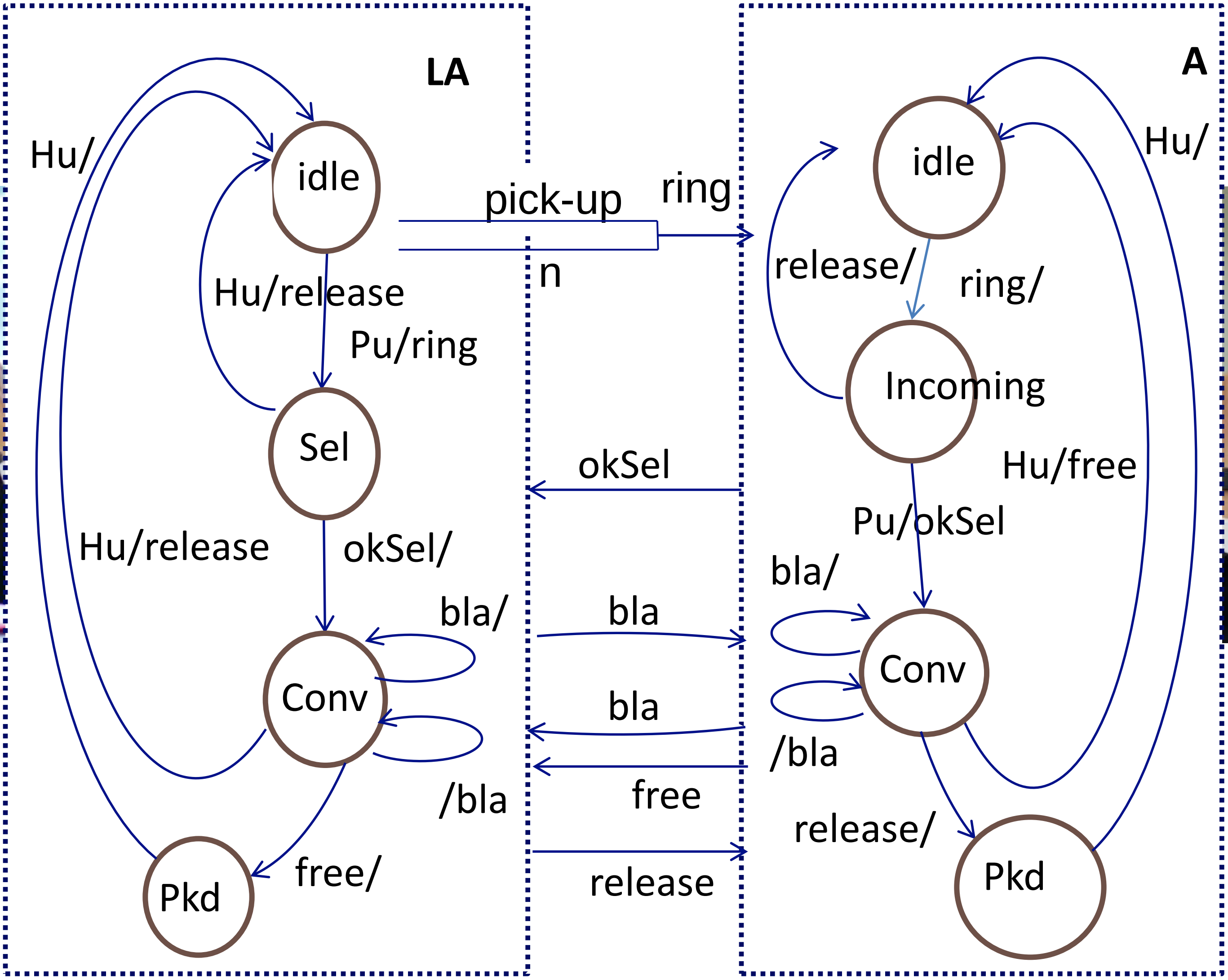
  {pick-up, release} from LA to A

  {okSel, free} from A to LA

A first informal approach



A second more precise UML-like description (receive/send)



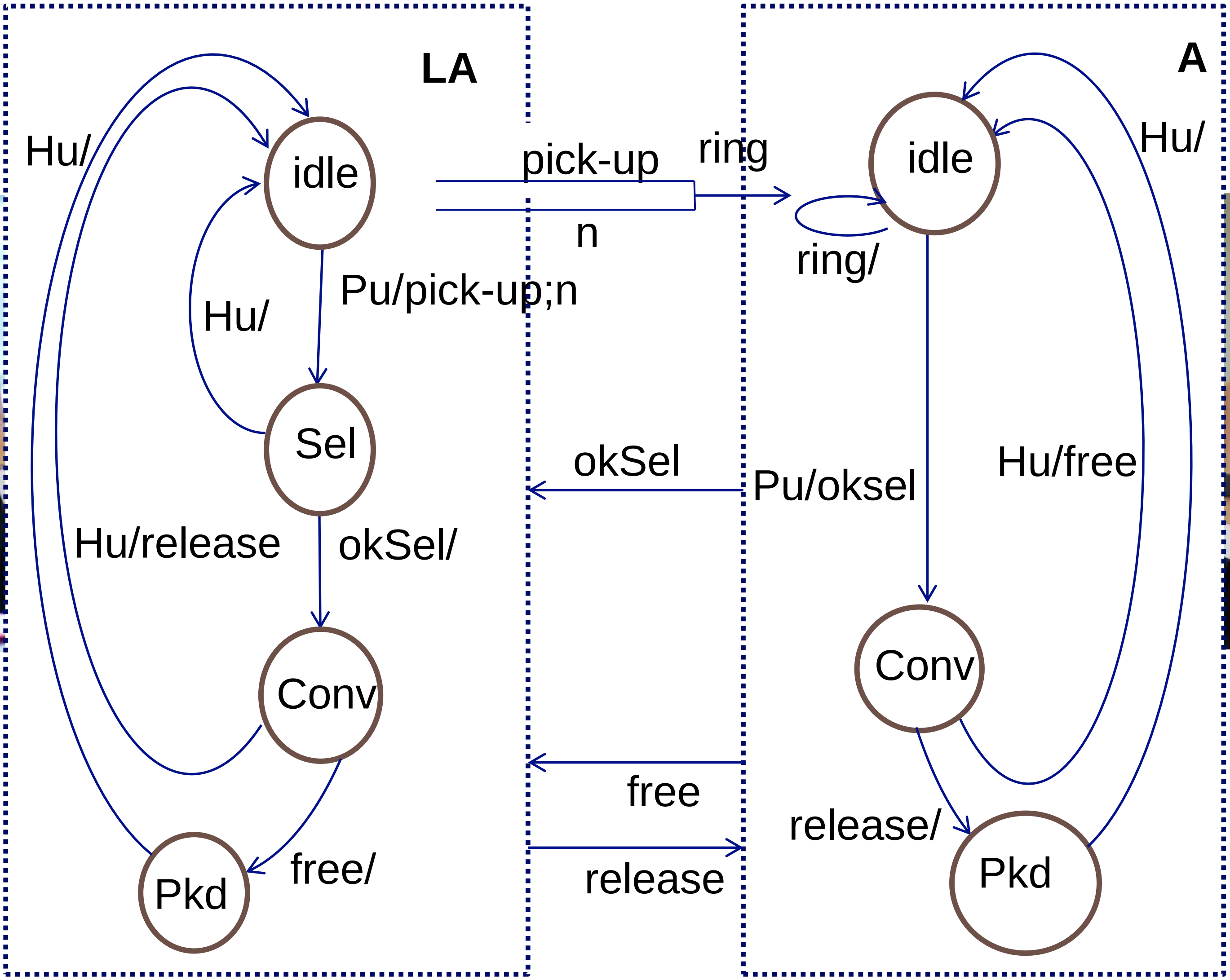
What do we want ?

- We want to model the phones LA and A as sequential processes exchanging messages (through queues) such that we can analyze their asynchronous behavior
- We want to understand/prove that LA and A can always both go back to their initial state leaving all control messaging queue ‘clean’ (no message left) for any order in which signals are exchanged

Few comments

- **The messages n, ring, and bla seem redundant or even useless from an analysis point of view**
  - n and ring always happen behind pick-up (if not a timeout bring you back to idle)
  - bla only happen when the two processes are in the conv state with no influence on the other parts of the processes
- **In the LA sequential process, we can loop over pick-up and hang up transitions and send an infinite number of pick-up signals**
- **free and release are symmetrical and can happen any time in any order depending on who is first hanging up**

A simplified UML-like description (receive/send)



Theoretical Models for Asynchronous Systems

Quick History

Carl Adam Petri 1926-2010

- **Petri Nets were invented in the 60's by Carl Adam Petri**
- **In the same period other models were introduced:**
  - Transition Systems
  - Vector Addition System (VAS)
- **Other formal models were introduced later:**
  - Milner's Calculus for Communicating Systems (CCS)
  - Ho's Discrete Events Dynamic Systems (DEDS)
- **Motivation was to model the notion of <u>asynchronous parallelism</u> or concurrency**
  - A finite State Machine (FSM) behaves fundamentally sequentially

Models: quick comparisons

- **In term of power of description:**

  FSM < PN < Turing Machines

- **Despite defined independently, it was proven that PN and VAS were ‘somewhat’ equivalent**
- **Parallel programs are seen as a set of sequential processes sharing resources (including exchanging messages)**
  - Each sequential process can be seen as a Finite State Machine

Transition Systems

- **A <u>transition system</u> is a tuple (S, T, →) where:**
  - S is a set of states,
  - T is a finite set of transitions and
  - $\rightarrow \subseteq S \times T \times S$ relates transitions with states.
- **Each transition can be seen as a function from S to S**
- **Most of the time, an element s_0 of S is distinguished and called <u>initial state</u>**
- **Most of the time, S will be an element of $\mathbb{N}^d$ to model amounts of d different kind of resources the system disposes of**
- **A trajectory (a run , or a trace) is a:**
  - word of T^*
  - sequence $s_1, s_2, s_3, \ldots s_k$ such that $\forall$ $i<k$, $\exists\ t \in T$, such that $s_i \xrightarrow{t} s_{i+1}$

Vector Addition Systems (ref: J. Leroux)

Definition

A vector addition system (VAS) is a finite set $\mathbf{A} \subseteq \mathbb{Z}^d$.

$\mathbf{A}$ set of actions.
$\mathbb{N}^d$ set of markings.

A run is a non-empty word $\rho = \mathbf{m}_0 \ldots \mathbf{m}_k$ of markings such that:

$$\forall j \in \{1, \ldots, k\} \qquad \mathbf{m}_j \in \mathbf{m}_{j-1} + \mathbf{A}$$

In this case, $\mathbf{m}_k$ is said to be reachable from $\mathbf{m}_0$.

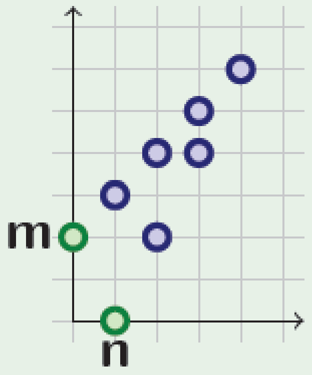
Theorem (Mayr 1981, Kosaraju 1982)

The reachability problem is decidable.

Reachability

Example

$\mathbf{A} = \{ \nearrow, \swarrow \} = \{ \binom{1}{1}, \binom{-1}{-2} \}$

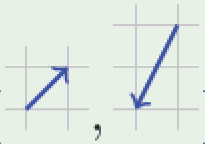


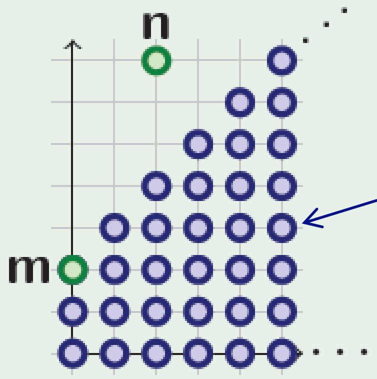
n is reachable from **m**.

$\rho = (0, 2)$ (

Unreachability

Example

$\mathbf{A} = \{$  $\}$



n is <u>not</u> reachable from **m**.

$$\phi(x_1, x_2) \;:=\; 0 \leq x_1 \;\wedge\; 0 \leq x_2 \;\wedge\; x_2 \leq x_1 + 2$$

Petri Nets

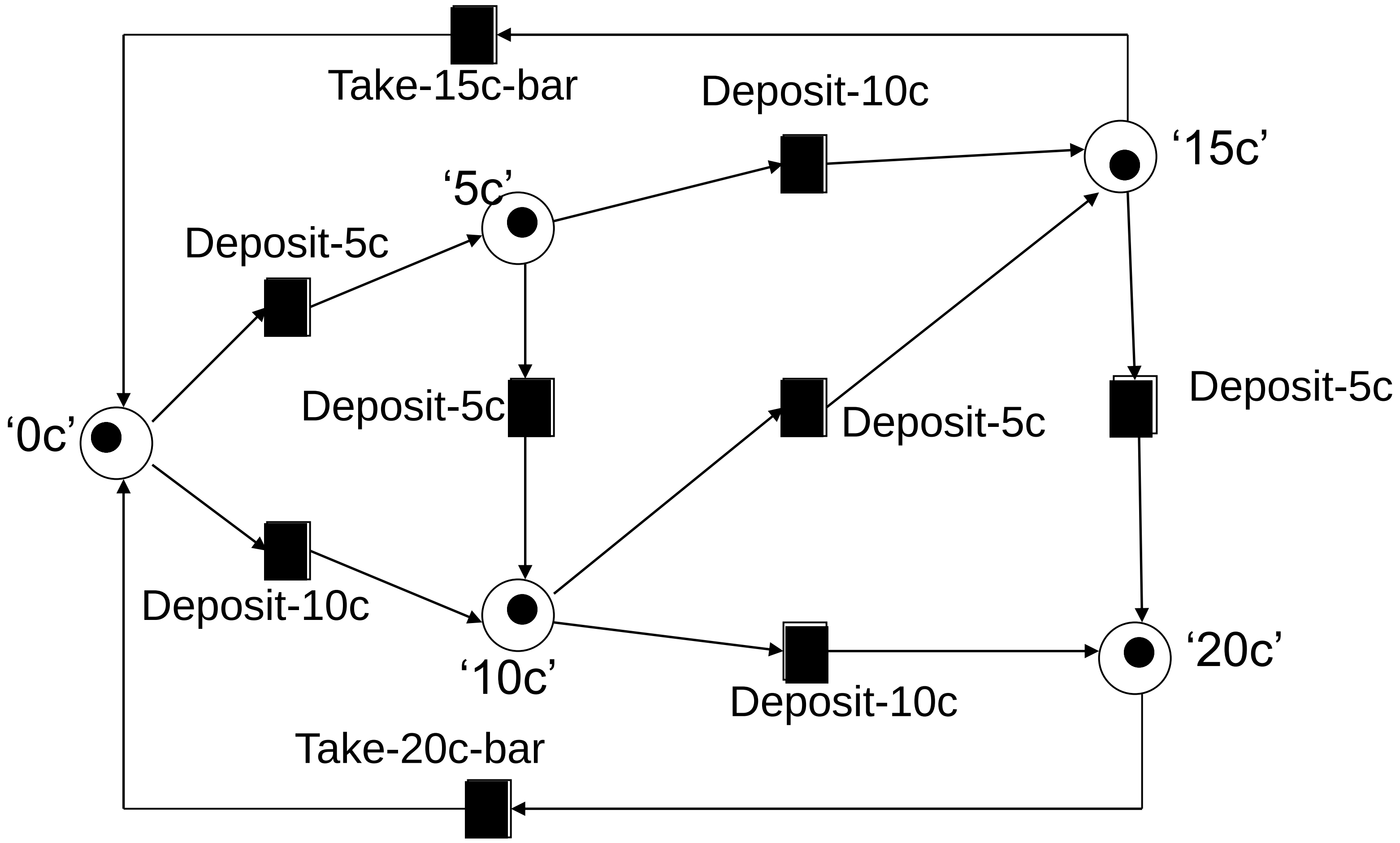
- **A Petri Net is a class of transition systems where P is a set of places (circles) , T a set of transitions (rectangles), an initial marking (or initial state) Mo is a function from P to $\mathbb{N}$.**
  - We dynamically move from one state to the next by ‘firing’ (executing, enabling) a transition t from M iff its preconditions (Pre) hold
    - Pre: PxT -> $\mathbb{N}$ gives the weight from places to a transitions
    - $M \geq Pre(\textbf{.},t)$
  - The relationship between P and T is described by a labeled bigraph

<P, T; Pre, Post> also represented by an incidence matrix

C = Post-Pre

- Post: PxT -> $\mathbb{N}$ gives the weight from transitions to places
- A sequence of transitions (or trace) R is associated with a vector $\vec{R}$ such that $\vec{R}_i$ is the number of occurrences of the transition t_i in $\vec{R}$
- From Mo, we reach M by enabling R with the equation:
  - $M = C\vec{R} + Mo$

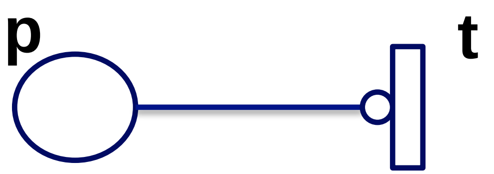
Example: Vending Machine (Token Games)



Test to 0 and Priority

- **Inhibitor edge:**

t is fireable iff M(p) = 0



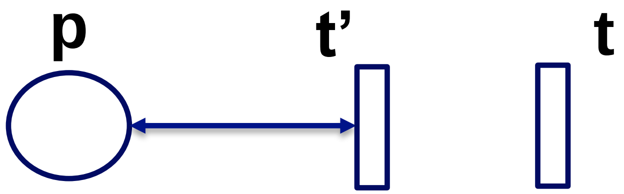
- **Priority:**

An order is defined over T, t is fireable from M iff

$M \geq Pre(.,t)$

$\nexists\ t' > t\ ,\ M \geq Pre(.,t')$

- **We can model an inhibitor edge between p and t using priorities**



By adding t' such that: t'>t and adding a loop between p and t':
t can't fire as long as there is a token in p

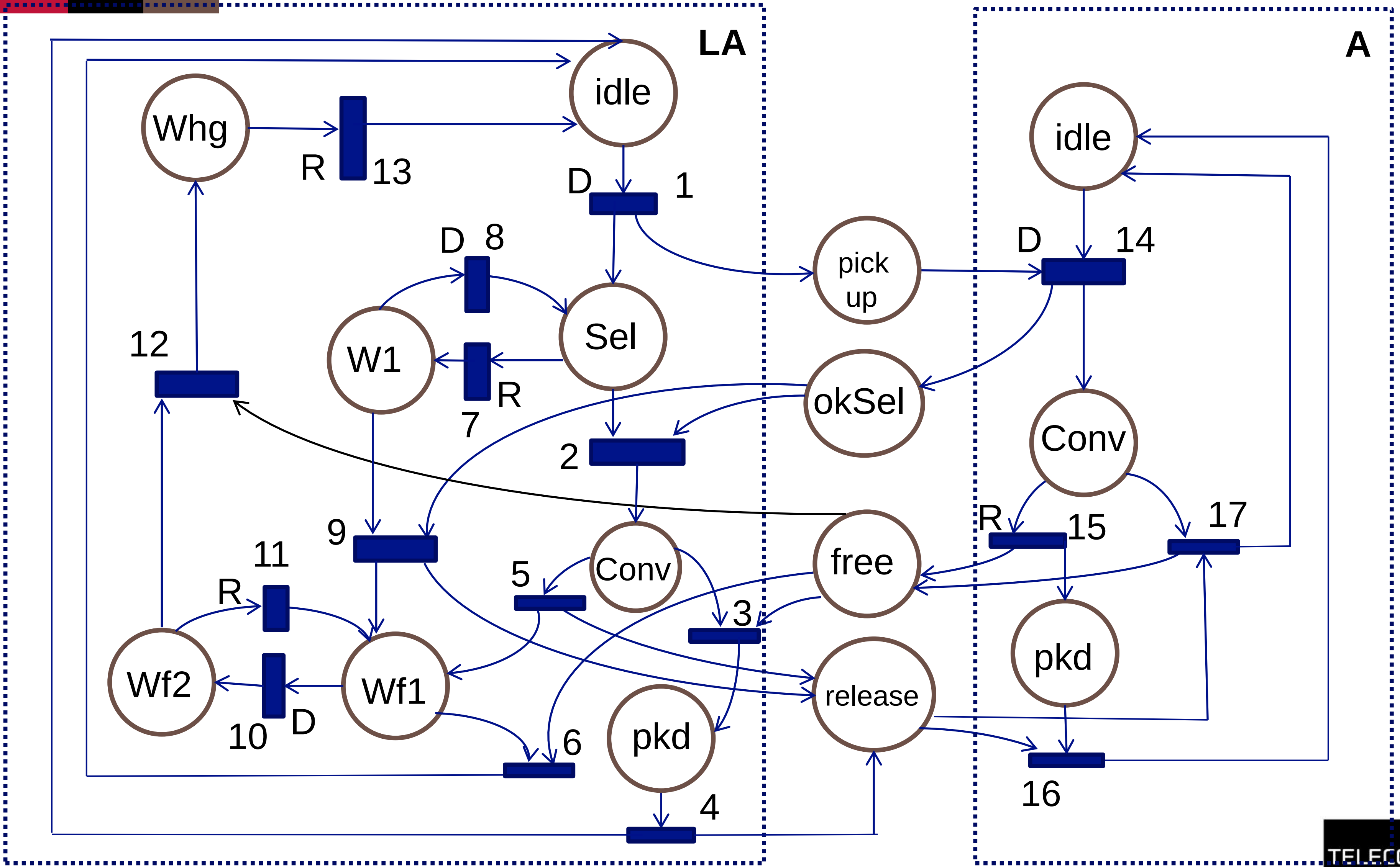
General properties

- A marking M is <u>reachable</u> iff it exists a sequence going from M0 to M
- A transition t is live iff from any reachable marking M it exists a sequence reaching a marking M' allowing to fire t
- A Petri Net is <u>live</u> iff all transition is live

- A place is bounded by an integer k>0 iff it does not exist a reachable marking M such that M(p)<k
- A Petri Net is <u>bounded</u> iff all place is bounded

- A marking H is a <u>home state</u> iff from any reachable marking M, it is possible to reach H

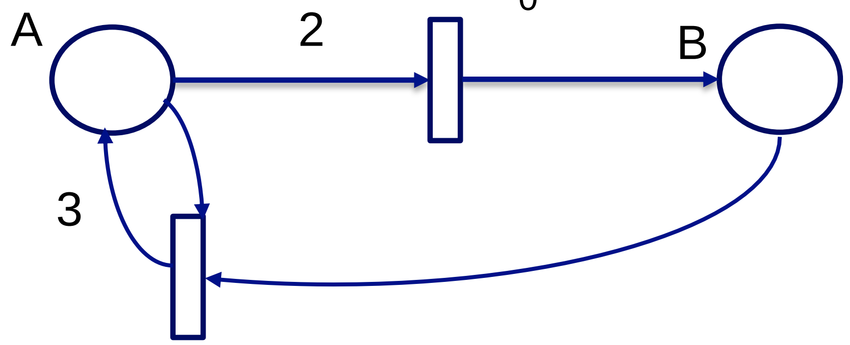
Modeling our example with a Petri Net



Invariants and linear invariants

Basic definitions

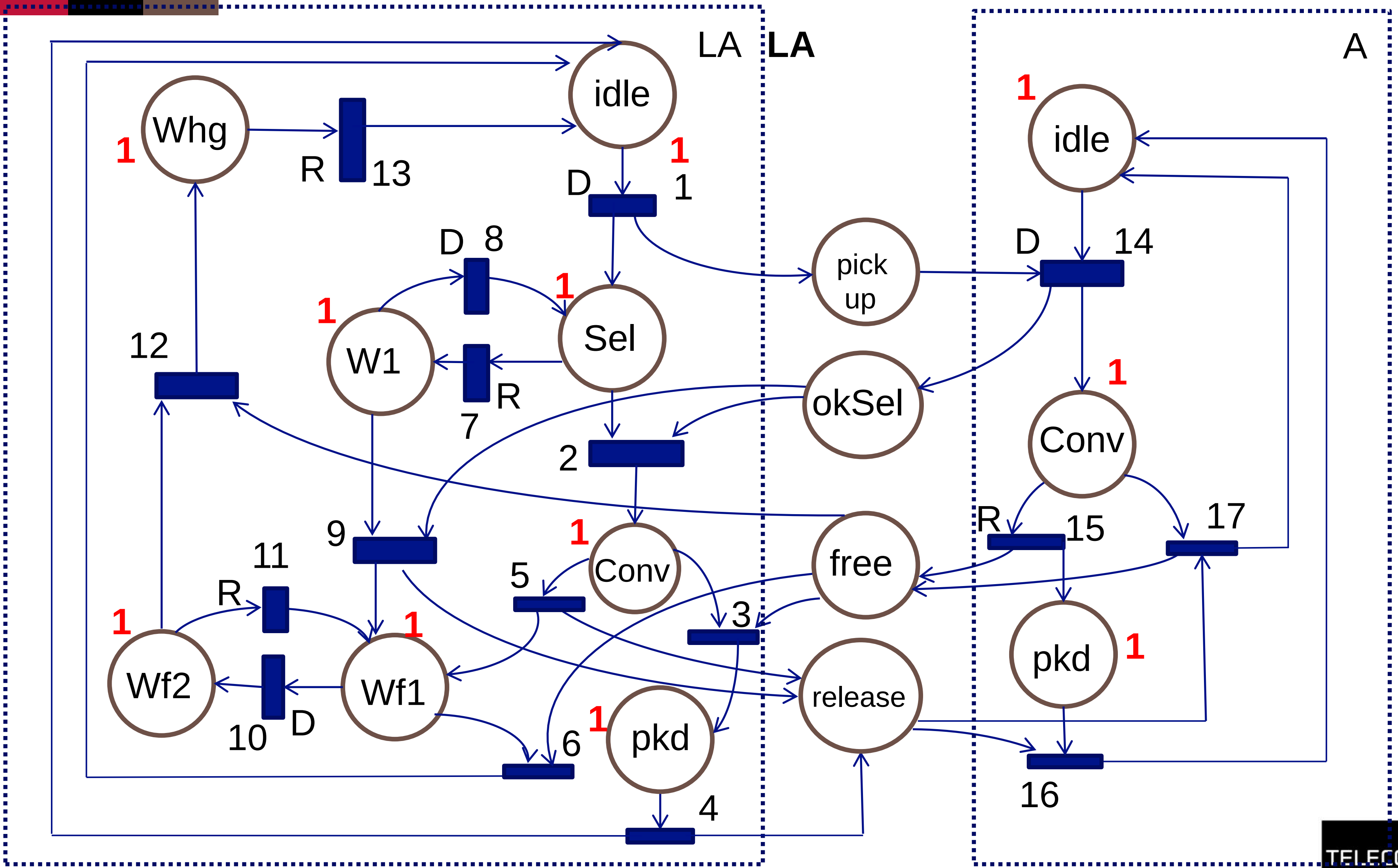
- **An invariant is a general property that is verified for all state of a transition system**
- **A linear invariant is a constant linear combinations over the values of the different resources (or places for a Petri Nets)**
  - For a Petri net, any linear invariant is associated with a weight function over the places named semiflow that verifies Kirchhoff ‘s law for each transition. We have : $f^TC = 0$ and $f^TM = f^TM_0$ for any marking M reachable from M_0 where $f^TM = \sum_{p \in P} f(p)\, M(p)$



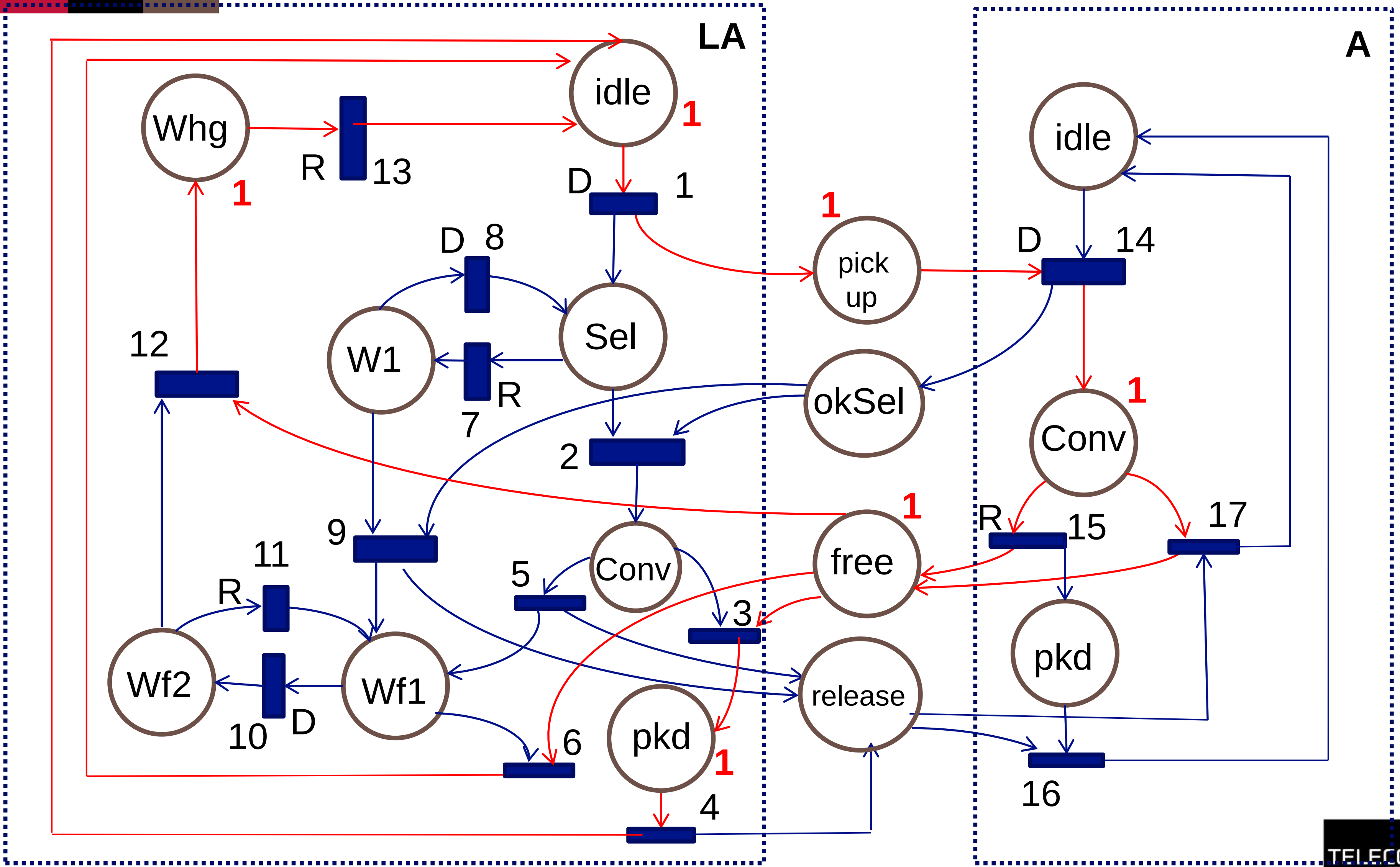
$f^T = (1, 2)$ is a semiflow

If $f^TM > 2$ and is odd then the net is live; if f^TM is even then the net is not live
Liveness is a non monotonic property !
Adding more token does not necessarily bring liveness

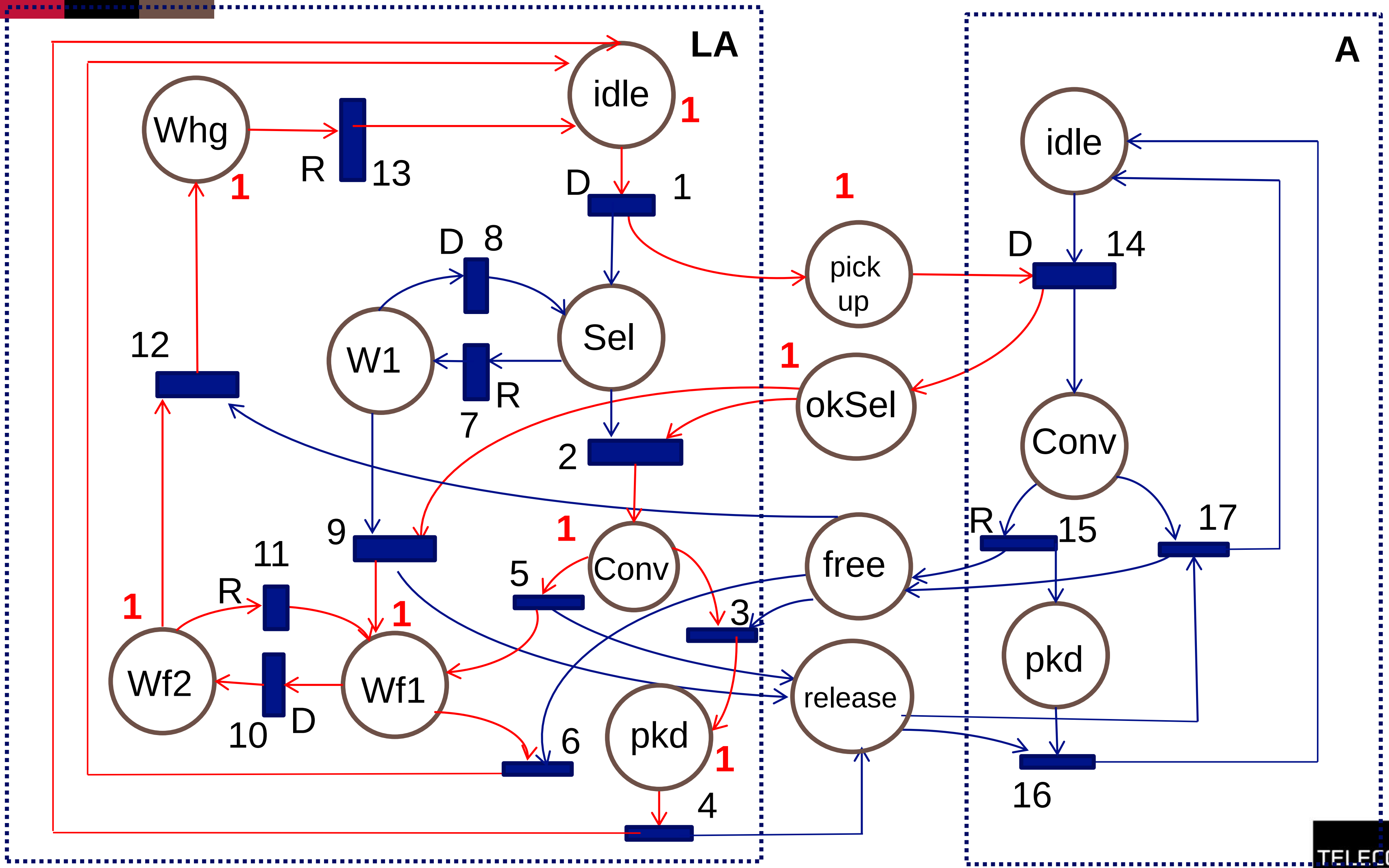
Linear invariants linked to LA and A



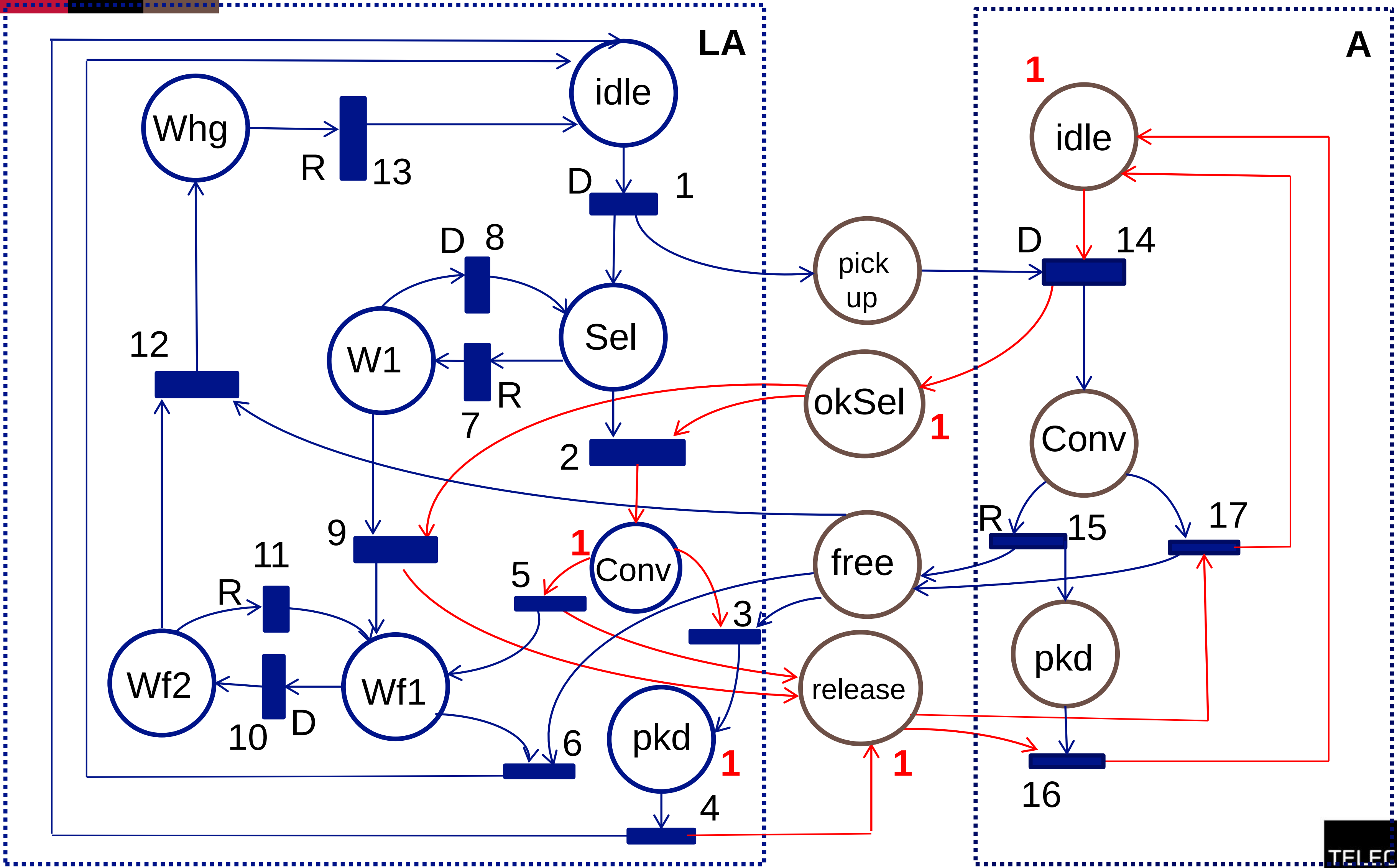
Linear invariant linked to {pick-up, free}



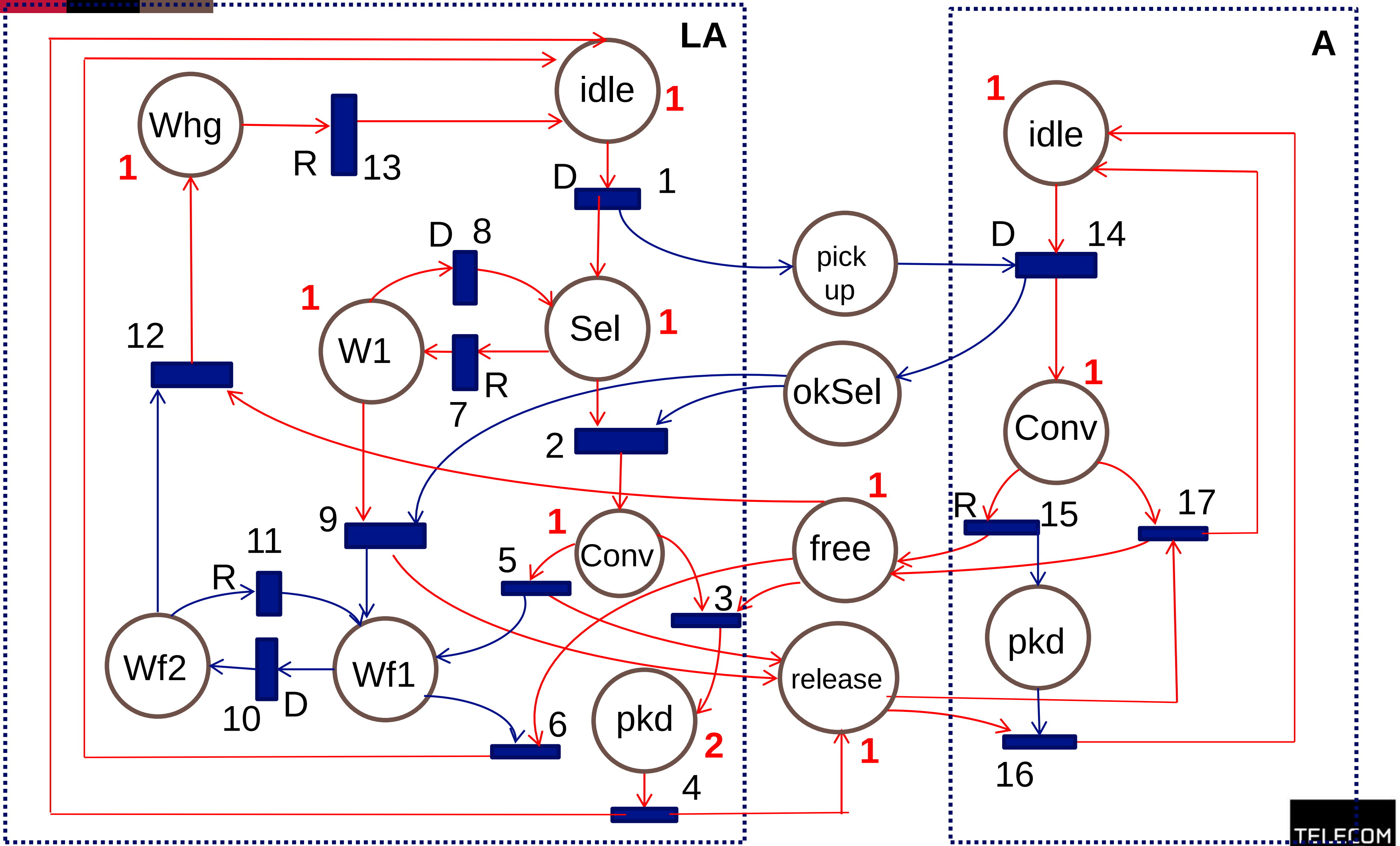
Linear invariant linked to {Pick up, Ok Sel}



Linear invariant linked to {okSel, release}



Linear invariant linked to {free, release}



Home spaces

- **A set of states H is a home space iff for any evolution of the system, it is always possible to go back in a state of H via a sequence of transitions.**
- **When |H| = 1, then , we say that H is a home state.**
  - If the initial marking is a home state then any reachable marking also is a home state
- **In our example, it can be proven that {(idle, idle, 0,0,0,0)} is a home state which answers our initial question:**

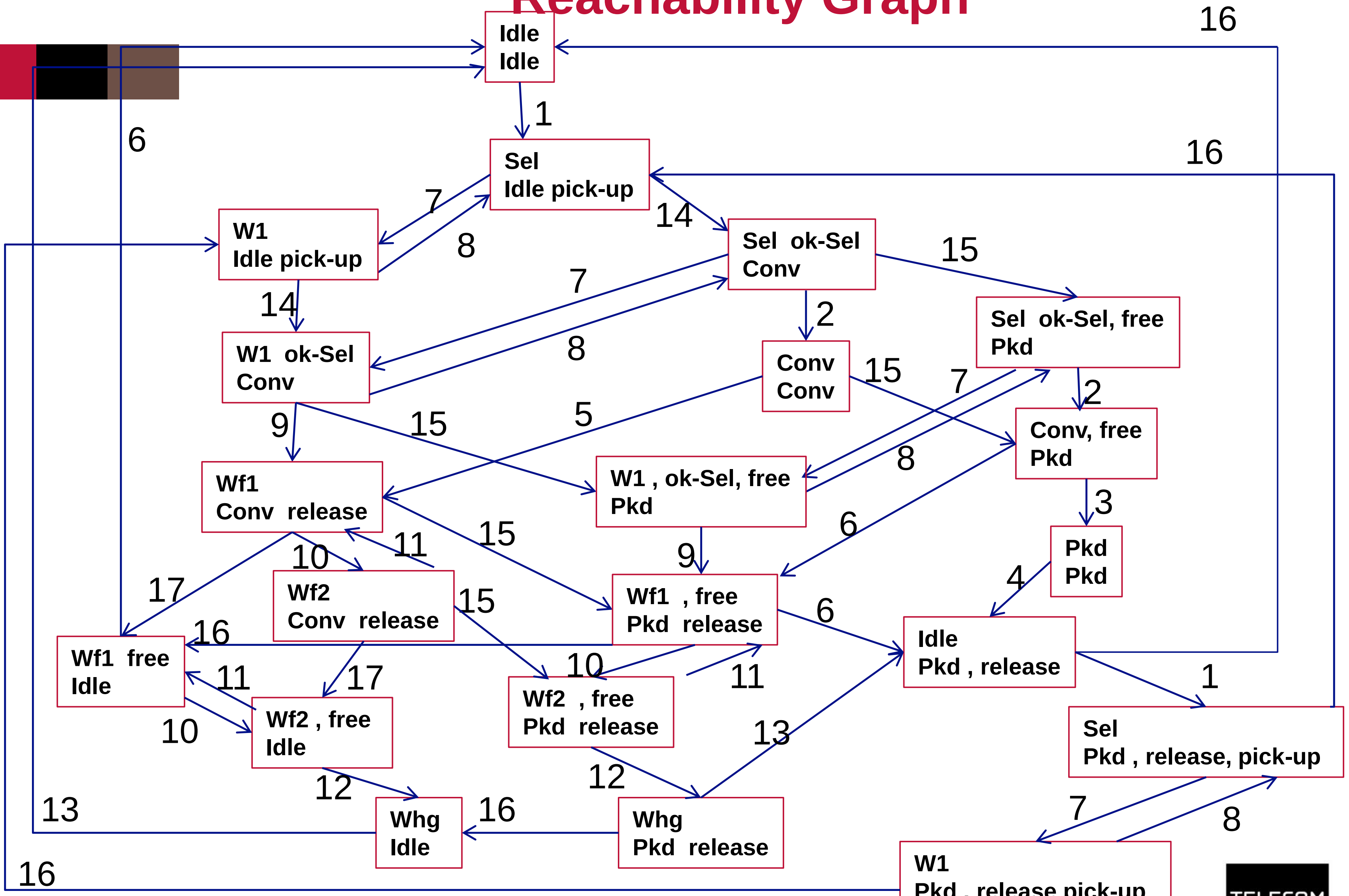
“*Can LA and A always both go back to their initial state leaving all control messaging queue clean?*”

Reachability Graph

Reachability Graph, definition and basic properties

- **A reachability graph is a graph where:**
  - Each node is labeled by a marking, each edge is labeled by a transition such that:
    - The edge (m, m’) is labeled by t iff m’= m + Post(.,t)-Pre(.,t)
    - Only one node is labeled by a given marking m
- **A reachability graph has a root labeled by m0, the initial marking**
  - For any node labeled by m, there exists a path from m0 to m.
  - m0 is a home state iff the reachability graph is strongly connected
- **The reachability is infinite iff some places are not bounded**

Reachability Graph



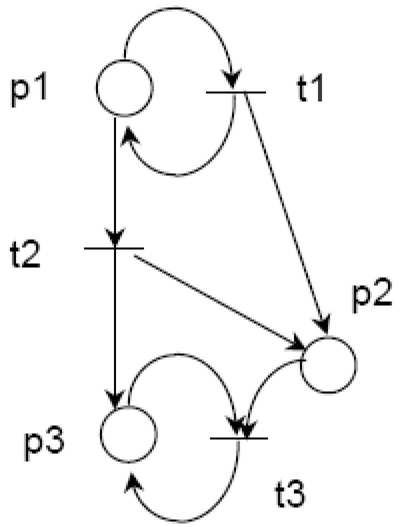
Coverability Tree

- **R=<P,T; Pre,Post; Mo>; AC(R) = <S,X> where S$\in\mathbb{N}_\omega^d$, where d=|P|,**
  - Mo labels r the root of the coverability tree AC
  - s of S labeled by Q of $\mathbb{N}_\omega^d$ has no successor iff
    - *Either* $\exists$ s' of S also labeled by Q in the path from r to s
    - *Or* there is no transition fireable from Q
  - If not, there are transitions fireable from Q. Then for each such t, we build a new edge labeled by t and ending by a node s' labeled by Q' such that for each place i of P:
    - Q'i = ω iff $\exists$ on the path from r to s' a label Q" such that:

      Q" ≤ Q + Post (.,t) – Pre(**.**,t) and Q"i < Qi + Post (i,t) –Pre(i, t)
    - Q'i= Qi + Post(i, t) - Pre(i, t) otherwise

Theorem (Karp & Miller 1969)
AC(R) is finite for any Petri Net R

Coverability Tree : an example (1/2) (ref : Hermann & Lin)



Initialization: M0 = (1,0,0) new

Step 1:

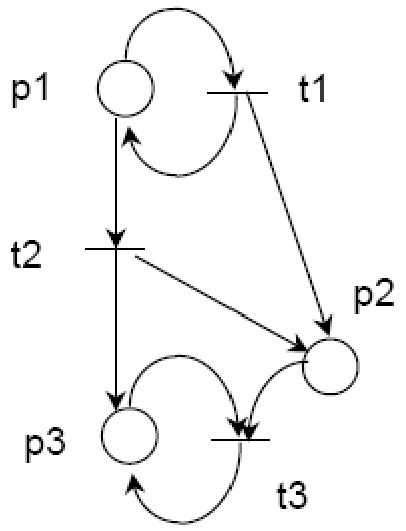
$$\begin{tikzcd}
 & (1,0,0) \text{ new} \arrow[dl, "t1"'] \arrow[dr, "t2"] & \\
\text{new } m1 = (1,\omega,0) & & m2 = (0,1,1) \text{ new}
\end{tikzcd}$$

m1 = (1,1,0) >= (1,0,0)

Step 2 (m1):

$$\begin{tikzcd}
 & & (1,0,0) \text{ new} \arrow[dl, "t1"'] \arrow[dr, "t2"] & \\
 & \text{new } m1 = (1,\omega,0) \arrow[dl, "t1"'] \arrow[d, "t2"] & & m2 = (0,1,1) \text{ new} \\
\text{old } (1,\omega,0) & \text{new } m3=(0,\omega,1) & &
\end{tikzcd}$$

Coverability Tree : an example (2/2)



Step 3 (m3):

$$\begin{tikzcd}
 & \text{(1,0,0) new} \arrow[dl, "t1"'] \arrow[dr, "t2"] & \\
\text{new m1=(1,}\omega\text{ ,0)} \arrow[d, "t1"'] \arrow[dr, "t2"] & & \text{m2=(0,1,1) new} \\
\text{old (1,}\omega\text{,0)} & \text{m3=(0,}\omega\text{,1) new} \arrow[d, "t3"] & \\
 & \text{old (0,}\omega\text{,1)} &
\end{tikzcd}$$

Step 4 (m2):

$$\begin{tikzcd}
 & \text{(1,0,0) new} \arrow[dl, "t1"'] \arrow[dr, "t2"] & \\
\text{new m1=(1,}\omega\text{ ,0)} \arrow[d, "t1"'] \arrow[dr, "t2"] & & \text{m2=(0,1,1) new} \arrow[d, "t3"] \\
\text{new (1,}\omega\text{,0)} & \text{m3=(0,}\omega\text{,1) new} \arrow[d, "t3"] & \text{m4 = (0,0,1) new} \\
 & \text{old (0,}\omega\text{,1)} &
\end{tikzcd}$$

Nothing is fireable from m4

Model Checking

- **Reachability graphs or coverability trees are used by the "Model checker" algorithm to prove a behavioral property often expressed in terms of temporal logic**
- **Model checking is highly complex.**
  - For instance, LTL model checking is coNP-hard and PSPACE-complete
- **'State Space explosion' prevents complete proof of most "real" property on any "large" system. However, it has been critical to find 'deep' errors (after several hundreds of thousands transitions).**
- **Coverability tree can be used to detect any unbounded resources, however, it cannot be used to prove any property, for instance, if a given state can be reached from the initial state?**
  - This question has been proven decidable for Petri Nets or equivalent models

Bibliography (1/2)

- GW Brams, “Réseaux de Petri , Théorie et Pratique, tome 1”, Masson 1983
- The World of Petri nets: http://www.daimi.au.dk/PetriNets/
- Y.C. Ho and X.R. Cao “Perturbation Analysis of Discrete Event Dynamic Systems” Kluwer Academic Publishers, Boston, 1991.
- R. Milner “Communication and Concurrency” Prentice Hall, International Series in Computer Science, ISBN 0-13-115007-3. 1989
- T. Murata “Petri nets: properties, analysis and applications.” Proceedings of the IEEE, 77(4), 541-80, (1989, April).
- W. Reisig and G. Rozenberg (eds) (1998). Lectures on Petri Nets 1: Basic Models. Springer-Verlag.
- M. Silva “Half a century after Carl Adam Petri's Ph.D. thesis: A perspective on the field.” Annual Reviews in Control 37(2): 191-219, 2013.

Bibliography (2/2)

- Slide 27 from Hermann J. & Lin E. “Petri Nets: Tutorial and Applications” Nov. 1997, http://www.isr.umd.edu/Labs/CIM/miscs/wmsor97.pdf
- Stephan Merz, Model Checking: A Tutorial Overview www.csee.usf.edu/~zheng/lib/verification/general/mc-tutorial.pdf