

Modeling with Petri nets

dr. Tamás Bartha

dr. András Pataricza

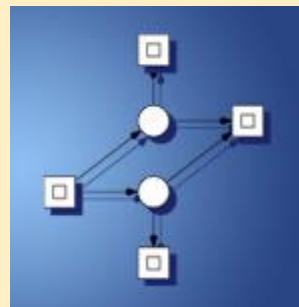
dr. István Majzik

BME Department of Measurement and Information Systems

Modeling tools: DNAnet, Snoopy, PetriDotNet



UNIVERSITY OF CAPE TOWN
Department of Computer Science



b.tu

Brandenburgische
Technische Universität
Cottbus

petridotnet
from BME-MIT



The PetriDotNet modeling tool

- Features
 - Graphical editor + token game + simulation
 - Easy to use, many convenience functions
 - **Extensions:** inhibitor arcs, timings, colored nets
 - Supports **hierarchical** Petri nets
 - Supports plug-ins, e.g. **analysis modules**
 - Dynamic properties, **CTL model checker**
 - Coloring, rotating elements, displaying arc weights
 - Standard PNML format, with INA export
- Developed by us: petridotnet.inf.mit.bme.hu

PetriDotNet screenshot

The screenshot displays the PetriDotNet software interface. The main window, titled "PetriDotNet - [Jatekautomata]", features a menu bar (File, Edit, View, Insert, Mode, Tools, Add-in, Window, Help) and a toolbar with various editing tools. The interface is divided into three main sections:

- Design/Simulation Panel:** Contains a "Toolbox" with buttons for "Select", "Place", "Transition", "Edge", and "Token", along with "Other elements". Below it is the "Properties" section, currently showing the "Identity" tab with fields for "Id" (n1) and "Name" (Net 1).
- Diagram Area:** A Petri net is shown on a grid. It consists of five places: "zseton" (top, 2 tokens), "nyer" (left, 2 tokens), "uzemben" (center, 1 token), "veszit" (center, 1 token), and "indul" (right, 1 token). There are also two transitions: "jatekban" (bottom) and an unlabeled transition between "nyer" and "uzemben". Edges connect these elements in a cycle.
- About PetriDotNet Dialog:** A dialog box is open in the bottom right corner, displaying the software's logo, version (1.3.4098.34586), and credits: "Petri Net Editor by Dániel Darvas, 2009-2011 at BME-MIT (BUTE DMIS) based on Petri.NET 1.0 by Bertalan Szilvási (advisor: Gábor Huszerl), 2008". It also lists advisors: "András Vörös (BME-MIT)" and "dr. Tamás Bartha (BME-MIT)". Contact information is provided: "http://petridotnet.inf.mit.bme.hu/" and "petridotnet@inf.mit.bme.hu". Buttons for "Send error feedback" and "OK" are at the bottom.

PetriDotNet analysis features

Properties of Net AlterBit

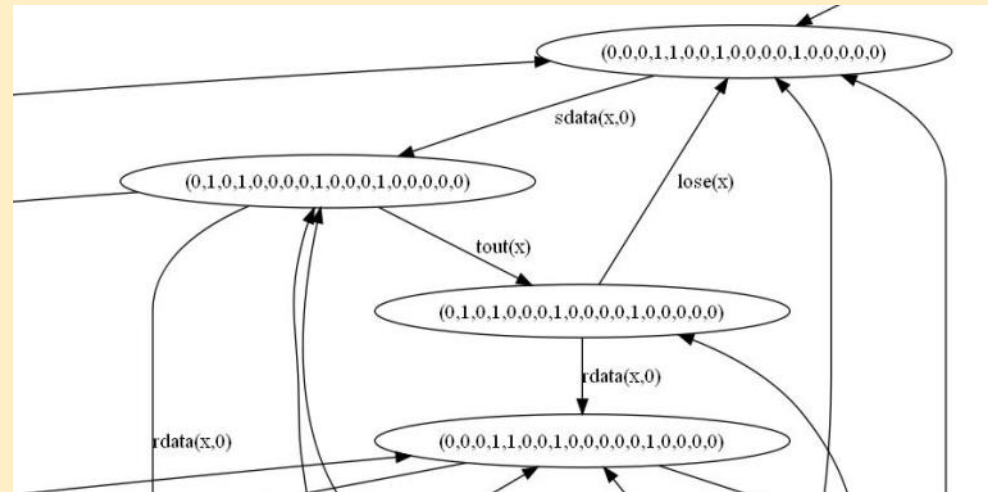
Dynamic Properties

Number of states:	108
Boundedness:	Bounded 1-bounded (safe net)
Deadlock freedom:	Deadlock free
Reversibility:	Reversible
Persistency:	Non-persistent

Static Properties

Most specific subclass:	Petri Net
Purness:	Not Pure

[Reachability check](#); [CTL check](#); [Save the reachability graph](#);
[Save adjacency matrix](#); [Search T-invariants](#); [Search P-invariants](#);
[Display token bounds of places](#);



CTL Expression Editor

AlterBit.buffer_x 0 Insert

Insert full expression

AF(AlterBit.wfa_0>0&EX(AlterBit.buffer_x>0))

OK

CTL MODEL CHECKING
Expression: AF(AlterBit.wfa_0>0&EX(AlterBit.buffer_x>0))
Model: AlterBit
Result: True
Runtime: 0,01 s

OK

PetriDotNet invariant analysis

The image shows a screenshot of the PetriDotNet software interface. On the left, a Petri net is visible with places labeled `ack_0`, `lose_0`, `empty_ack`, `lose_1`, and `ack_1`. The net consists of orange circles (places) and rectangles (transitions) connected by orange lines (edges).

Four windows are overlaid on the net:

- Show invariants on the net** (top left): A dropdown menu shows the invariant `{1 × ack_0, 1 × ack_1, 1 × empty_ack_}` and a `Show` button.
- Show invariants on the net** (top right): A dropdown menu shows the invariant `{lose_x, sdata_x,0, tout_x_}` and a `Show` button.
- P-Invariants** (center): A window titled "List of P-Invariants calculated by Martinez-Silva algorithm". It contains the text: "Calculation finished in 2,00 ms. (places=18, transitions=22)" followed by a list of invariants:
 - `{1 × ack_0, 1 × ack_1, 1 × empty_ack_}`
 - `{1 × data_x, 1 × empty_data_, 1 × data_y}`
 - `{1 × rts_x, 1 × queue_x, 1 × wfa_0, 1 × rts_y, 1 × wfa_1, 1 × queue_y}`
 - `{1 × wait_0, 1 × buffer_x, 1 × ok_x, 1 × ok_y, 1 × buffer_y, 1 × wait_1}`An `OK` button is at the bottom.
- T-Invariants** (right): A window titled "List of T-Invariants calculated by Martinez-Silva algorithm". It contains the text: "Calculation finished in 1,00 ms. (places=18, transitions=22)" followed by a list of invariants:
 - `{lose_x, sdata_x,0, tout_x_}`
 - `{lose_y, sdata_y,1, tout_y_}`
 - `{rack_1, put_x, sdata_x,0, rack_0, sdata_y,1, put_y, drop_y, sack_0, drop_x, sack_1}`
 - `{lose_1, sdata_y,1, tout_y, drop_y, sack_1}`
 - `{drop_1, sdata_y,1, tout_y, drop_y, sack_1}`
 - `{lose_y, rack_1, put_x, sdata_x,0, rack_0, sdata_y,1, put_y, tout_y, drop_y, sack_0, drop_x, sack_1}`
 - `{lose_0, sdata_x,0, tout_x, sack_0, drop_x}`
 - `{sdata_x,0, tout_x, drop_0, sack_0, drop_x}`
 - `{lose_x, rack_1, put_x, sdata_x,0, tout_x, rack_0, sdata_y,1, put_y, drop_y, sack_0, drop_x, sack_1}`
 - `{lose_x, lose_y, rack_1, put_x, sdata_x,0, tout_x, rack_0, sdata_y,1, put_y, tout_y, drop_y, sack_0, drop_x, sack_1}`An `OK` button is at the bottom.

Basic principles of modeling

Purpose of system modeling

- IT systems are usually well decomposed
 - Building systems by integrating components
 - Steps, processes, threads, ...
 - Relationships between basic components:
 - **Explicit** logical relationship: order, causality
 - **Implicit** dependency: e.g. using shared resource
- Model-based analysis: qualitative and/or quantitative
 - **Qualitative**: proving logical correctness
 - **Quantitative**: performance analysis, reliability, availability, safety analysis

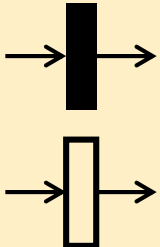
Building a model

- Three main model element categories:
 - Processes, containing activities
 - Resources (including: data, messages, channels)
 - Interactions between processes and resources
- Modeling: hierarchical and functional
 - Bottom up:
 - Basic activities -> (Composite activities ->)
 - Subprocesses -> Composite processes
- Steps:
 - Building individual model elements
 - Integration

Typical steps of system modeling

1. The **process** model (without detailed resource usage and communication)
2. The **resource** model
 - A finite automaton part with busy/idle/... states
 - Message queue (if needed)
3. **Integration**: Fusion of corresponding transitions in the process and resource models
 - E.g.: *Occupying* fused with transition *Idle* → *Busy*
 - E.g.: sending message puts message into queue

Modeling activities in Petri nets

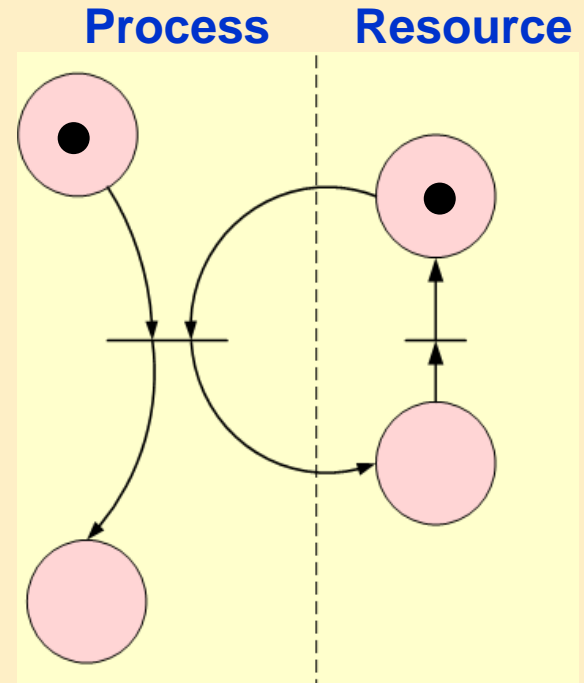
- Basic activity: firing a transition
 - Resources used: input / output places
 - Execution time
 - deterministic
 - stochastic
- }  deterministically timed transition
exponentially timed transition

Questions regarding enabledness:

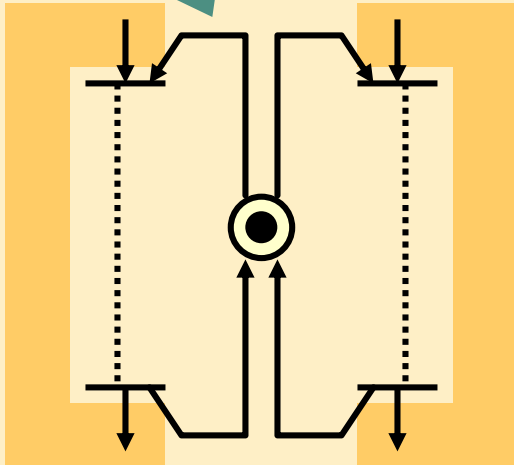
- Untimed transitions fire first (higher “priority”)
- What happens with time after becoming disabled?
 - Restarts (new random): “restarts” activity
 - Remains (previous time): “continues” activity

Example: Modeling resource allocation

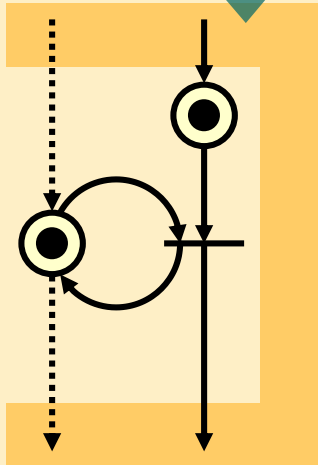
- Allocation of required resource
- Mutual exclusion
- Using limited resource



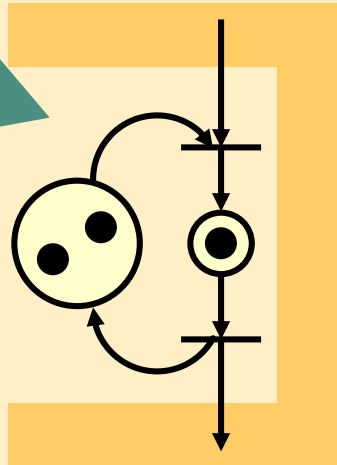
Modeling mutual exclusion



Reading state variable

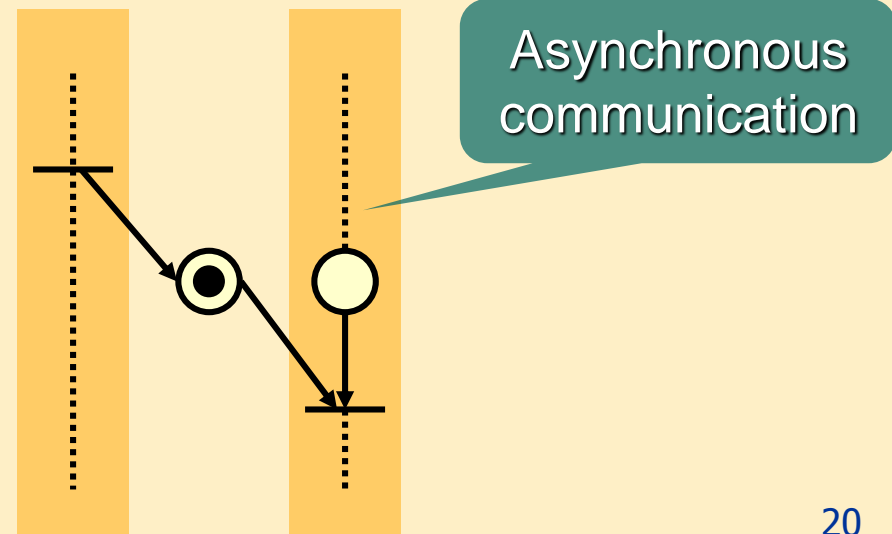
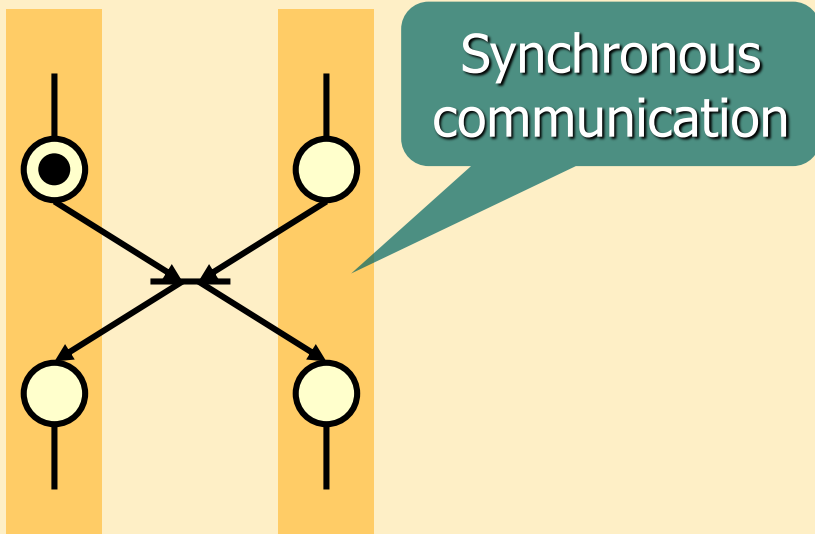
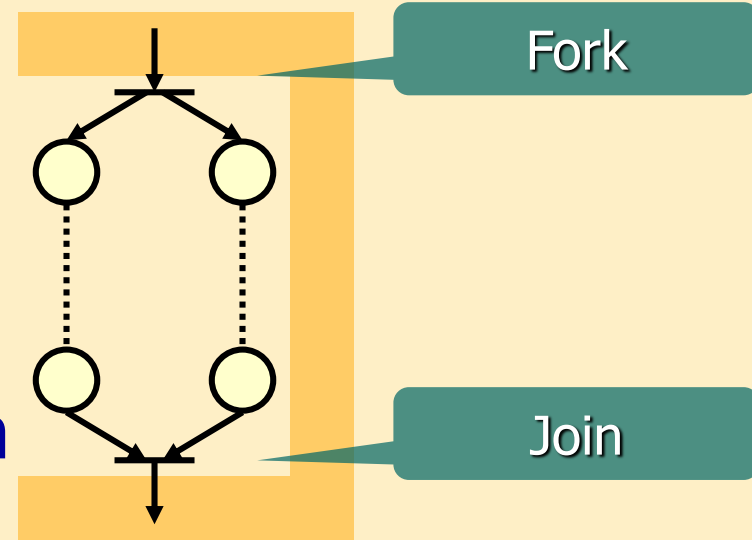


Modeling limited resource capacity



Example: Relationships between processes

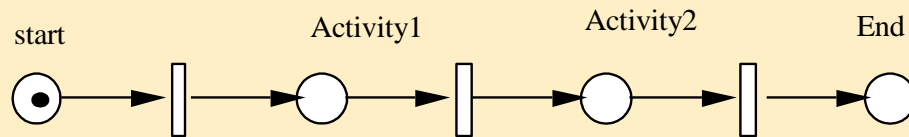
- Parallelism
 - Fork and join
- Synchronous communication
 - Wait for the other
- Asynchronous communication
 - Like a mailbox



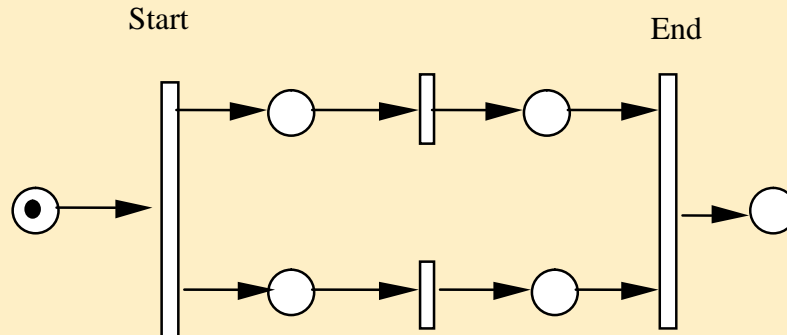
Example: Modeling a production cell

Processors

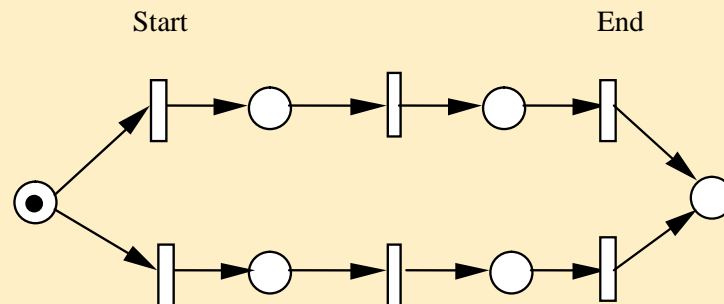
- Sequential processors:



- Parallel processor:

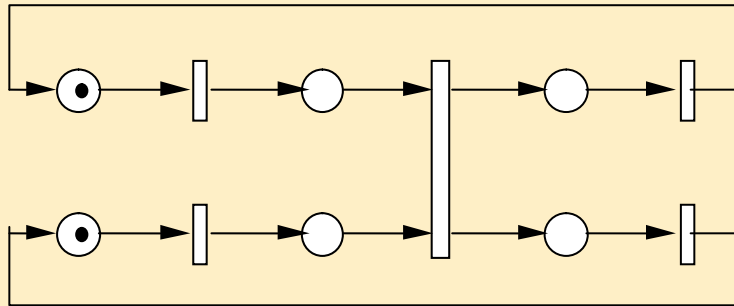


- Alternative processor:

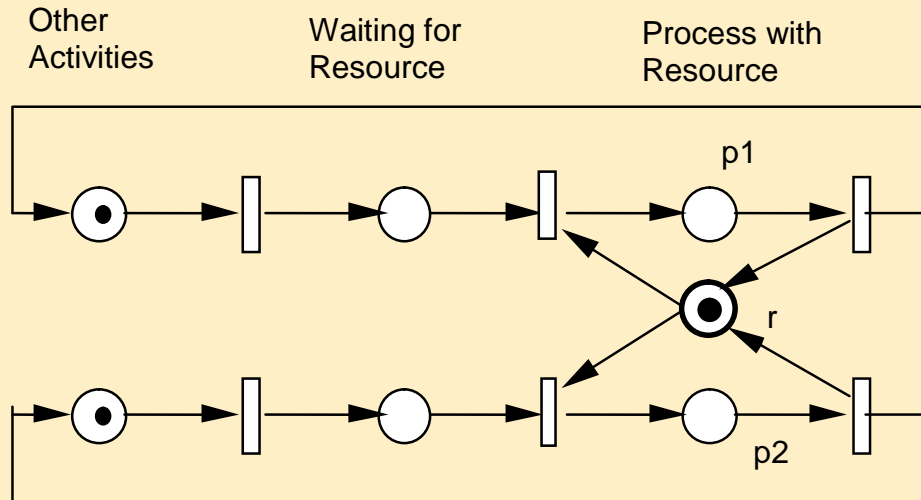


Interactions

- Synchronization:

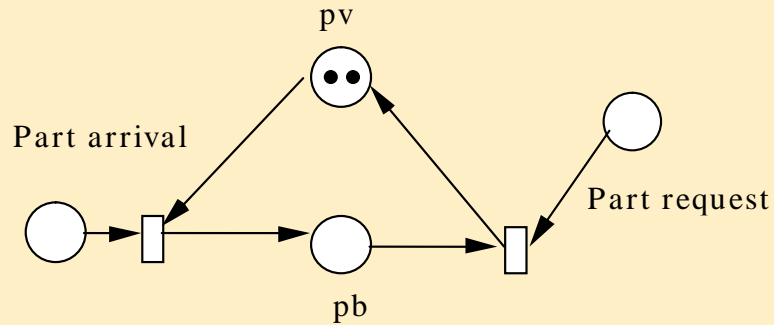


- Shared resource:

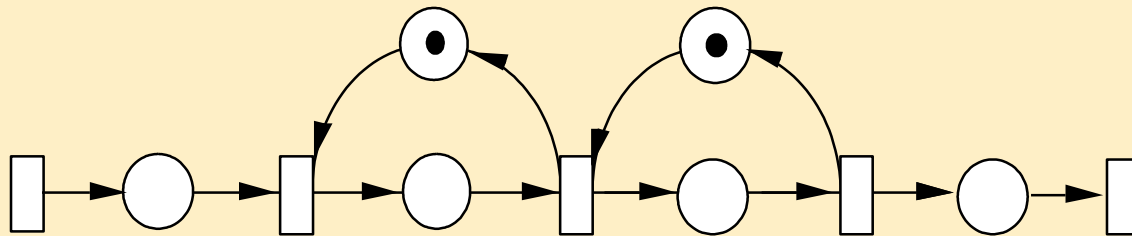


Containers for processors

- Bounded capacity container:

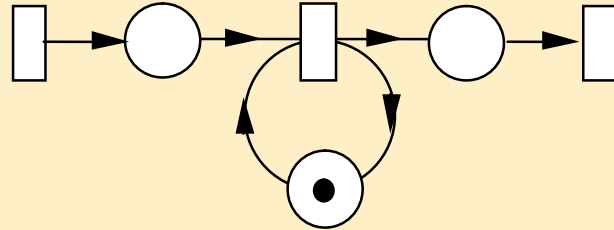


- FIFO container:

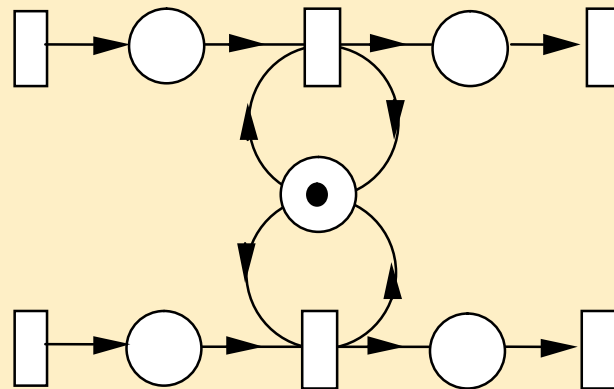


Using machines

- Process with dedicated machine:

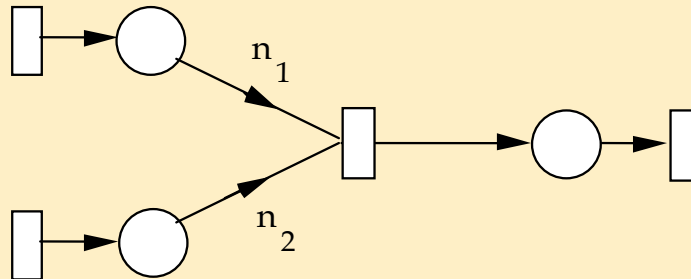


- Process with shared machine:

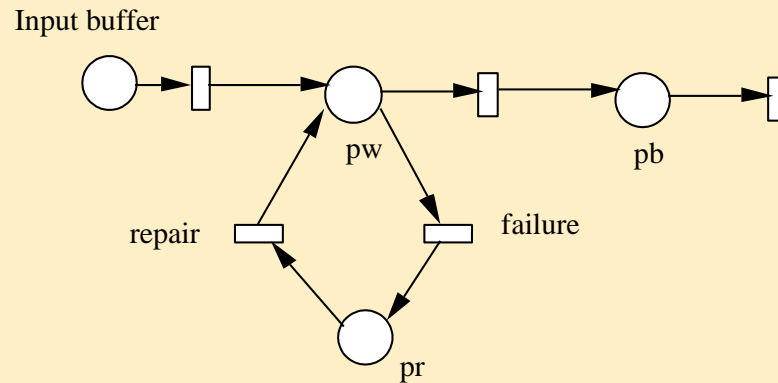


Assembly

- Assembling parts:

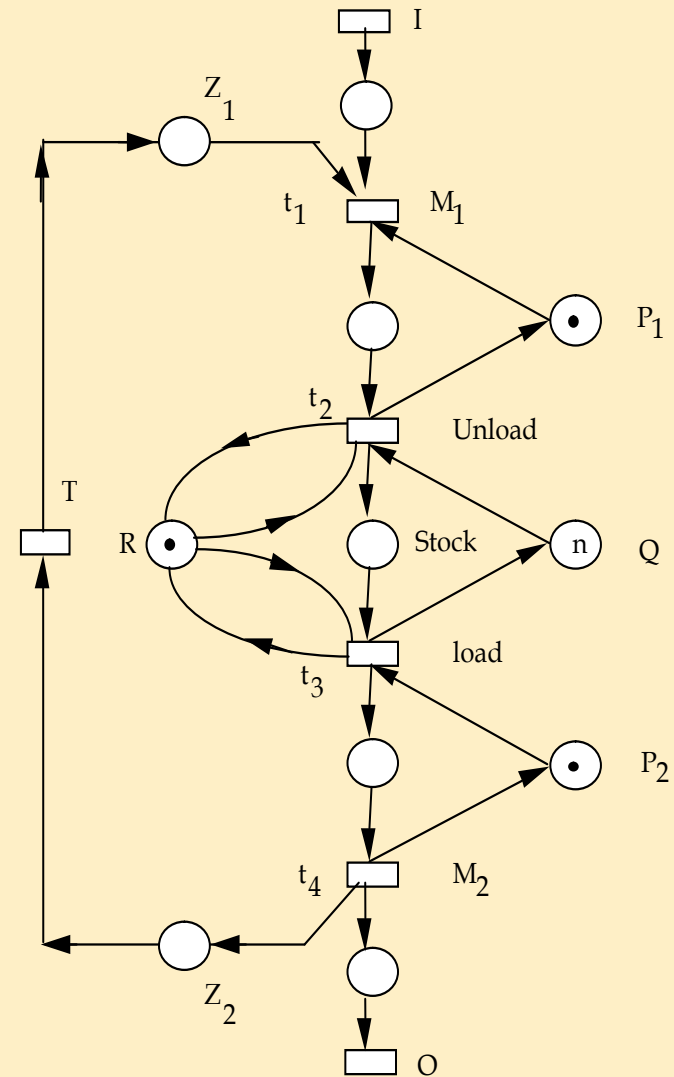
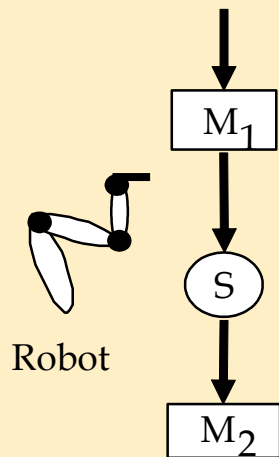


- Failure during process:



Robot cell

- Activities
- Containers (bounded capacity)
- Resources
- Cyclic behavior



Example Petri net: Alternating bit protocol

The modeling task

Alternating Bit Protocol

- Transmission protocol for faulty channels
 - Messages can get lost (a finite number of times)
 - Contents of messages cannot change
- Goal: the protocol should ensure (with a bounded number of steps) that the message is transmitted to the receiver

Sender process

- Attaches a checking bit to the message
- Received messages are confirmed by the receiver, with the same checking bit
- If the bit attached to the message is b^0 , then
 - if the message is lost, the sender detects the lack of confirmation with a timeout → sends again
 - if the sender receives a confirmation with a bit b^0 (which is expected), then a negated bit is attached $b^1 = \neg b^0$ to the next message
 - if the sender receives a confirmation with a bit $b^1 = \neg b^0$ (despite expecting b^0), then the confirmation is discarded (and a timeout will occur due to the lack of confirmation)

Receiving process

- Confirms receiving the message by sending back the same checking bit
- If a message with checking bit b^0 is received, then it is confirmed by sending b^0 back, then
 - If the bit of the next message is b^1 (correct), then sends b^1 back to acknowledge
 - If the bit of the next message is b^0 (incorrect), then the message is discarded, but sends a confirmation (assuming that it was a repeated message due to the lack of confirmation)

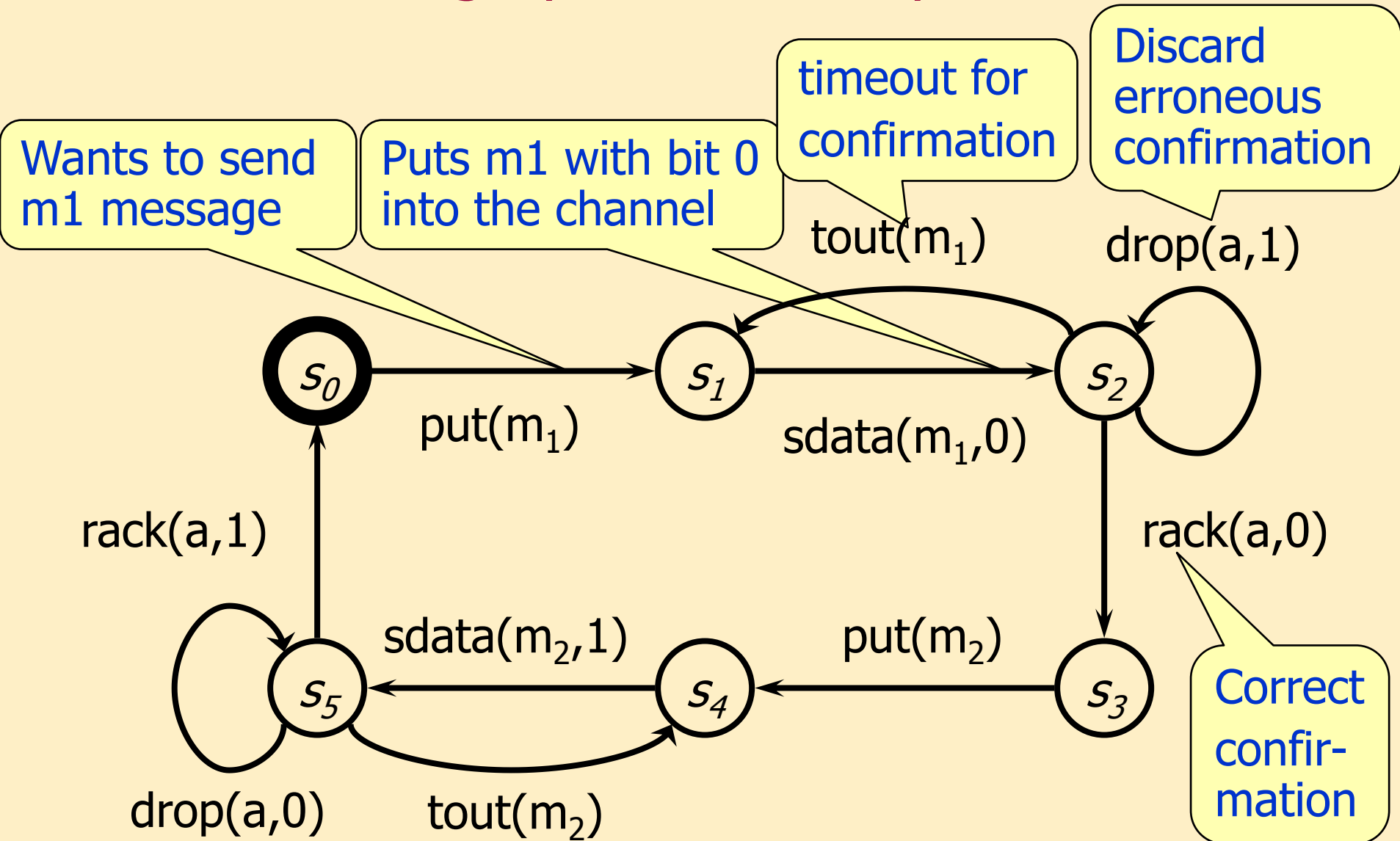
Steps of building the model

1. Decompose the task to actors and resources
2. Determine the states of actors
3. Determine states of resources and message buffers
4. Create Petri net models from state-based models
5. Integrate actor and resource models
6. Check integrated model
7. Use the model to solve the task

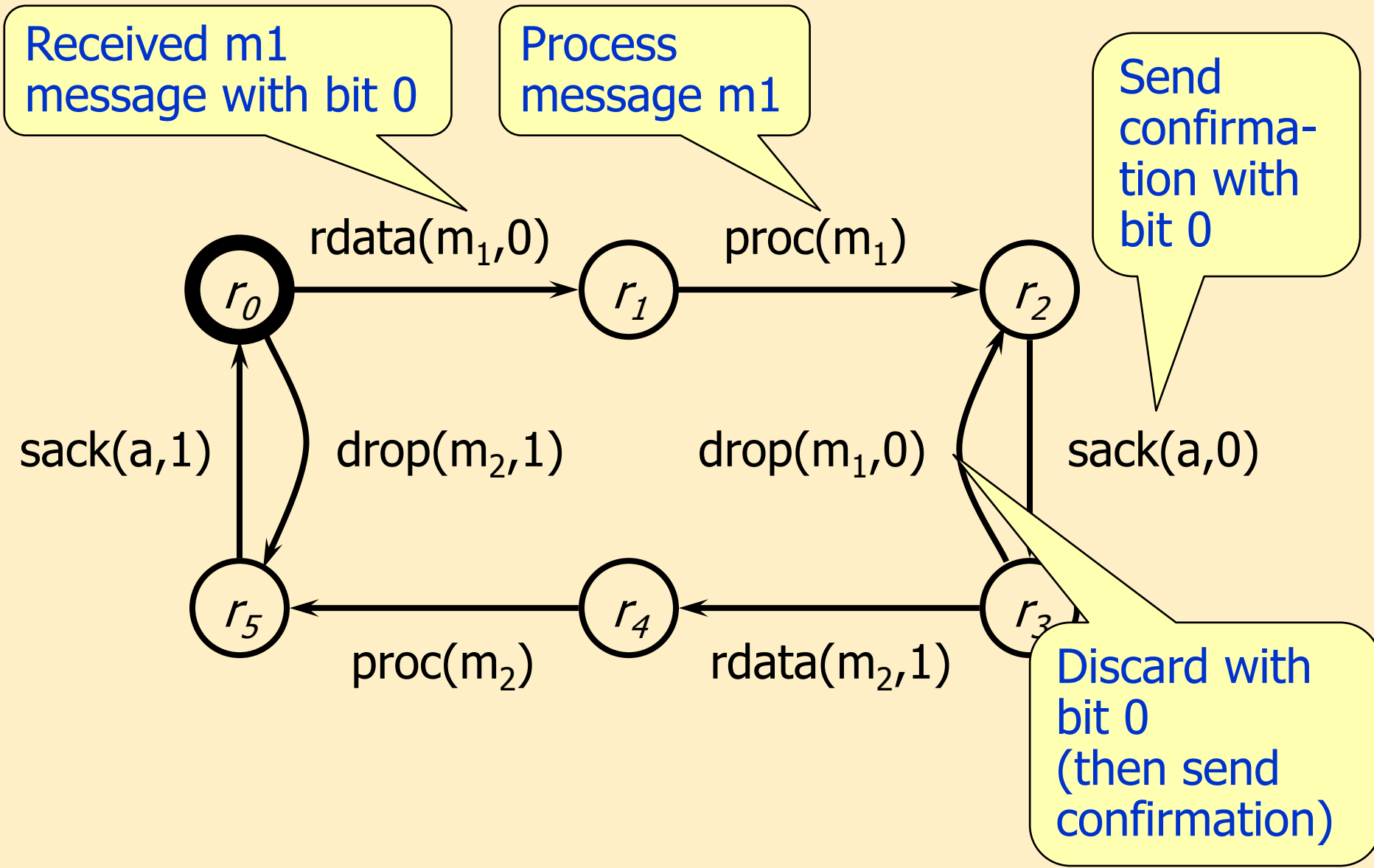
Components and states

- Components (subsystems)
 - Actors: sender process, receiver process
 - Resources: data channel, confirmation channel
- Each components have its own state
 - State graph: states are circles, events are arcs
- Same events happen at the same time:
synchronization

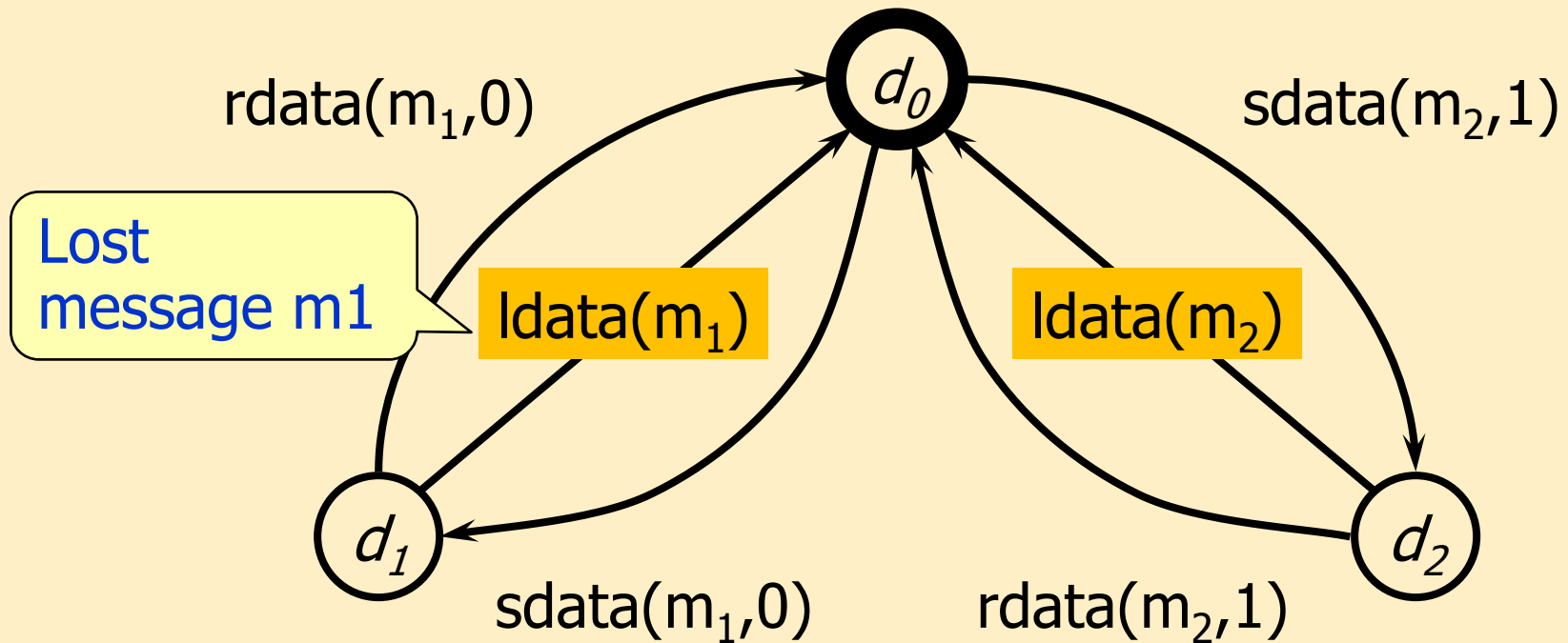
State graph of sender process



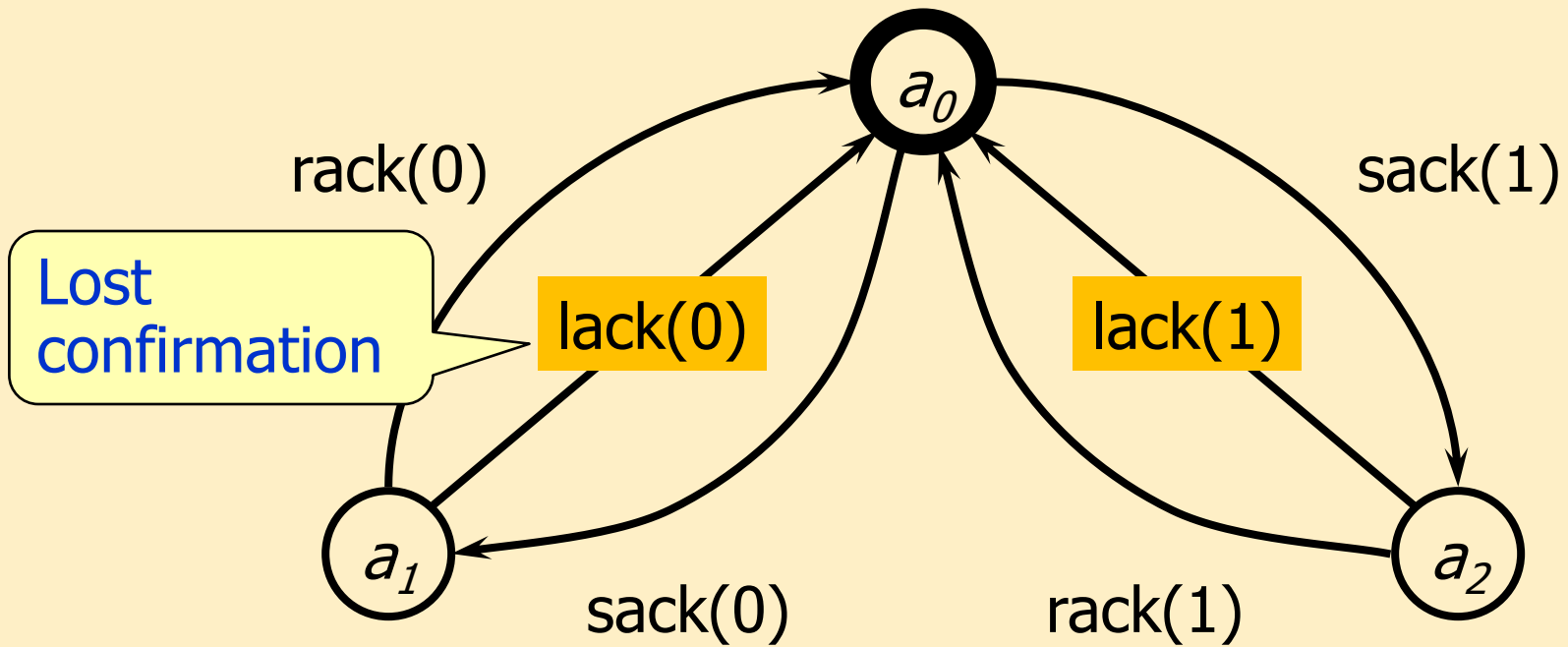
State graph of receiver process



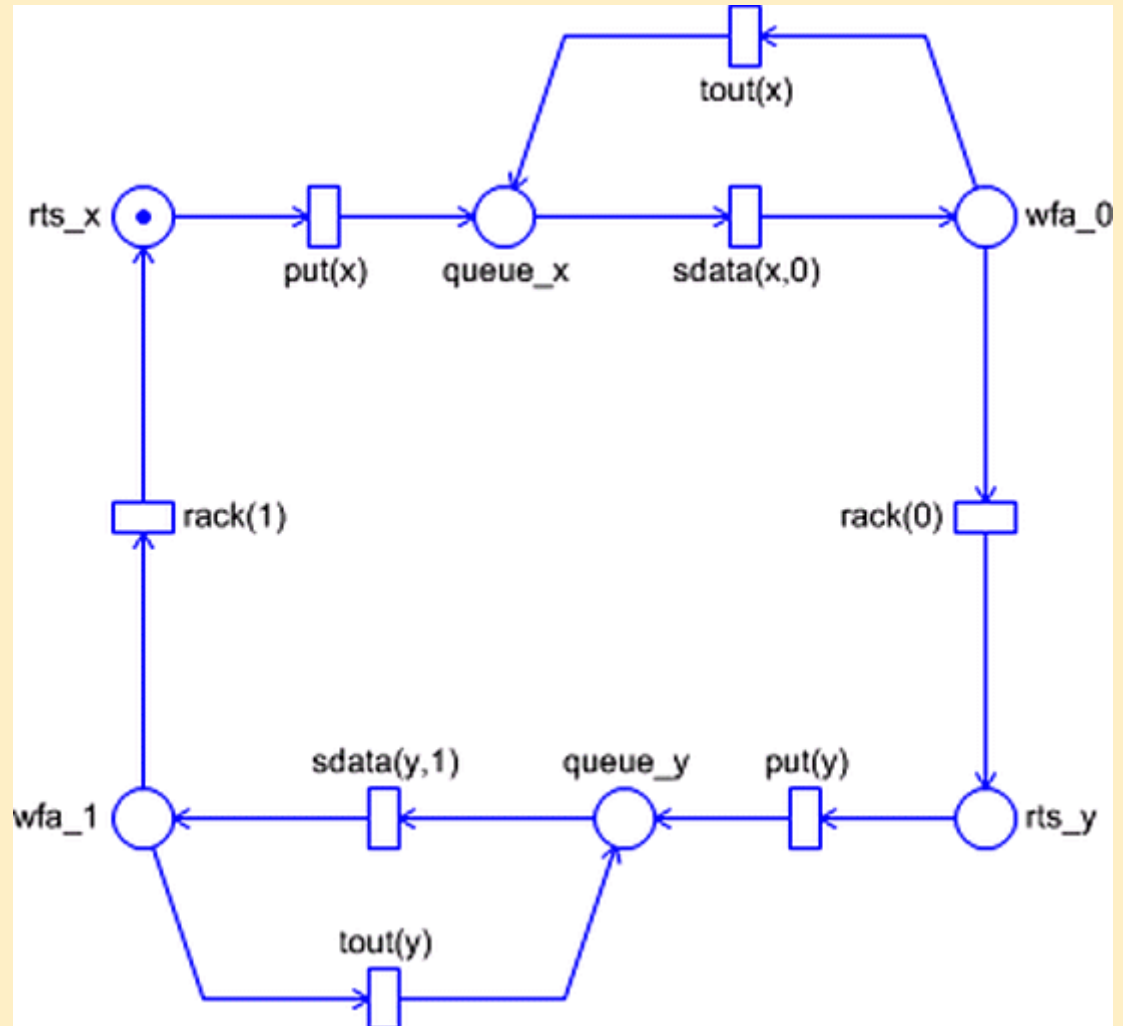
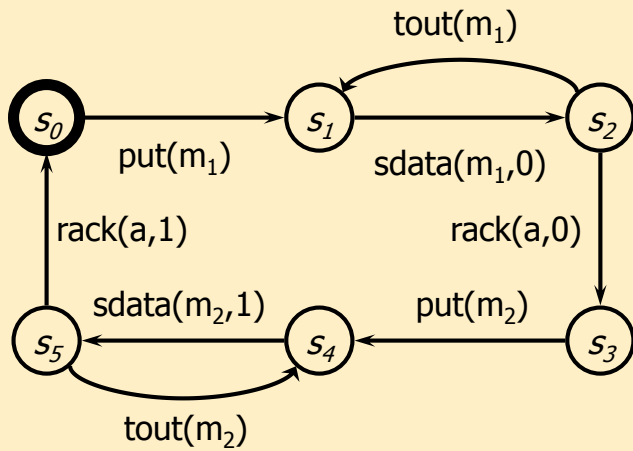
State graph of data channel



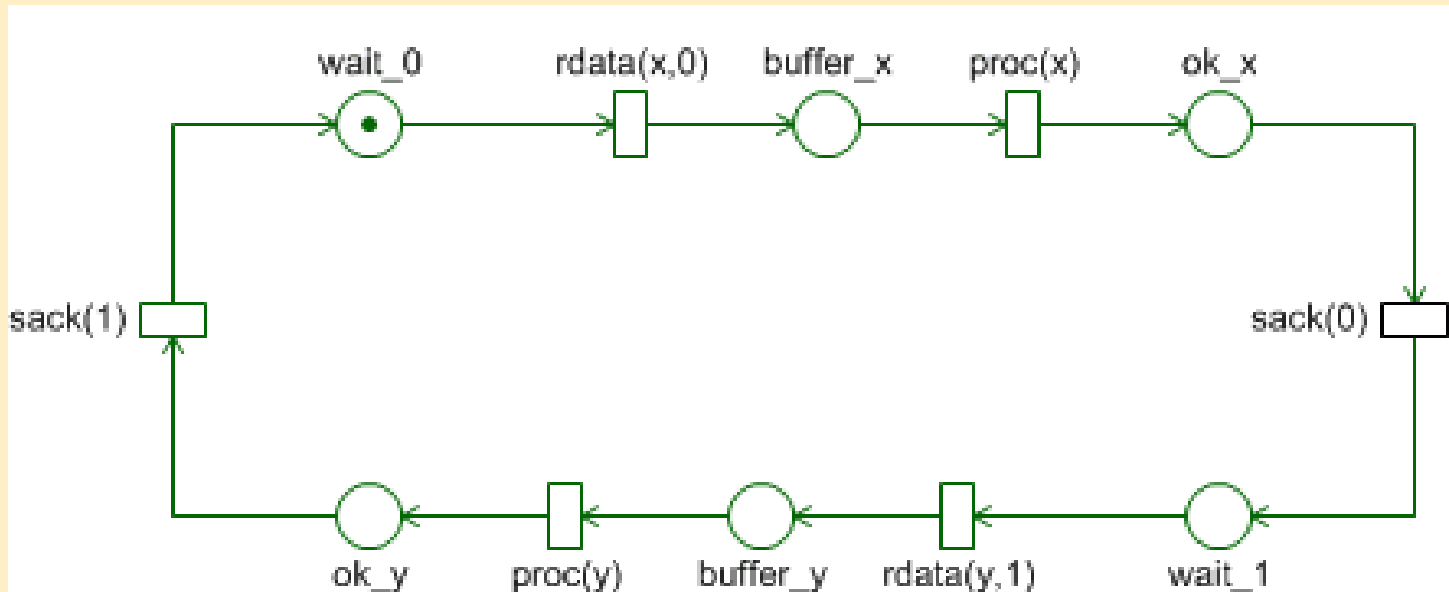
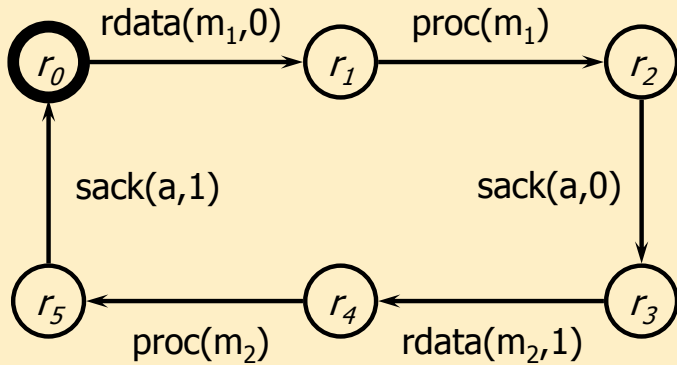
State graph of confirmation channel



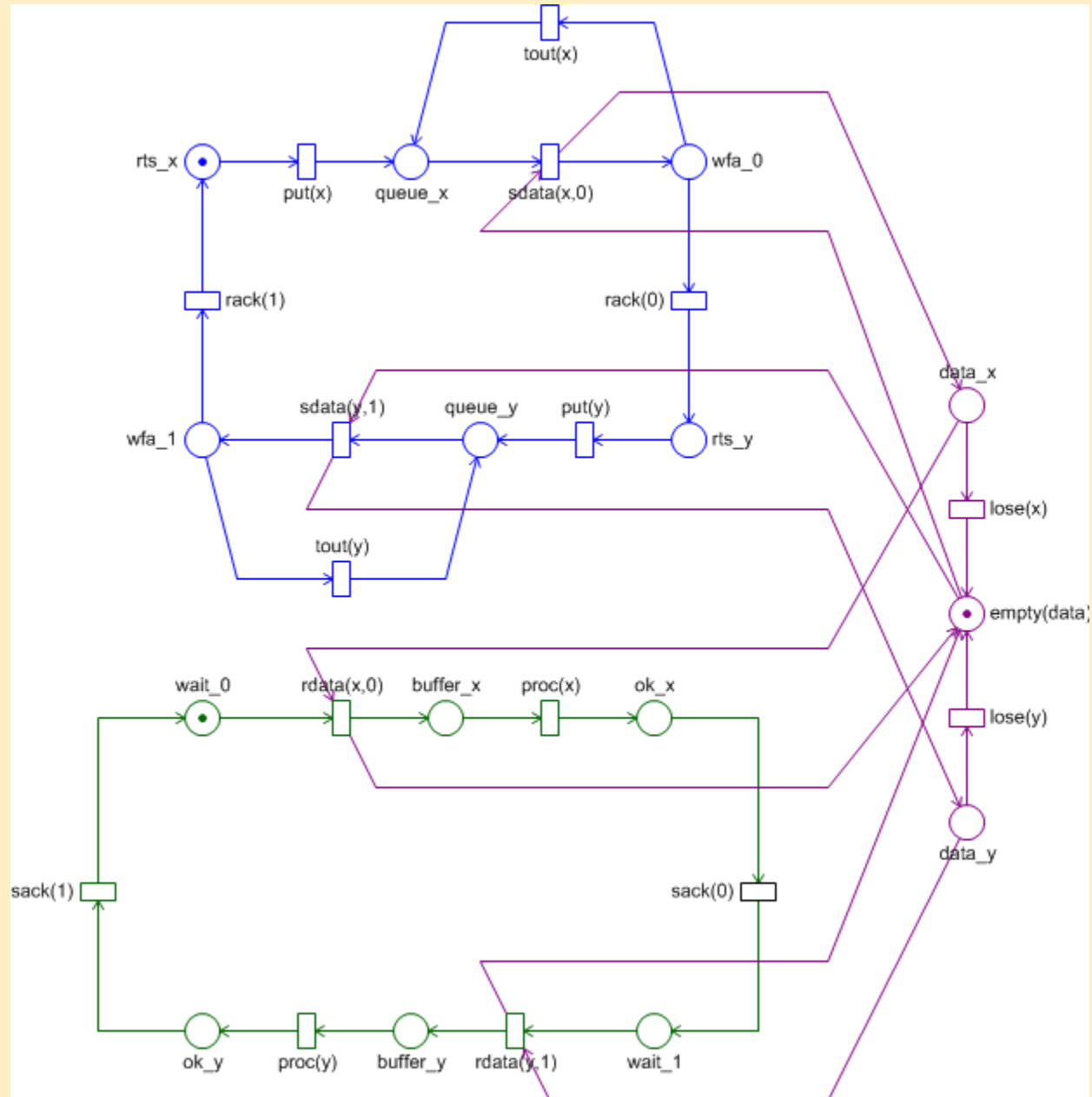
Petri net model of sender process (main loop)



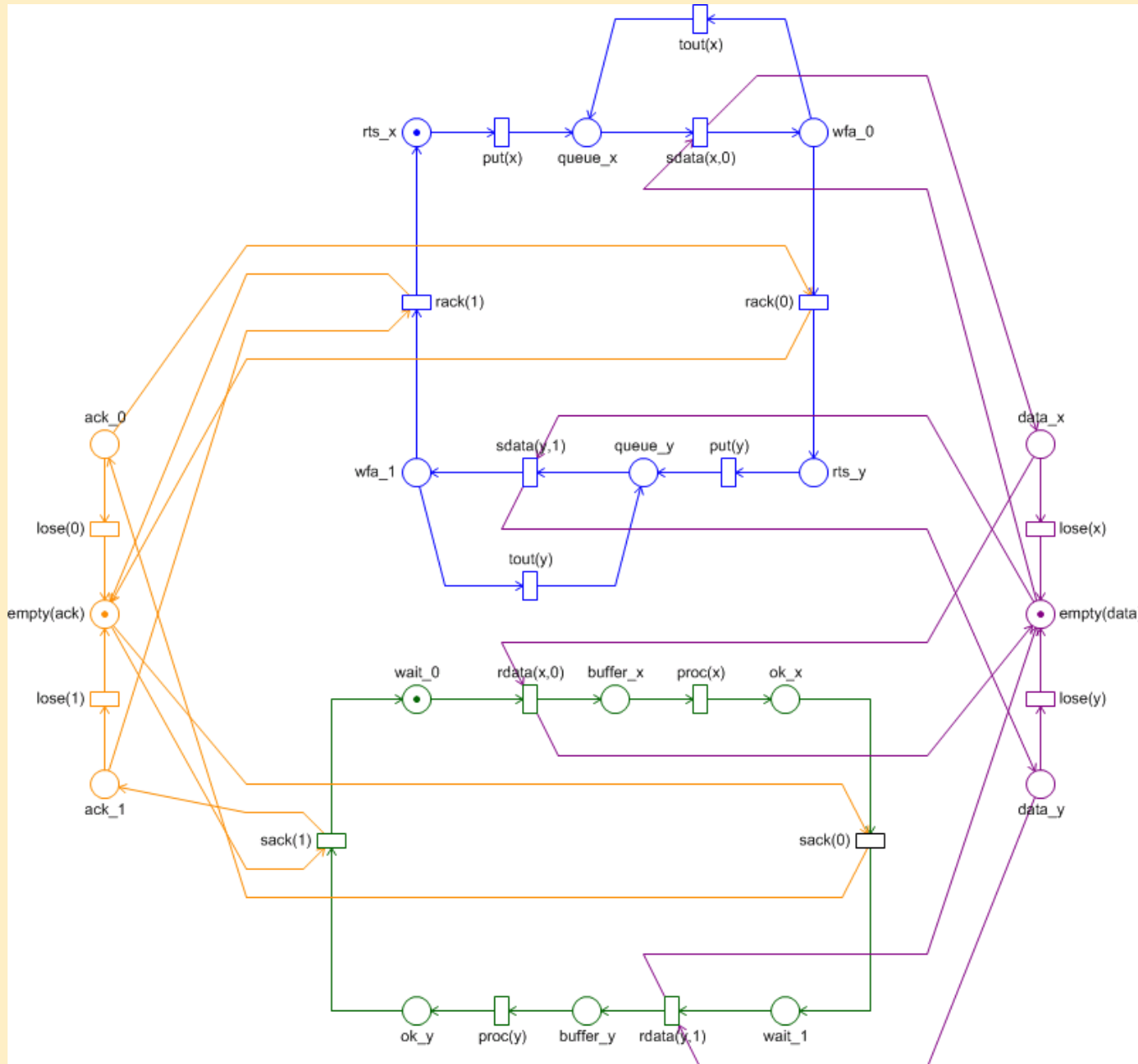
Petri net model of receiver process (main loop)



Data channel and data transmission (main loop)



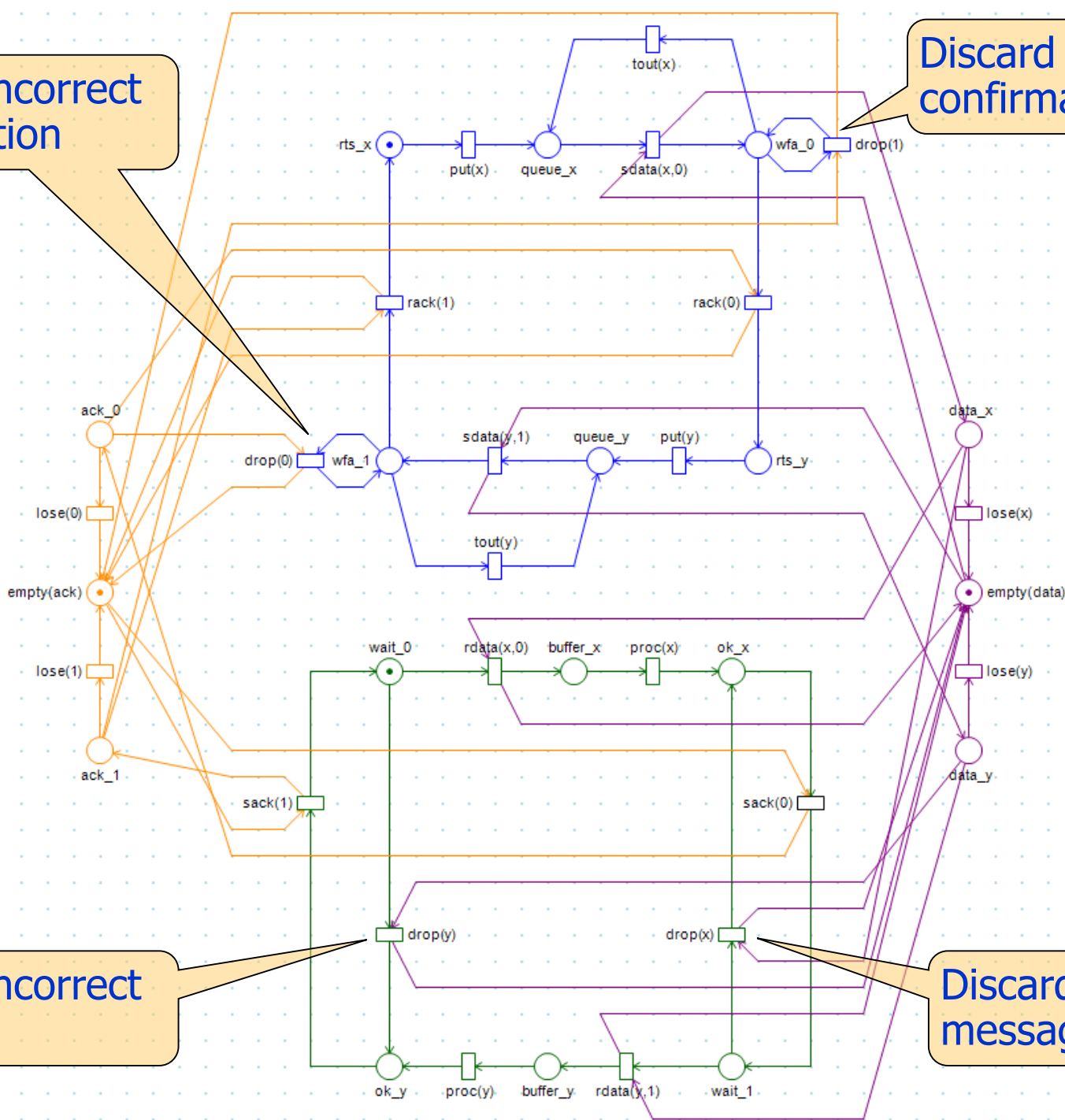
Confirmation channel and confirmation (main loop)



Discard incorrect confirmation

Discard incorrect confirmation

Extensions



Discard incorrect message

Discard incorrect message

Example Petri net: Alternating bit protocol

PetriDotNet: Dynamic properties of the model

Properties of Net AlterBit

Dynamic Properties

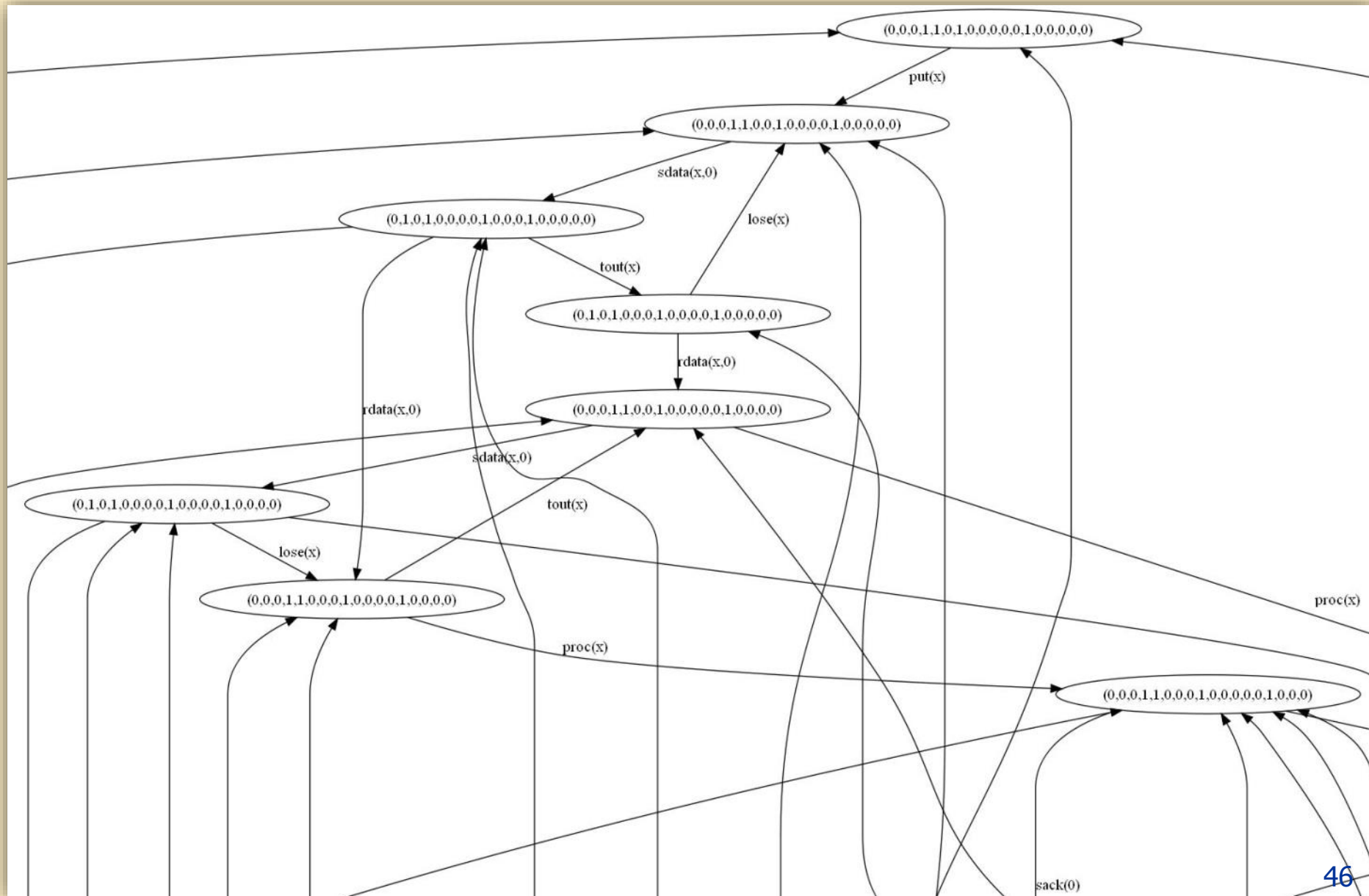
Number of states:	108
Boundedness:	Bounded 1-bounded (safe net)
Deadlock freedom:	Deadlock free
Reversibility:	Reversible
Persistency:	Non-persistent

Static Properties

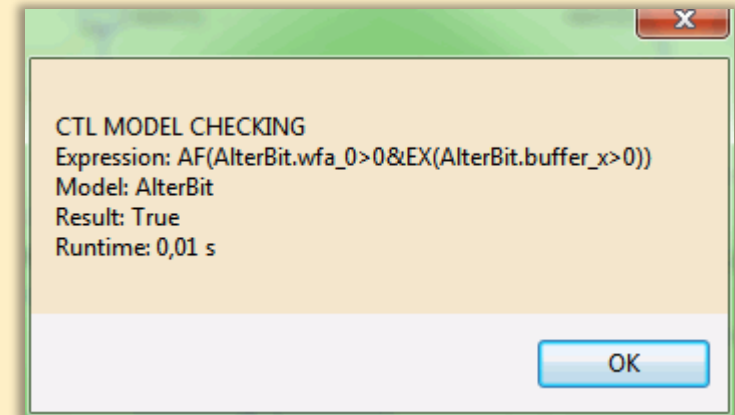
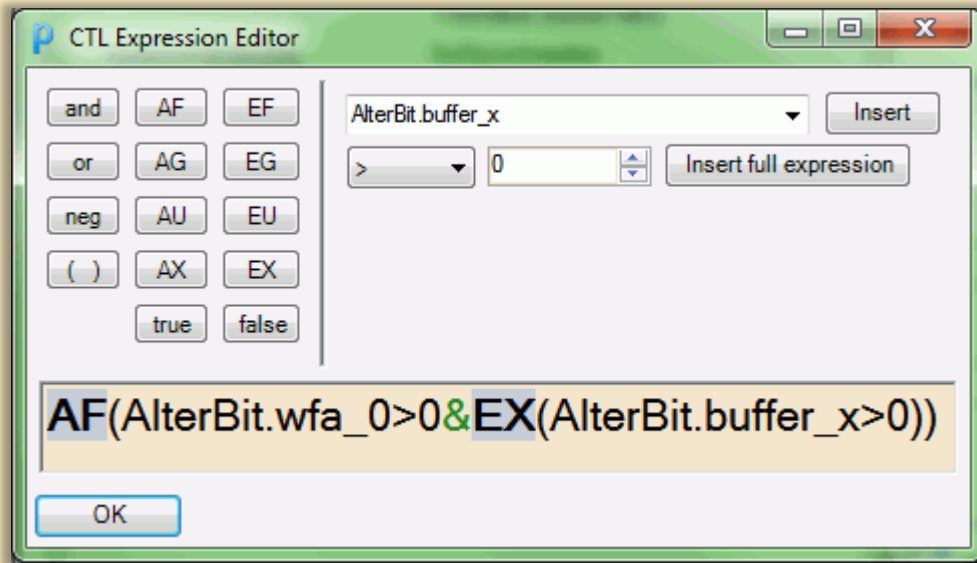
Most specific subclass:	Petri Net
Purness:	Not Pure

[Reachability check](#); [CTL check](#); [Save the reachability graph](#);
[Save adjacency matrix](#); [Search T-invariants](#); [Search P-invariants](#);
[Display token bounds of places](#);

PetriDotNet: Reachability graph (GraphViz)



PetriDotNet: CTL model checking



AF(AlterBit.wfa_0>0 & **EX**(AlterBit.buffer_x>0))

⇒ True

AG(**AF**(AlterBit.buffer_y>0))

⇒ False

AF(**EG**(AlterBit.buffer_x=0))

⇒ True

EF(AlterBit.wfa_0>0 & AlterBit.data_x=0)

⇒ True

AF(AlterBit.queue_x>0 & **AX**(AlterBit.wfa_0>0 & AlterBit.data_x>0)) ⇒ True

PetriDotNet: Invariant analysis

The screenshot displays the PetriDotNet application interface with three windows open:

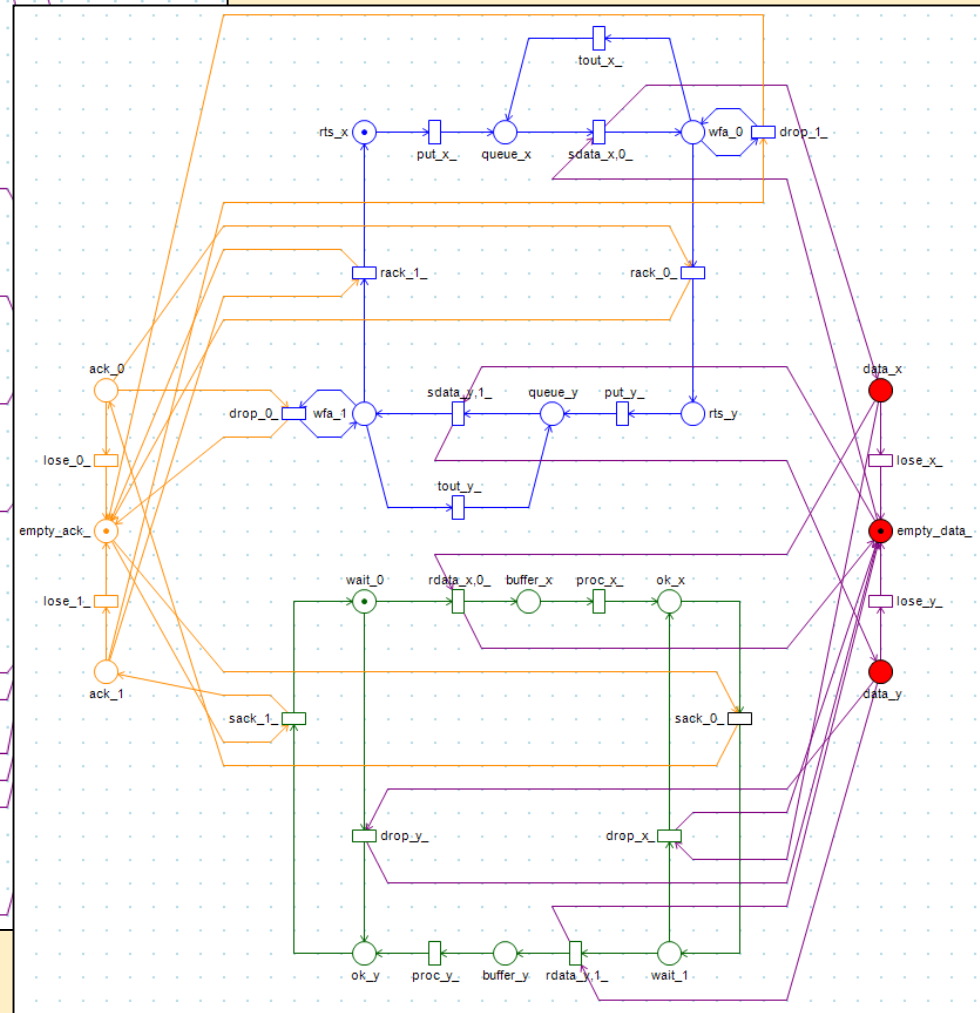
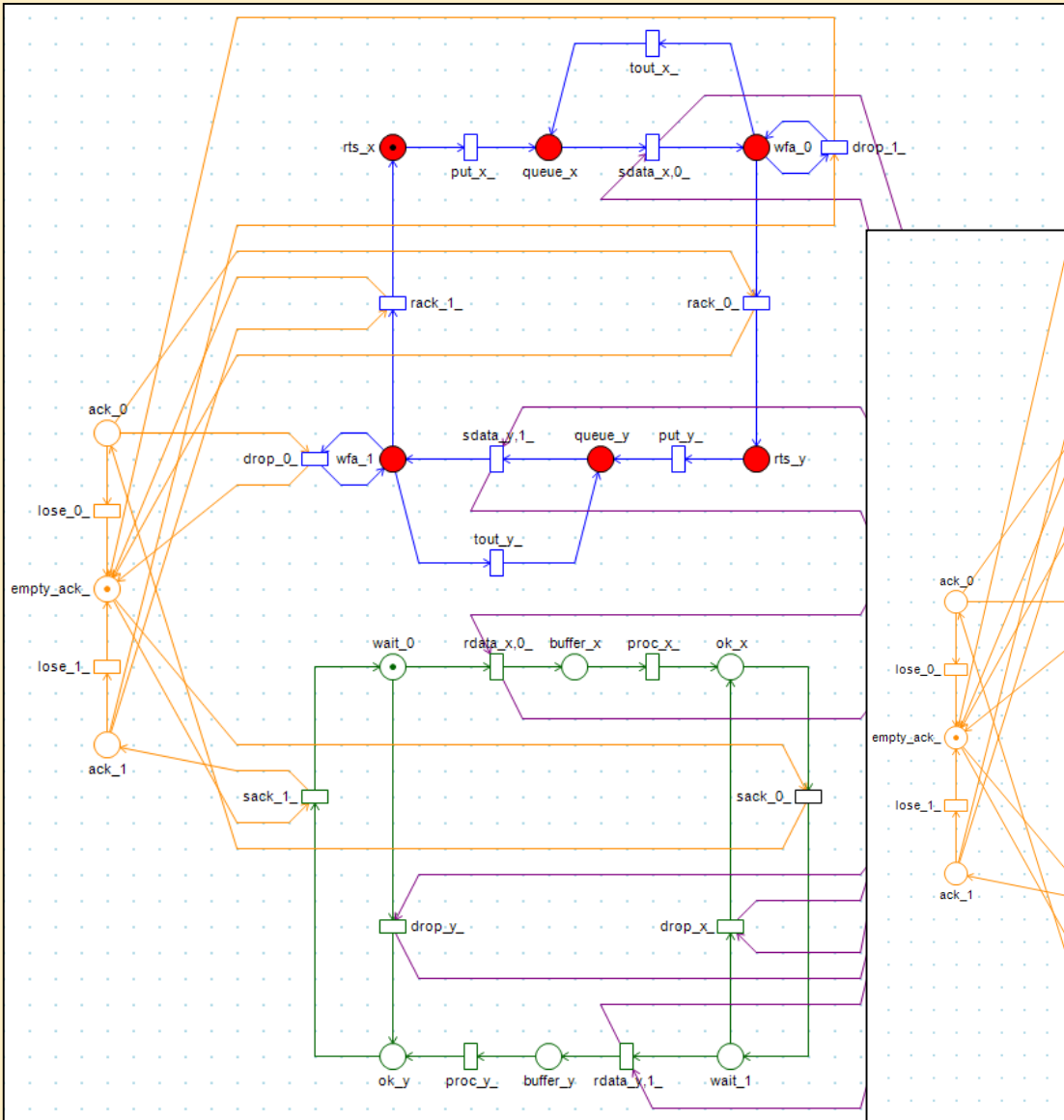
- Háló tulajdonságai**: The main window, containing two **ShowInvariants** sub-windows. The top one shows the expression `{lose(x), sdata(x,0), tout(x)}` and the bottom one shows `{ack_0, ack_1, empty(ack)}`.
- P-Invariants**: A dialog box titled "List of P-Invariants calculated by Martinez-Silva algorithm". It reports "Calculation finished in 0,00 ms. (places=18, transitions=22)" and lists the following invariants:

```
{ack_0, ack_1, empty(ack)}
{data_x, empty(data), data_y}
{rts_x, queue_x, wfa_0, rts_y, wfa_1, queue_y}
{wait_0, buffer_x, ok_x, ok_y, buffer_y, wait_1}
```
- T-Invariants**: A dialog box titled "List of T-Invariants calculated by Martinez-Silva algorithm". It reports "Calculation finished in 15,60 ms. (places=18, transitions=22)" and lists 22 invariants, including:

```
{lose(x), sdata(x,0), tout(x)}
{lose(y), sdata(y,1), tout(y)}
{rack(1), put(x), sdata(x,0), rack(0), sdata(y,1), put(y), drop(y), sack(0), drop(x), sack(1)}
{lose(1), sdata(y,1), tout(y), drop(y), sack(1)}
{drop(1), sdata(y,1), tout(y), drop(y), sack(1)}
{lose(y), rack(1), put(x), sdata(x,0), rack(0), sdata(y,1), put(y), tout(y), drop(y), sack(0), drop(x), sack(1)}
{lose(0), sdata(x,0), tout(x), sack(0), drop(x)}
{sdata(x,0), tout(x), drop(0), sack(0), drop(x)}
{lose(x), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), drop(y), sack(0), drop(x), sack(1)}
{lose(x), lose(y), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), tout(y), drop(y), sack(0), drop(x), sack(1)}
{rack(1), put(x), sdata(x,0), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), sack(0), proc(y), rdata(y,1), sack(1)}
{lose(x), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), sack(0), proc(y), rdata(y,1), sack(1)}
{rack(1), put(x), sdata(x,0), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), drop(y), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{lose(0), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), drop(0), rdata(x,0), proc(x), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{lose(x), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), drop(y), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{lose(x), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), rdata(x,0), proc(x), drop(y), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{lose(x), lose(y), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), tout(y), rdata(x,0), proc(x), drop(y), sack(0), drop(x), proc(y), rdata(y,1), sack(1)}
{lose(x), lose(1), rack(1), put(x), sdata(x,0), tout(x), rack(0), sdata(y,1), put(y), tout(y), rdata(x,0),
```


PetriDotNet: P-invariants (examples)

State machines
of components



PetriDotNet: T-invariants (examples)

Cyclic behavior (here: correct, data loss)

