Hardware Synthesis with Petri Nets: Appendix

Alex Yakovlev, Victor Khomenko,
Andrey Mokhov and Danil Sokolov
Newcastle University, UK
async.org.uk
workcraft.org

PN/ACSD’15: Advanced Tutorial, Brussels, June 2015

Appendix

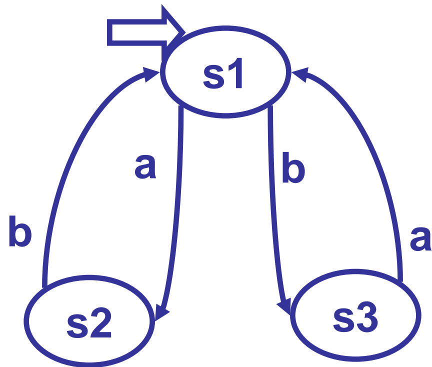
- **Synthesis of LPNs from Transition Systems (Using Region Theory)**
- **Direct synthesis of circuits from LPNs**
- **Examples**
- **Tools**

Synthesis from transition systems

- Modelling behaviour in terms of a sequential capture – *Transition System*
- Synthesis of LPN (distributed and concurrent object) from TS (using *theory of regions*)
- Examples: one place buffer, counterflow pp

Transition Systems

Original TS specification



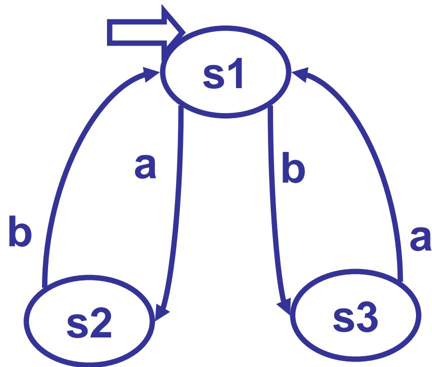
Not (semi-)elementary

The relationship between Transition Systems and Petri nets and conditions for synthesizability of a PN from a TS are based on *Theory of Regions*

(Ehrenfeucht, Rozenberg, Nielsen, Thiagarajan, Reisig, Mukund, Darondeau et al.)

Transition Systems and regions

Original TS specification

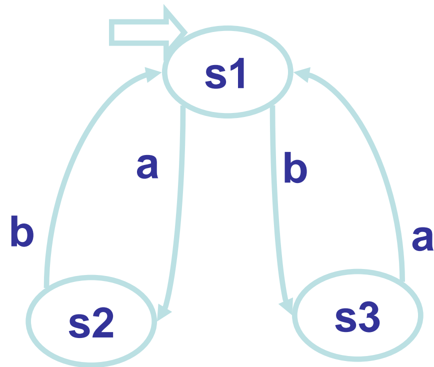


Not (semi-)elementary

No non-trivial regions!

Transition Systems and regions

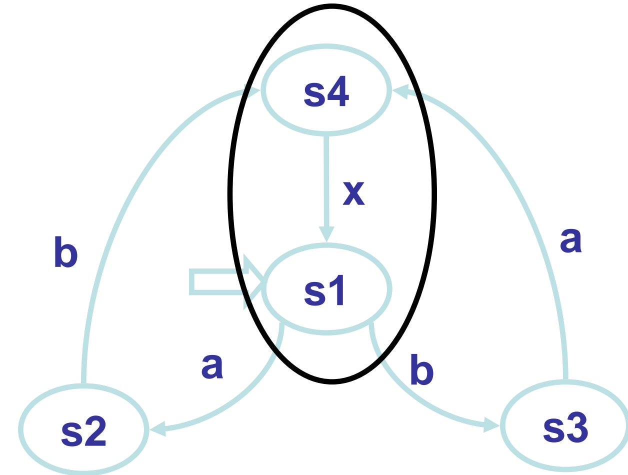
Original TS specification



Splitting states

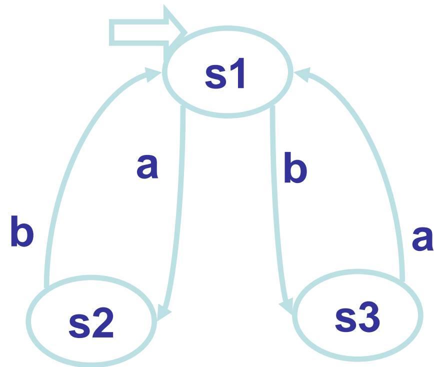
Inserting dummy events:

(x)



Transition Systems and regions

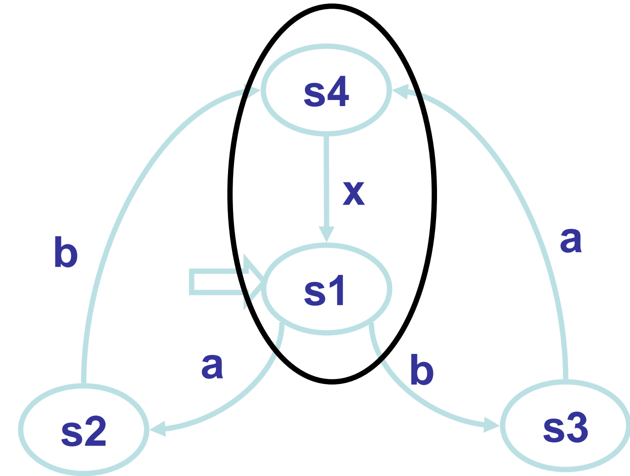
Original TS specification



Splitting states

Inserting dummy events:

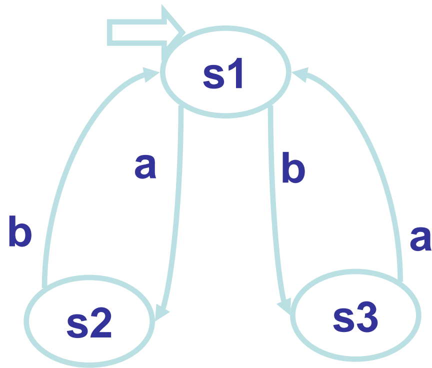
(x)

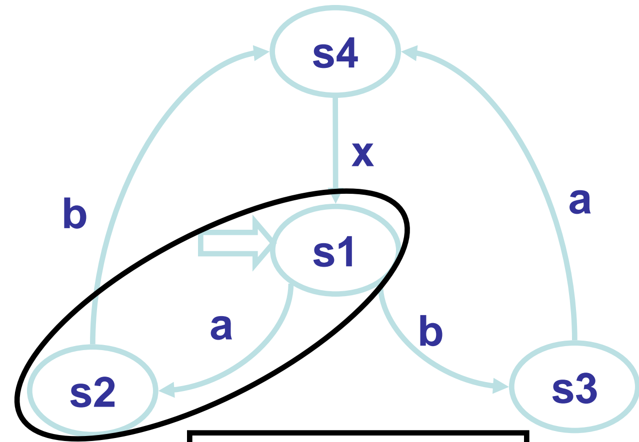


This transformation preserves observational equivalence

Transition Systems and regions

Original TS specification





Region r1:

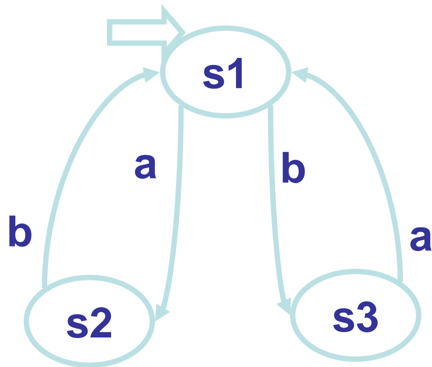
exit(b)

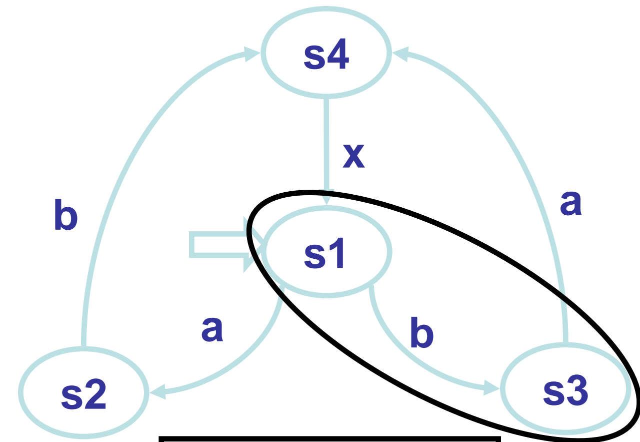
enter(x)

no-cross(a)

Transition Systems and regions

Original TS specification





Region r2:

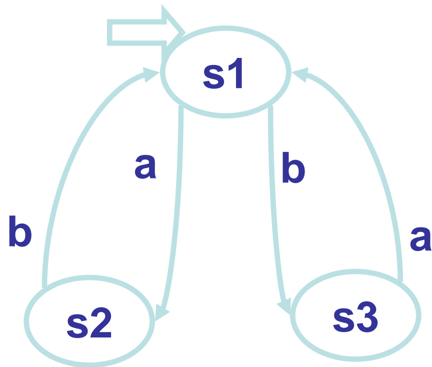
exit(a)

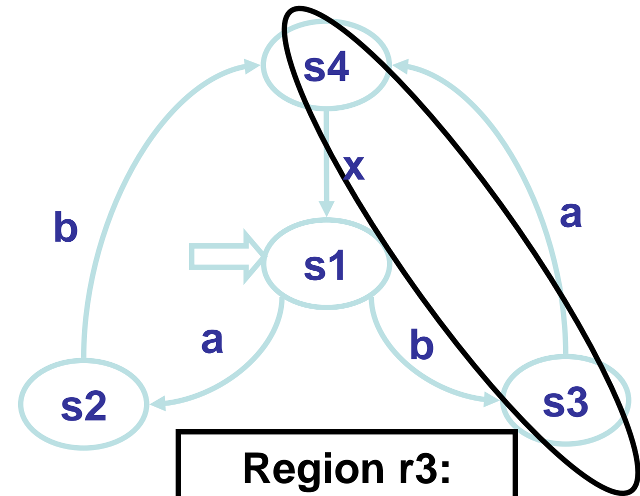
enter(x)

no-cross(b)

Transition Systems and regions

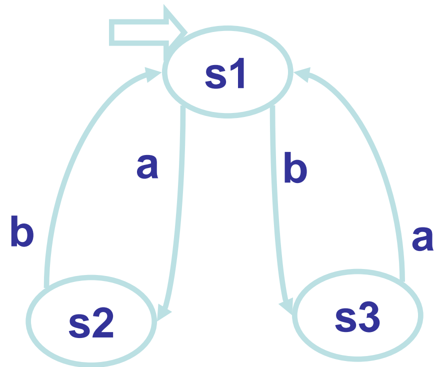
Original TS specification

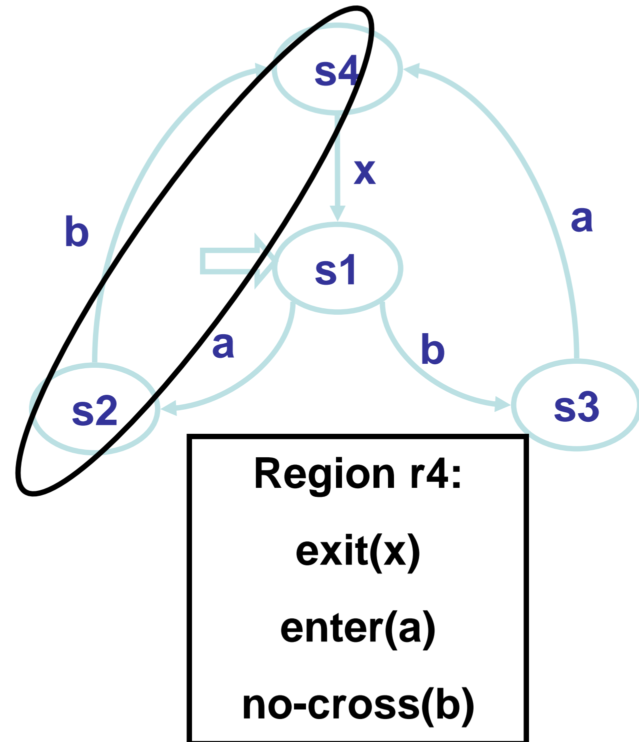




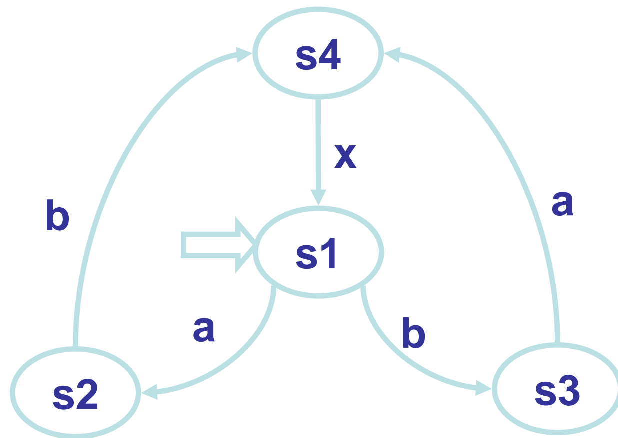
Transition Systems and regions

Original TS specification





From Transition System to LPN

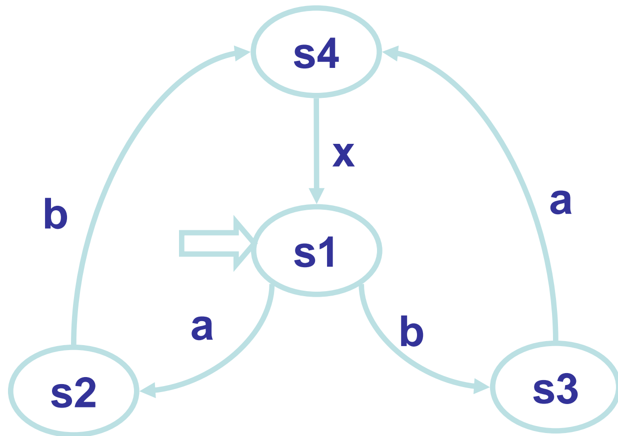


Regions in the TS are associated with places in the LPN

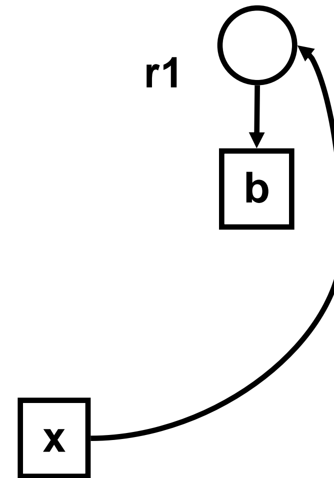
Events are associated with transitions

Exit/entry/no-cross relations are associated with pre/post relations

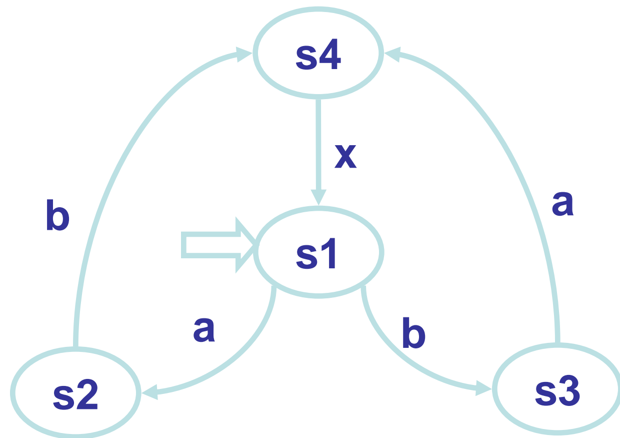
From Transition System to LPN



r1: exit (b), enter(x), no-cross(a)

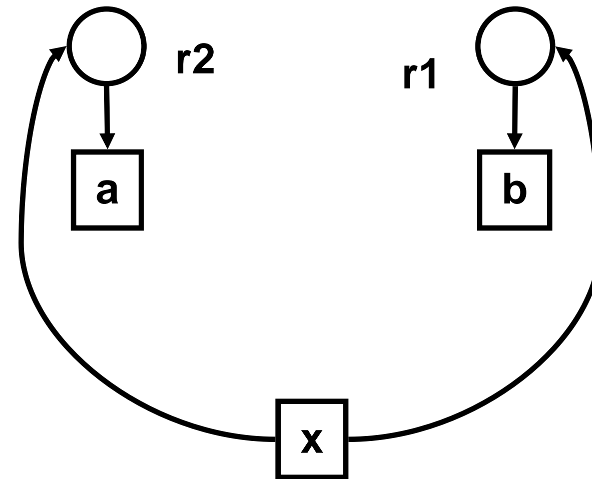


From Transition System to LPN

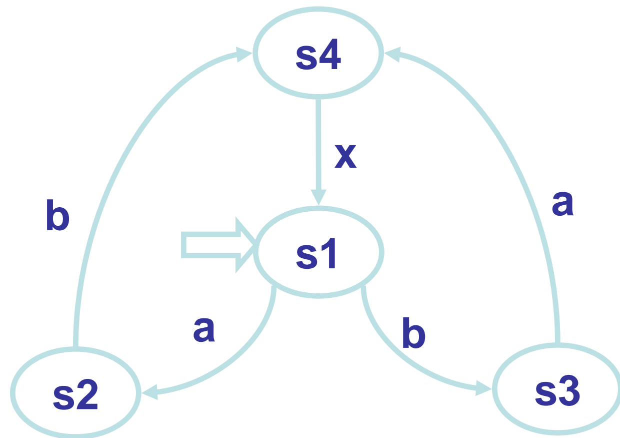


r1: exit (b), enter(x), no-cross(a)

r2: exit (a), enter(x), no-cross(b)



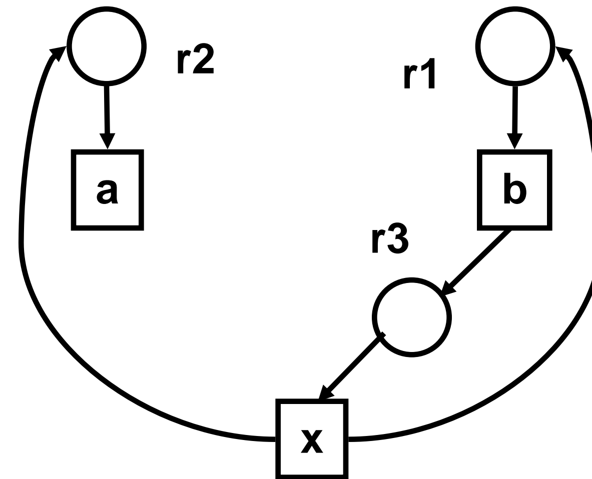
From Transition System to LPN



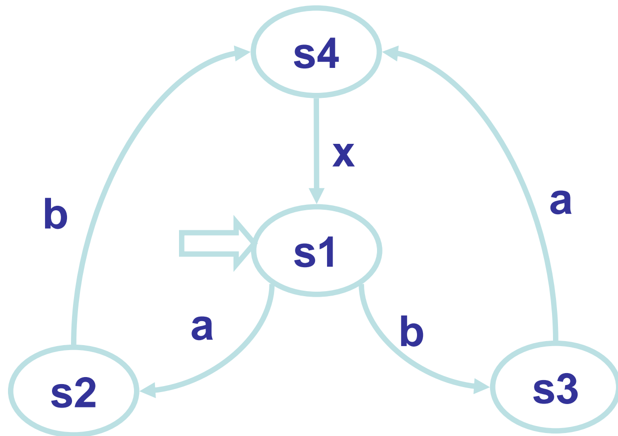
r1: exit (b), enter(x), no-cross(a)

r2: exit (a), enter(x), no-cross(b)

r3: exit (x), enter(b), no-cross(a)



From Transition System to LPN

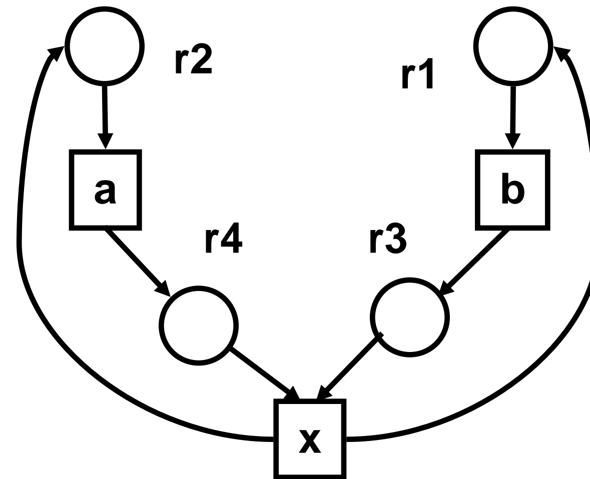


r1: exit (b), enter(x), no-cross(a)

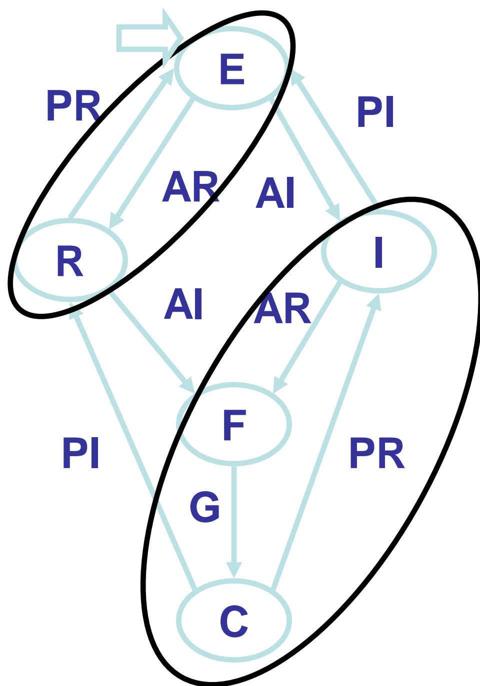
r2: exit (a), enter(x), no-cross(b)

r3: exit (x), enter(b), no-cross(a)

r4: exit (x), enter(a), no-cross(b)



Example: counterflow pipeline



Examples of regions:

r1={E,R} – pre-region(AI), post-region(PI)

r2={I,F,C} – pre-region(PI), post-region(AI), co-region(G)

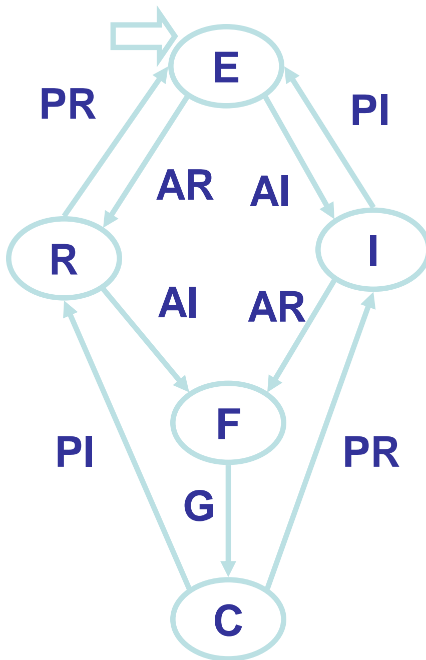
Notation: exit(a) -> pre-region(a), entry(a) -> post-region(a), inside(a) -> co-region(a)

Violation of semi-elementarity:

1. **Intersection of pre-regions (only r2!) for PI {I,F,C} is not equal to Excitation Region for PI {I,C}**
2. **Intersection of pre-regions (empty!) for G is not equal to Excitation Region for G {F}**

Example: counterflow pipeline

Molnar’s 5-state TS:

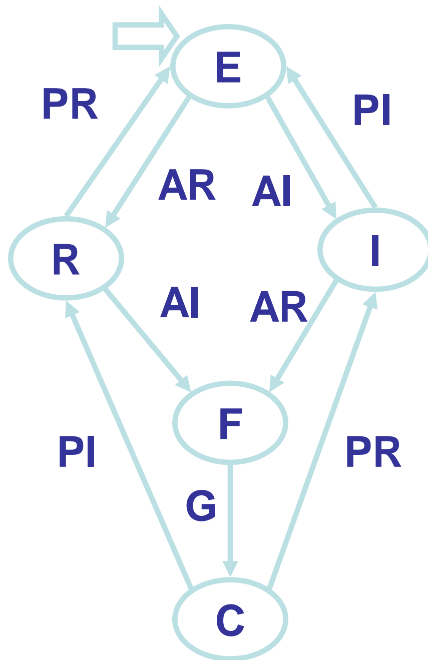


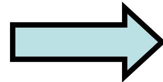
Solution:

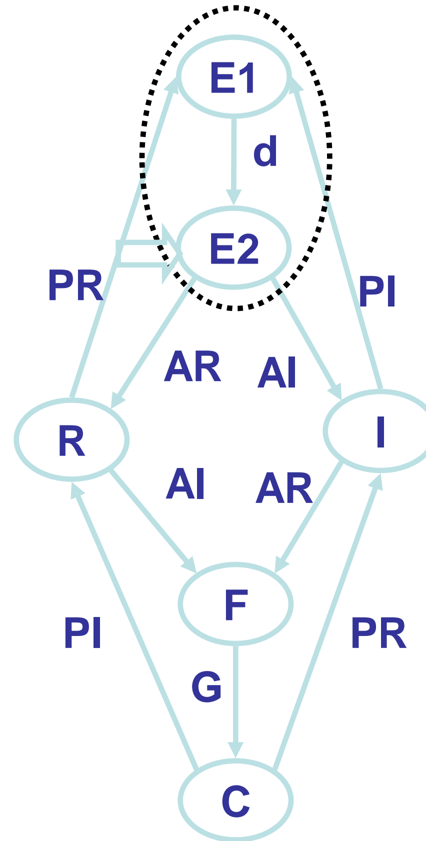
Split a state (E) and insert a silent action (d), preserving behavioural (observational) equivalence

Example: counterflow pipeline

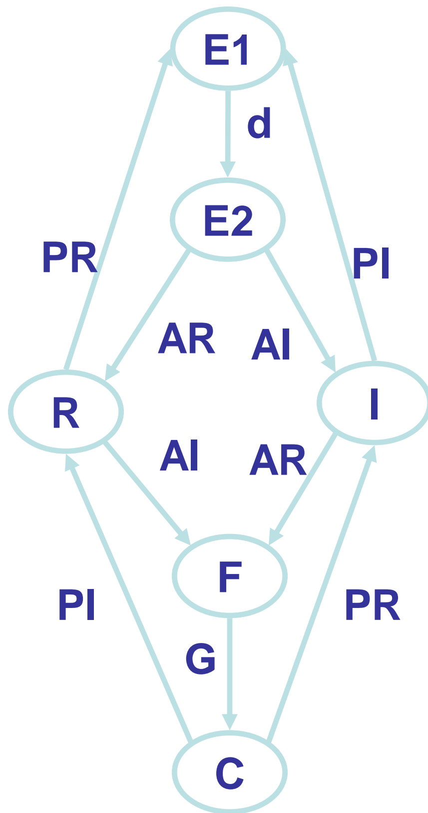
Molnar's 5-state TS:







Example: counterflow pipeline



Semi-elementary TS

Minimal set of regions:

r1 = {E1,E2,I} <- pre(AR), post(PR)

r2 = {E1,E2,R} <- pre(AI), post (PI), co(d)

r3 = {R,F,C} <- pre(PR), post(AR), co(G)

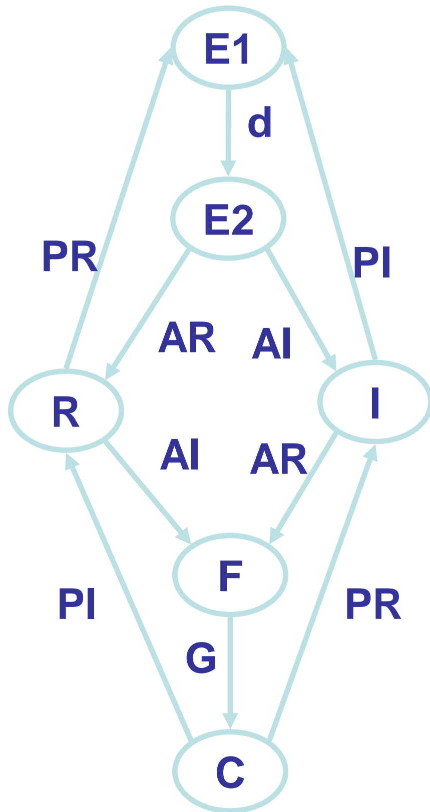
r4 = {I,F,C} <- pre(PI), post(AI)

r5 = {E2,I,C} <- pre(PI,AR), post(G,d)

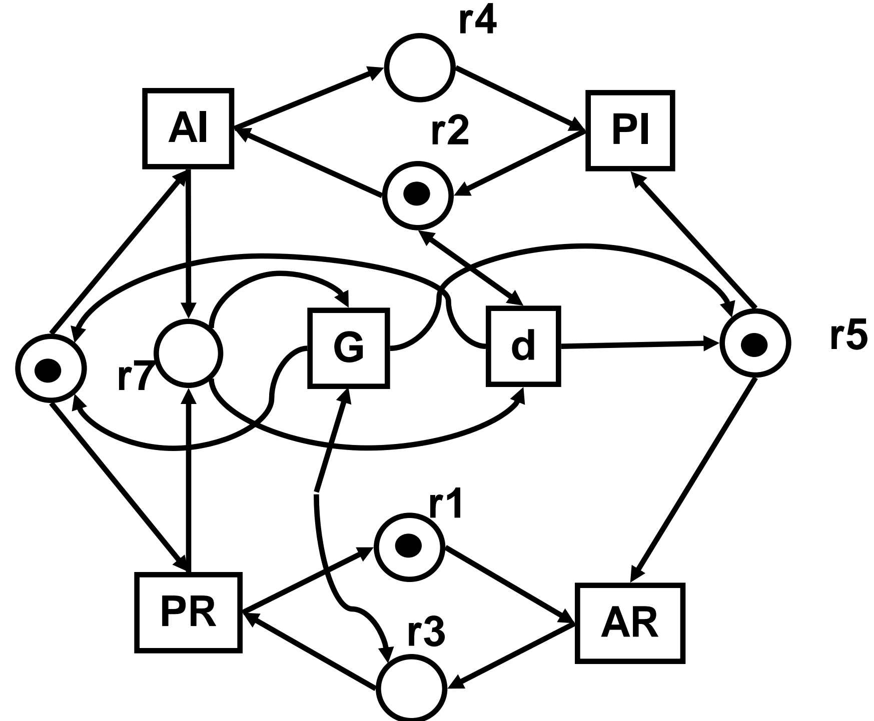
r6 = {E2,R,C} <- pre(PR,AI), post(G,d)

r7 = {E1,I,F} <- pre(G,d), post(PR,AI)

Example: counterflow pipeline



Semi-elementary TS



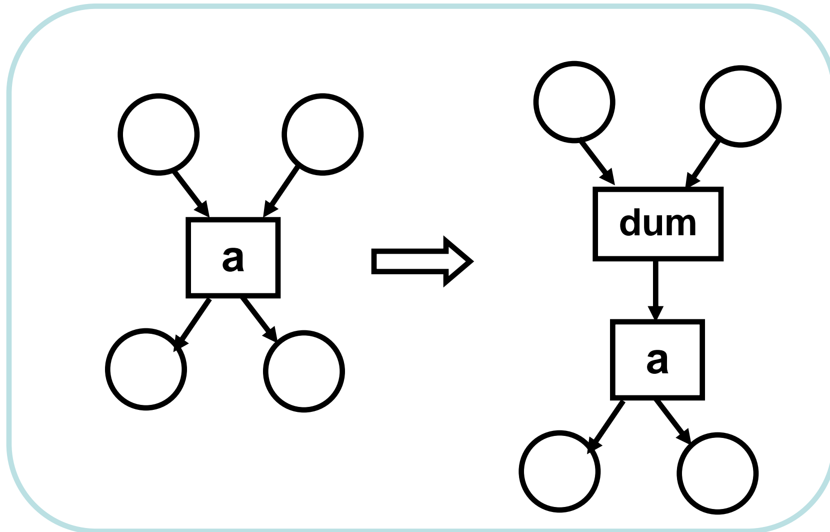
Semi-elementary Petri net

Refinement at the LPN level

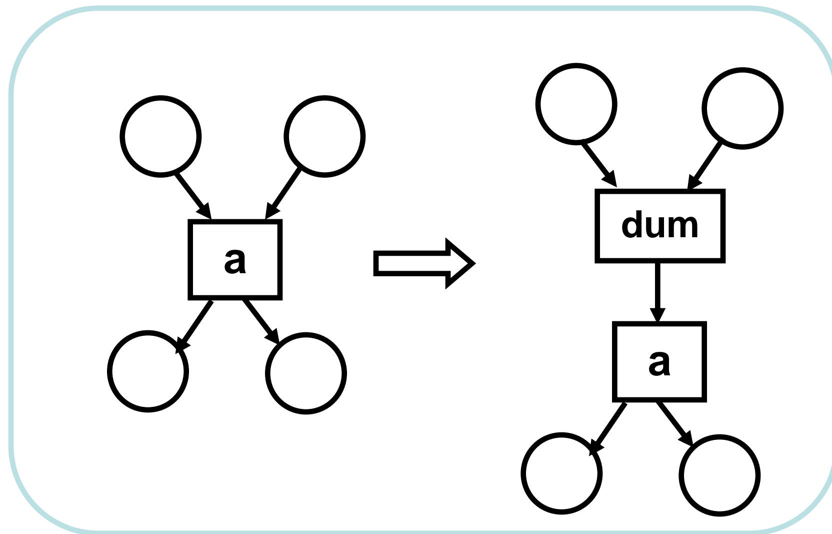
- Examples of refinements:
  - Introduction of “silent” events
  - Handshake refinement
  - Signalling protocol refinement (return-to-zero versus non-return-to-zero)
  - Arbitration refinement (Appendix)

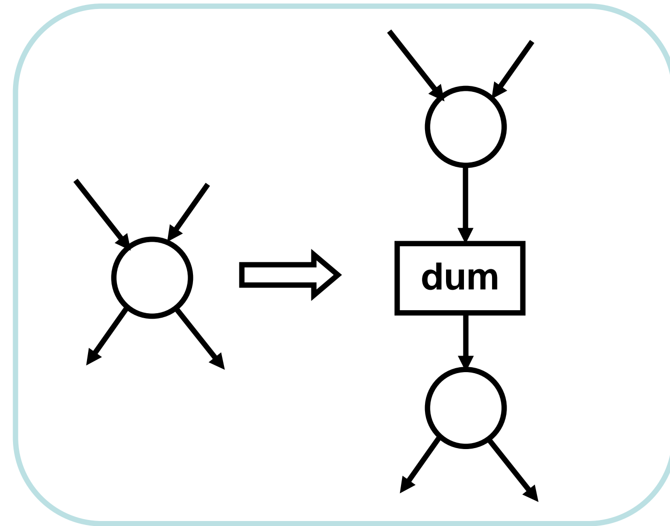
All these refinements must preserve behavioural equivalence (discussed below) and some other properties at the STG level (discussed later)

Structural refinement in LPN

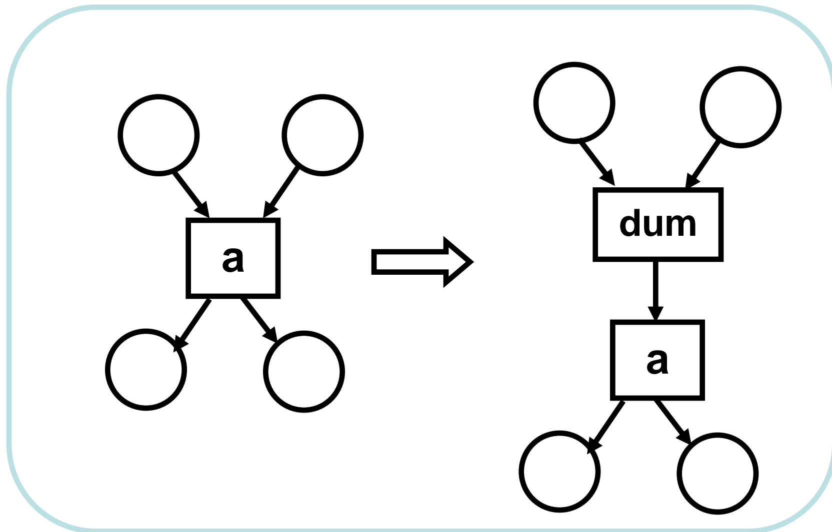


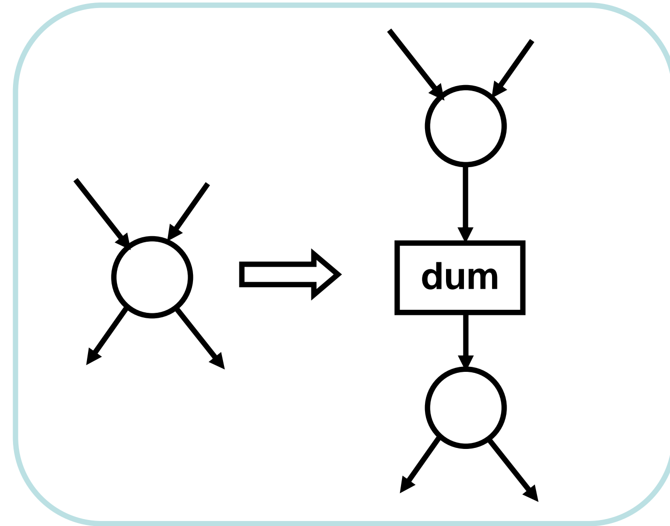
Structural refinement in LPN

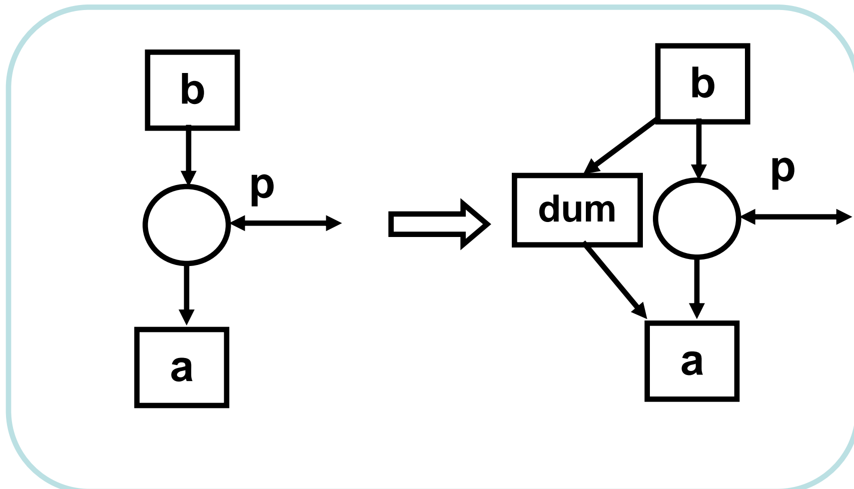




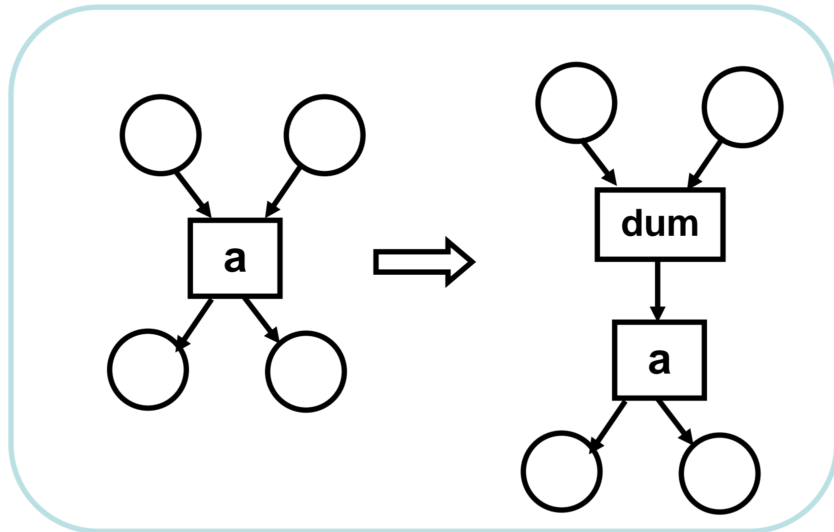
Structural refinement in LPN

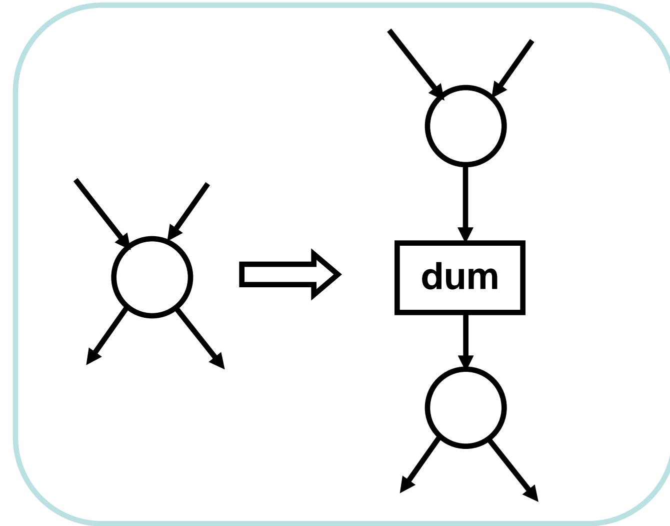


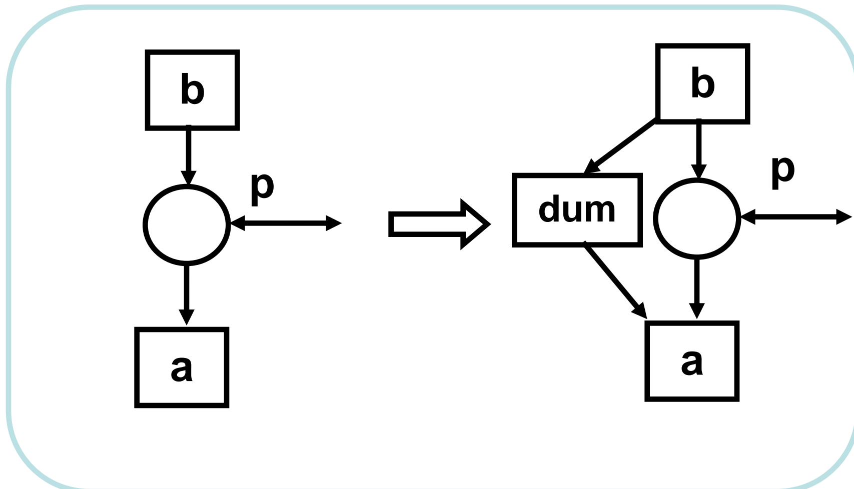


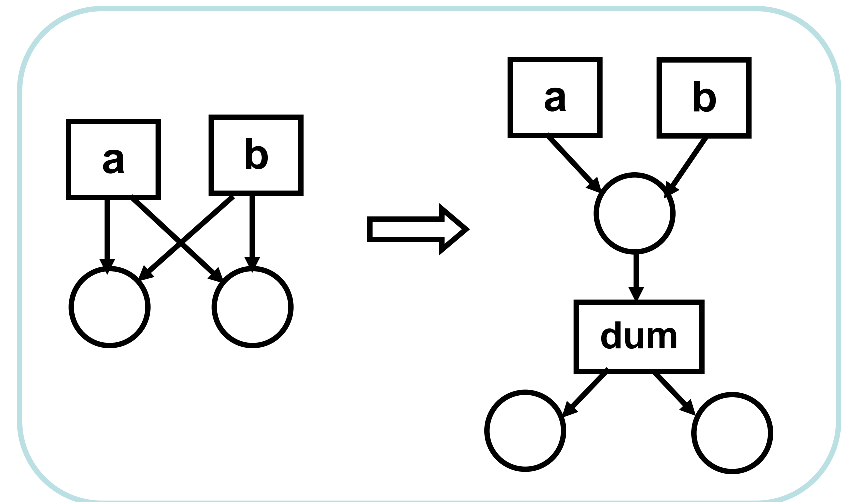


Structural refinement in LPN

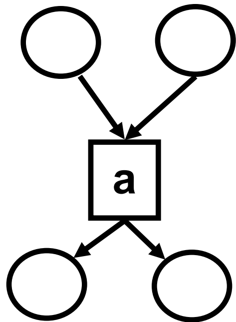






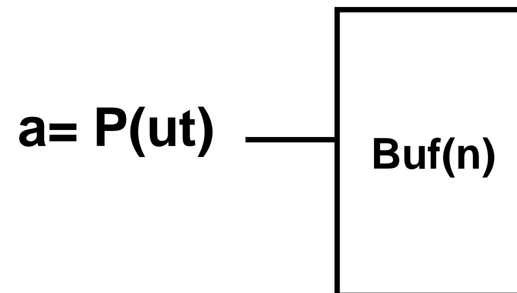


Handshake refinement



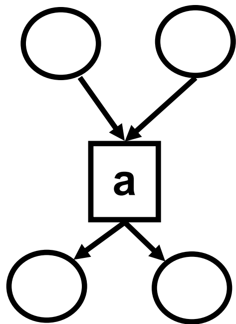
Let abstract event (action) “**a**” be associated with some port of the control circuit

E.g.

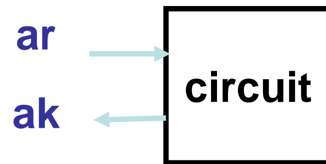


This may lead to the following refinements at the circuit level

Handshake refinement

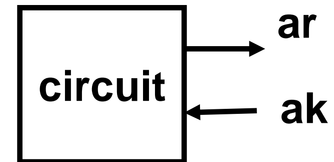


Passive handshake

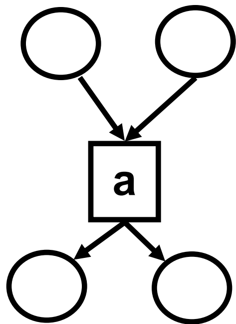


Environment produces (first) request

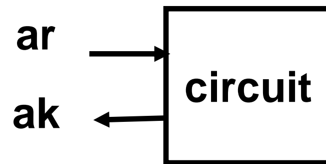
Active handshake



Handshake refinement

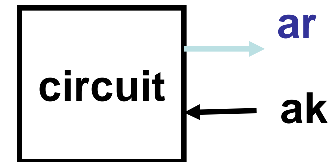


Passive handshake

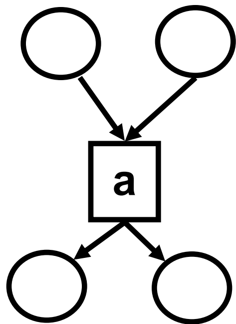


Environment produces (first) request

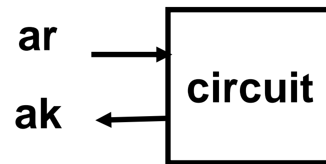
Active handshake



Handshake refinement

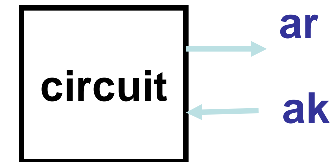


Passive handshake



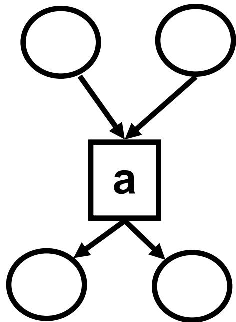
Environment produces (first) request

Active handshake

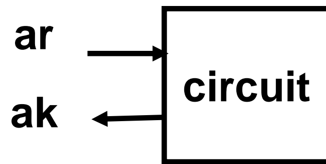


Circuit produces (first) request

Handshake refinement

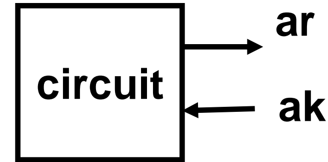


Passive handshake



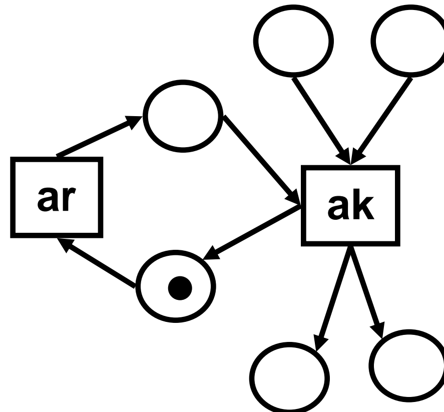
Environment produces (first) request

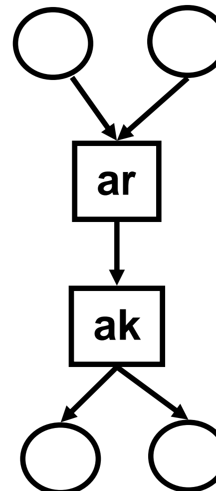
Active handshake



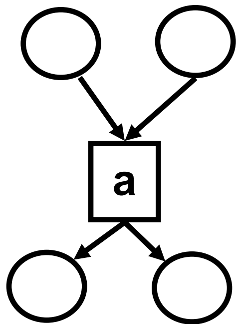
Circuit produces (first) request

Two phase, non-return-to-zero (NRZ) protocol

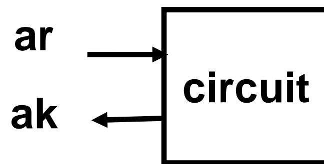




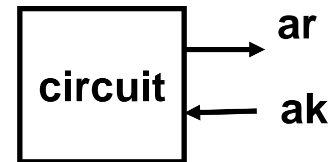
Handshake refinement



Passive handshake



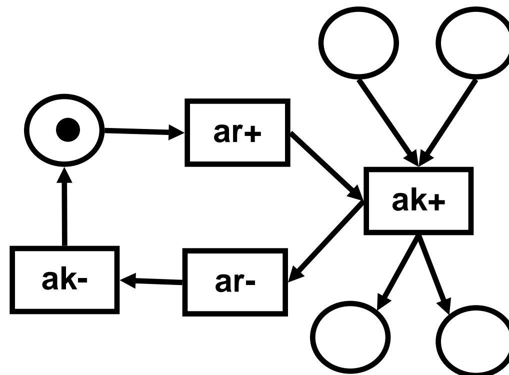
Active handshake

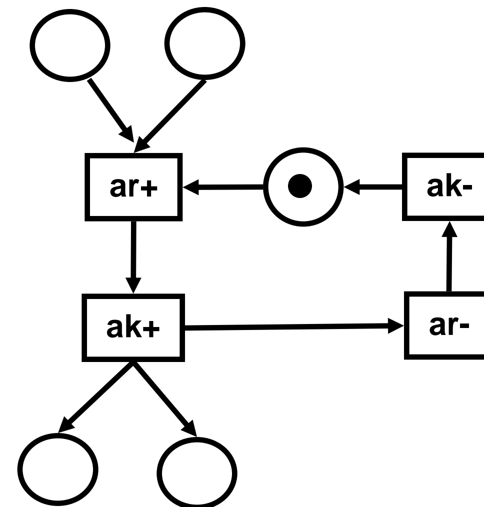


Environment produces (first) request

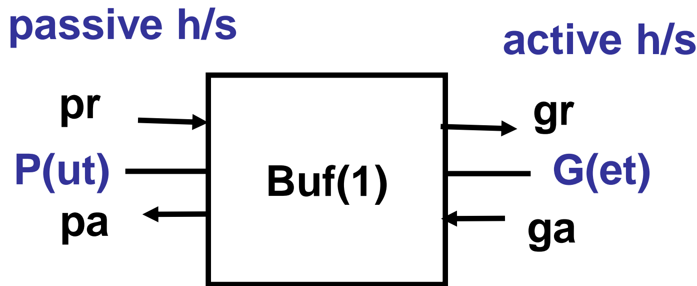
Circuit produces (first) request

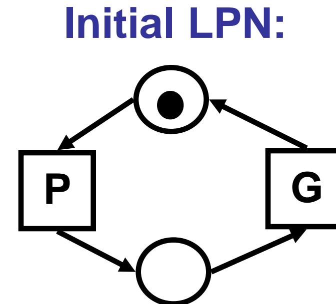
Four phase, return-to-zero (RTZ) protocol



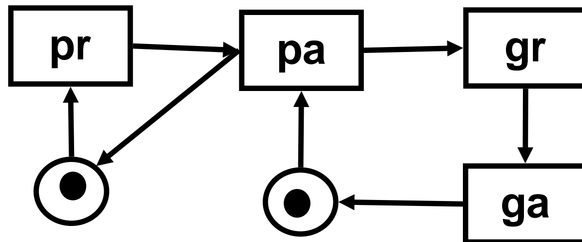


Handshake refinement example

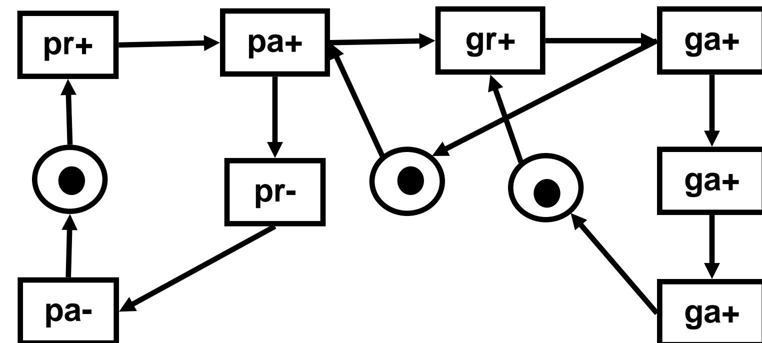




Two phase (NRZ) protocol:



Four phase (RTZ) protocol:

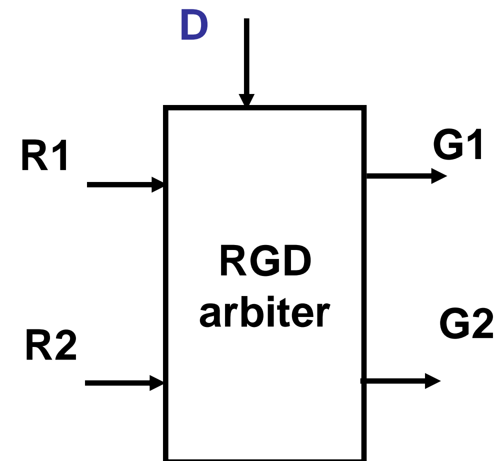


Subsequent transformations are possible at STG level – e.g. re-shuffling of non-critical (resetting) transitions (discussed later)

Arbitration refinement

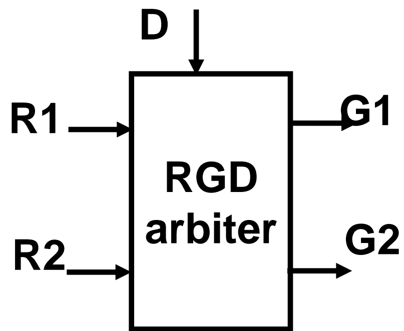
- Asynchronous circuits often require elements to resolve *conflicts* which are intentionally "pre-programmed" in specifications
- These elements are *similar to semaphores* (etc.) in concurrent programs
- These elements are different from logical gates because they involve *internally analogue* components
- The LPN model must be refined to explicitly *"factorise" non-persistent behavior* from the rest of the model – the latter can be synthesized using logic gates

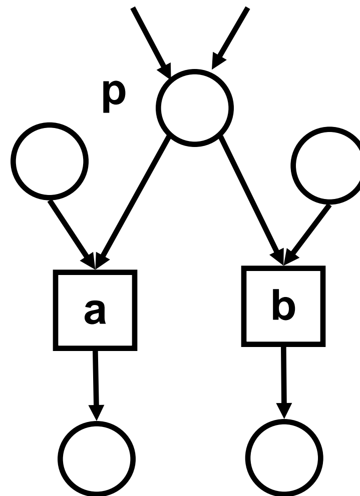
E.g. Request-Grant-Done (RGD) arbiter



Arbitration refinement

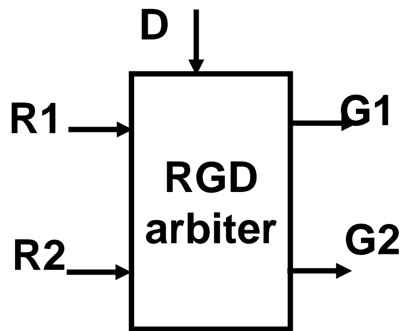
E.g. Request-Grant-Done (RGD) arbiter

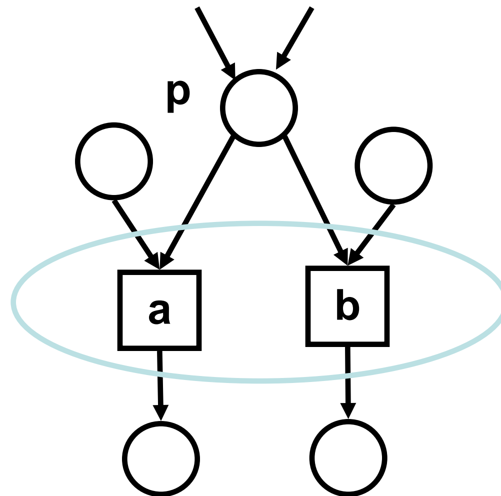




Arbitration refinement

E.g. Request-Grant-Done (RGD) arbiter

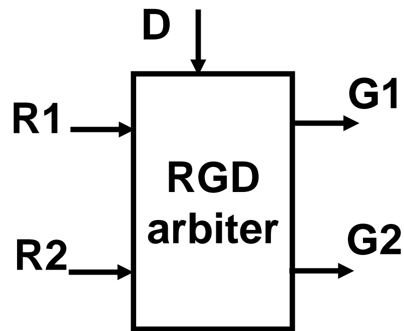


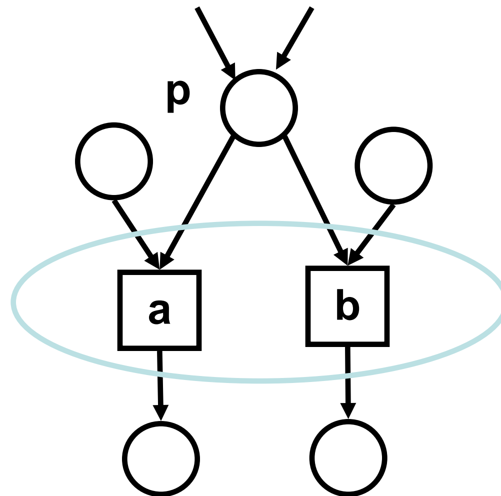


Assume **a** and **b** are circuit actions that are in conflict (may disable each other) and need to be protected

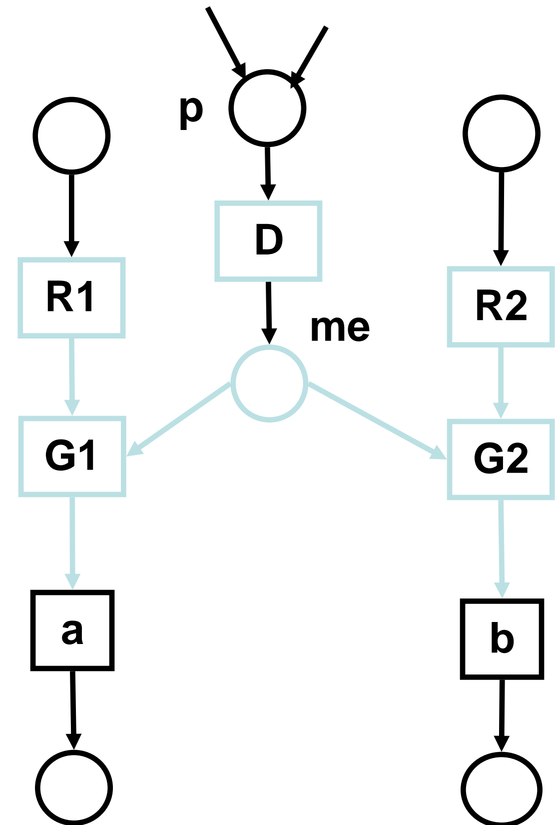
Arbitration refinement

E.g. Request-Grant-Done (RGD) arbiter



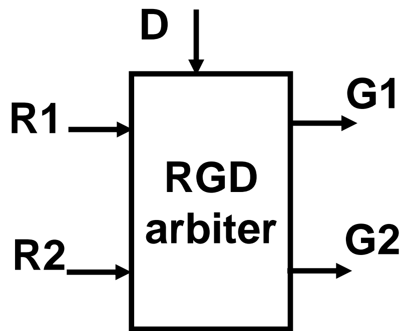


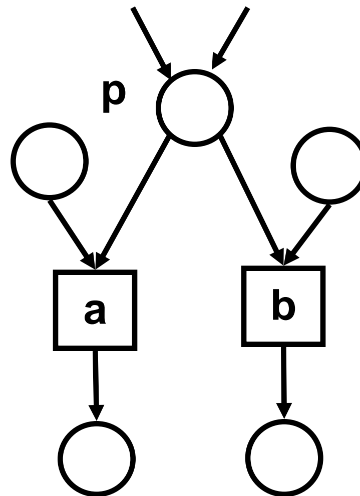
Assume **a** and **b** are circuit actions that are in conflict (may disable each other) and need to be protected

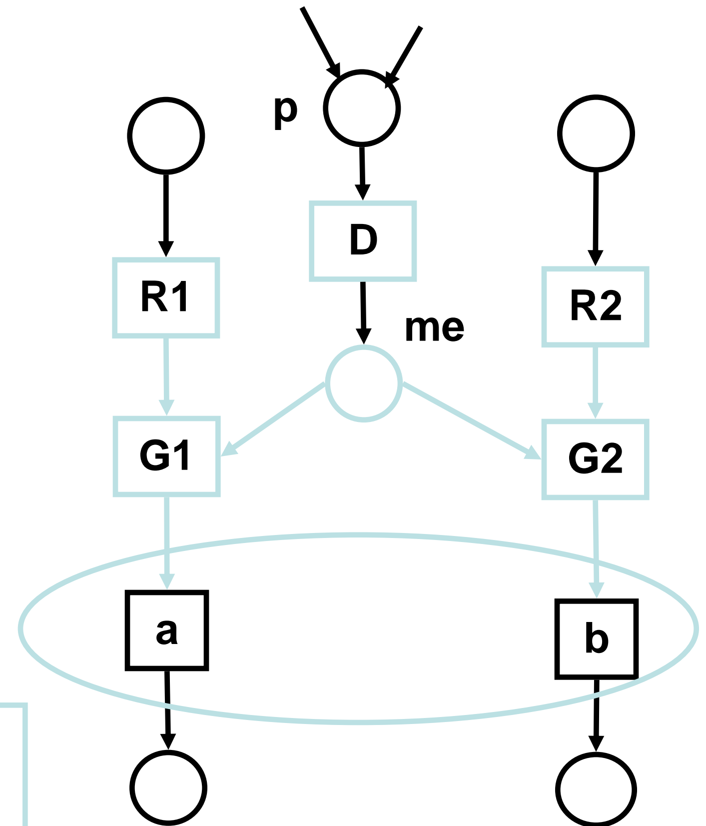


Arbitration refinement

E.g. Request-Grant-Done (RGD) arbiter







a and b are protected now (they are no longer disabled)

Translation of LPNs to circuits

- After appropriate refinements have been made one can translate Labelled Petri nets (or Signal Transition Graphs) into circuits
- Either by syntax-direct translation (discussed below)
- Or by using Logic Synthesis using STGs

Why direct translation?

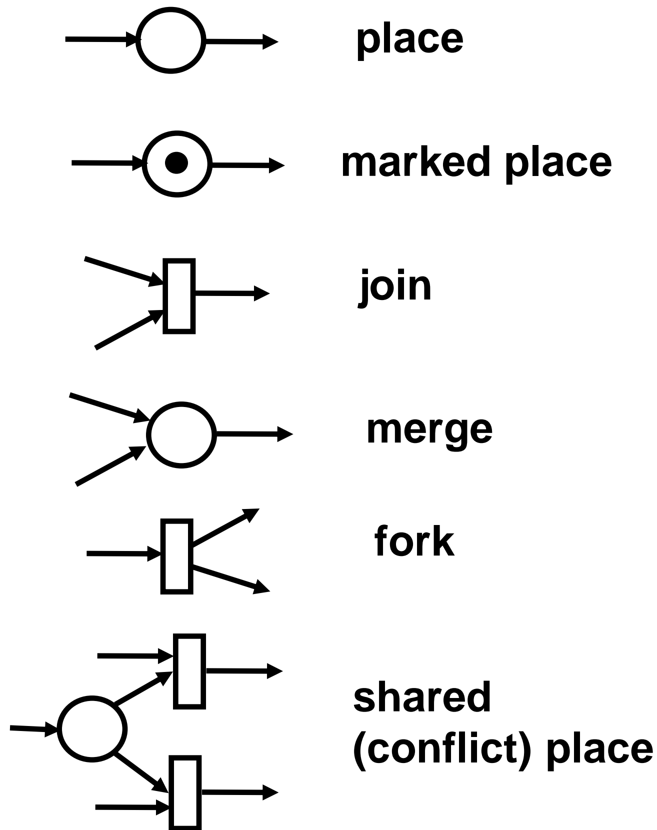
- Direct translation has linear complexity but can be area inefficient (inherent one-hot encoding)
- Logic synthesis has problems with state space explosion, repetitive and regular structures (log-based encoding approach)

Direct Translation of Petri Nets

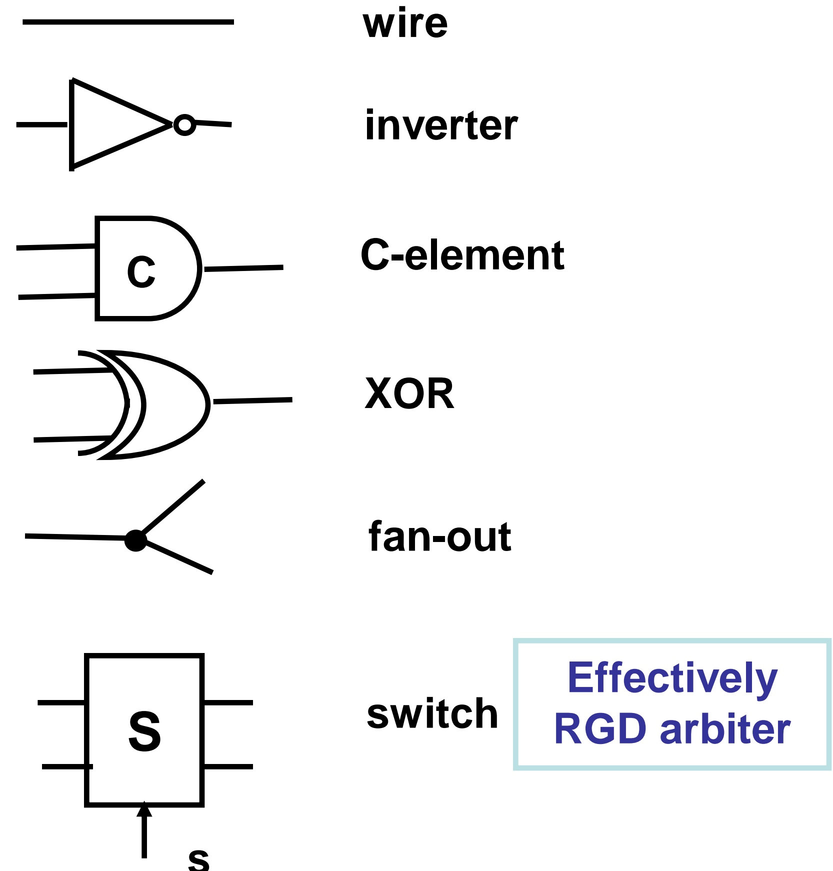
- Previous work dates back to 70s
- Synthesis into *event-based (two-phase)* circuits (similar to Sutherland's micropipeline control)
  - S.Patil, F.Furtek (MIT)
- Synthesis into *level-based (4-phase)* circuits (similar to synthesis from one-hot encoded FSMs)
  - R. David ('69, translation FSM graphs to CUSA cells)
  - L. Hollaar ('82, translation from parallel flowcharts)
  - V. Varshavsky et al. ('90,'96, translation from PN into an interconnection of David Cells)

Patil's set of modules

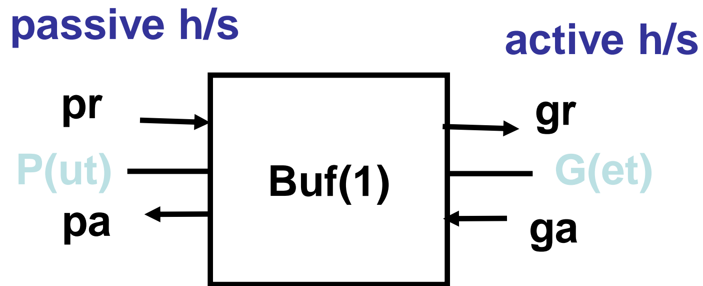
Petri net fragment:



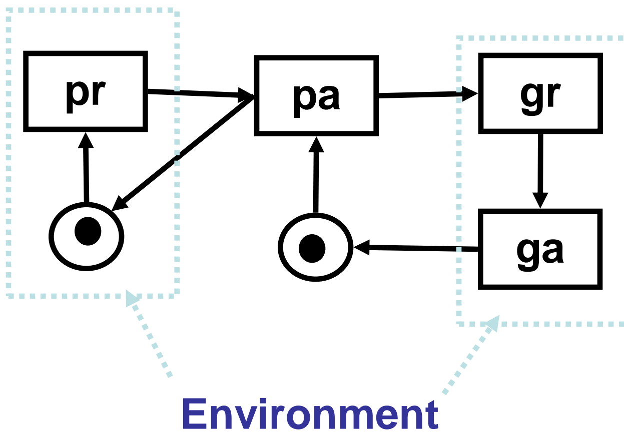
Circuit equivalent:



Example

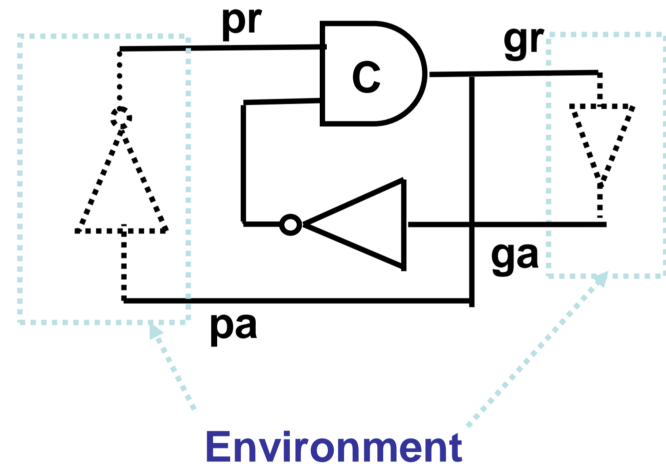


Two phase (NRZ) protocol:



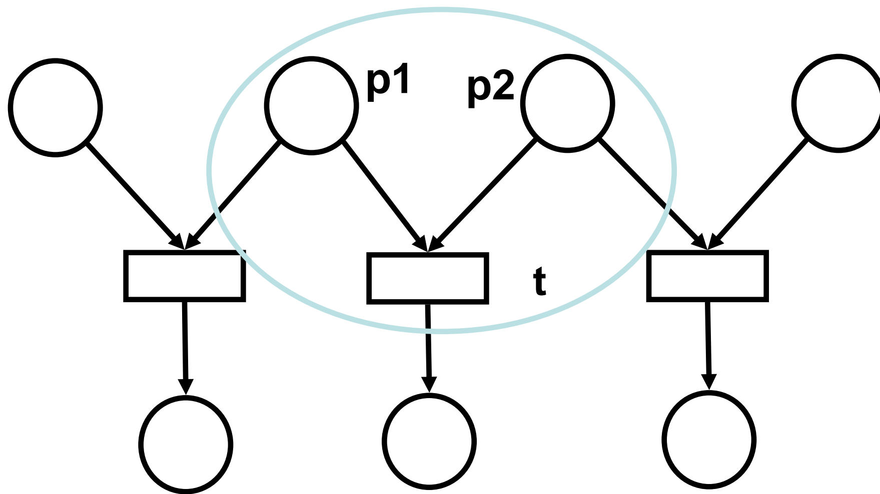
Two-phase implementation

(using Patil's elements):



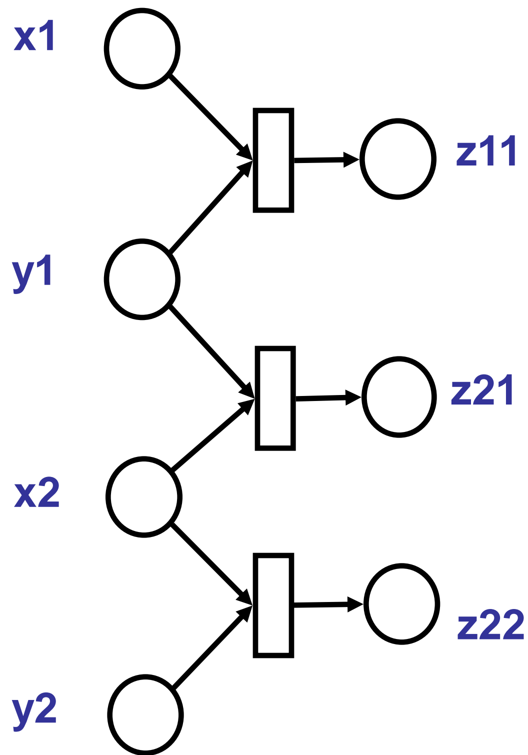
Simple Net restriction

Patil's translation was restricted to (1-safe) Simple Nets



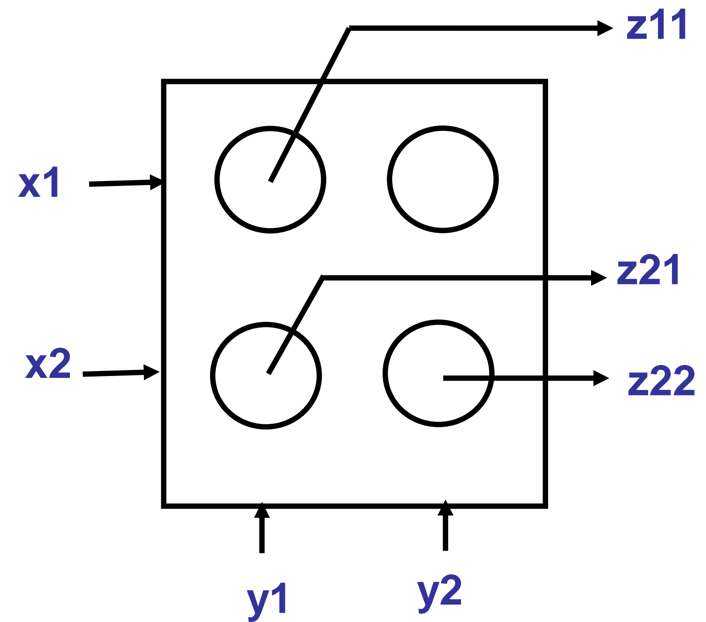
Violation of simplicity: transition t has more than one input place (p1 and p2) that is input to other transitions

Extension to Simple Nets

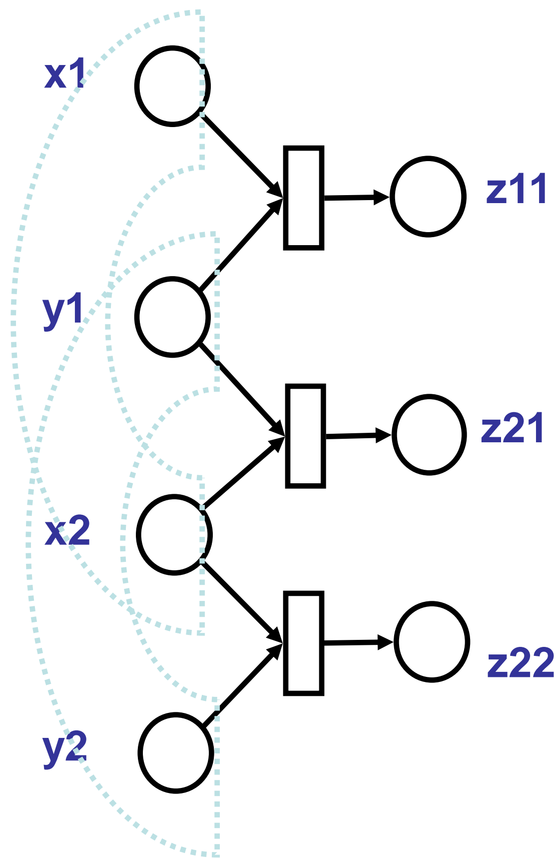


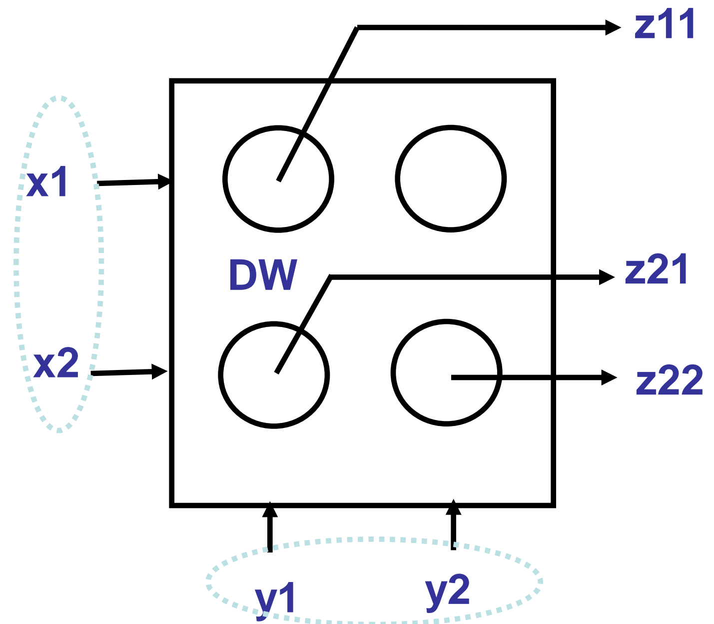
2-by-2 Decision-Wait element

(multi-way Join)



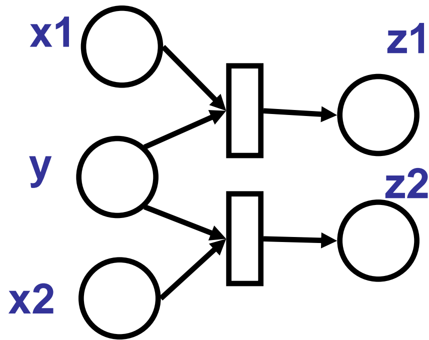
Extension to Simple Nets

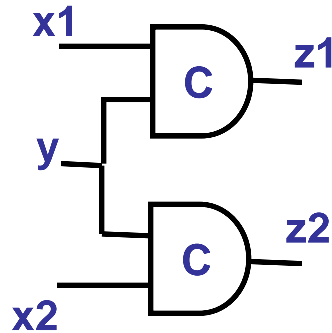




(x1 and x2) and (y1 and y2) must pairs of mutually exclusive events

Problems with C-elements

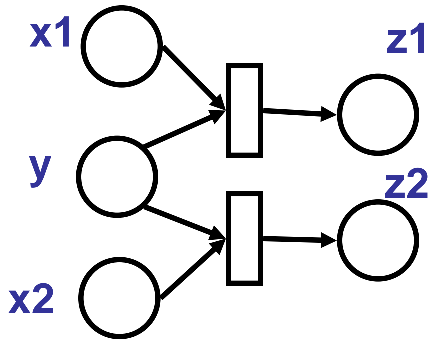


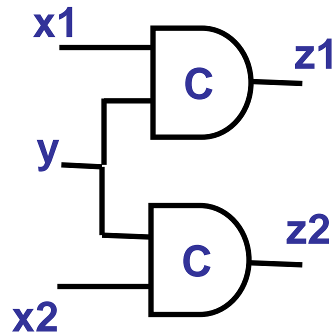


Can we just use a pair of C-elements to implement a 2-by-1 Decision wait?

x1 and x2 are mutually exclusive – so no need for a S-switch (RGD arbiter)

Problems with C-elements





x1 and x2 are mutually exclusive – so no need for a S-switch (RGD arbiter)

Can we just use a pair of C-elements to implement a 2-by-1 Decision wait?

No.

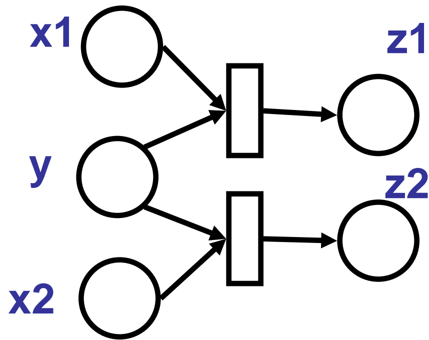
C-elements can only synchronise:

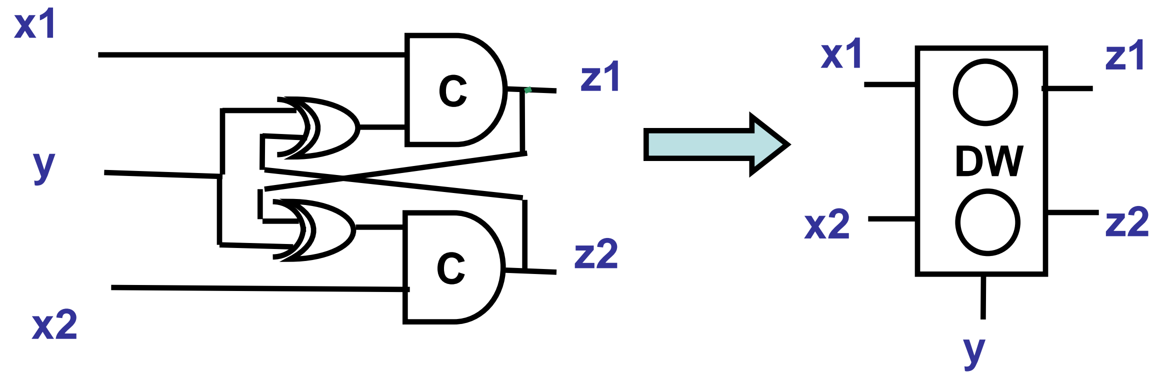
rising (0-1) with rising (0-1) or

falling (1-0) with falling (1-0) but

not rising (0-1) with falling (1-0)

Problems with C-elements

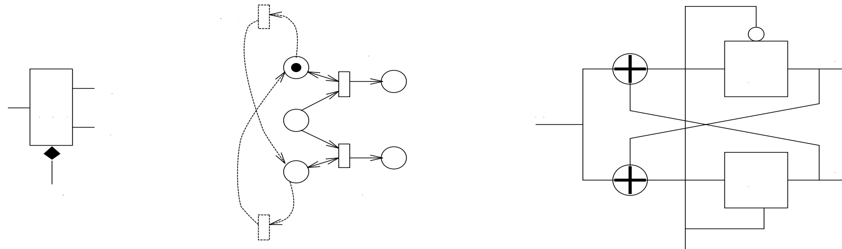




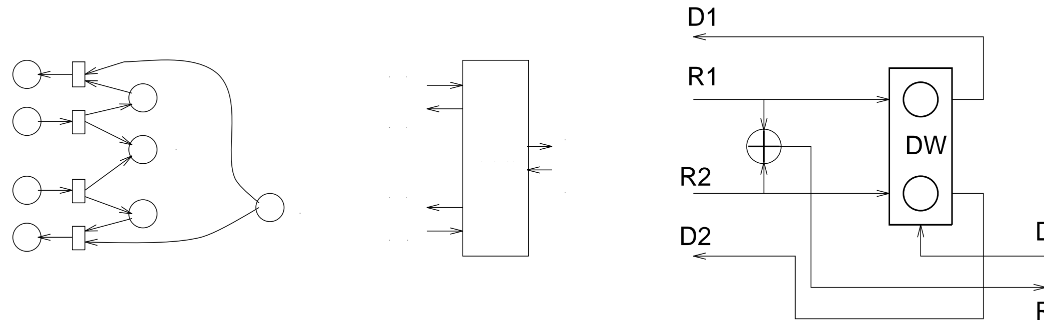
x1 and x2 are mutually exclusive – so no need for a S-switch (RGD arbiter)

Other useful elements

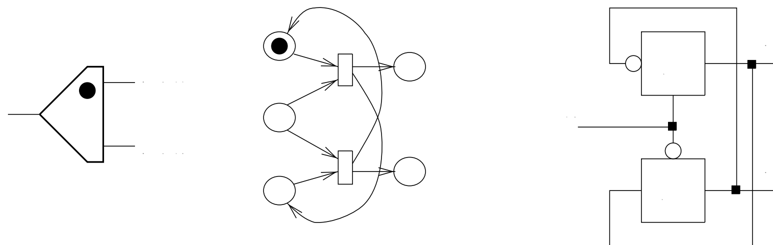
Select:



Call:

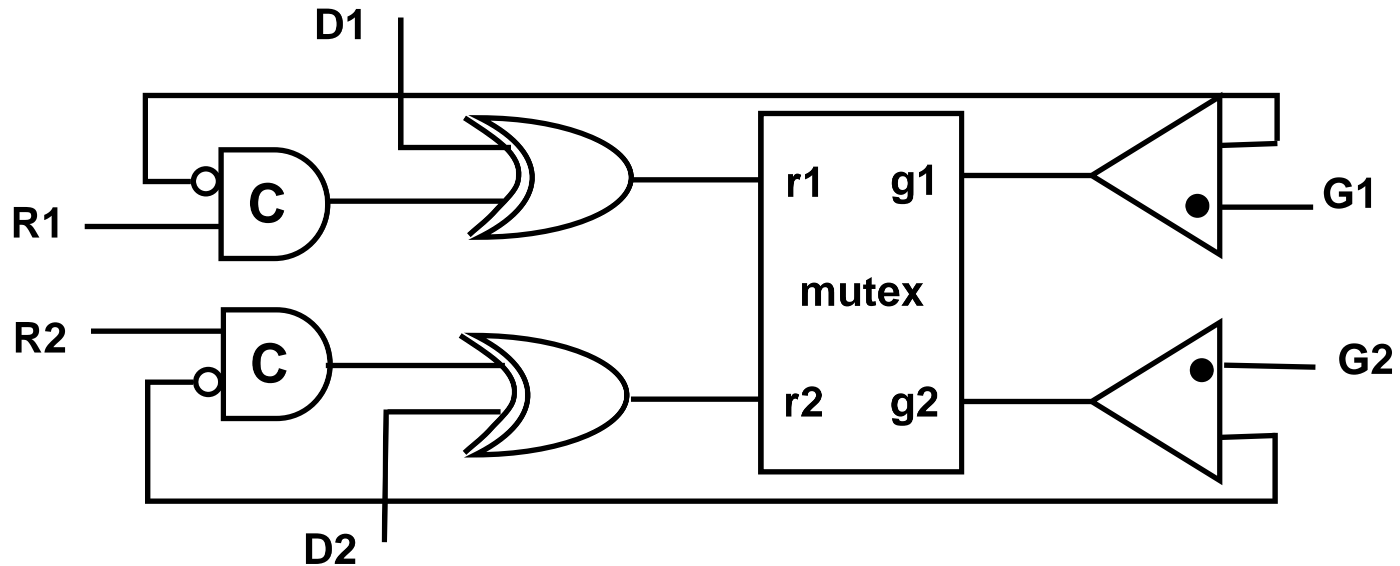


Toggle:

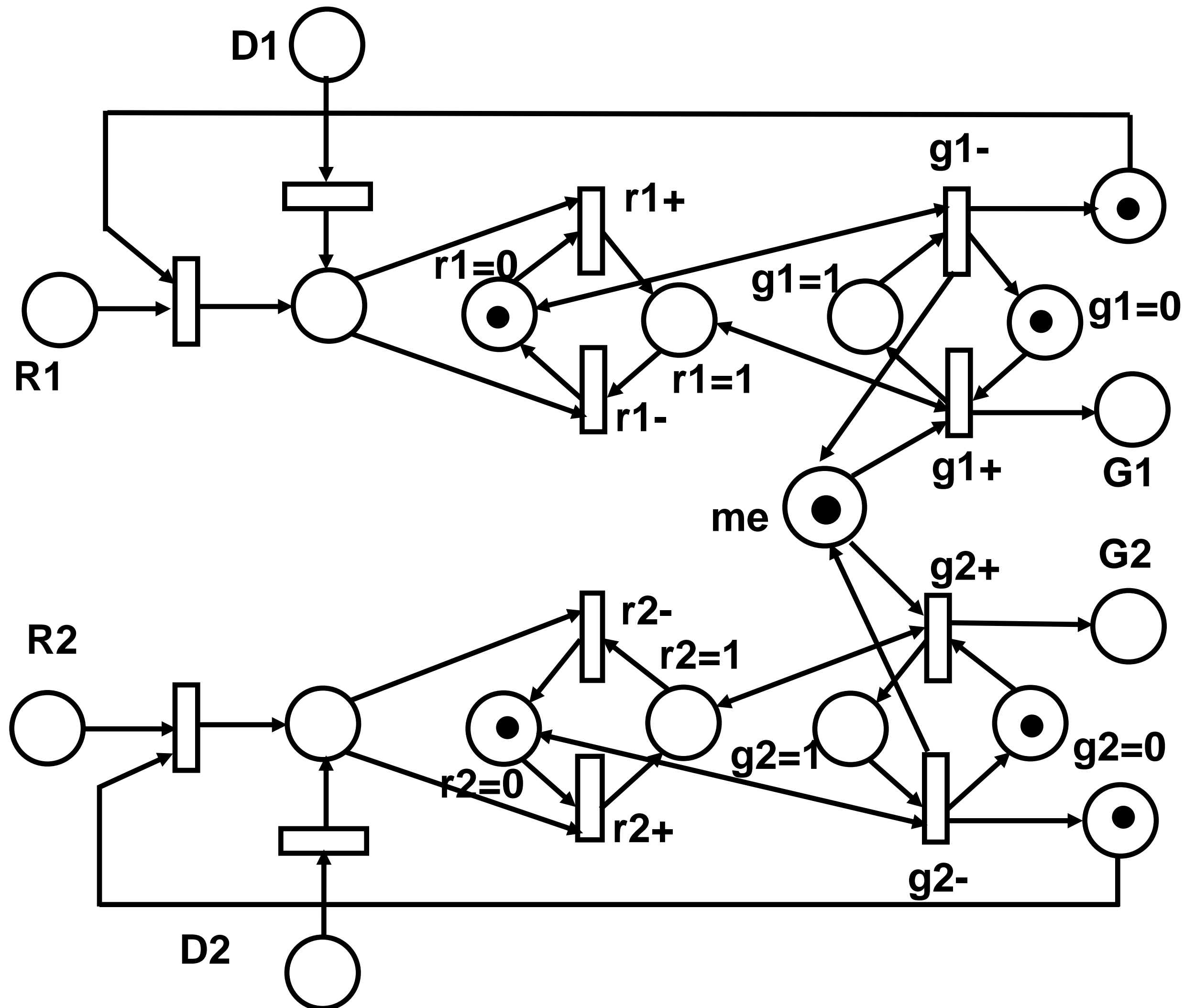


More examples

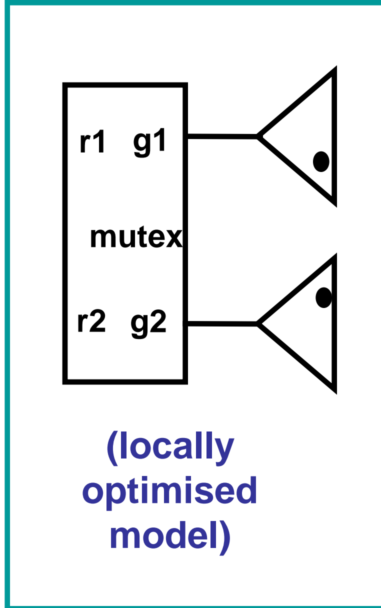
Request-Grant-Done (RGD) arbiter



Request-Grant-Done (RGD) arbiter

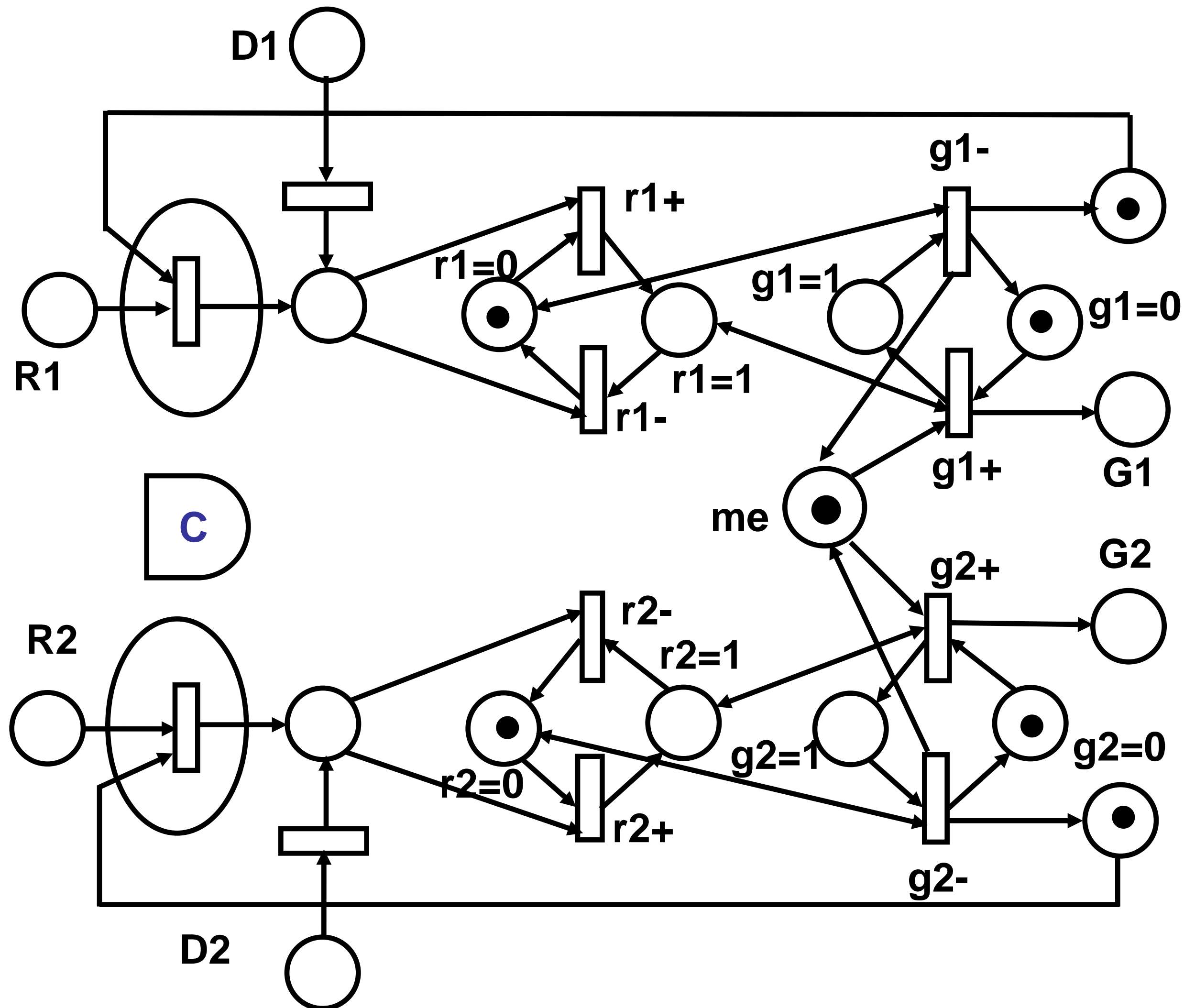


Request-Grant-Done (RGD) arbiter

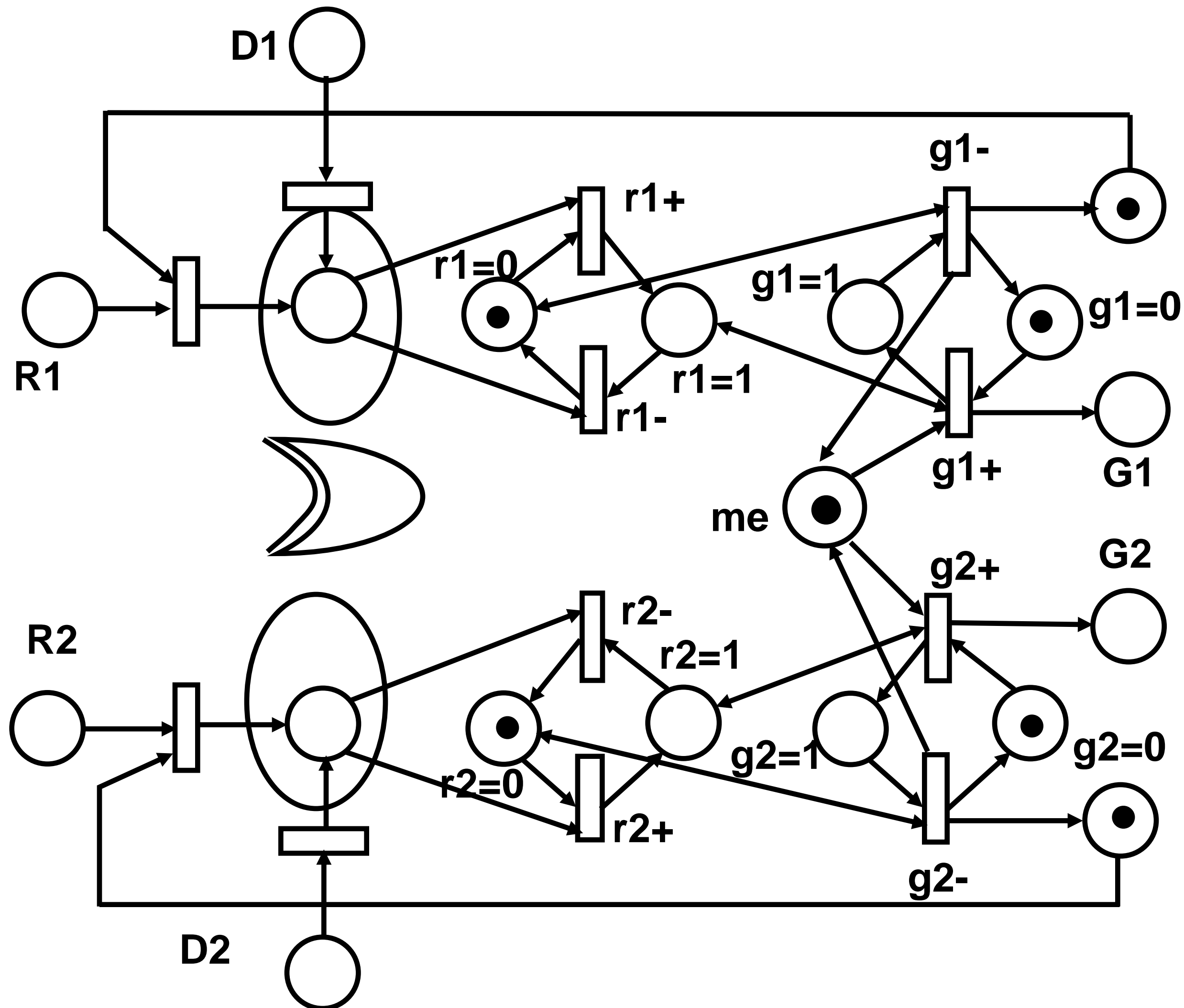


(locally optimised model)

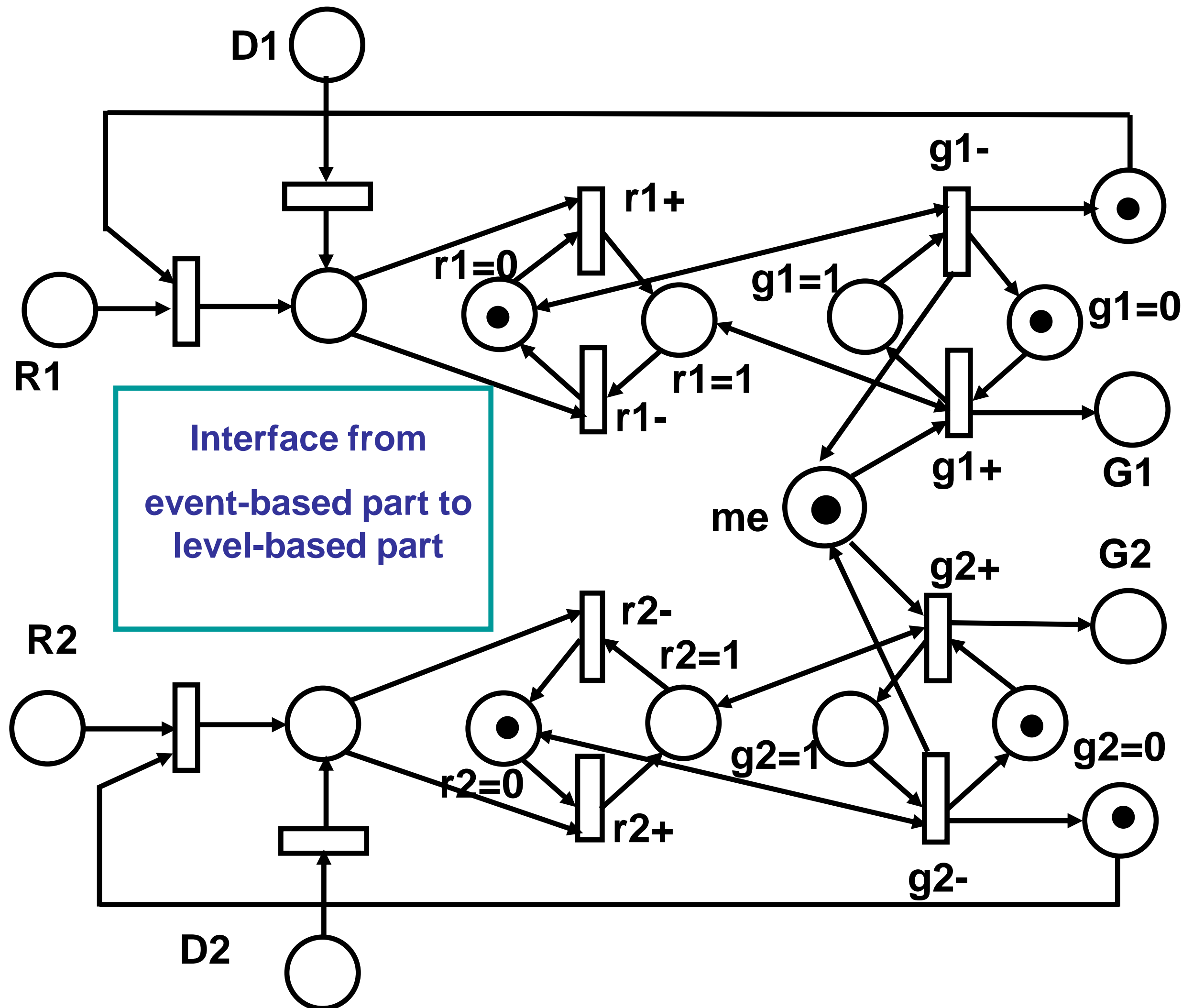
Request-Grant-Done (RGD) arbiter



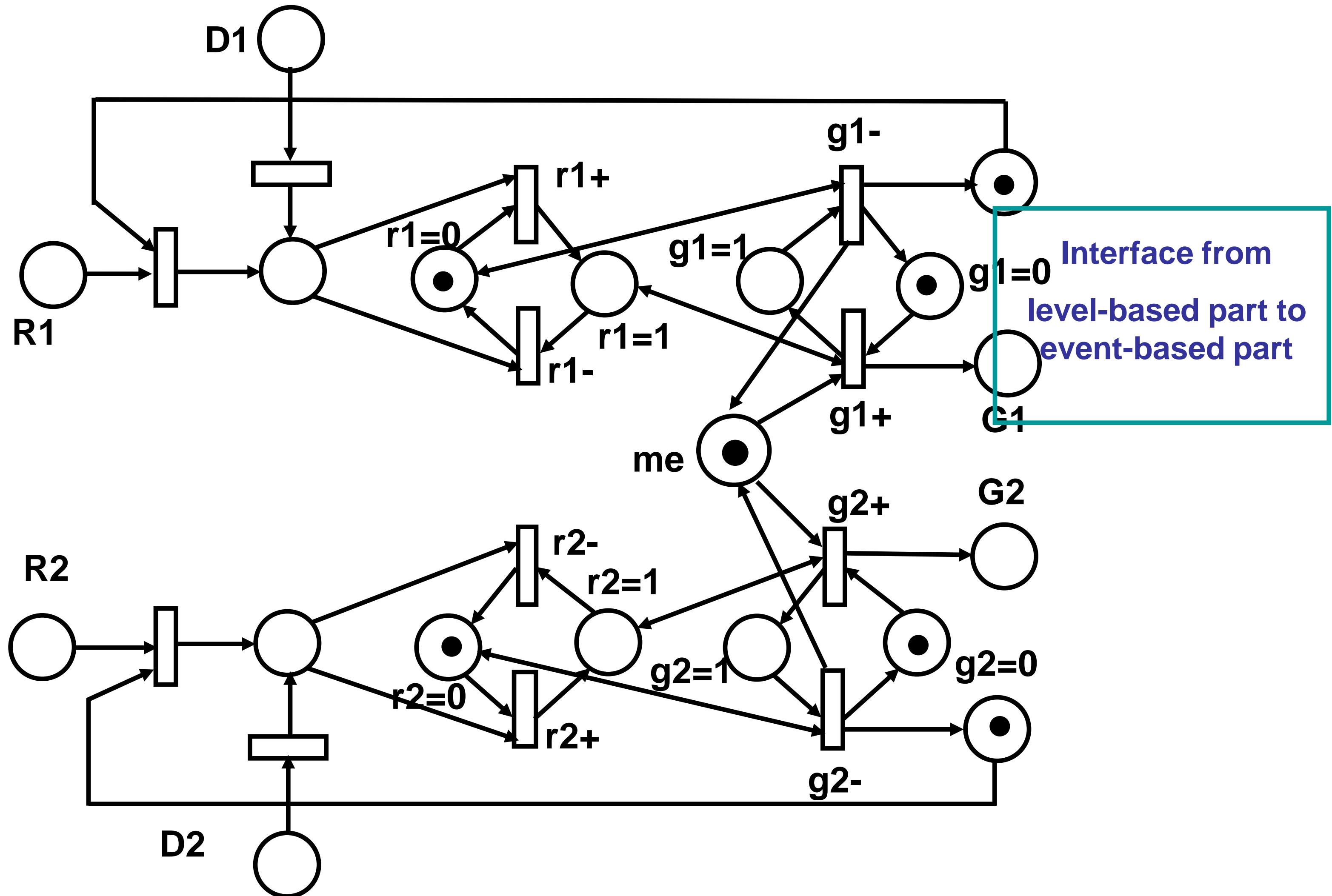
Request-Grant-Done (RGD) arbiter



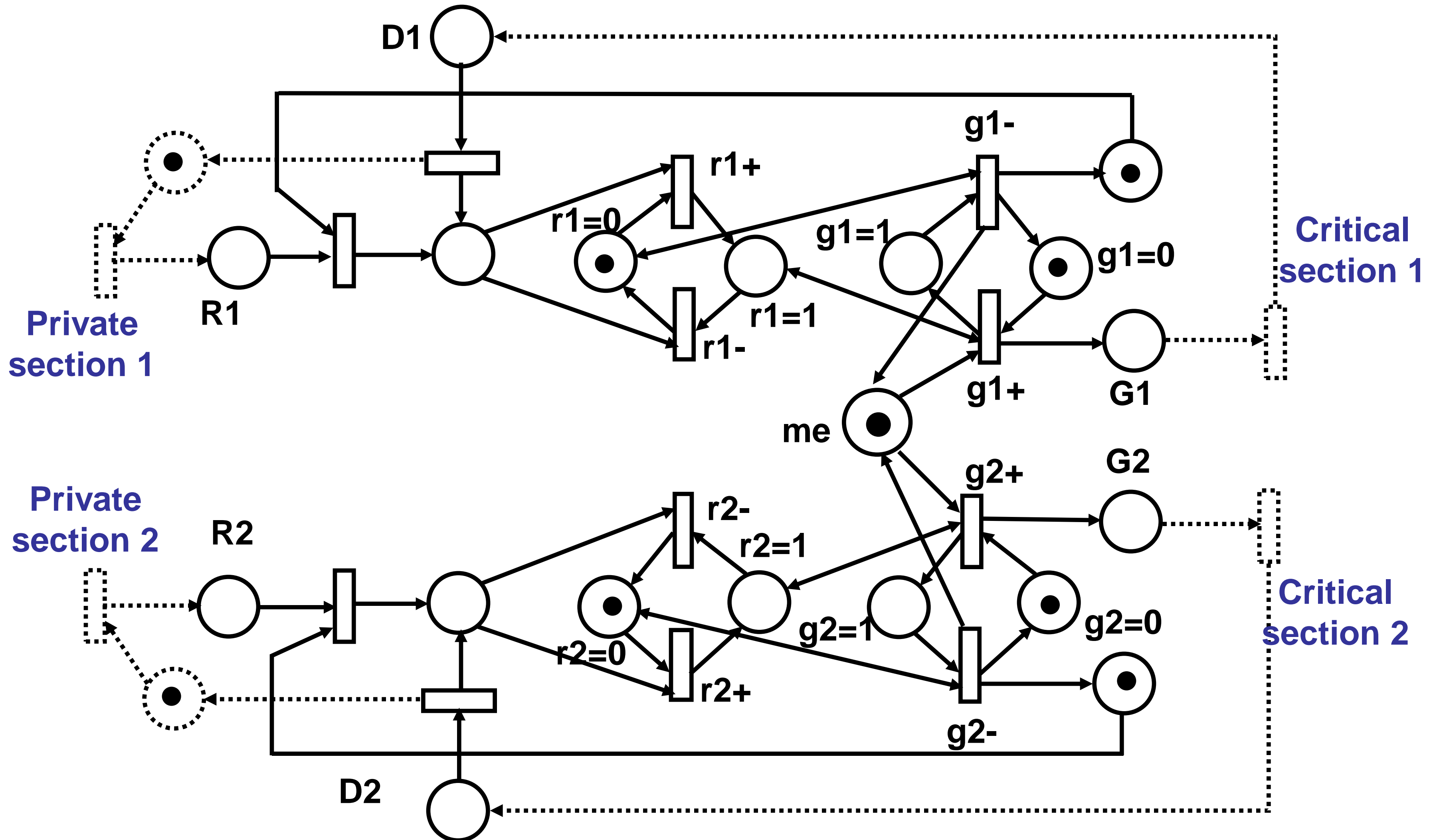
Request-Grant-Done (RGD) arbiter



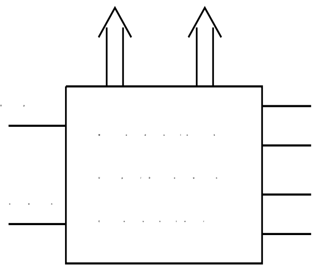
Request-Grant-Done (RGD) arbiter

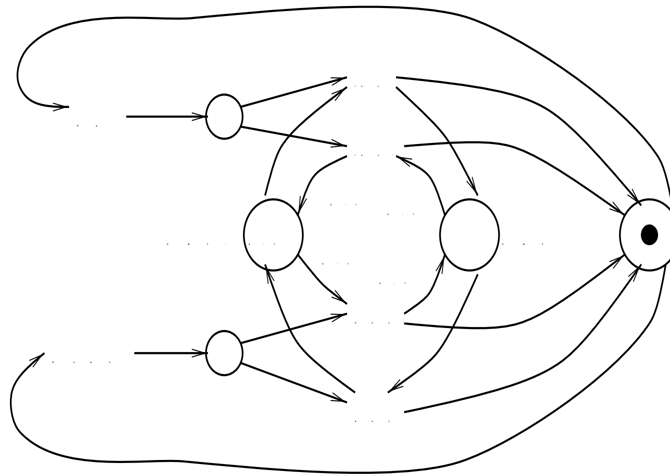


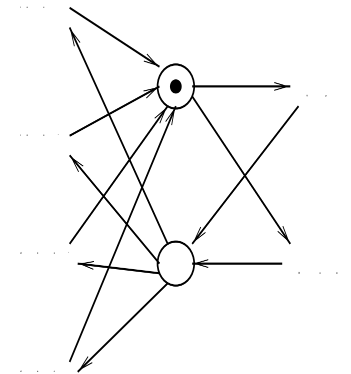
Request-Grant-Done (RGD) arbiter with environment



Direct synthesis example (modulo-k Up-Down counter)



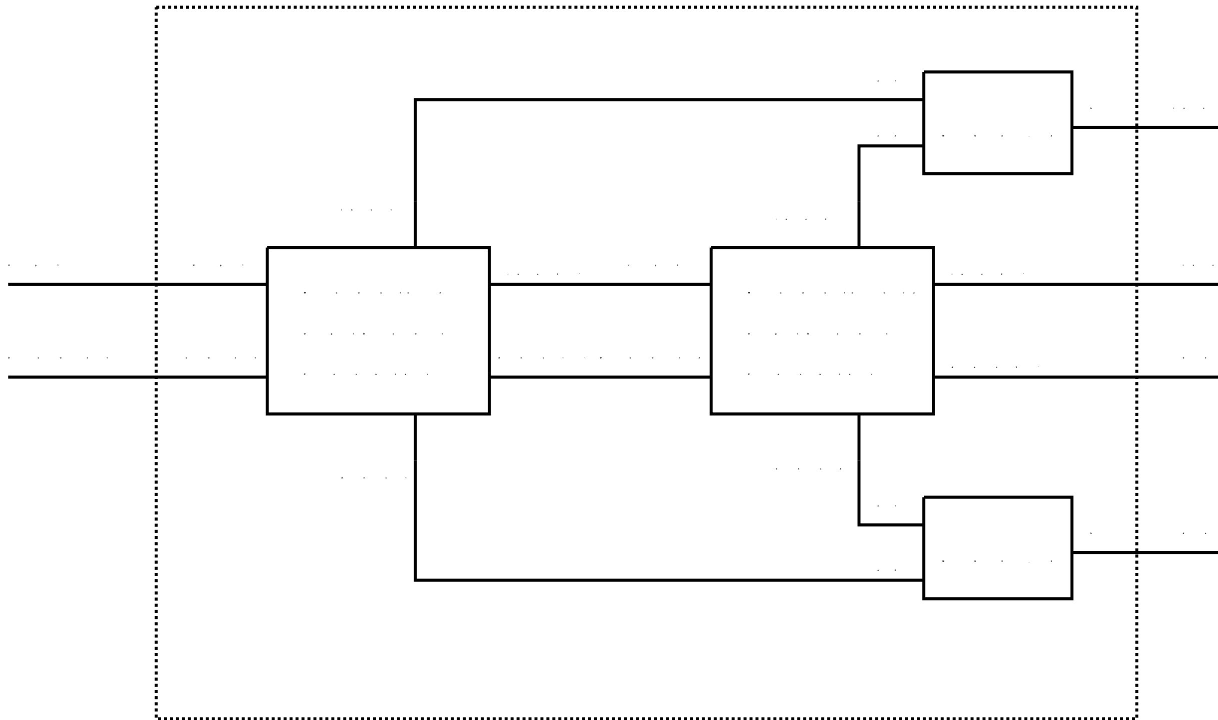




Mod-k counter LPN

Environment LPN

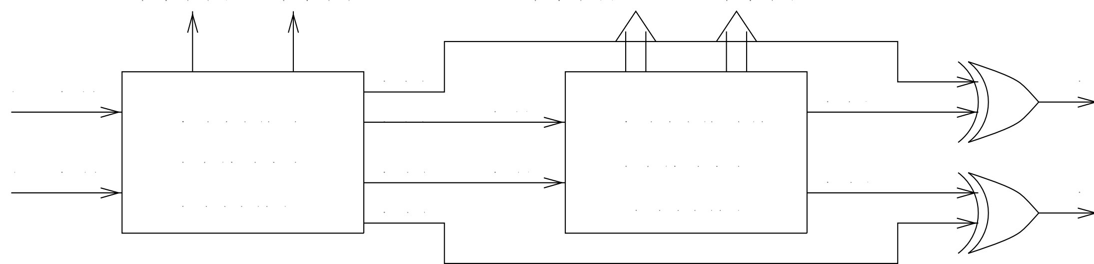
Direct synthesis example (modulo-k Up-Down counter)



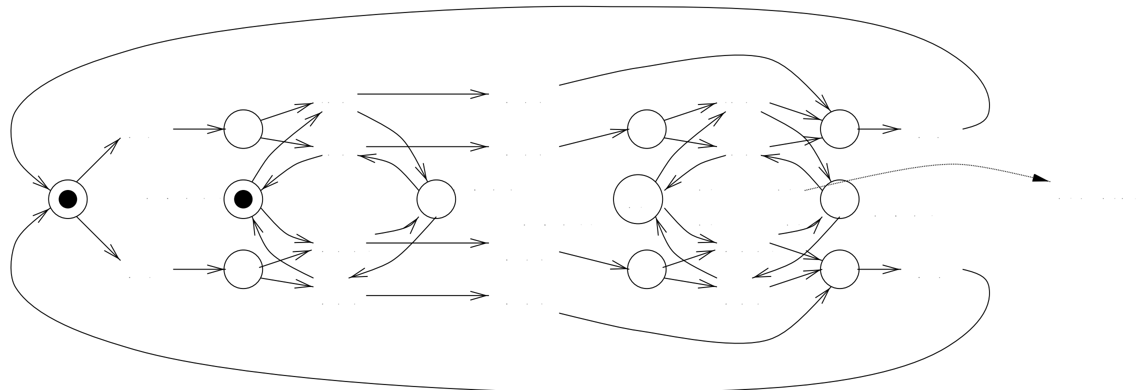
Decomposition (structural view)

Direct synthesis example (modulo-k Up-Down counter)

structure

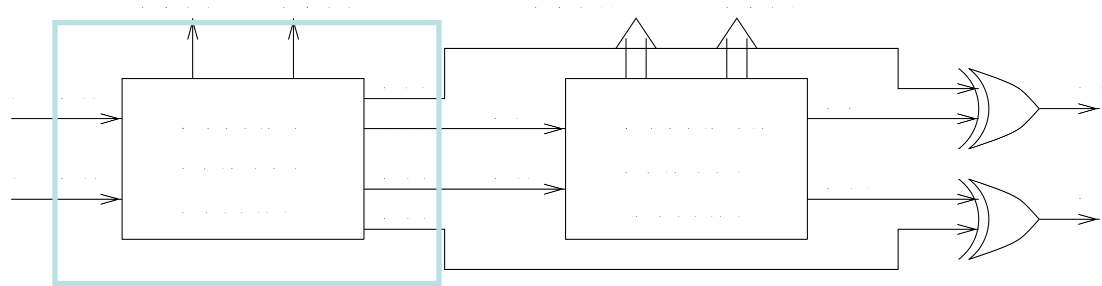


LPN

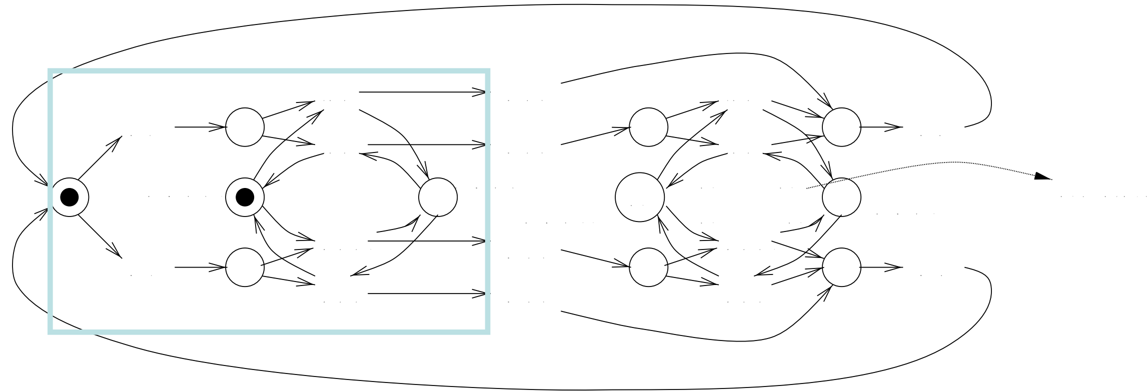


Direct synthesis example (modulo-k Up-Down counter)

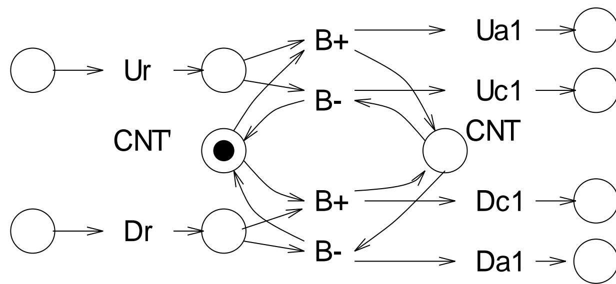
structure



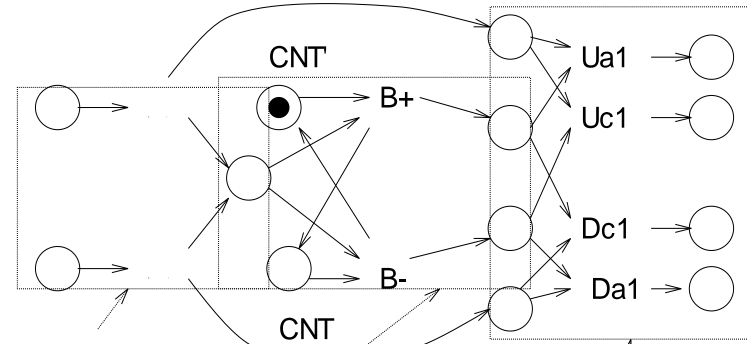
LPN



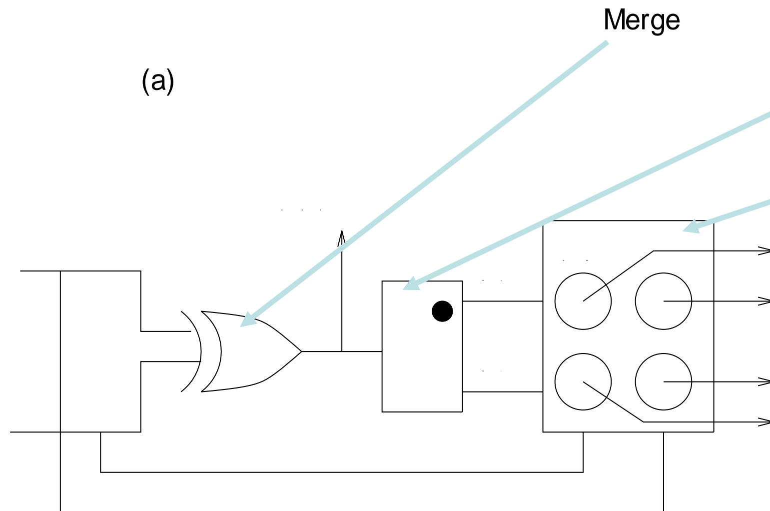
Direct synthesis example (modulo-k Up-Down counter)



(a)

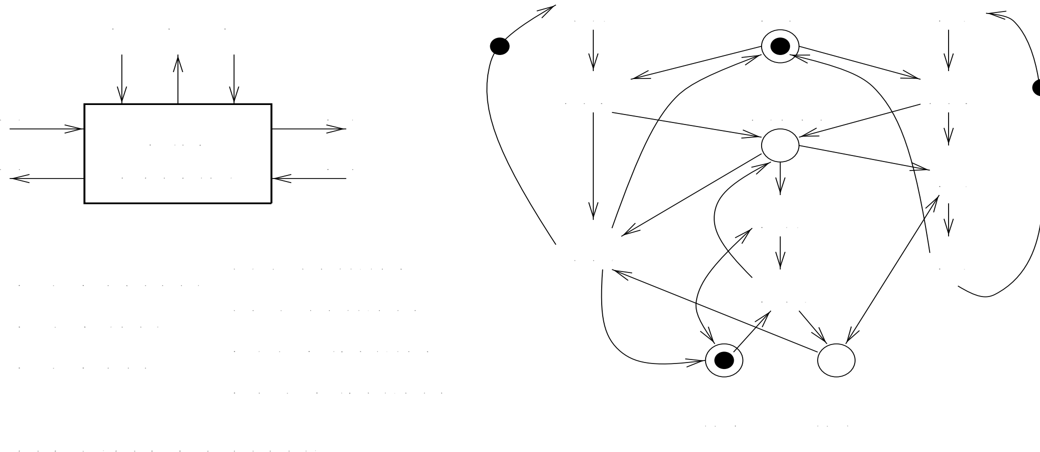


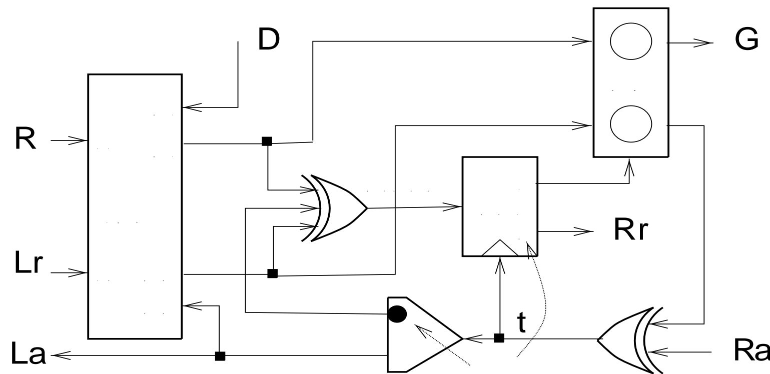
(b)



(c)

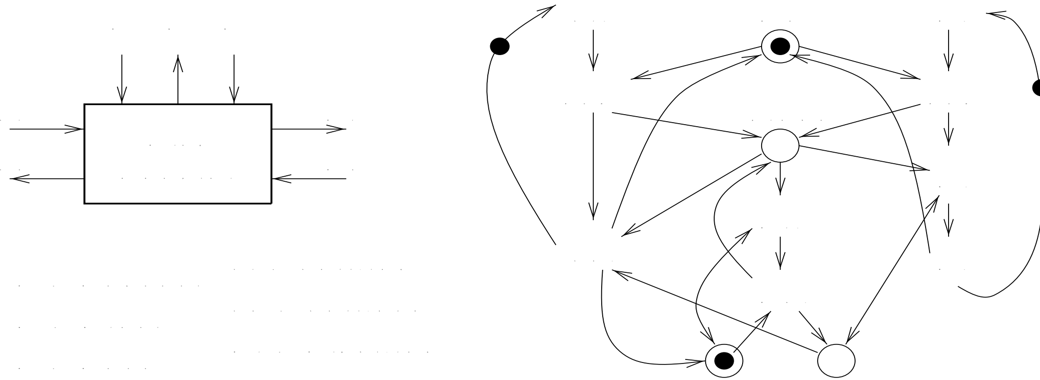
Direct synthesis example (lazy token ring adapter)

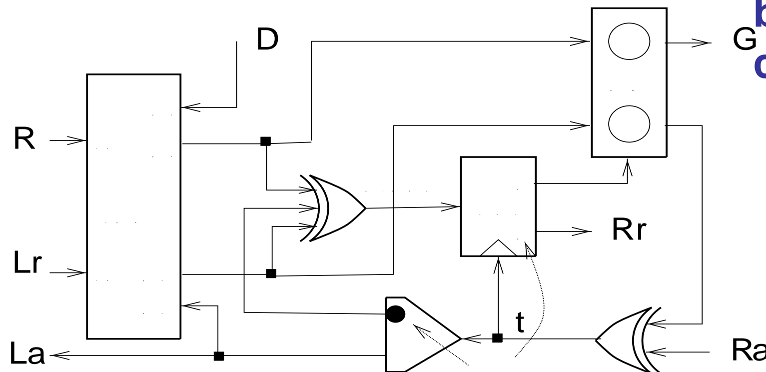




(b)

Direct synthesis example (lazy token ring adapter)



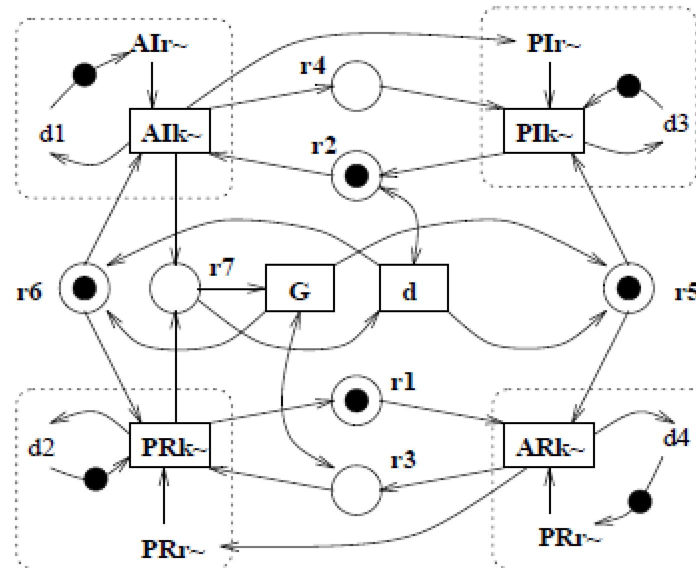


(b)

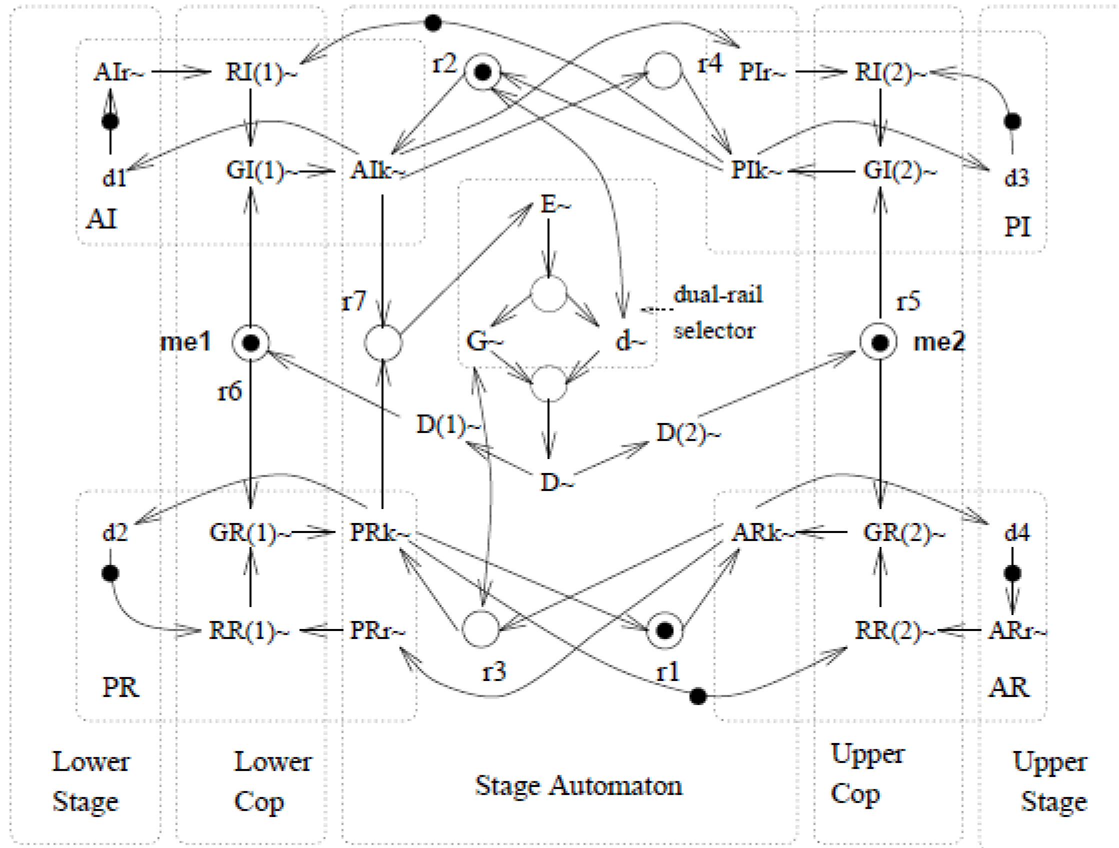
Exercise:

Refine this initial LPN and map it (fragment-by-fragment) to the circuit

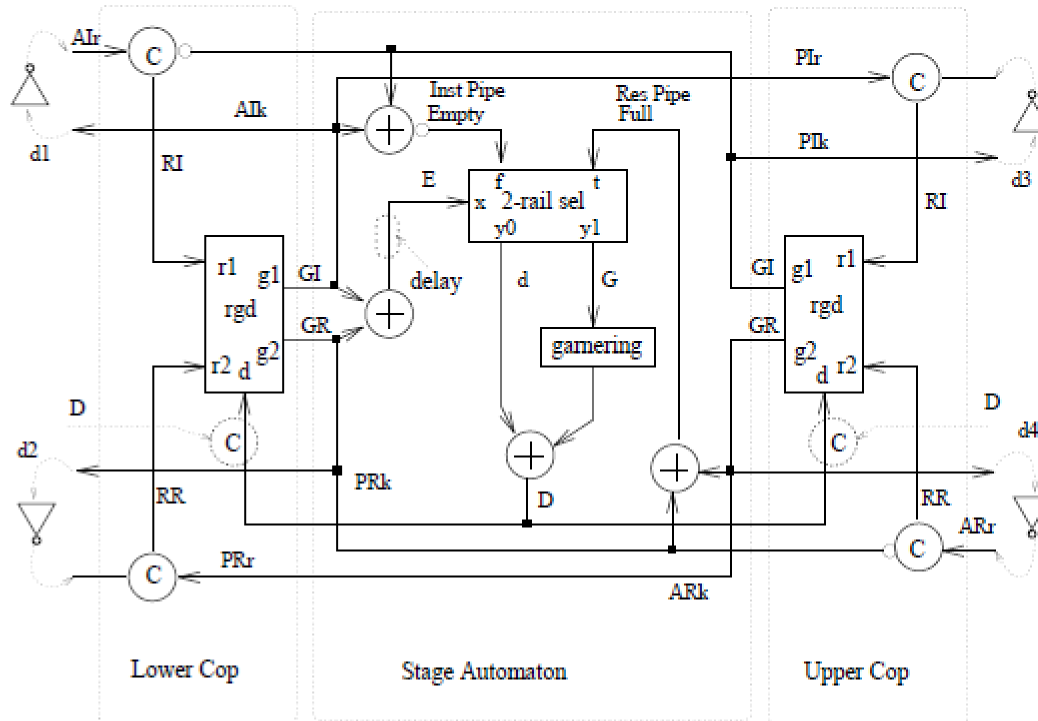
Implementation of Counterflow Pipeline



Counterflow pipeline implementation



CFPP Implementation



Workcraft

GUI-based tool from Newcastle to work with interpreted graph models

Latest downloadable version - http://workcraft.org

Tools at UPC, Barcelona

- Petrify *(J. Cortadella)*
  - http://www.lsi.upc.edu/~jordicf/petrify/
- Genet (*J. Carmona)*
  - http://www.lsi.upc.edu/~jcarmona/genet.html
- Rbminer *(M. Solé)*
  - http://www.lsi.upc.edu/~jcarmona/rbminer/

Tools by other groups

- SYNET *(B. Caillaud)*
  - *http://www.irisa.fr/s4/tools/synet/*
- VipTool (*Bergenthum et al.)*
  - *http://viptool.ku-eichstaett.de/wiki/doku.php*
- Parikh Miner *(B. Van Dongen)*
  - http://prom.win.tue.nl/research/wiki/prom/start

Petrify *(J. Cortadella et al.)*

- Features:
  - Synthesis/mining of safe Petri nets
  - Synthesis of asynchronous controllers
  - Stable, efficient, long-standing tool
  - Used in universities and industry.

- http://www.lsi.upc.edu/~jordicf/
  - Binaries for Linux/Sun/Windows
  - Related papers
  - Tutorial.

Genet *(J. Carmona)*

- Features:
  - Synthesis/mining of general Petri nets
  - Decompositional/Divide-and-Conquer approaches
  - GenetGUI: graphical user interface (by J. Muñoz)

- http://www.lsi.upc.edu/~jcarmona/genet.html
  - Binaries for Linux/Sun
  - Related papers
  - Tutorial.

Rbminer (M. Solé)

- Features:
  - Mining of general Petri nets
  - Helper applications (log2ts, ...)
  - Efficiency

- http://www.lsi.upc.edu/~jcarmona/rbminer/
  - Binaries for Linux/
  - Related papers
  - Tutorial.