Tutorial Notes

Petri Net Based Software Dependability Engineering

Monika Heiner

Brandenburg University of Technology
Department of Computer Science

Karl - Marx - Straße 17
D - 03013 Cottbus
Germany

mh@informatik.tu-cottbus.de
phone: (+ 49 - 355) 69 - 3884
fax: (+ 49 - 355) 69 - 3830

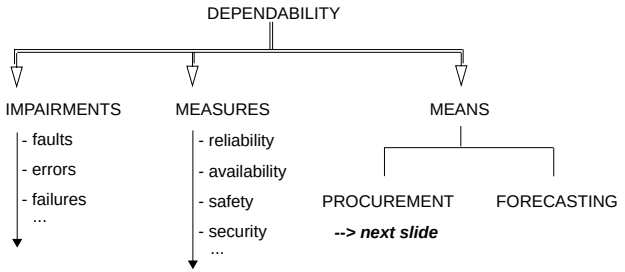
Outline:

1. Dependability terminology

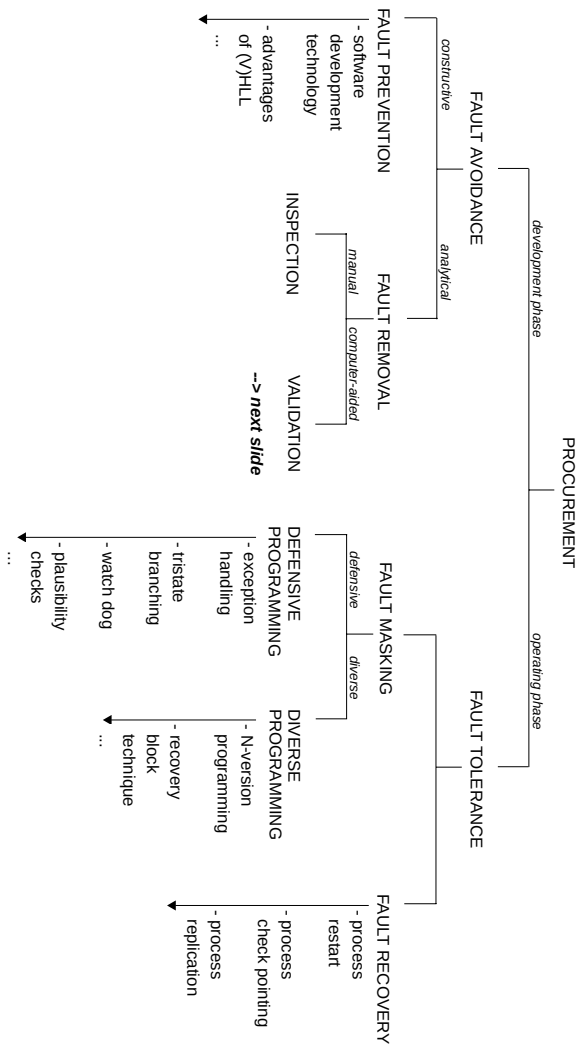
Dependability:

* ability of a system
  to fulfill its predefined task
  (in spite of any hardware and/or software faults)

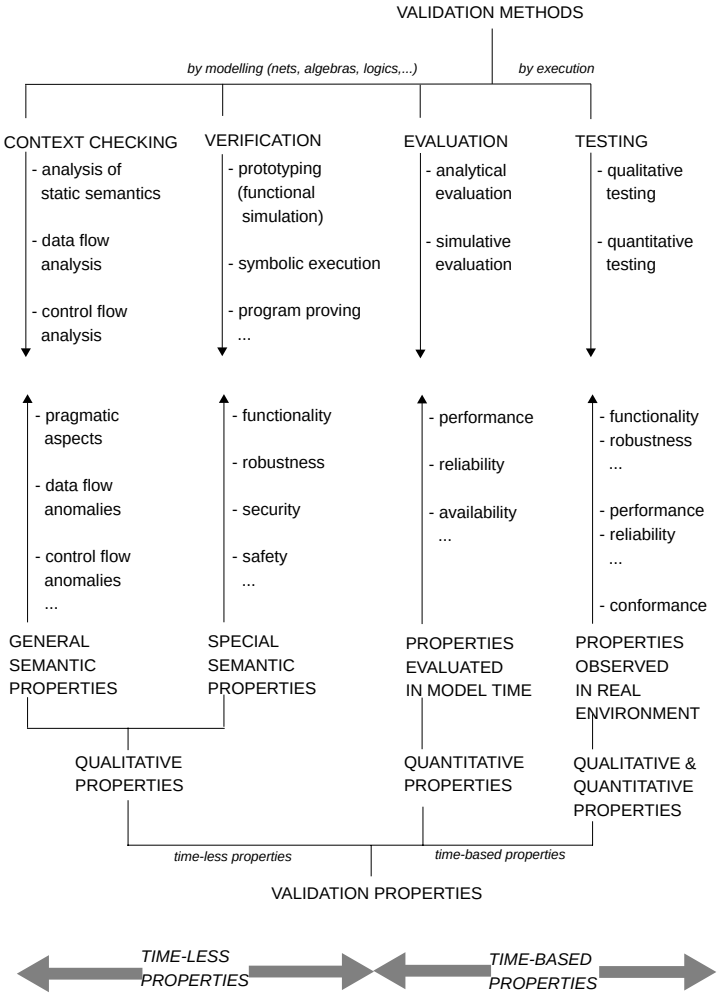
Taxonomy of dependable computing: [Laprie 95]



Means of software dependability procurement:



Computer-aided software validation techniques:



Summary:

* Validation methods can be classified according to the properties they aim at.

* The different validation methods do not compete, but complement each other.

* A recommended order of validation methods takes into account that

  ** validation should be applied as early as possible,

  ** the proper functionality is a prerequisite for an evaluation of quantitative properties, and

  ** the expected functionality can only be guaranteed in any case if all consistency conditions of context checking have been fulfilled.

* A thourough software validation is expensive and requires an adequate mathematical foundation.

* *Software validation:*
  waste of money
  or
  worthwhile investment?

Procurement / forecasting by

* dependability modelling

  ⇨ Which kind of models?

  ⇨ Where do the models come from?

* engineer's basic principle:
  Keep everything as simple as possible!

  ⇨ dedicated models
  for different kinds of properties;

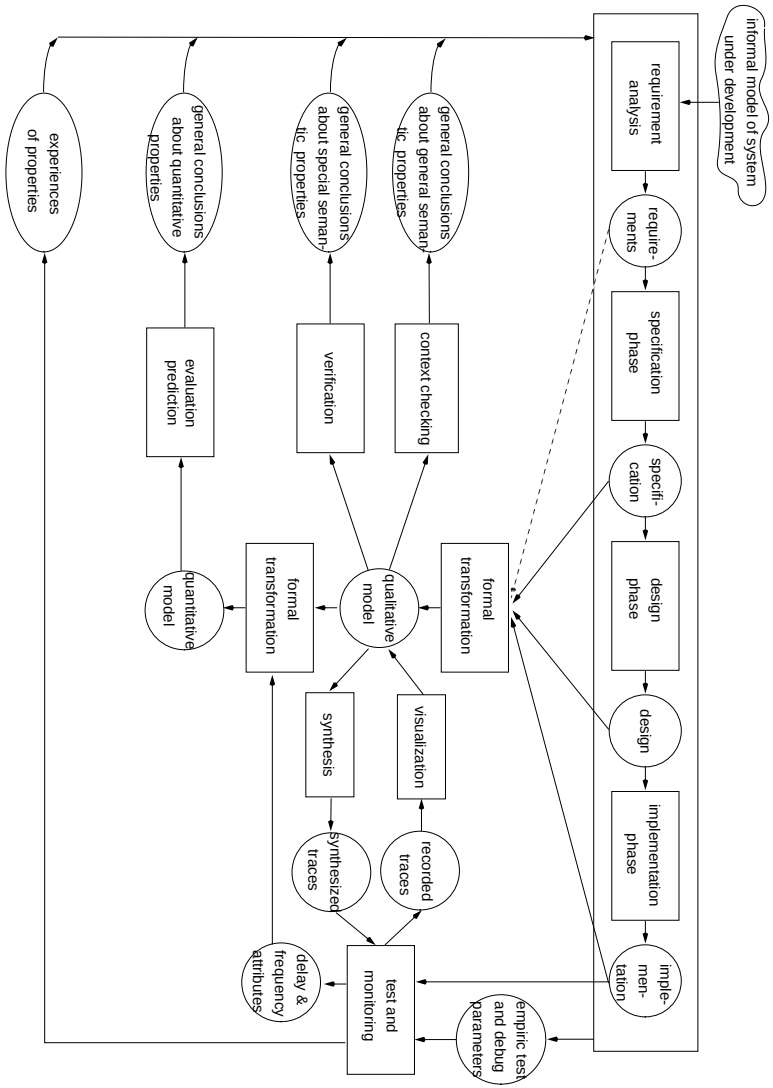
2. Petri net approach

Why Petri nets?

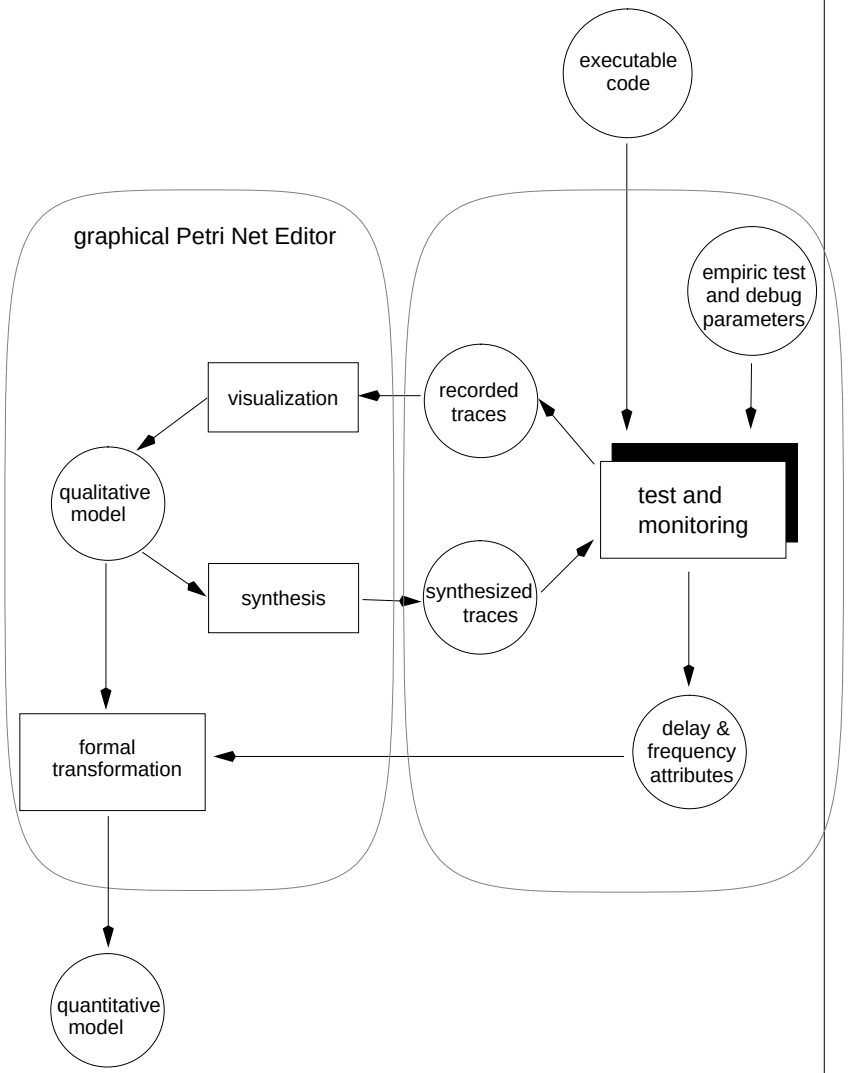
* a suitable intermediate representation for
  ** different (specification/programming) languages,
  ** different phases of software development cycle,
  ** different validation methods;
* modelling power
  ** true concurrency semantics
  ** applicable on any abstraction level
  ** specification of limited resources possible
* analyzing power
  (not restricted to reachability graph)

BUT: modelling power <-> analyzing power

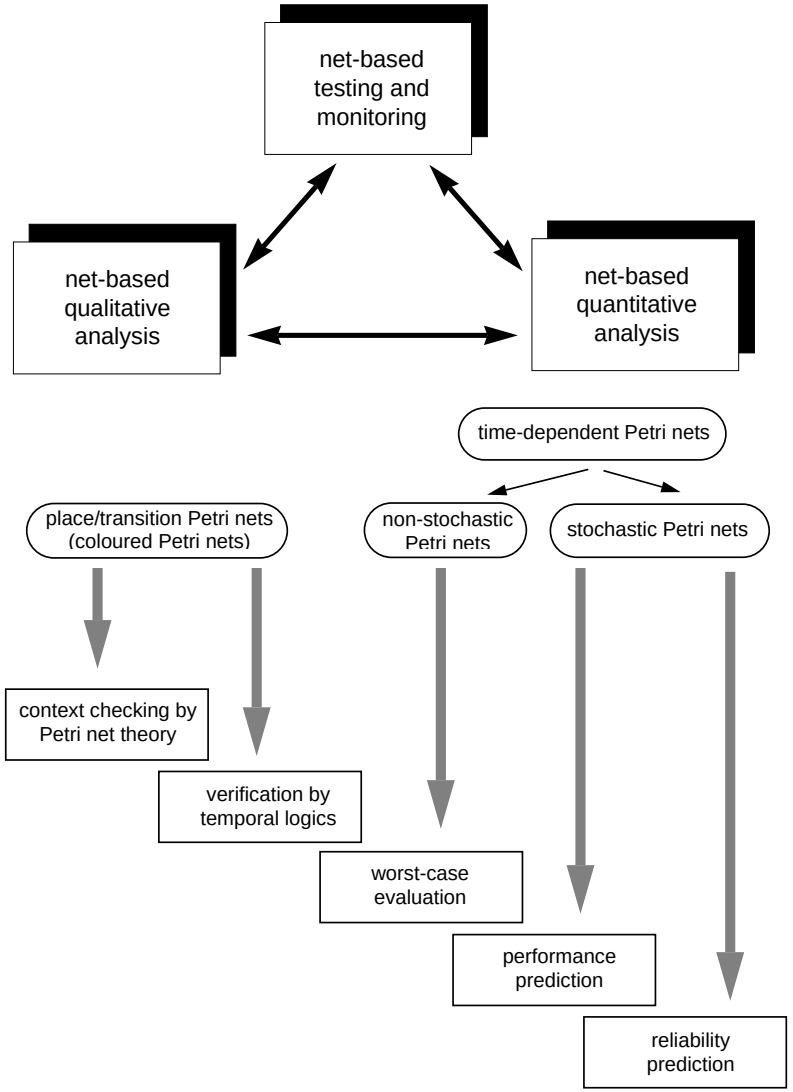
Logical architecture of a net-based tool kit:



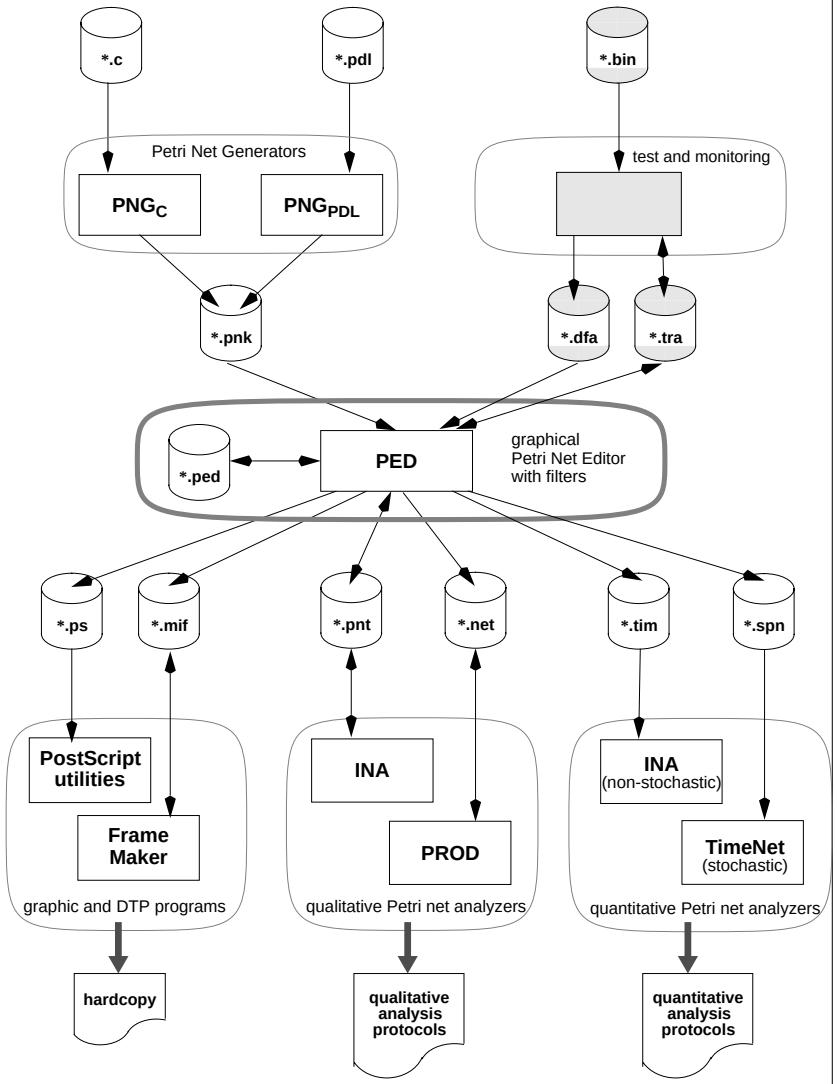
The role of net based testing and monitoring:



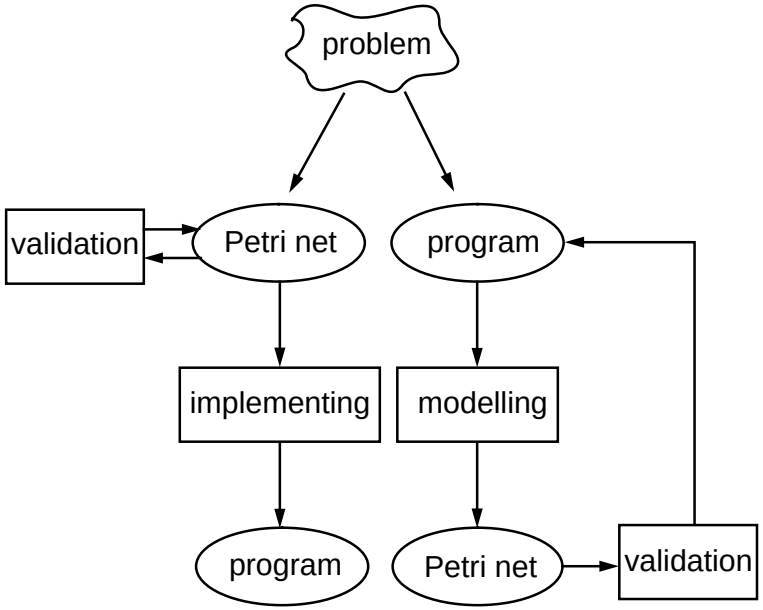
The Petri net approach to Integrate Qualitative and Quantitative Analysis:



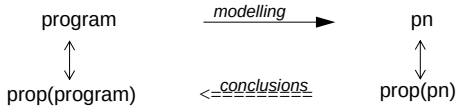
Physical architecture of a net based tool kit:



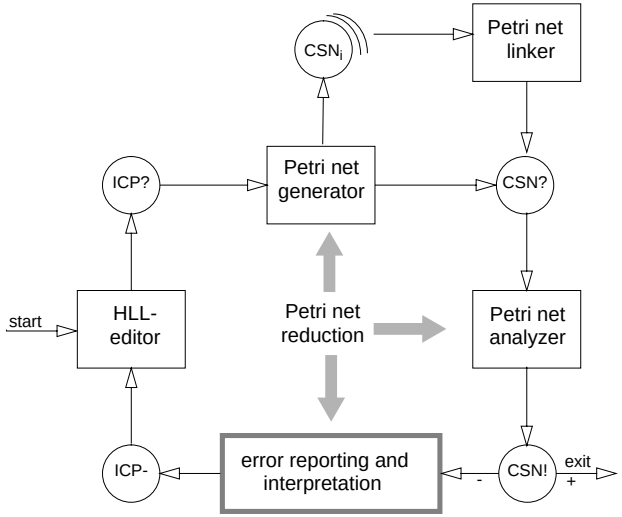
Software engineering & Petri nets:



Basic principle of Petri net based software validation:



Petri net framework:



ICP - Interacting Concurrent Processes
CSN - Control Structure Net

3.
Modelling
with Petri nets

Petri net generator:

* preconditions:
  ** dedicated language constructs
     for all process interactions
  ** (quasi-) static amount of processes
  ** no run-time dependencies
     (like recursion, dynamic loops)
* generated granularity:
  ** coarse-grained control structure
     (communication skeleton,
     purely sequential program parts -> transition)
  ** fine-grained control structure
     (statement structure,
     each statement -> transition)
* support of cross referencing
  between program and net structure:
  node names with source text line numbers
* automatic layout of a sequential process' structure
* program complexity measure:
  Number of Acyclic Paths - NAP
  (number of structurally possible paths)

Step-wise modelling:

* according system structure

  ** each process separately

  ** composition of process system
     interacting concurrent processes (ICP)

* according process structure

  ** control structure model (csm)
     as place/transition net

  ** control flow model (cfm)
     as coloured net

     cfm = csm + control variables

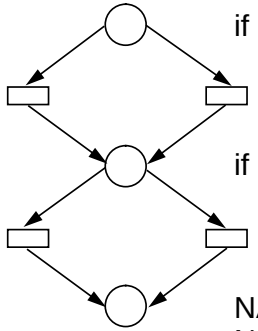
  ** data flow model

Model abstractions by (place/transition) Petri nets:

program -> finite place/transition nets

PN:
firing without time
general firing rule
single step

TPN:
firing duration
earliest firing rule
max step



$NAP_S = 4$
$NAP_D = 2$

$R_{TPN} \subseteq R_{PN} \supseteq R_D$

$\exists$-properties: $\overline{prop}$ (pn) -> $\overline{prop}$ (program)
$\forall$-properties: prop (pn) <- prop (program)

The csm grammar of reference language:

process ::= process_id@ ":" **process**
statement_sequence
#process process_id@ .

statement_sequence ::= statement_sequence ";" statement
| statement .

statement ::= if_statement
| case_statement
| loop_statement
| jump_statement
| send_statement
| receive_statement
| wait_statement
| simple_statement@ .

if_statement ::= **if** if_expression@
then statement_sequence
[**else** statement_sequence]
#if .

case_statement ::= **case** case_expression@ **of**
case_label@ ":" statement_sequence
{ "**|**" case_label@ ":" statement_sequence }*
[**default** ":" statement_sequence]
#case .

loop_statement ::= [loop_ label@ **":"**]
loop [loop_expression@]
statement_sequence
#loop [loop_ label@] .

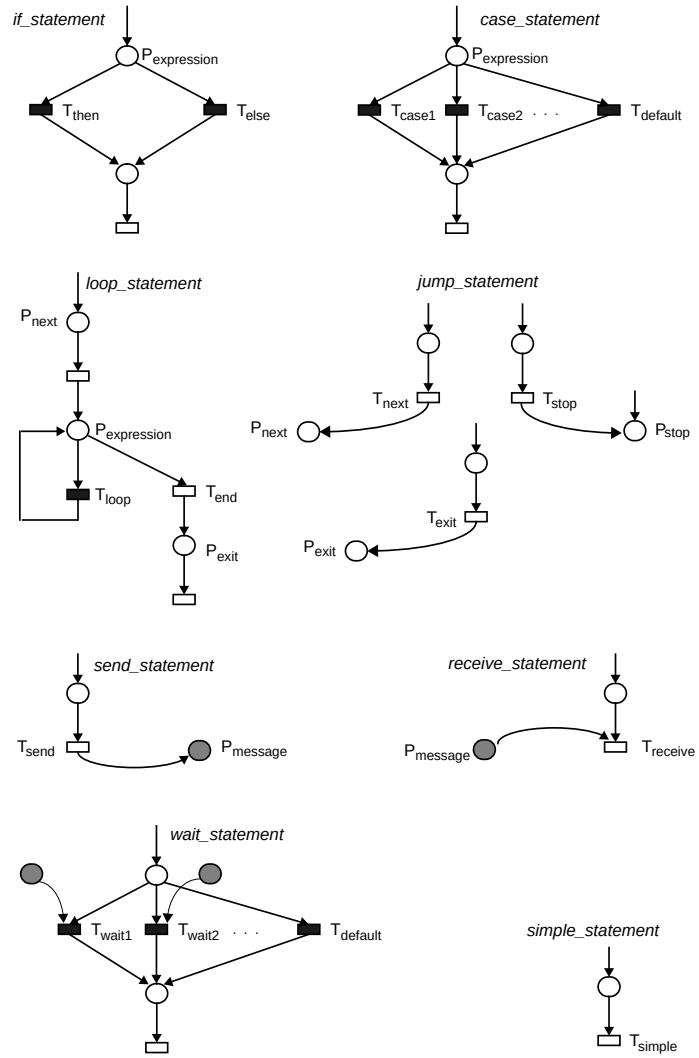
jump_statement ::= **next** loop_label@
| **exit** loop_label@
| **stop** .

send_statement ::= **send** message@ [**to** process_id@] .

receive_statement ::= **receive** message@ [**from** process_id@] .

wait_statement ::= **wait**
message@ [**from** process-id@] ":" statement_sequence
{ "|" message@ [**from** process-id@] ":" statement_sequence }*
#wait .

Petri net components for csm grammar of reference language:



Sequential process' control structure model (without data dependencies):

* pure
* ordinary
  ↷ homogeneous
* conservative
  ↷ structurally bounded
* marked with exactly one token
  ↷ safe (1-bounded)
* all static conflicts are dynamically realizable
* SM,
  SCSM (if only structured goto's)

No zero test in place/transition nets!

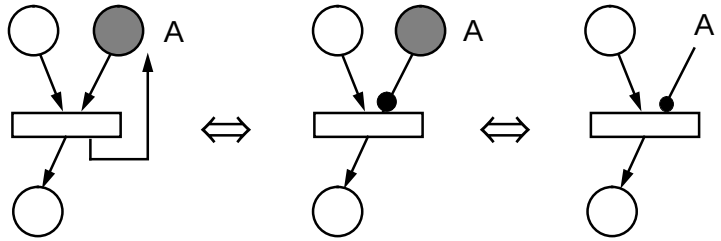
→ wayout for finite, discrete data types:

each variable is modelled by as many places as there are values in the type,

e. g. for a Boolean variable P (initializid with true)

⊙ P ○ / P

→ notation agreement for test edge:



A - connector (fusion node)

The cfm grammar of reference language (in addition to csm grammar):

case_expression@$_0$ :: = Bool_expression
| ordinal_expression@.

loop_expression@$_0$ :: = Bool_expression
| ordinal_expression@ .

Bool_expression :: = Bool_operand
| **not** Bool_expression
| Bool_expression **or** Bool_operand
| Bool_expression **and** Bool_operand
| Bool_expression “=“ Bool_operand
| Bool_expression “/=“ Bool_operand .

Bool_operand :: = Bool_denotation
| Bool_variable
| “(“ Bool_expression “)” .

Bool_denotation :: = **true**
| **false** .

Bool_variable :: = identifier@ { of type Boolean } .

statement :: = Bool_declaration
| Bool_assignation .

Bool_declaration :: = **Bool** Bool_variable “:=” Bool_denotation .

Bool_assignation :: = Bool_variable “:=” Bool_expression .

Remarks to cfm grammar:

* Each declaration of a Boolean variable encloses its initialization!

* Grammar rules defining the same metanotion are additive.

* The order of declarative and applied occurrence of each Boolean variable has to follow the usual data flow restrictions of static semantics.

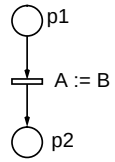
* Boolean expressions can be of any complexity.

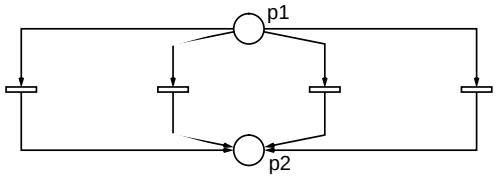
Declaration of a Boolean variable A:

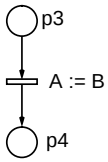
* two places A and /A are generated;

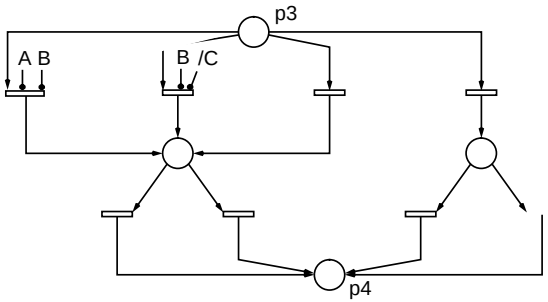
* initial marking according to the given initialization of the variable;

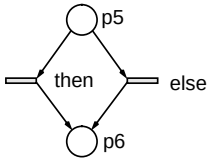
Examples of place/transition net components for Boolean operations:

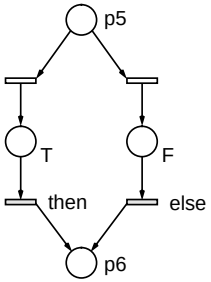












Coloured Petri nets as short-hand notation for place/transition Petri nets:

(Trivial) Folding of place/transition net produced by reduced (0) grammar:

- $\forall$ node: colour (node) = {dot}
- $\forall$ edge: Identity function

Folding of Boolean operations:

Folding of places

- ***Bool*** V := denotation $\rightarrow$ $^{V}\bigcirc$ colour (V) ={T, F}

  A Boolean place is always marked with T xor F.

- ***true*** $\rightarrow$ (T) true
- ***false*** $\rightarrow$ (F) false

The Boolean denotations are modelled by Boolean places, always marked with T resp F.

(-> evaluation of Boolean denotation is a special case of evaluation of Boolean variable)

Folding of transitions:

* all transitions of one Boolean operation (representing the different possible value combinations) are folded to one transition with colour set
  - {T, F}, if 1 pre-Boolean place
  - {T, F} x {T, F}, if 2 pre-Boolean places

* edges: function of type
  Colour (T_B) ∈ {T, F} (no bugs)

  e.g. Identity I
  (I + 1) (t_B) = (t_B mod 2) + 1 (= not)

(x, y) \ f	F (first)	S (econd)	OP			
			or	and	equal	un-equal
(T, T)	T	T	T	T	T	F
(T, F)	T	F	T	F	F	T
(F, T)	F	T	T	F	F	T
(F, F)	F	F	F	F	T	F
					F + S + 1	F + S

Sequential process' control flow model (with data dependencies):

* not pure

* ordinary
  - ⇝ homogenous

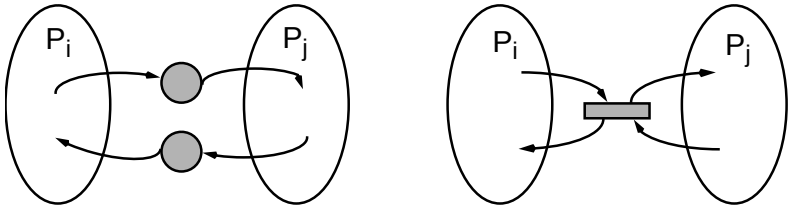
* conservative
  - ⇝ structurally bounded

* not marked with exactly one token
  - one process counter
  - one token for each pair of Boolean places
  - ⇝ safe (1-bounded)

* no dynamic conflicts
  - ⇝ max outdegree of a reachability graph node = 1

* not ES

Basic principles of process connection:



ICP

ICP - Interacting Concurrent Processes

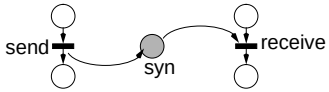
Classification of language constructs for process communication:

PROCESS COMMUNICATION

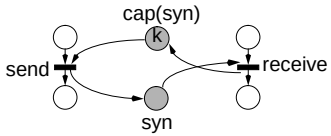
- SYNCHRONIZATION (of sender)
  - asynchronous — *(no-wait-send, the general case requires infinite buffer)*
  - semi-asynchronous — *(delay, if finite buffer full)*
  - synchronous — *(delay until message has been received)*
  - remote invocation — *(delay until a response has been given)*
  - rendezvous — *(delay until message has been exchanged, no buffering, direct transfer)*
- ADDRESSING
  - direct — *(one-to-one communication: sender and receiver know each other)*
  - semi-direct-by-sender — *(many-to-one communication: only the sender knows the receiver, not vice versa)*
  - semi-direct-by-receiver — *(one-to-many communication: only the receiver knows the sender, not vice versa)*
  - indirect — *(many-to-many communication: via common global objects like channels, mail boxes, monitors)*
- WAITING (of receiver)
  - deterministic — *(the choice of the message to receive occurs independently from the progress of neighbouring processes)*
  - non-deterministic — *(receiving is influenced by the available messages provided by neighbouring processes)*

Petri net components for process synchronization:

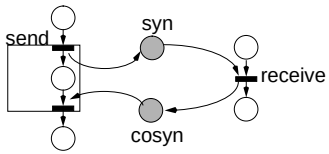
asynchronous
(possibly unbounded)



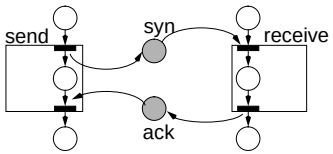
semi-asynchronous
(k-bounded,
locally conservative)



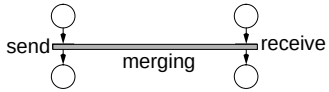
synchronous
(safe,
globally conservative)



remote invocation
(safe,
globally conservative)

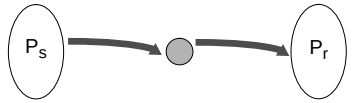


rendezvous
(safe,
locally conservative)

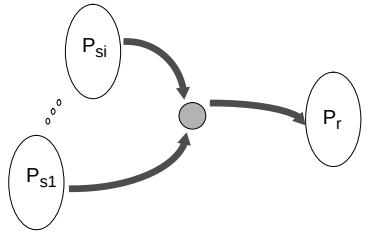


Petri net components for process addressing:

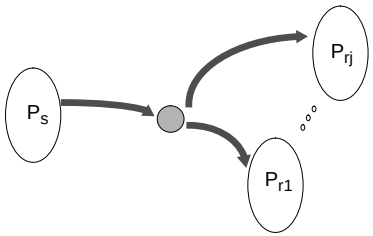
direct
(only
static channel conflicts)



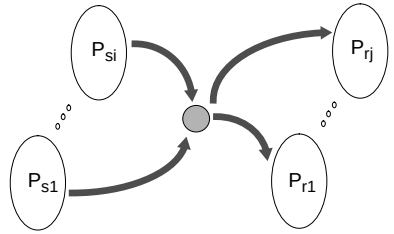
semi-direct-by-sender
(only
static channel conflicts)



semi-direct-by-receiver
(dynamic channel conflicts
possible)



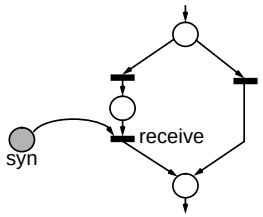
indirect
(dynamic channel conflicts
possible)



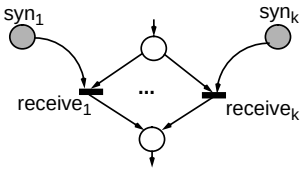
The pre-process P_s can send from different control points, and the post-process P_r can receive from different control points.

Petri net components for process waiting:

deterministic
(confusion impossible)
pure control flow conflicts)



non-deterministic
(confusion possible)



Modell of communicating sequential processes:

* ordinary
  - homogeneous
* safe (1-bounded), if only
  synchronous/remote invocation/rendezvous communication
* bounded, if also
  semi-asynchronous communication

Simplified view[1] of the influence of communication patterns on net structure class:

addressing waiting\	direct / semi-direct-by-sender	indirect / semi-direct-by-receiver
determininistic	EFC	ES
non-deterministic	ES	CSM

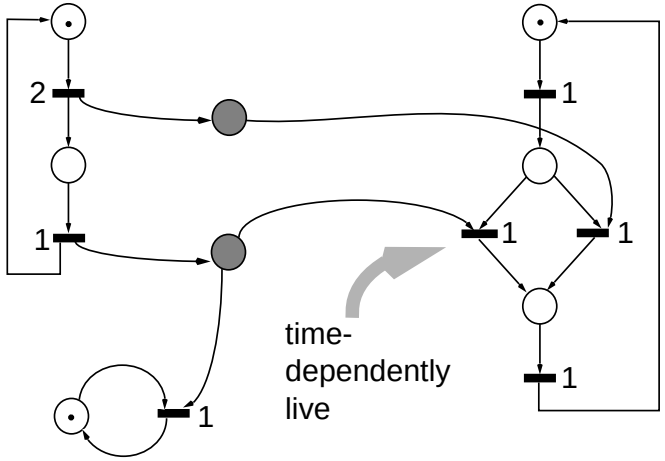
1. Provided, pre- and postprocesses do not access the same communication object from different control points.

☐ Known to be time-independently live [Starke 90], i.e. a live net remains live under any constant delay timing (duration net).

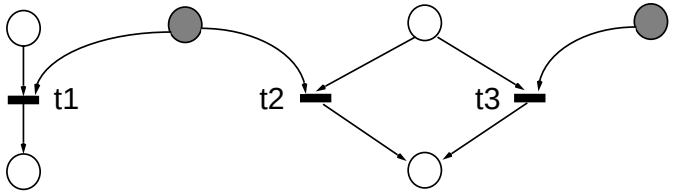
The influence of communication patterns on conflict structures:

\addressing waiting	direct / semi-direct-by-sender	indirect / semi-direct-by-receive
deterministic	no dynamic channel conflicts	channel and control flow conflicts appear only separately
non-deterministic		confusing combination of channel and control flow conflicts possible

Example of a time-dependently live (duration) CSN:

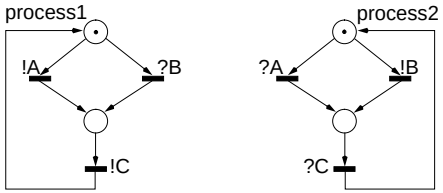


Confusing combination of channel and control flow conflict:

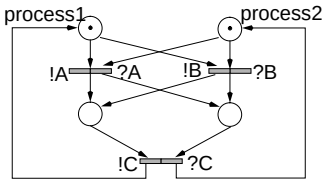


A simple protocol in three variants [Diaz 82]:

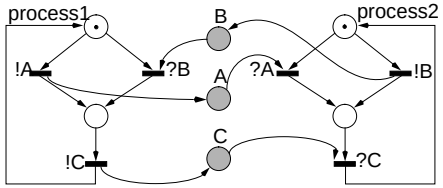
(1)
given
original
process
patterns



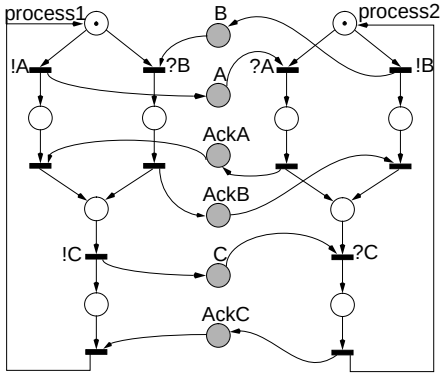
(1a)
rendezvous:
EFC,
safe,
live



(1b)
asynchronous:
ES,
unbounded,
live

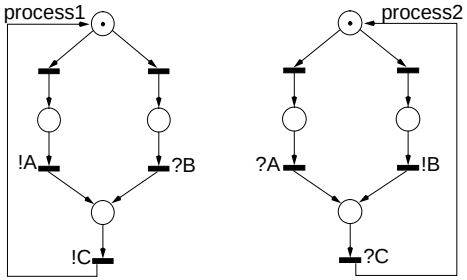


(1c)
remote invocation:
ES,
safe,
not live

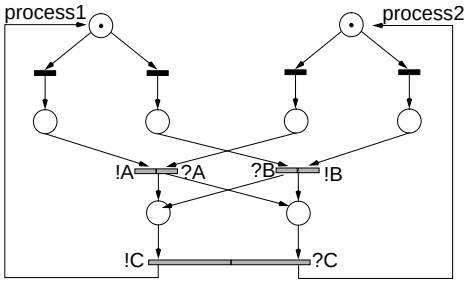


A modified simple protocol in three variants:

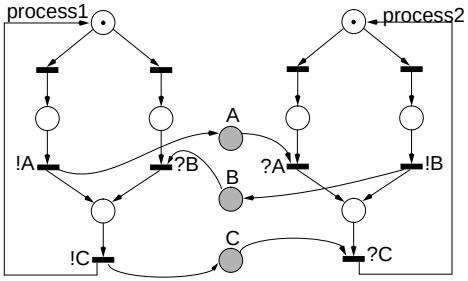
(2)
given
modified
process
patterns



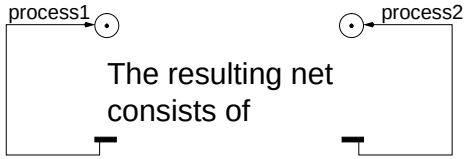
(2a)
rendezvous:
(E)FC,
safe,
not live



(2b)
asynchronous:
(E)FC,
unbounded,
not live

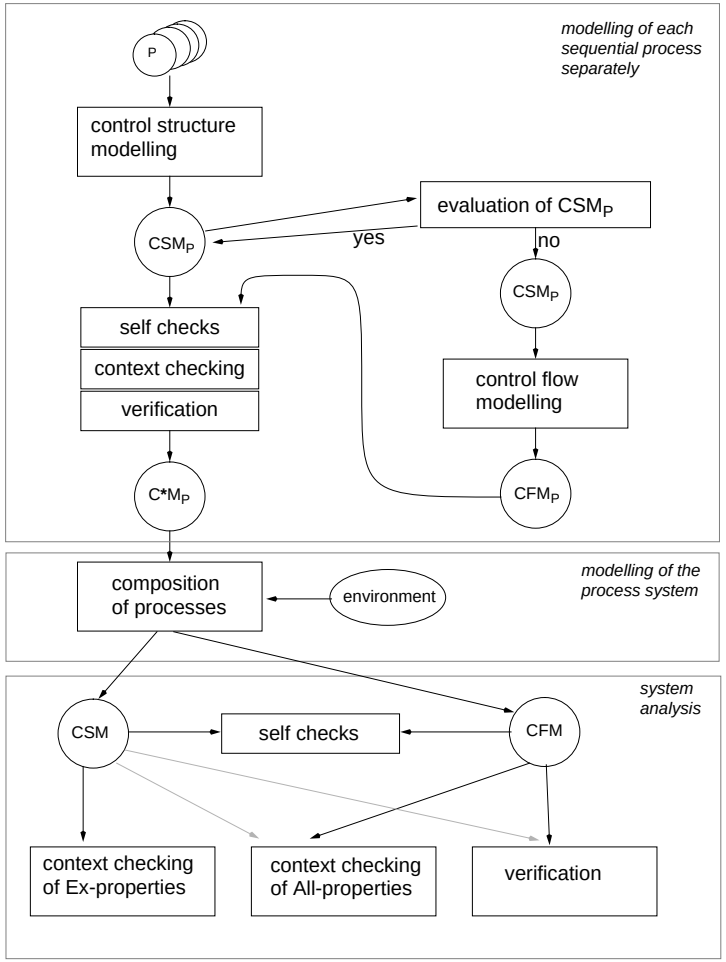


(2c)
remote invocation:
(E)FC,
safe,
not live



4.
Qualitative analyzing with Petri nets

Methodology overview:



CSM - Control Structure Model
CFM - Control Flow Model

Qualitative properties:

structural properties;

especially valuable:
local(ly decidable) structural properties;

certain combinations of structural properties
allow conclusions to
behavioural properties;

behavioural properties
see below;

Petri Net Properties - Overview:

1. Simple Structure Properties

ORD	ordinary *(1-multiplicity of all arcs)*
HOM	homogeneous *(all output arcs of a given place have the same multiplicity)*
NBM	non-blocking multiplicity *(for each place applies: MIN multiplicity of input arcs >= MAX multiplicity of output arcs)*
PUR	pure *(no side conditions)*
CSV	conservative *(any firing preserves token amount)*
SCF	static conflict free
CON	connected
SC	strongly connected
Ft0	there is a transition without pre-place
tF0	there is a transition without post-place
Fp0	there is a place without pre-transition
pF0	there is a place without post-transition
MG	marked graph *(synchronization graph)*
SM	state machine
FC	free choice net
EFC	extended free choice net
ES	extended simple net

2. More Expensive Structure Properties

DTP deadlock trap property

SMC state machine coverable
(covered with SM components)

SMD state machine decomposable
(covered with SCSM components)

SMA state machine allocatable

CPI covered with place invariants

CTI covered with transition invariants

SB structurally bounded

3. Behaviour Properties

B bounded

REV reversible *(the initial state m_0 can be reached again from all reachable states: home state)*

DSt dead states *(a state where no transition is enabled)*

BSt bad states *(a state where a fact is enabled)*

DTr dead transitions *(at the initial state)*

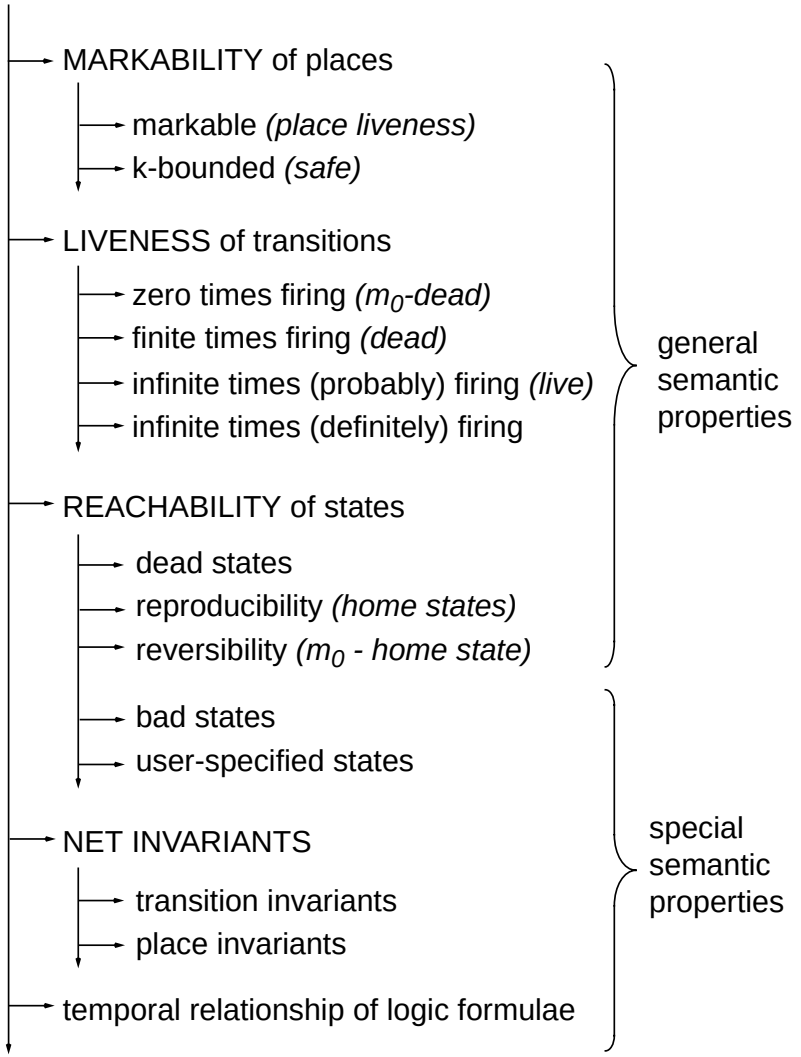
DCF dynamically conflict free

L live

LV live, excepted transitions dead at the initial marking
(live, excepted implicit facts)

L&S live & safe *(1-bounded)*

Behavioural net properties:



Software-oriented interpretation of net properties:

* **Dead code:**

  statements which will never be executed;

  pn: the corresponding transition never fires (dead at the initial marking);

  rg: transition does not appear at any edge;

* **Total deadlock:**

  the arrival in some system state from which there is no exit;

  pn: dead marking;

  rg: final nodes (sheets);

* **Partial deadlock**:

  not all parts of the system are available for all times;

  pn: there are no dead markings, but dead transition(s);

  rg: not all strongly connected components contain all transitions;

* **Well-structuredness:**

  all parts of the system may be executed for ever;

  pn: the net ist live;

  rg: all strongly connected components contain all transitions;

* **Livelock**:

  parts of the system may be blocked for ever (due to the scheduler‘s strategy or something else not contained in the model);

  pn: live, but not livelock-free;

  rg: not all circles contain all transitions;

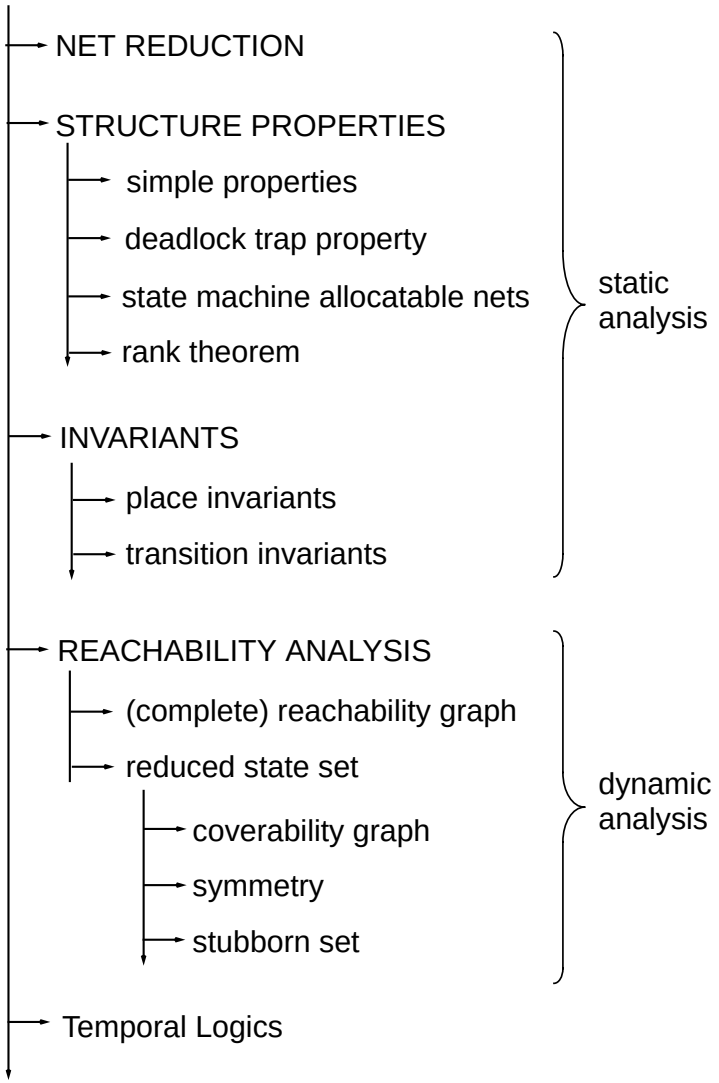
* **Fault tolerance and self-synchronization**:

  after a failure or from any abnormal state, the software will return to normal execution (recovery from failure) within finite time;

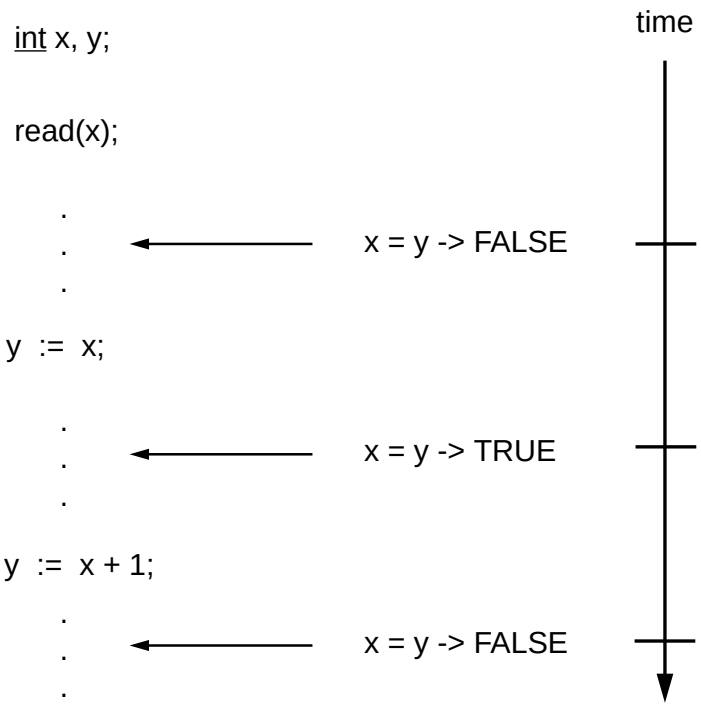
  pn: reproducibility / reversibility;

  rg: from any state, the home state (initial state) is reachable again;

Qualitative analysis methods:



Temporal logics, introduction:



Temporal logics, overview:

semantics / time	interleaving	true concurrency
linear	traces (no conflict, no concurrency) Manna & Pnueli, Kröger	runs (no conflict, but concurrency) Reisig
branching	reachability graph (conflict & concurrency not distinguishable) Emmerson, Clarke, PROD	branching processes (conflicts & concurrency) PEP?

interleaving semantics (PROD):

linear OPERATOR
~ branching „OnAllBranch“ OPERATOR

CTL operators:

E on some branch
A on all branches

X next
F eventually
G henceforth
U until

○ ◇ □ U

EX NextOnSomeBranch (f)

AX NextOnAllBranches (f)

EF EventuallyOnSomeBranch (f)

AF EventuallyOnAllBranches (f)

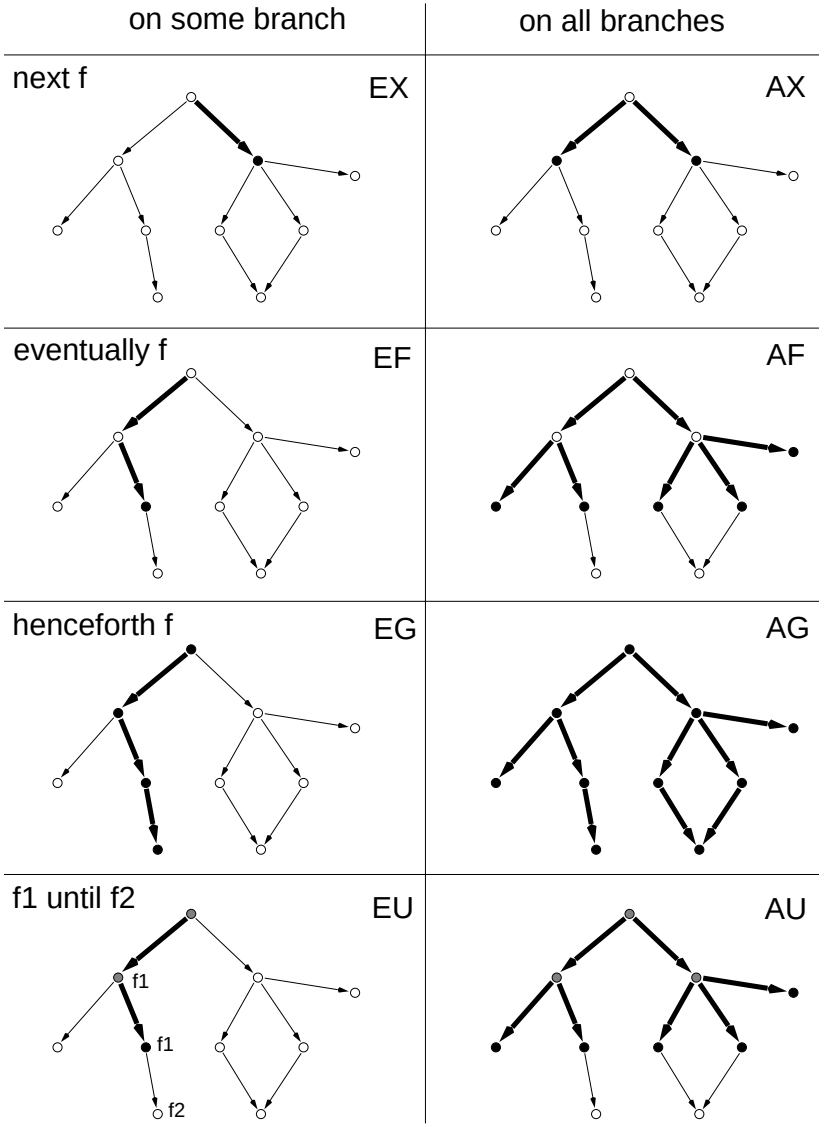
EG HenceforthOnSomeBranch (f)

AG HenceforthOnAllBranch

EU UntilOnSomeBranch (f1, f2)

AU UntilOnAllBranches (f1, f2)

CTL operators (cont.):



Examples (A):

(1) query node card (mutex) > 1

*(**mutex:** all rg nodes where more than one process are executing the mutex region)*

(2) query node ((card (messages) == n) && \
(card (acks) == n))

*(**alternating bit protocol:** all rg nodes where both* FIFO's are full)

(3) query node \
Not (Or (And (red1 = = < . 1 . >, \
not (yellow1 = = < . 1 . >)),\
And (red2 = = < . 1 . >, \
not (yellow2 = = < . 1 . >))))

*(**traffic lights:** is there any state where not at least one of the (both) traffic lights shows pure red)*

Examples (B):

{ All / One } acyclic paths

- from the current node
- to all nodes satisfying the expression

query { bpath (true, false) / dpath (true, false) / bspan (true) / dspan (true, false) } < formula > { mute / verbose / super-verbose }

(1) query bspan (true) $0

*(**philosophers:** determine one path from the current node to the terminal node in $0)*

(2) query bspan (true) not step true

***(philosophers:** is it possible, starting from the current node, to reach eventually a node without successors)*

Examples (C):

(1) difference between
sequential / parallel producer consumer system

if the producer is „ready to deliver“ (rd)
and at least one of the buffer cells is empty,
then the producer can immediately go by one step
to „ready to produce“ (rp);

$$\Box\ (rd \wedge\ (e1 \vee e2\ ) \rightarrow \bigcirc\ rp\ )$$

```
query node HenceforthOnAllBranches          \

        (If Then                            \
            (And  (rd = = < . 1 . >,        \
                        Or  (  e1 = = < . 1 . >,    \
                               e2 = = < . 1 . >     \
                            )                       \
                    ),                              \
                    step rp = = < . 1 . >           \
            )                                       \
        )
```

-> requires true concurrency semantics!

Examples (D), typical questions case study:

(1) a channel cannot be free and full at the same time:

$$\Box\ (\neg\,(ch-free \wedge ch-full)\ )\ )$$

(2) dependent input/output behaviour (belts):

$$\Box\ (transporting \rightarrow \neg\,(chCFfree \vee chCFfull \vee chFTfree \vee chFTfull\ ))$$

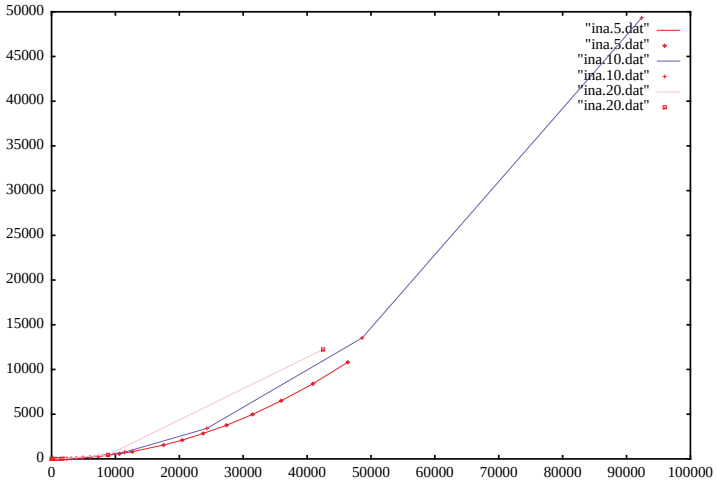
(3) the arms cannot be active at the same time:

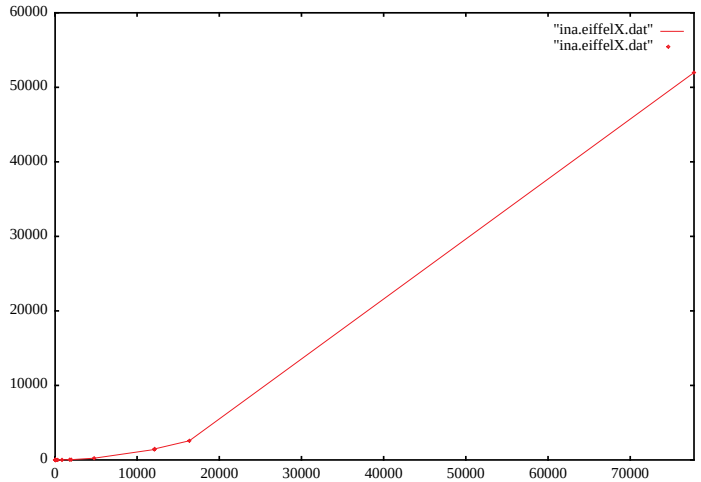
$$\Box\ (\neg\,(A1loading \wedge A2loading \vee A1loading \wedge A2unloading \vee A1unloading \wedge A2loading \vee A1unloading \wedge A2unloading\ ))$$

(4) Is it possible that both arms hold a plate
at the same time:

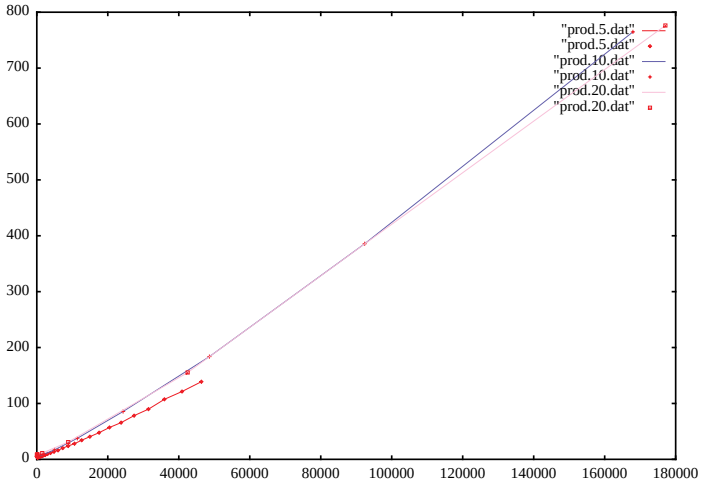
$$\Diamond\ (A1storing \wedge A2storing)$$

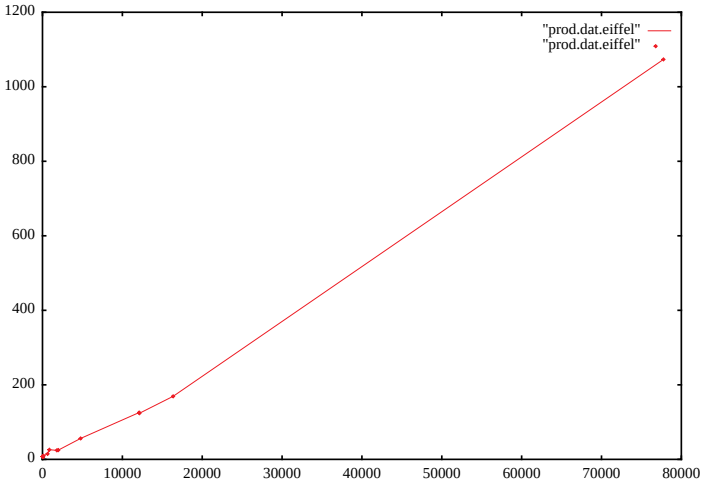
Analysis efforts - INA



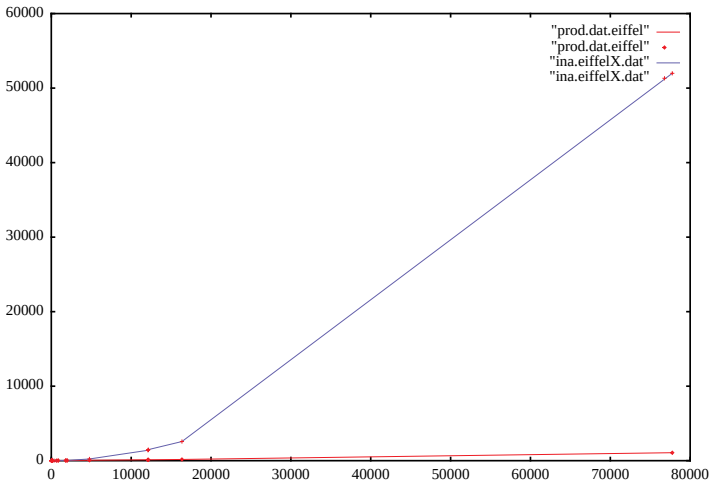


Analysis efforts - PROD





Analysis efforts - INA- PROD comparison:



Wayout?

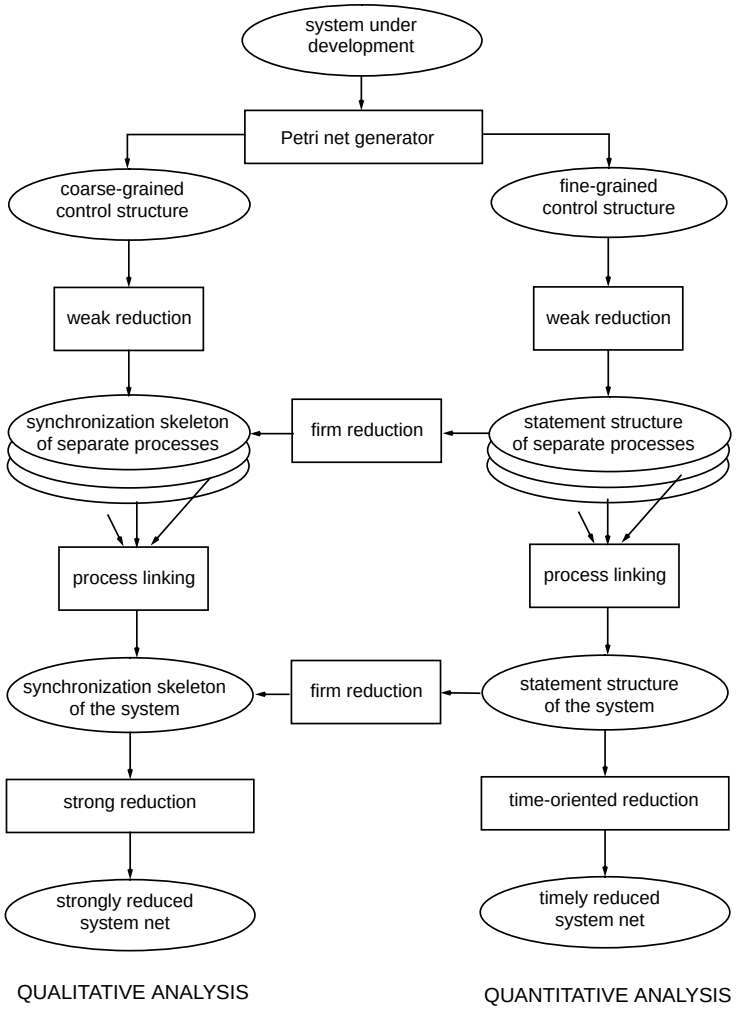
* on the fly verification of
  'next time'-free linear time temporal logic
  via stubborn set reduced reachability graph [PROD 95];

* very fast for small formula:
  splitting of a big question into a lot of small questions;

  e.g. the net is LIVE iff:

5.
Quantitative analyzing with Petri nets

Where does the model come from?



Which kind of time model?

* atomic program parts assigned to transitions
  - time assigned to transitions

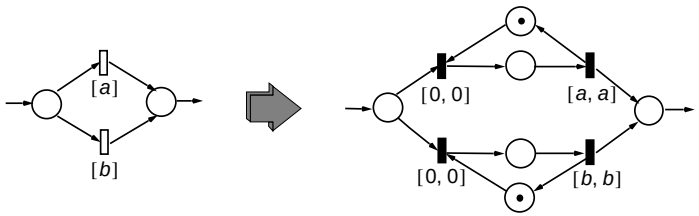
* as simple as possible
  - duration nets (timed nets) ?

* adequate characterization of time consumption
  (alternatives, iterations)
  - interval nets (time nets) ?

* structural simplicity, e. g. alternative as

duration net
(with token reservation)
(constant delay)
(firing consumes time)

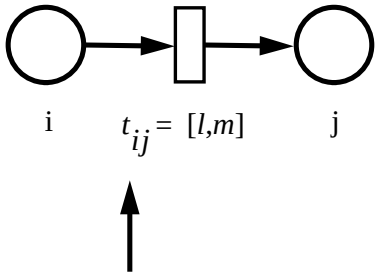
interval net
(no token reservation)
(interval times)
(firing itself timeless)



- duration interval net!
  (with token reservation,
  interval delay)

Input parameters (worst-case evaluation):

*

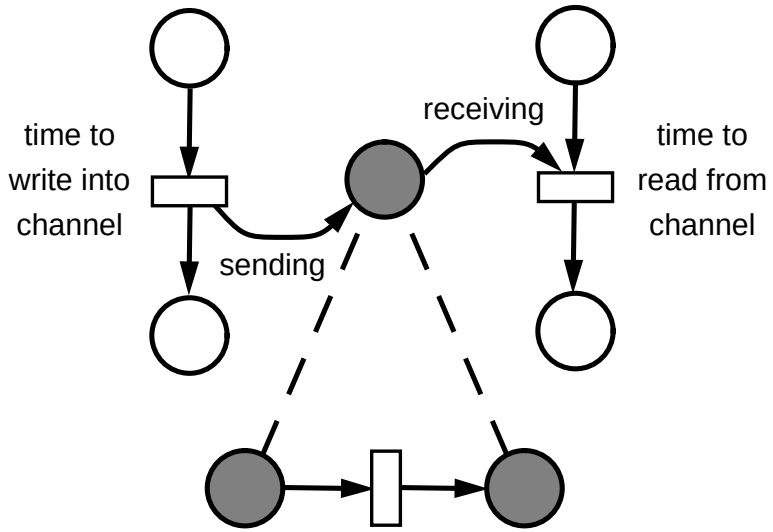


time consumption of sequential program parts
- at least **l** time units
- at most **m** time units
- or any (continuous) time in between

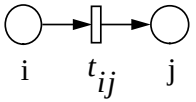
measured by monitoring OR
calculated from computer instructions

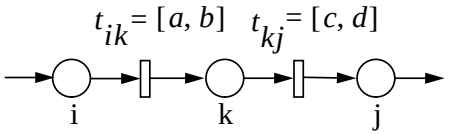
* no explicit
branching probabilities

Communication time model:

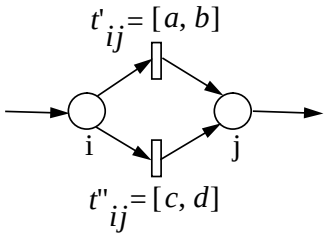


Structural compression within sequential parts (worst-case evaluation):

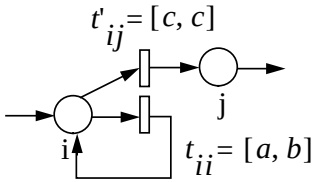




$$t_{ij} = [a + c, b + d]$$



$$t_{ij} = [min(a, c), max(b, d)]$$

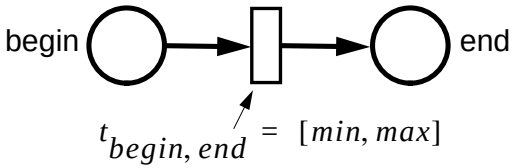


$$t_{ij} = [m \cdot a + c, n \cdot b + c]$$

assumption:

$\left\{ \begin{matrix} \text{lower} \\ \text{upper} \end{matrix} \right\}$ bound $\left\{ \begin{matrix} m \\ n \end{matrix} \right\}$ of iterations given

Output parameters (worst-case evaluation):



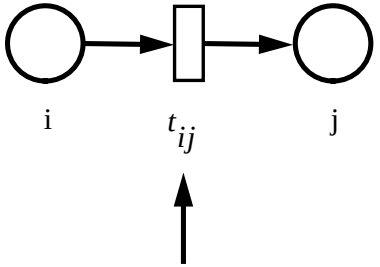
* min execution duration
  (shortest path),
  max execution duration
  (largest path);

* esp. valuable for systems
  which require predictably timing behaviour
  (to meet given deadlines);

* calculations can be based on
  discrete reachability graph
  (only integer states);

Input parameters (performance):

*



time consumption of sequential program parts
given by
- time duration (average time),
- time interval (average time interval) or
- probability distribution

calculated from computer instructions,
measured by monitoring,
OR
just estimated;

* maybe branching probabilities

Symbol conventions (performance):

transition types

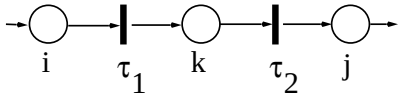
- immediate transition (zero time)
- transition with constant delay
- transition with (modified) geometric delay
- transition with DDP-distributed delay (Defective Discrete Phase [Ciardo 95])

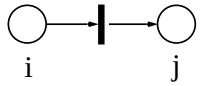
Note:

all transition types with token reservation
(in opposite to
stochastic Petri nets like GSPN and DSPN)

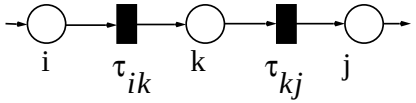
Structural compression within sequential parts (performance):

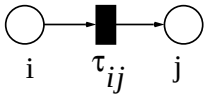
(any sequence of) immediate transitions





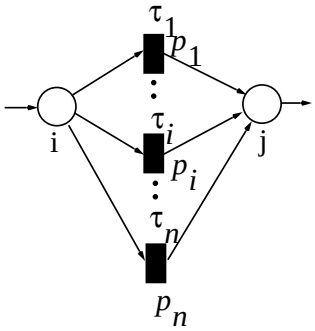
(fixed number of) constant delays

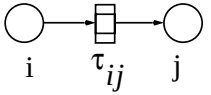




$$\tau_{ij} = \tau_{ik} + \tau_{kj}$$

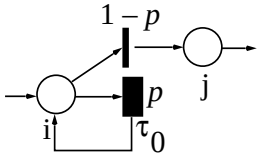
delay with a fixed number of alternatively chosen constant delays

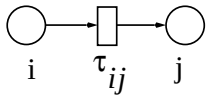




$$P\left(\tau_{ij} = \tau_i\right) = p_i$$

modified geometric delay
(geometrically distributed number of identical constant phases decreased by one phase)

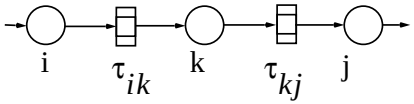


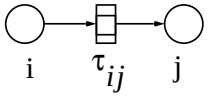


$$P\left(\tau_{ij} = n\tau_0\right) = (1-p)\,p^n$$

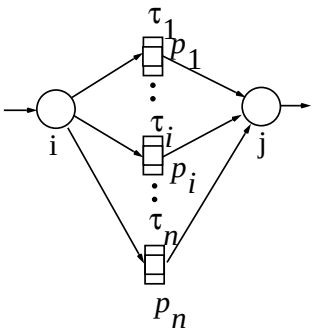
Structural compression within sequential parts (performance) cont:

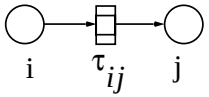
sequence
of DDP-distributed delays



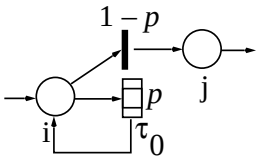


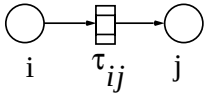
alternative
of DDP-distributed delays





iteration
of DDP-distributed delays





Output parameters (performance):

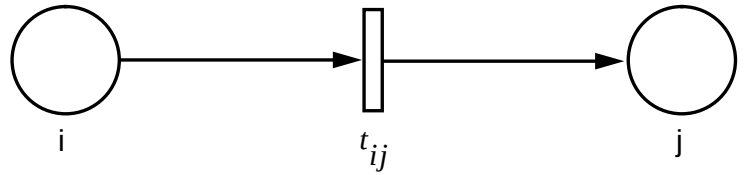
typical questions:

* average utilization of a machine
  (average number of firings of a transition);

* average number of parts, processed per time unit
  (average number of tokens
  at steady-state in a certain place);

* average steady-state delay
  spent in traversing a subnetwork;

Input parameters (reliability):



probability of entering path (i,j)

e.g.
* equal for program branches
* measured

probability of successfully executing path (i,j) given that program has entered path (i,j)

e.g.
* proportion of bugs in the total program

$$\left(\frac{\text{amount of errors}}{\text{lines of code}}\right)$$

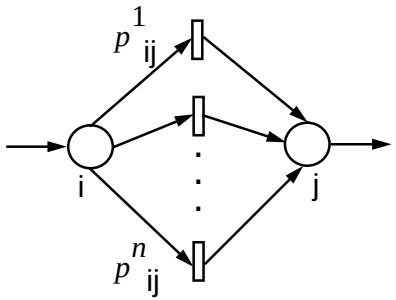
* evaluated by testing programs
  - of the same characteristics and size
  - developed by the same team

p_{ij}: joint probability of entering AND executing the path successfully

Input parameters (reliability) cont.:

* (place, transition) - arcs

p_{ij}: probability of entering the path and executing the path successfully



in failure free case:

$$\sum_{k=1}^{n} p^k_{ij} = 1$$

in case of failure

$$\sum_{k=1}^{n} p^k_{ij} < 1$$

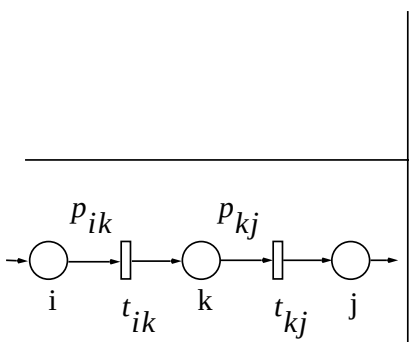
probability of unsuccessfully executing path (i,j)

$$1 - \sum_{k=1}^{n} p^k_{ij}$$

* transitions:

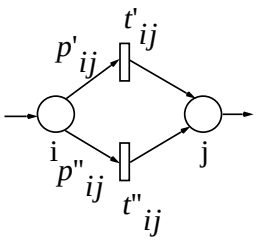
t_{ij}: (average) time to execute path (i,j)

Structural compression within sequential parts (reliability):



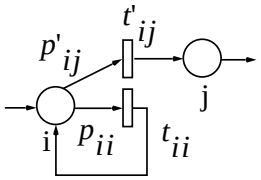
$$p_{ij} = p_{ik} \cdot p_{kj}$$

$$t_{ij} = t_{ik} + t_{kj}$$



$$p_{ij} = p'_{ij} + p''_{ij}$$

$$t_{ij} = \frac{p'_{ij}\ t'_{ij} + p''_{ij}\ t''_{ij}}{p'_{ij} + p''_{ij}}$$



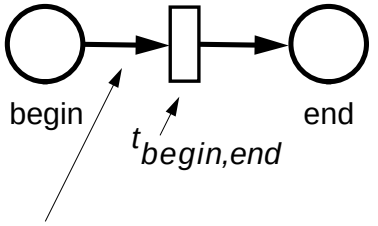
$$p_{ii} = \frac{p'_{ij}}{1 - p_{ii}}$$

$$t_{ij} = t'_{ij} + \frac{p_{ii} \cdot t_{ii}}{1 - p_{ii}}$$

Output parameters (reliability):

* **failure rate z:**
proportion of
components failed in a given time unit
in relation to
total amount of components still working error-free

* of a sequential program:



$$z_{begin,\,end} = \frac{1 - p_{begin,\,end}}{t_{begin,\,end}}$$

* of communicating sequential programs:

transformation into stochastic model

- branching probabilities
- deterministic firing delays

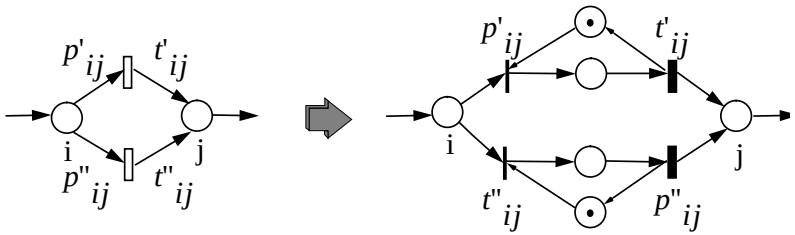
e.g. TimeNET

Transformation into stochastic model:

* immediate transitions for proper conflict solution

  (alternatives, iterations)

* weights (probabilities) for immediate transitions



| immediate transitions

▮ deterministically delayed, no token reservation

▯ deterministically delayed, with token reservation

Conclusions:

* Validation/reliability prediction requires time and resources;
  contradiction to “time to market” principle!

* compositional validation, structural induction;

* dream of the future:
  theory of structured programming
  (of sequential processes)
  extended to
  theory of structured programming of dependable interacting concurrent processes;

References:

[Ciardo 95]
Ciardo, G.:
Discrete-time Markovian Stochastic Petri Nets;
in: Stewart, W. J. (Ed.) Numerical Solution of Markov Chains '95, Raleigh, NC, January 1995.

[Clarke 86]
Clarke, E. M.; Emerson, E. A.; Sistla, A. P.:
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications;
ACM Trans. on Programming Languages and Systems 8(86)2, pp. 244-263.

[Diaz 82]
Diaz, M.:
Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models;
Computer Networks 6(82), pp. 419-441.

[German 94]
German, R.; Kelling, C.; Zimmermann, A.; Hommel, G.:
TimeNET - A Tool Kit for Evaluating Non-Markovian Stochastic Petri Nets;
Research Report 1994-19, TU Berlin, 1994.

[Heiner 92]
Heiner, M.:
Petri Net Based Software Validation - Prospects and Limitations;
Techn. Report ICSI Berkeley/CA, TR-92-022.

[Heiner 93]
Heiner, M.:
A Petri Net View of Process Communication;
GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme", Magdeburg, June 1992, in König, H. (ed.): , FOKUS-Reihe, Saur-Verlag 1993.

[Heiner 94a]
Heiner, M.; Ventre, G.; Wikarski, D.:
A Petri Net Based Methodology to Integrate Qualitative and Quantitative Analysis;
Information and Software Technology 36(94)7, 435-441.

[Heiner 94b]
Heiner, M.; Wikarski, D.:
An Approach to Petri Net Based Integration of Qualitative and Quantitative Analysis of Parallel Systems;
Techn. Report BTU Cottbus, I-09/1994.

[Heiner 95a]
Heiner, M.:
Petri Net Based Software Dependability Engineering;
Proc. RELECTRONIC '95, Budapest, Oct. 1995, pp. 181-186.

[Heiner 95b]
Heiner, M.:
Petri Net Based Software Dependability Engineering;
Tutorial Notes, Int. Symposium on Software Reliability Engineering, Toulouse, Oct. 1995.

[Heiner 95c]
Heiner, M.; Popova, L.:
Worst-case Analysis of Concurrent Systems with Duration Interval Petri nets;
submitted 1st Int. Workshop on PDSE, Berlin, March 1996.

[Heiner 95d]
Heiner, M.; Deussen, P.:
Petri Net Based Qualitative Analysis - A Case Study;
Techn. Report BTU Cottbus, 1995 (to appear).

[Laprie 95]
Laprie, J.-C.:
Dependability - Its Attributes, Impairments and Means;
in Randell, B.; Laprie, J.-C.; Kopetz, H.; Littlewood, B. (eds.): Predictably Dependable Computing Systems, Springer 1995, pp. 3-24.

[Popova 91]
Popova, L.:
On Time Petri Nets;
J. Information Processing and Cybernetics EIK 27(91)4, 227-244.

[Popova 95a]
Popova, L.:
An Algorithm for Computing the Shortest and the Largest Path in Time Petri Nets;
Informatik-Bericht, Humboldt-University zu Berlin, 1995 (to appear).

[Starke 90]
Starke, P. H.:
Analysis of Petri Net Models (in German);
B.G.Teubner 1990.

[Starke 92]
Starke, P. H.:
INA - Integrated Net Analyzer, Manual (in German);
Berlin 1992.

[Varpaaniemi 95]
Varpaaniemi, K.; Halme, J.; Hiekkanen, K.; Pyssysalo, T.:
PROD Reference Manual;
Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, August 1995.

Appendix: Case study

[Lewerentz 95]

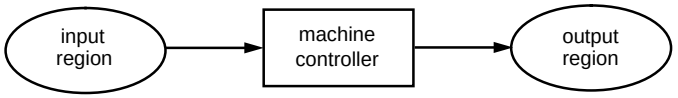
Eiffel approach:

[Casais 94a/b],
[Michaelis 93]

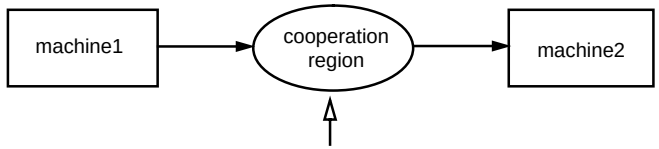
Basic design principles:

* production cell = pipeline of machines

* each machine
  - takes plates from some input places;
  - processes them;
  - puts plates on some output places;

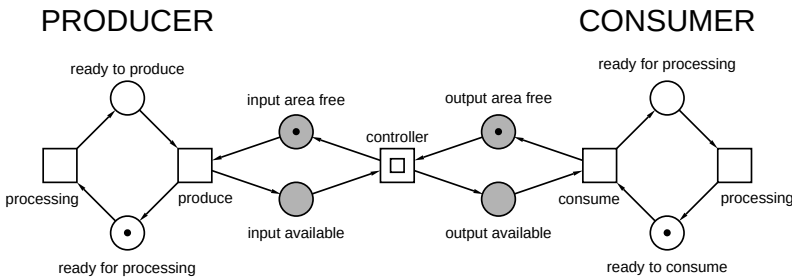


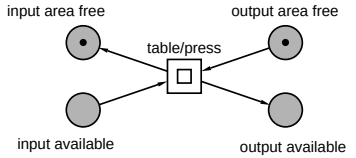
* cooperation region between two consecutive machines

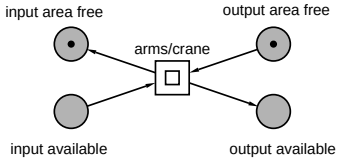


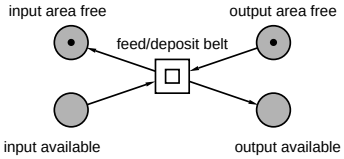
* mutual exclusive shared resources
  - robot swivel
    (to rotate both arms)
  - physical regions
    (intersection of trajectories of different machines)

Producer / consumer relation:









Three types of cooperation pattern:

(1) **table/press:**

the controller must always hold a lock on one of its cooperation regions;

(2) **arms/crane:**

step-wise synchronization with only one of its adjacent controllers;

(3) **belts:**

simultaneous control of input and output region necessary;

Three types of cooperation pattern (cont.):

table / press
(mutually exclusive input / output)

arms / crane
(independent input / output)

feed / deposit belt
(dependent input / output)

Three arm versions:

version1

version2

version3

Source text examples [Casais 94a,b]**:**

arm: procedure to take a plate

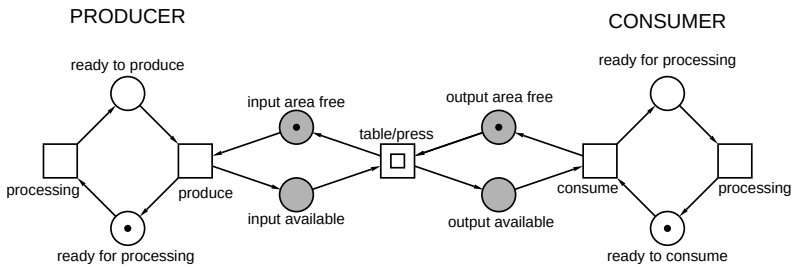
```
Take /* version2 */
    acquire locks on shared resources (swivel)
    acquire lock on input area
        move_arm_to_grasppos
        do_grasp
        go_in
    release lock on input area
    release locks on shared resources (swivel)

Take /* version3 */
    acquire lock on input area
    acquire locks on shared resources (swivel)
        move_arm_to_grasppos
        do_grasp
        go_in
    release lock on input area
    release locks on shared resources (swivel)
```

belt: procedure to transport a plate

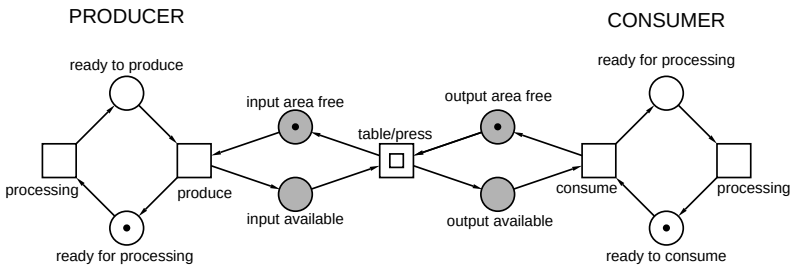
```
Transport
    acquire lock on input area
    acquire lock on output area
        transport
    release lock on input area
    release lock on output area
```


table / press: (mutually exclusive input / output) (with init part)



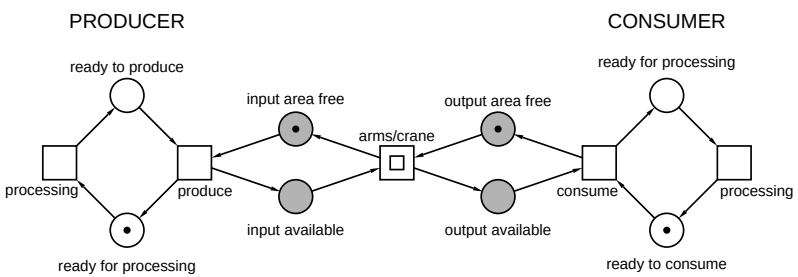
ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	N	Y	N	N	Y	N	N	N	Y	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
N	N	N	N	?	N	Y	?	N	N	N	N	Y	N	N	N	

table / press:
(mutually exclusive input / output)
(without init part)



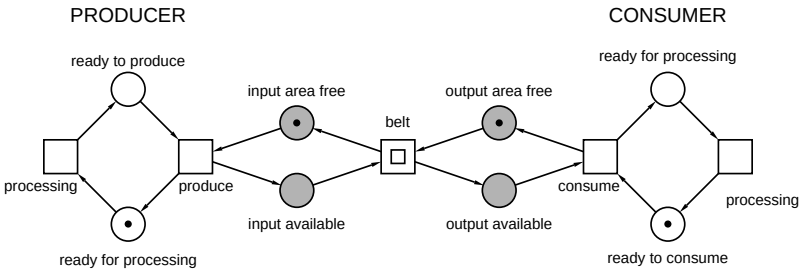
ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	Y	Y	Y	N	N	N	N	Y	N	Y	Y	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	

arms / crane:
(independent input / output)
(version 2 / 3)



ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	

feed / deposit belt: (dependent input / output)

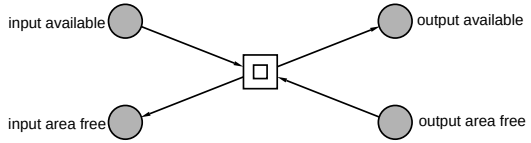


ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	Y	Y	Y	N	N	N	N	Y	N	Y	Y	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	

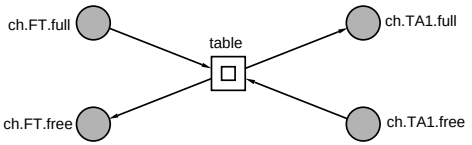
Instantiation of the analyzed controller patterns:

table / press instantiation

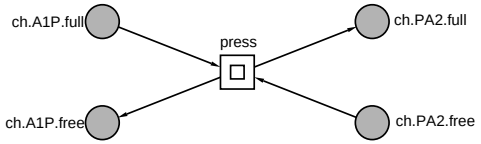
(mutually exclusive input /output)



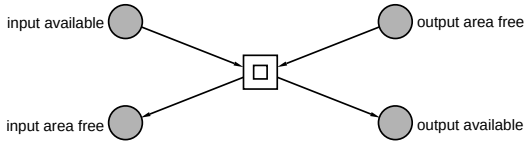
coarse structure of the table



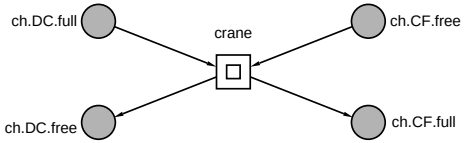
coarse structure of the press



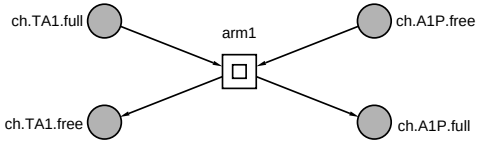
arms / crane instantiation
(independent input /output)



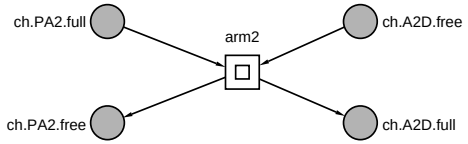
coarse structure of the crane



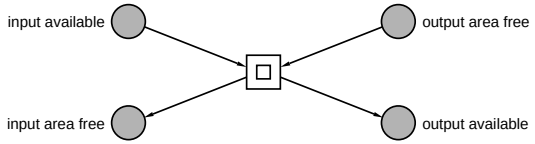
coarse structure of arm1



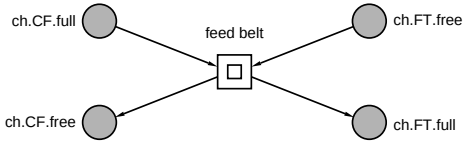
coarse structure of arm2



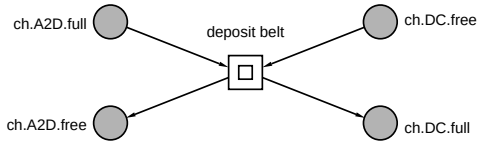
feed / deposit belt instantiation
(dependent input /output)



coarse structure of the feed belt

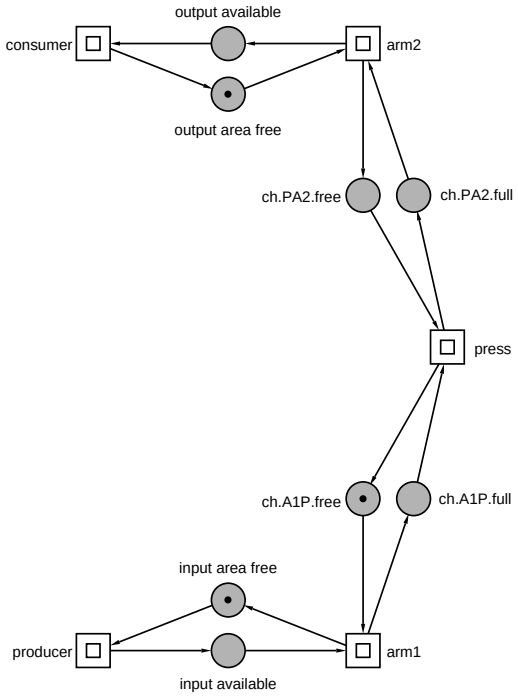


coarse structure of the deposit belt



Step-wise composition:

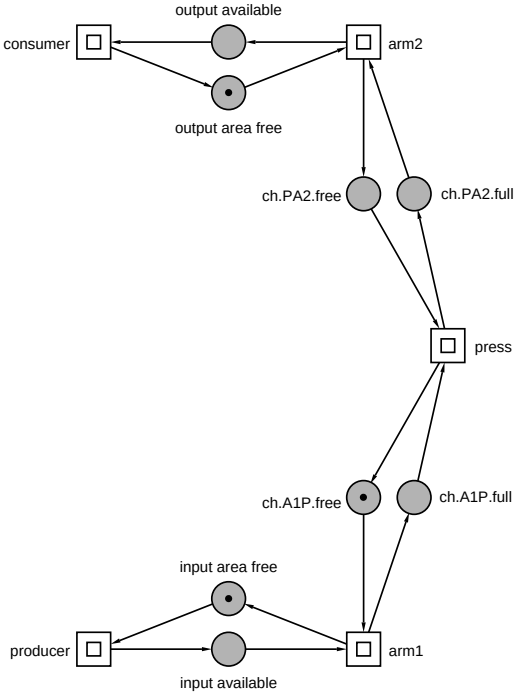
e.g. subsystem: arm1 - press - arm2 (arms: version2)



ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	N	Y	Y	Y	Y	N	Y	N	N	?	N	N	N	

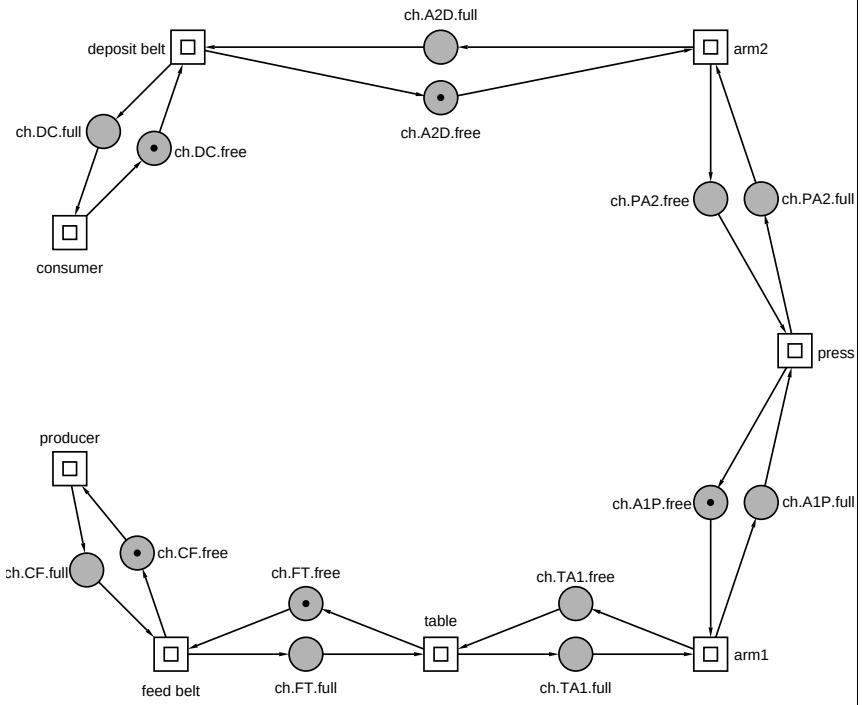
Step-wise composition:

e.g. subsystem: arm1 - press - arm2 (arms: version3)



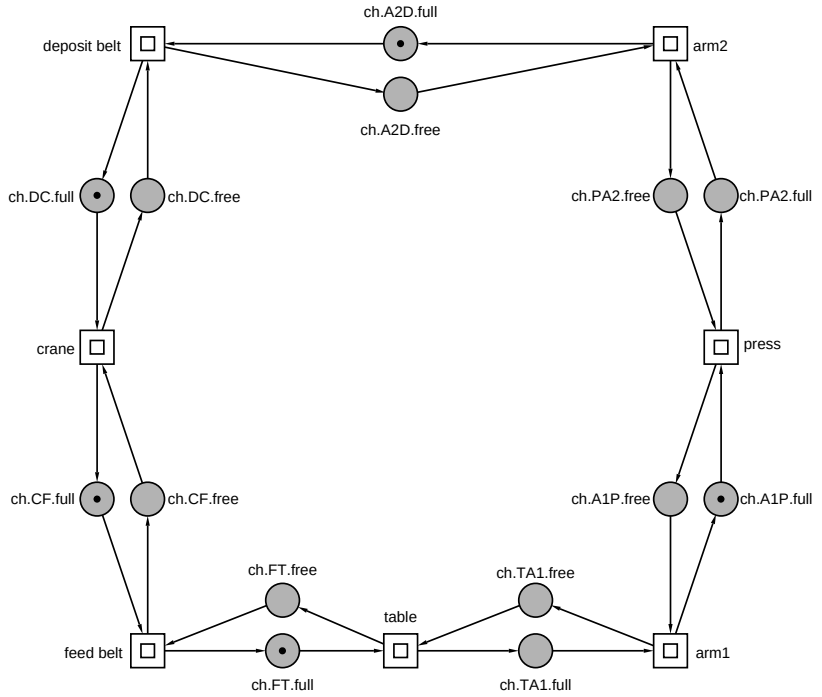
ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	

Coarse structure of the open system:



ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	

Coarse structure of the closed system:



ORD	HOM	NBM	PUR	CSV	SCF	CON	SC	Ft0	tF0	Fp0	pF0	MG	SM	FC	EFC	ES
Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	N	N	N	N	Y
DTP	SMC	SMD	SMA	CPI	CTI	B	SB	REV	DSt	BSt	DTr	DCF	L	LV	L&S	
Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	

Net sizes and analysis efforts done:

	places/ transitions	DTP[a]	R_{stub}[b]	R[b]
table / press with init part without init part	 13 / 9 12 / 8	 (N) 28	 12 8	 28 24
crane	12 / 8	31	11	48
arms version 1 version 2 version 3	 13 / 8 17 / 12 17 / 12	 38 109 88	 11 15 15	 48 112 96
belts	12 / 8	26	8	36
subsystem with arm version 1 arm version 2 arm version 3	 25 / 16 33 / 24 33 / 24	 175 3.851 (N) 725	 47 75 140	 640 1.984 1.800
open system	51 / 36	1.145	299	77.760[c]
closed system with 1 plate with 2 plates with 3 plates with 4 plates with 5 plates	51 / 36	1.140	 36 7 2 94 98 121	 864 4.776 12.102 16.362[d] 12.144[e]

a) processed candidates to check the Deadlock Trap Property
b) number of states generated
c) after about 8.5 h on PC 80486, 75 MHz
d) after about 45' computing time
e) after about 20' computing time

////// just for fun

Outlook:

* model refinement
  interactions with hardware interface
  (sensors, actors)

* analysis of safety reguirements
  by temporal logic

* quantitative analysis
  by different types of time-dependent Petri nets
  Duration Interval Nets,
  Stochastic Nets)

* synthesis of the control software

* fault tolerance

References:

[Casais 94a]
Casais, E.:
Eiffel; A Reusable Framework for Production Cells Developed with an Object-oriented Programming Language;
pp. 241-256.

[Casais 94b]
Casais, E.:
An Experiment in Framework Development;
pp. 95-124.

[Lewerentz 95]
Lewerentz, C.; Lindner, T.:
Formal Development of Reactive Systems - Case Study Production Cell;
LNCS 891, 1995.

[Michaelis 93]
Michaelis, M.:
Objektorientierte Modelleirung einer Fertigungszelle mit Eiffel (in German);
Diplomarbeit Univ. Karlsruhe, Fakultät für Informatik, June 1993.