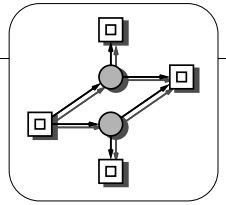
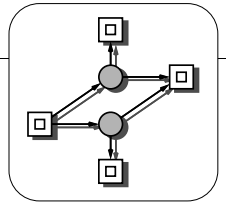


BTU Cottbus,
Informatik

ZUR SICHERHEITSTECHNISCHEN ZERTIFIZIERUNG VON SPS-ANWENDERPROGRAMMEN MIT PETRINETZEN

MONIKA HEINER

mh@informatik.tu-cottbus.de
http://www.informatik.tu-cottbus.de



ZUSAMMEN-FASSUNG

Es wird ein Projekt vorgestellt zur Entwicklung eines dedizierten Petrinetz-Arbeitsplatzes für Automatisierungstechniker.

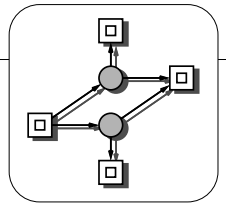
Eine weit verbreitete Realisierungsform in der Automatisierungstechnik sind speicherprogrammierbare Steuerungen (SPS). Zur Beschreibung von SPS-Anwenderprogrammen gibt es einen internationalen Standard (IEC 1131-3), der u.a. eine Petrinetz-nahe Ablaufsprache enthält. Diese IEC-1131-konformen Steuerprogramme werden zunächst auf “klassische” (Platz/Transitions-) Petrinetze zurückgeführt und um ein Petrinetz-Modell der zu steuernden Anlage (Umgebungsmodell) ergänzt.

Das eröffnet die Möglichkeit, zur Validierung/Zertifizierung der automatisierungstechnischen Lösung auf ein breites Arsenal von Petrinetz-basierten Methoden zurückgreifen zu können, für die teilweise mittlerweile auch leistungsfähige und stabile Werkzeuge existieren. Dabei wird insbesondere Wert gelegt auf eine geeignete Kombination von

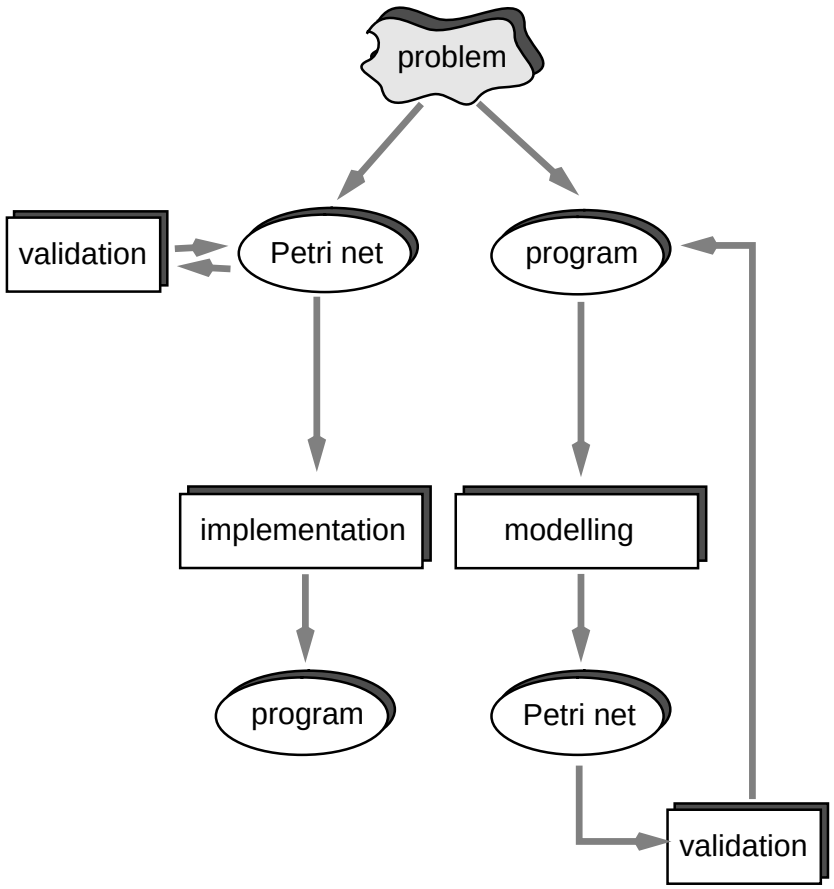
* informalen (Animation),
* semi-formalen (systematisches Testen) und
* formalen (Analyse)

Methoden.

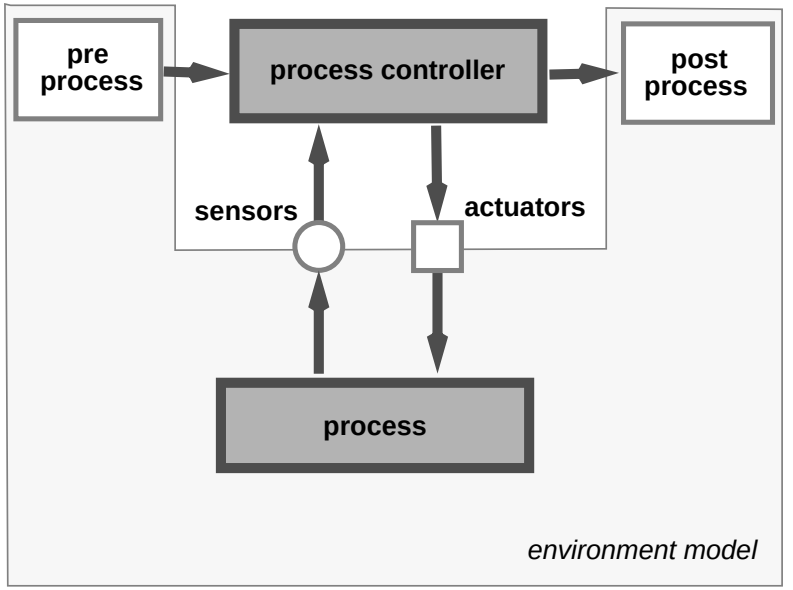
Es werden Entwurfskriterien für ein allgemeines Framework eines offenen und konfigurierbaren Petrinetz-Werkzeugkastens abgeleitet.



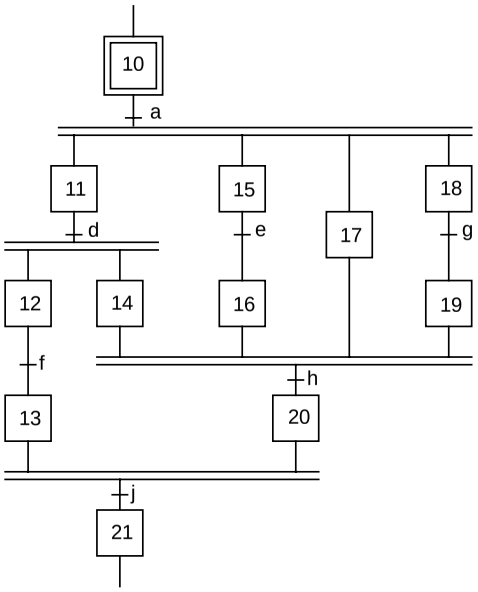
SOFTWARE ENGINEERING & PETRI NETS

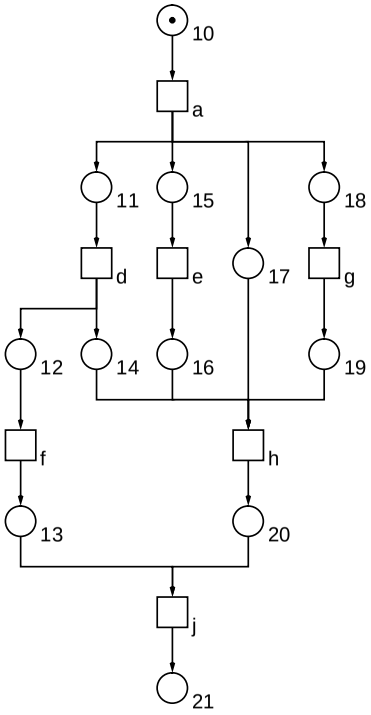


Basic structure of reactive systems:



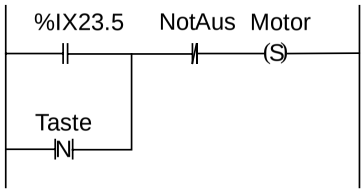
Ablaufsprache (AS) -> Petrinetz:



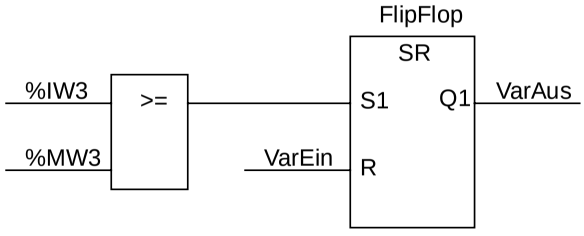


[Gouma 96, p. 84]

SPS-Programmierung mit IEC 1131-3



Kontaktplan (KOP)



Funktionsbausteinsprache (FBS)

```
LD      %IX23.5
ORN     Taste
ANDN    NotAus
S       Motor
```

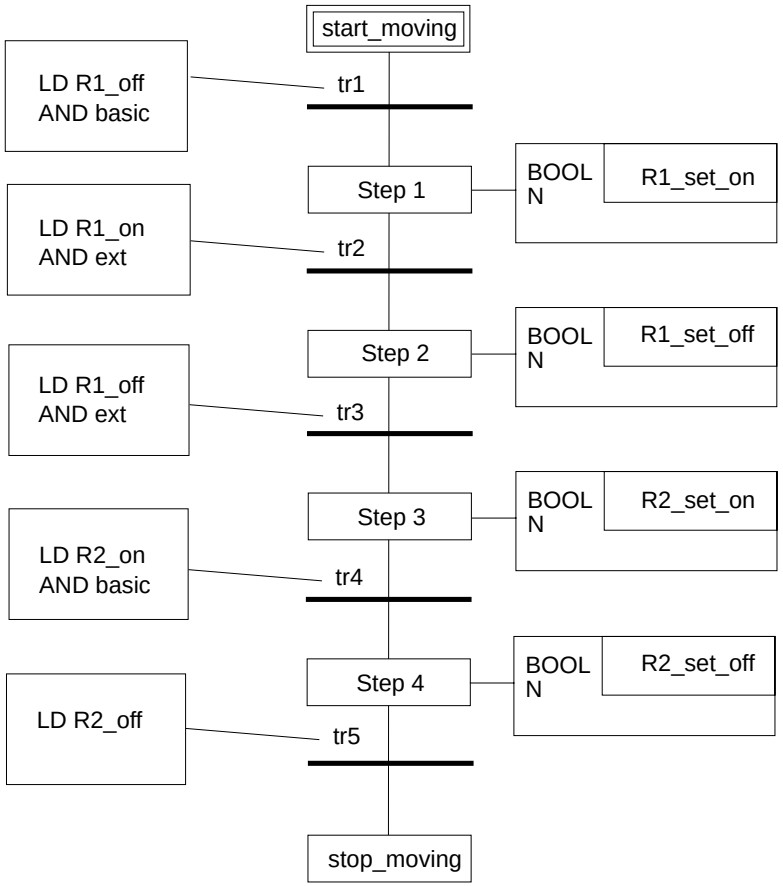
Anweisungsliste (AWL)

```
CAL FlipFlop ( S1 := (%IW3 >= %MW3),
               R := VarEin );

VarAus := FlipFlop.Q1;
```

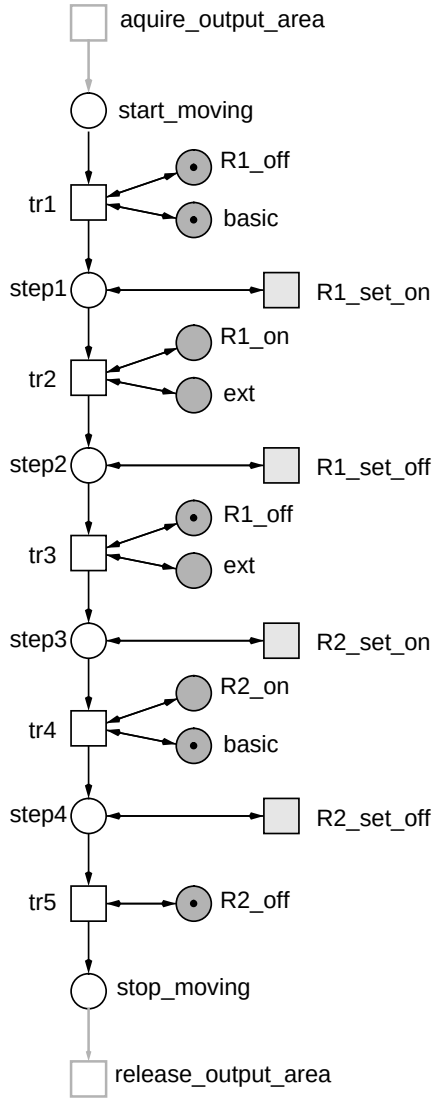
Strukturierter Text (ST)

Pusher controller as PLC program:

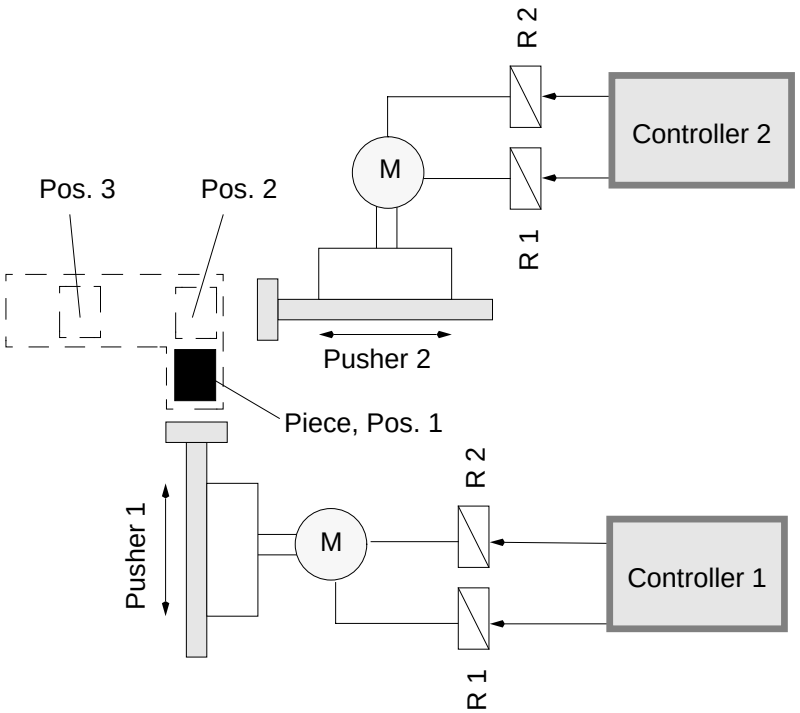


[Rausch 96]

Controller as place/transition net, control model:

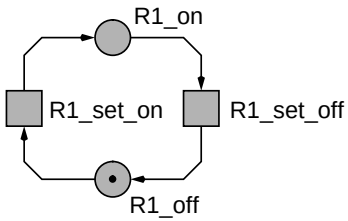


Example - concurrent pushers:

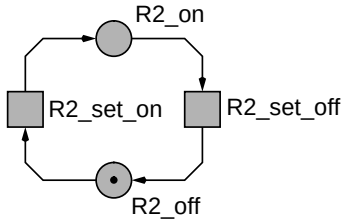


Environment model, without explicit error states:

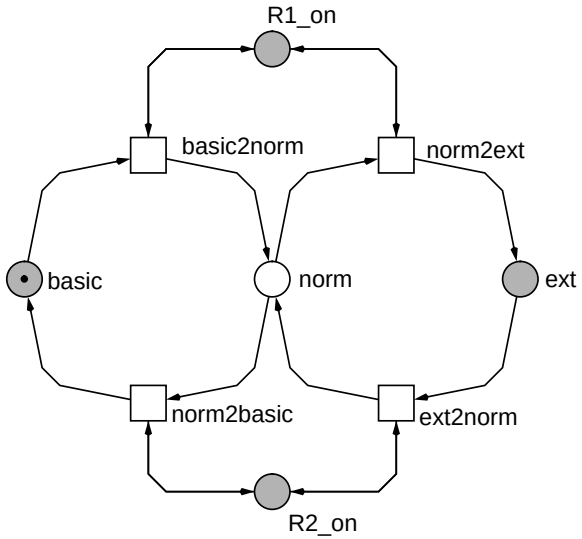
RELAY R1



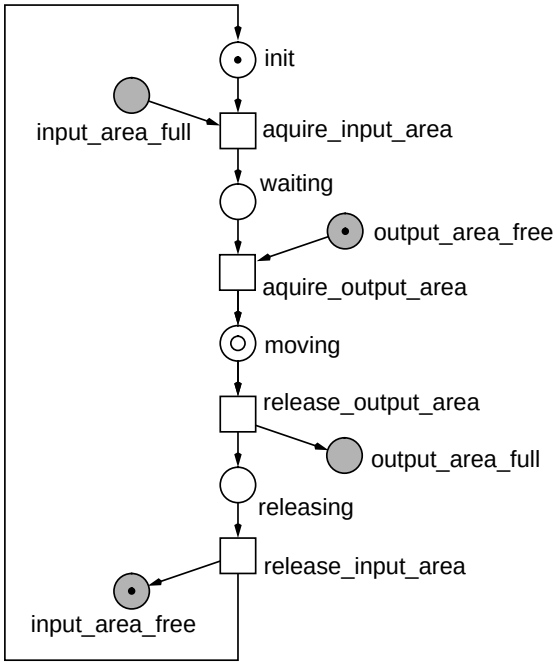
RELAY R2



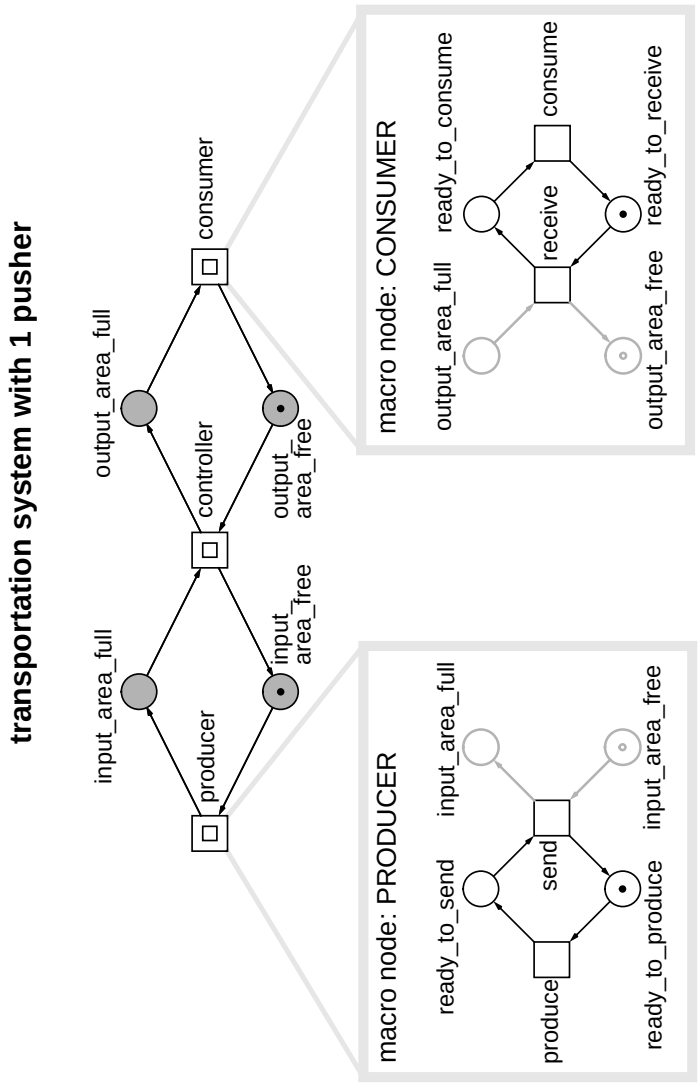
PUSHER
without error states



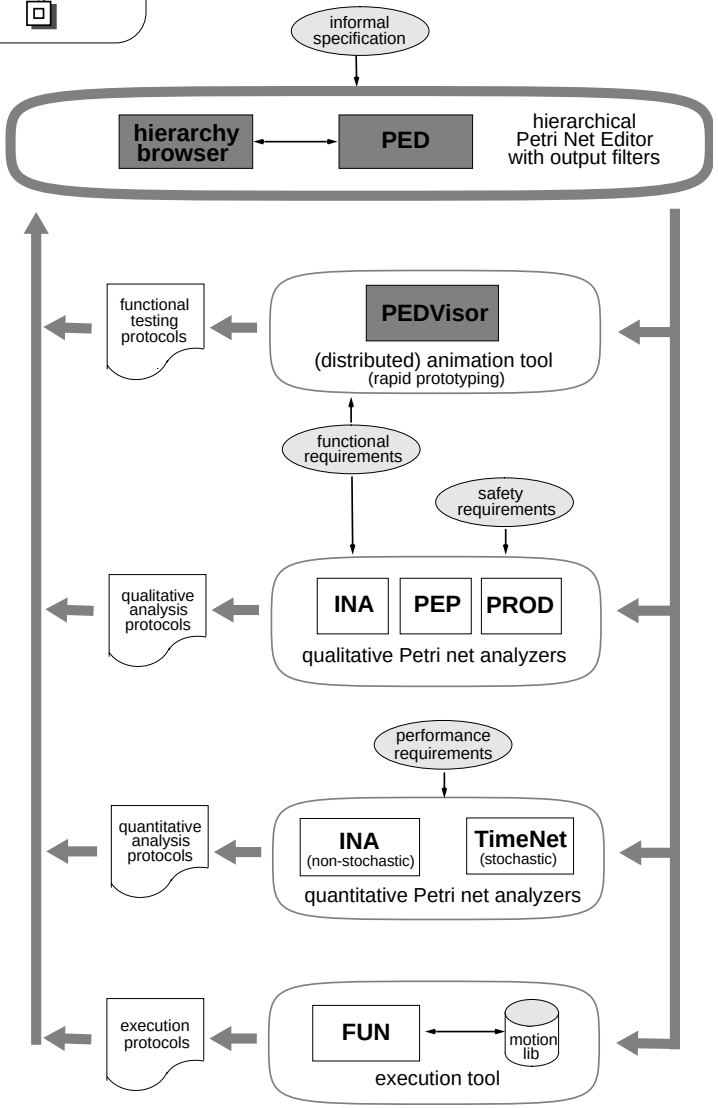
Controller as place/transition net, cooperation model:



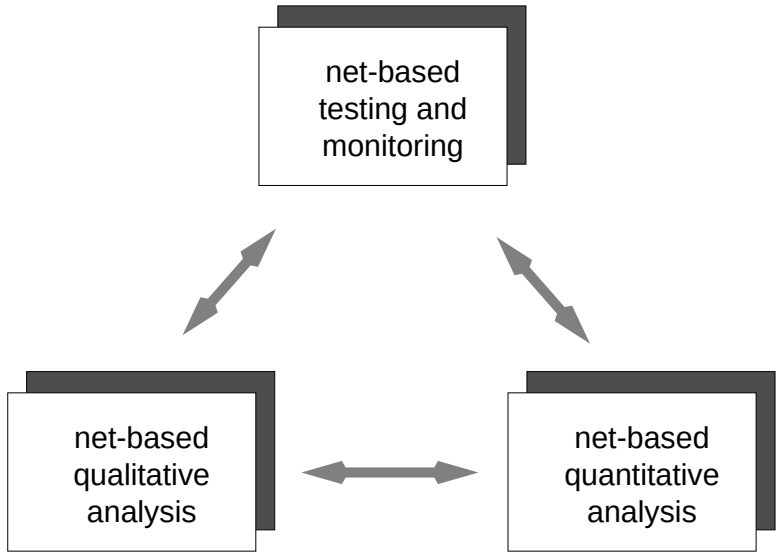
Pusher, producer consumer relation:

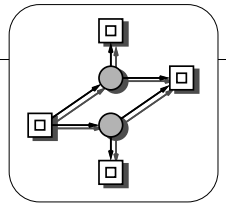


TOOL OVERVIEW:



INTEGRATION OF QUALITATIVE & QUANTITATIVE ANALYSES





MODEL CLASSES

- PETRI NETS
  - PLACE/TRANSITION PETRI NET (COLOURED PN)
    - context checking by Petri net theory
    - verification by temporal logics
  - TIME-DEPENDENT PN
    - NON-STOCHASTIC PETRI NET
      - worst-case evaluation
    - STOCHASTIC PETRI NET
      - performance prediction
      - reliability prediction

QUALITATIVE ANALYSIS METHODS:

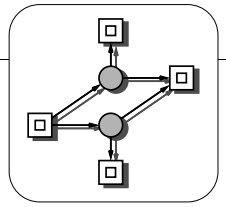
- NET REDUCTION
- STRUCTURE PROPERTIES
  - ***simple properties***
  - ***deadlock trap property***
  - state machine allocatable nets
  - rank theorem
- INVARIANTS
  - ***place invariants***
  - ***transition invariants***

static analysis

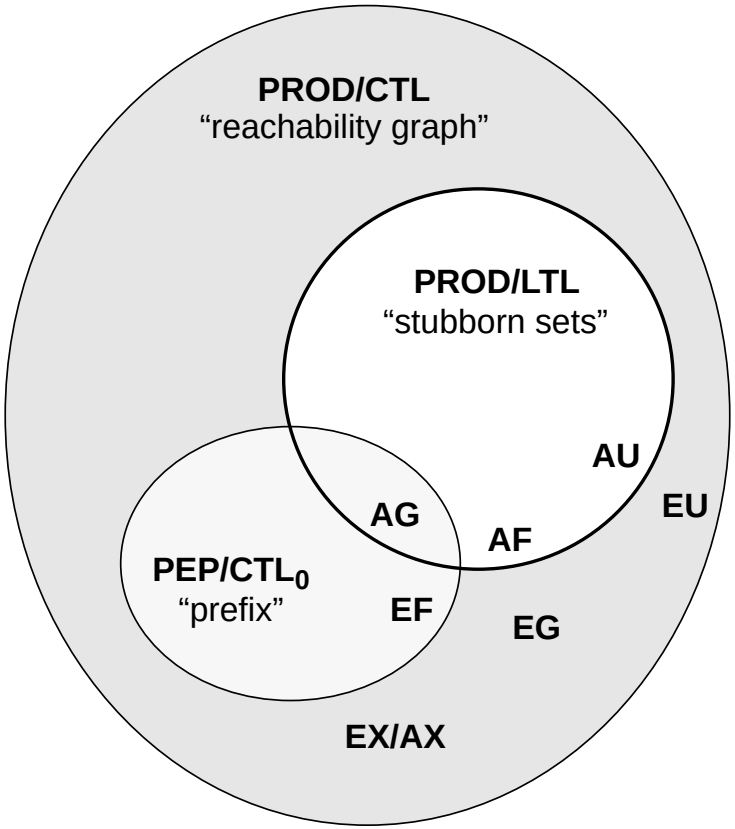
- REACHABILITY ANALYSIS
  - (complete) reachability graph
  - reduced state set
    - coverability graph
    - symmetry
    - ***stubborn set***

dynamic analysis

- ***MODEL CHECKING***



TEMPORAL LOGICS:



Concurrent pushers, analysis efforts:

# pushers	P / T	R	R_{stub}	prefix B / E
1	24 / 21	88	22	96 / 45
2	42 / 38	464	42	213 / 99
3	60 / 55	3.088	79	366 / 170
4	78 / 72	18.848	133	555 / 258
5	96 / 89	118.624	204	780 / 363
6	114 / 106	738.368	292	1041 / 485
7	132 / 123	4.614.208*)	397	1338 / 624
8	150 / 140	?	519	1671 / 780
9	168 / 157	?	658	2040 / 953
10	186 / 174	?	814	2445 / 1143

*) about 5 days, 900 MByte

Concurrent pushers, typical safety properties:

- At any time, a pusher may be driven in one direction only, e.g.:

  $\mathbf{AG}\ (\neg(R1\_on \wedge R2\_on))$

- To avoid collision, it is not allowed to move adjacent pushers at the same time, e.g.:

$$\left(\sum_{i=1}^{2} Ri\_on\_Pj + \sum_{i=1}^{2} Ri\_on\_Pk\right) \leq 1 \quad , \forall j,k : j+1 = k$$

- No pusher motion must be driven too far, e.g.:

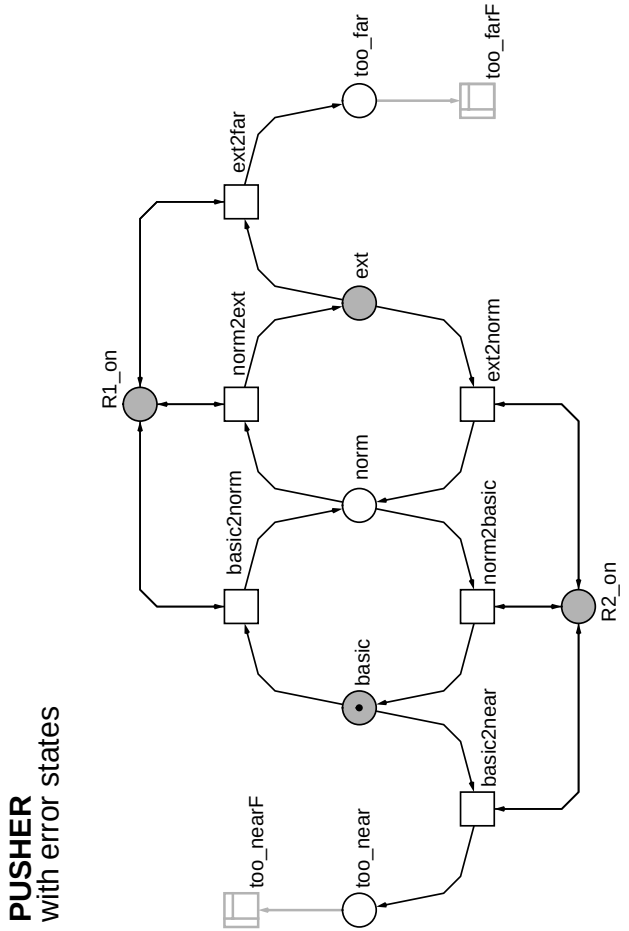
  $\mathbf{AG}\ (\neg P1\_too\_near)$

  $\mathbf{AG}\ (\neg P1\_too\_far)$

- While moving a pusher1, a new work piece must not arrive in its input position, e.g.:

  $\mathbf{AG}\ (ch01full \rightarrow P1\_basic)$

Environment model, with explicit error states:

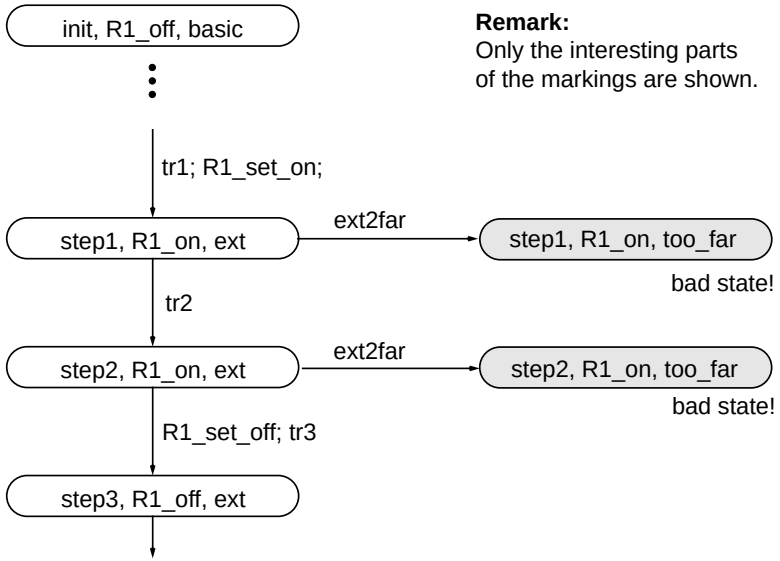


PUSHER
with error states

Time-dependent Petri nets, overview:

firing principle / times	**non-preemptive** (token reservation)	**preemptive** (no token reservation)
constant	timed nets [Ramchandani 74] ***-> (non-preemptive) duration nets***	preemptive duration nets
interval	non-preemptive interval nets	time nets [Merlin 74] ***-> (preemptive) interval nets***

Concurrent pushers, (part of the) reachability graph:



Remark:
Only the interesting parts of the markings are shown.

-> (preemptive) interval nets

unreachability of bad states, m_0-dead(ext2far) if:

$$lft(tr2) < eft(ext2far) \wedge$$
$$lft(set\_R1\_off) < eft(ext2far)$$

PLC workbench, basic components:

* dedicated technical language(s)
  -> *functional requirements*
  -> *safety requirements*
  -> *performance requirements*

* batch processing of requirement specifications
  -> *distributed over different tools & processors*

* libraries of reusable Petri net components
  -> *environment model*
  -> *cooperation patterns*

PedFrame, basic concepts:

* hierarchy of Petri net classes
  -> *extension / adaptation*
  (net elements, attributes of net elements, graphical representation, analysis methods)
  -> *configuration*

* free dynamic changes of current Petri net class
  -> *global data structure*
  -> *Petri net class = view on global data*

* hidden (foreign) tool interfaces
  -> *inline tools*
  -> *offline tools (e.g. PEP, PROD/LTL)*
  -> *online tools (e.g. INA, PROD/CTL)*

* heuristics of suitable analysis steps
  -> *property ==> method*

* visualization of analysis results
  -> *esp. negative results*

* class library of standard algorithms