Proc. 2nd IMACS Symposium on Mathematical Modelling (MATHMOD VIENNA ‘97), February 1997, pp. 171-176

ON EXPLOITING THE ANALYSIS POWER OF PETRI NETS FOR THE VALIDATION OF DISCRETE EVENT SYSTEMS

Monika Heiner
Brandenburg University of Technology at Cottbus, Computer Science Institute
Postbox 101344, D-03013 Cottbus, Germany
mh@informatik.tu-cottbus.de, http://www.informatik.tu-cottbus.de

Abstract: The development of provably error-free concurrent systems is still a challenge of practical system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. Among those Petri net analysis techniques suitable for strong verification purposes there is an increasing amount of promising methods avoiding the construction of the complete interleaving state space, and by this way the well-known state explosion problem. This paper claims to demonstrate that the available methods and tools are actually applicable successfully to at least medium-sized systems. For that purpose, the step-wise validation of various system properties (consistency, safety, progress) of the concurrent control software of a reactive system is performed. If possible, different analysis techniques are applied and compared with each other concerning its efforts.

keywords: programmable logic controller, hierarchical place/transition nets, verification, temporal logics, model checking, interval nets;

1 Introduction

Petri nets enjoy several advantages with respect to modelling and analysis of discrete event systems with inherent concurrency. Worth mentioning is especially the ability of combining different methods on a common representation. This variety ranges from informal (animation) via semi-formal (systematic testing) up to formal (exhaustive analysis) methods and comprises qualitative as well as quantitative evaluation techniques. But maybe most valuable is the fact that among the formal methods suitable for strong verification purposes there is an increasing amount of promising methods avoiding the construction of the complete interleaving state space, and by this way the well-known state explosion problem.

This paper gives an overview on these methods and demonstrates their strength and limitations by a running example. The discussion covers

- **static analysis techniques**, constructing no state space at all,
  e.g. structural properties allowing conclusions on behavioural properties, linear-algebraic analysis revealing invariants, local reduction rules to minimize the net structure,
- **lazy state space construction**, building reduced (interleaving) state spaces, which are generally much smaller than the complete state space for highly concurrent systems,
  e.g. stubborn set reduction to decide deadlock freedom or un-/reachability of special states, and to prove the validity of formulae in a nexttime-free linear time temporal logic (LTL\X),
- **alternative state space construction**, exploiting concurrency to build partial order (true concurrency) descriptions of the system behaviour,
  e.g. finite prefix of branching processes for model checking, and - just emerging - concurrent automata (CA), combining the advantages of reachability graphs and branching processes (the nodes are global states; the arcs are labelled with semi-words of transition events; each branching corresponds actually to a conflict;).

As example serves an adopted version of the pusher problem for which in [7] a control program has been synthesized automatically. By this way, this paper presents a reversal check for that synthesis. General transformation rules are sketched to transform programmable logic controller (plc) programs into ordinary place/transition Petri nets. Therefore, the rich amount of Petri net analysis techniques and tools can be applied for computer-aided analysis of plc programs.

The tool kit used comprises PED (hierarchical Petri net editor) [5], INA (structural properties, place/transition invariants, stubborn set reduced deadlock and reachability analysis, net reductions) [9], PROD (stubborn set reduced deadlock analysis and model checking - LTL\X) [10], and PEP (prefix-based model checking - CTL_0, linear-algebraic analysis) [1]. For short descriptions of the tool features which have been proven to be suitable see e.g. [2]. More details can be found in the referred tool manuals.

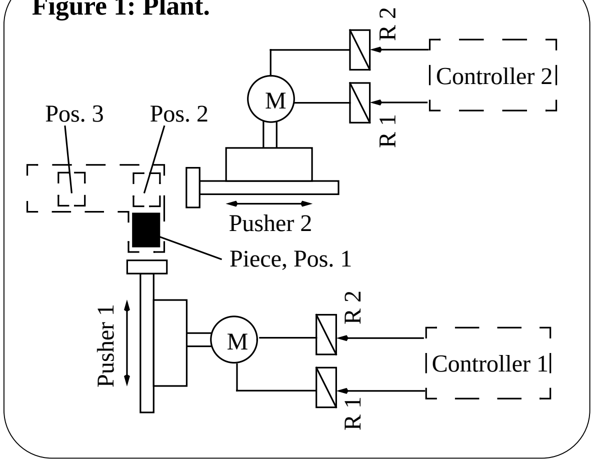
2 Task Description

To make the paper self-contained, the running example, adopted from [7], is shortly sketched.

The example consists basically of two concurrently working pushers moving work pieces (see figure 1). The work piece is moved from position one to position two by the first pusher, and from position two to position three by the second pusher. Both pushers are driven by electric motors which can be controlled by corresponding relays into two moving directions.

Starting from this basic situation, chains of concurrent pushers may be constructed in order to move pieces step by step from the input position via a number of inner positions to the output position.

Figure 1: Plant.



3 Requirement Specification

In addition to the task description, a list of informally specified safety and progress properties is given. Typical properties of this type are:

(a) safety

- At any time, a pusher can be driven in one direction only.
- To avoid collisions, it is not allowed to move adjacent pushers at the same time.
- No pusher motion must be driven too far/near.
- While moving a pusher, a new work piece must not arrive in its input position.

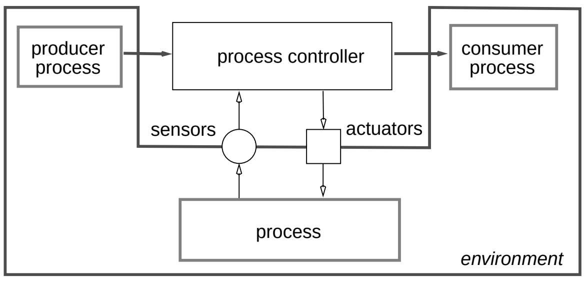
(b) progress

- After an active phase of a pusher, its successor will be activated before the predecessor will be started again.
- It is guaranteed that each pusher works infinitely often (livelock freedom).
- Any work piece entering the plant will finally leave the plant.

(c) consistency
Additional properties to be verified emerge during modelling reflecting useful (self-) consistency checks (see section 4.1).

4 Modelling with Hierarchical Petri Nets

The model of the total system may be characterized by a strong separation of control program and environment into different parts. The control program consists generally of a finite and static set of communicating processes. The environment model is composed of small reusable components: the producer/consumer processes of the work flow, and the devices of the controlled process.



4.1 Environment Model

For each device type exists a net component - building step-by-step a growing reusable component library to describe the uncontrolled plant behaviour. Each physical device is basically characterized by its finite set of discrete states, and additionally by the commands (externally visible transitions - they grey ones) forcing the device to change its current state (see fig. 3). Obviously, each device must be in one and only one state at any time. In terms of Petri net theory, the states of a device form a place invariant. In our example, there are two types of devices (relays, pushers). Accordingly, there are two consistency conditions. E.g. it holds for all pushers *Pi*:

(P1) $(Pi\_too\_near + Pi\_basic + Pi\_norm + Pi\_ext + Pi\_too\_far) = 1$

or expressed as temporal formula ($\dot{\vee}$ stands for exclusive or):

(P1*) $\mathbf{AG}\ (Pi\_too\_near\ \dot{\vee}\ Pi\_basic\ \dot{\vee}\ Pi\_norm\ \dot{\vee}\ Pi\_ext\ \dot{\vee}\ Pi\_too\_far)$

The composite model is structured into two layers. The top layer of a transport system with two pushers (Figure 2) consists of six macro components. Each macro transition P1 and P2 contains the process environment model given in figure 3, but prefixed with the instance names P1 or P2, respectively. For a more systematic analysis procedure (see section 5 and section 6), two versions of pushers are considered: without and with explicit error states (too_near, too_far). In the initial state (marking), all relays are off, the pushers are in their basic positions, and the plant is empty (contains no work piece).

4.2 Control Program Model

The pattern of the essential parts of the controller macro transitions (con1, con2) is given in Figure 4. The original plc programs are written in IEC 1131-3 (see left part). These programs are (automatically) translated into ordinary place/transition nets. For an example, how this could look like, consider the right side in Figure 4.

In order to avoid unnecessary restrictions of the concurrency degree, it could be helpful to exploit a special test arc feature for modelling of the transitions' side conditions. In that case, the amount of data, which has to be searched through during the analysis steps, may become much smaller, provided the analysis tools are prepared to handle test arcs.

4.3 Requirement Specification in Model Terms

Finally, the informally given requirement specifications have to be transformed into the terms of the formal model.

(a) safety

- At any time, a pusher can be driven in one direction only:

  (P2) $\mathbf{AG}\ (\neg(Pi\_R1\_on \wedge Pi\_R2\_on))\ ,\ \forall i$

- To avoid collisions, it is not allowed to move adjacent pushers at the same time:

  (P3) $\left(\sum_{i=1}^{2} Pj\_Ri\_on + \sum_{i=1}^{2} Pk\_Ri\_on\right) \le 1\ ,\ \forall i, \forall j, k : j+1 = k$

- No pusher motion must be driven too far/near:

  (P4) $\mathbf{AG}\ (\neg Pi\_too\_near)\ ,\ \forall i$

  (P5) $\mathbf{AG}\ (\neg Pi\_too\_far)\ ,\ \forall i$

- While moving a pusher, a new work piece must not arrive in its input position:

  (P6) $\mathbf{AG}\ (posi\_full \rightarrow Pi\_basic)\ ,\ \forall i$

(b) progress

- After an active phase of a pusher, its successor will be activated before the predecessor will be started again:

  (P7) $\mathbf{AG}\ (Pi\_norm \vee Pi\_ext \rightarrow \mathbf{AF}\ (\neg(Pi\_norm \vee Pi\_ext)\mathbf{AU}(Pj\_norm \vee Pj\_ext)))\ ,\ \forall i, j : i+1 = j$

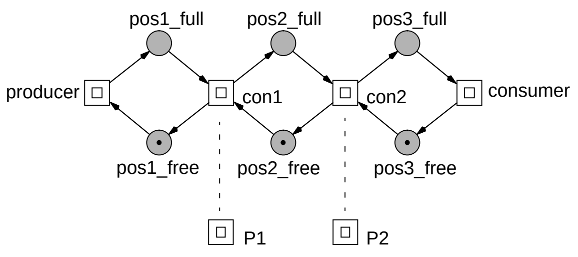
- It is guaranteed that each pusher works infinitely often (livelock freedom), e.g. (*en(t)* stands for the conjunction of all preplaces of *t*:

  (P8) $\mathbf{AG}\ (\mathbf{AF}\ (en(Pi\_basic2norm)))\ ,\ \forall i$

- Any work piece entering the plant will finally leave the plant (which may be considered as a consequence of (P7) and (P8)), i.e. in case of a two-pusher chain:

  (P9) $\mathbf{AG}\ (pos1\_full \rightarrow \mathbf{AF}\ pos3\_full)$

Figure 2: Top layer of a transportation system with two pushers.



Drawing convention:
Shadowed nodes are so-called logical (fusion) nodes. They serve as connectors to avoid immoderate edge crossing. All logical nodes with the same name are logically identical.

Figure 3: Process environment model of each controller.

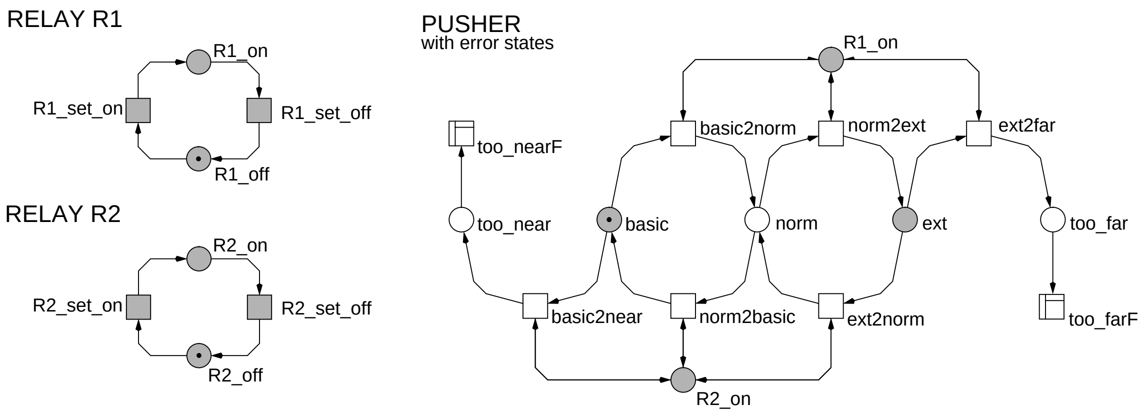
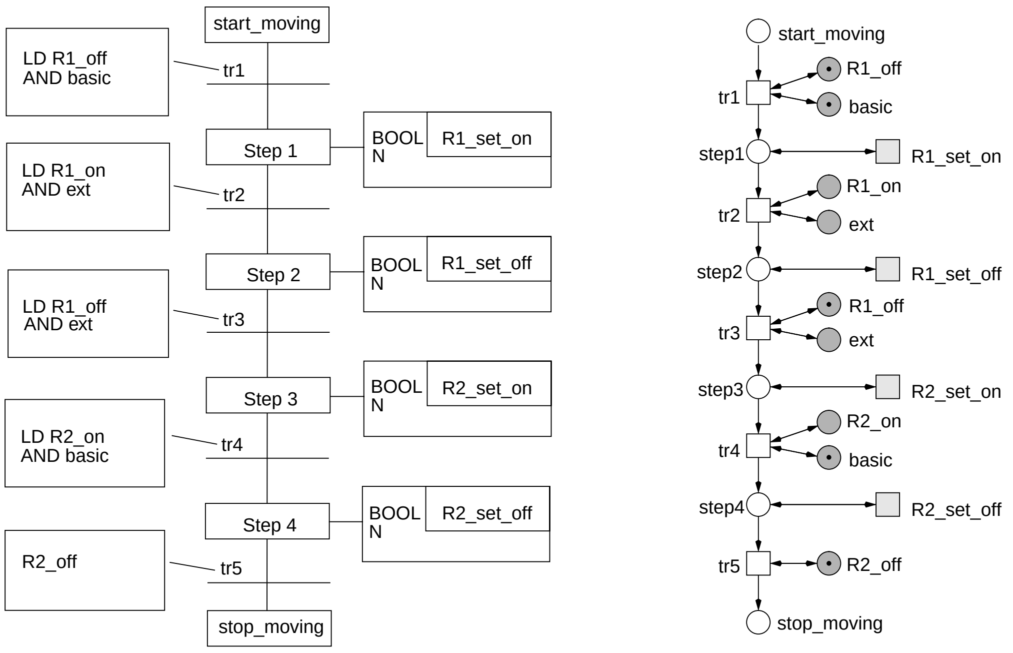


Figure 4: Part of the controller program and its Petri net model.



5 Qualitative Analysis

We present a two-step analysis. At first, the pusher model without explicit error states is discussed in this section. Afterwards, the error states are integrated leading to the notion of time (see section 6).

5.1 General Analysis

General analysis deals with properties which should be valid independently of the intended functional behaviour of the system. Basically, these are boundedness and liveness.

boundedness: The net is covered by semi-positive place invariants (INA). Moreover, the token sum of all these place invariants equals to 1. So we are able to conclude the 1-boundedness of the net (a necessary precondition for PEP's model checker).

liveness: The deadlock freedom can be proven efficiently by construction of stubborn set reduced reachability graphs (INA, PROD), which are generally much smaller than the complete state space. Additionally, it can be shown efficiently that the net is covered by semi-positive transition invariants as necessary (but not sufficient) condition for liveness. But liveness (no dead system parts) can't be proven by classical Petri net theory for longer pusher chains, due to the lack of suitable net structures (the given nets are not Extended Simple, net reduction does not help), and due to the state explosion by considering all interleaving transition sequences (reachability graph). However, based on the branching processes' prefix, for each transition the liveness has been proven (PEP) by model checking the temporal formula: $\mathbf{AG}\ \mathbf{EF}\ (en(t))$.

5.2 Special Analysis

(a) safety

There are different analysis techniques available to prove the unreachability of unsafe states (P2) - (P6):

Facts (INA): The unsafe states may be modelled as facts (special transitions which are expected to become never enabled). But, the evaluation of bad states (a state where a fact is enabled) by the given tool kit requires the reachability graph. That's why we will avoid this approach.

Stubborn set reduction (INA): The net is transformed in such a way that the unsafe states become dead states. Then the stubborn set reduced reachability graph has to be constructed. Because any dead states are preserved under this reduction, the original net does not contain any unsafe states if the transformed net does not reach any dead states. This technique could be useful if the required net transformation is done by the analysis tool.

Place invariants (INA): A sufficient condition for the unreachability of a given marking m is fulfilled if the there exists at least one place invariant x for which the token conservation equation

$$\sum_{p \in P} x(p) \cdot m_0(p) = \sum_{p \in P} x(p) \cdot m(p)$$

is not valid. To check this equation, complete markings must be specified. But unsafe states are usually given in terms of submarkings (containing "don't care" places). This main disadvantage is overcome in the next approach.

Trap equation (PEP): Based on a linear upper approximation of the state space, a sufficient condition for linear properties of the type $A \cdot m \leq b$ has been introduced in [4]. The implementation is integrated in the latest version of PEP. We use it to prove (P3).

Model checking of temporal formulae: Model checking, combined with stubborn set reduction (PROD, LTL\X) or based on the finite prefix of branching processes (PEP, CTL_0), provides generally the most convenient method to raise safety questions, esp. because set of (unsafe) states may be characterized in a concise manner. Both model checkers run very fast. Due to the evaluation method, they are applicable also to larger systems of which the size of the interleaving state space is unknown.

(b) progress

(P7) - (P9) use a richer set of (temporal) logical operators. Therefore, model checking facilities are unavoidable. Due to the **AF** and **AU** operators, these properties can be proven only by PROD. We use it to prove (P7) - (P9) for any pusher chain.

(c) consistency

For any pusher chain, (P1) is analyzable by INA, and in the version of (P1*) by PROD or PEP. But for larger systems, it is generally a cumbersome task to prove this type of properties by finding the suitable place invariants.

A summary on the analysis efforts necessary to gain the results mentioned above is given in the following table.

Table 1: Overview on analysis efforts.

# pushers	P / T	R	R_{stub}	prefix (B / E)	CA[a)], events
1	24 / 21	88	22	96 / 45	26
2	42 / 38	464	42	213 / 99	45
3	60 / 55	3.088	79	366 / 170	82
4	78 / 72	18.848	133	555 / 258	119
5	96 / 89	118.624	204	780 / 363	173
6	114 / 106	738.368	292	1041 / 485	228
7	132 / 123	4.614.208	397	1338 / 624	299
8	150 / 140	?	519	1671 / 780	372
9	168 / 157	?	658	2040 / 953	460
10	186 / 174	?	814	2445 / 1143	551

a) 2 nodes (global states) and 2 arcs (labelled with semi-words of events), for any pusher chain.

6 Quantitative Analysis

In case of explicit error states within the model (Pi_too_far, Pi_too_near), it has to be proven that a pusher, after having reached the expected extension, is switched off fast enough. Obviously, we have now to take into consideration also the timing behaviour of the given system.

In terms of interval Petri nets [6] this means that the error transitions modelling the pusher motions into unsafe states (Pi_ext2far, Pi_basic2near) may be enabled, but will never fire due to the influence of time. Therefore, the proof of the unreachability of explicit error states ((P4), (P5)) can be traced back to the proof that the related error transitions are dead at the initial state.

This may e.g. happen because the transitions Ci_tr2 and Pi_R1_set_off (disabling Pi_ext2far) fire always before Pi_ext2far is willingly to fire. Generally, a proof like that depends essentially on the chosen interval times (but can be done by INA, at least as long as the reachability graph fits into memory). But in this concrete case, we are able to conclude - by evaluating a suitable part of the reachability graph (or at best a non-interleaving version of it) - that for any time intervals for which the relations

$$lft(Ci\_tr2) < eft(Pi\_ext2far) \wedge lft(Ri\_set\_off) < eft(Pi\_ext2far)$$

hold, the dangerous transitions Pi_ext2far will never fire. Similar relations hold for Pi_basic2near.

7 Conclusions

All qualitative (i.e. timeless) properties have been proven without construction of the reachability graph (interleaving state space). Up to now, the quantitative (i.e. time-dependent) analysis of interval nets is based on reachability graph construction and evaluation. But in [8], a method has been proposed to describe the behaviour of interval nets by a finite prefix of branching processes. It seems to be worth thinking over how to combine both approaches. Nevertheless, all proves were carried out automatically by help of general Petri net analysis tools. Therefore, they are reproducible in an objective way.

References

[1] BEST, E.; GRAHLMANN, B.: PEP - Programming Environment Based on Petri Nets, Documentation and User Guide; Univ. Hildesheim, Institut für Informatik, Nov. 1995.

[2] HEINER, M.; DEUSSEN, P.; SPRANGER, J.: A Case Study in Developing Control Software of Manufacturing Systems with Hierarchical Petri Nets; Proc. 1st Int. Workshop on Manufacturing and Petri Nets held at ICATPN ’96, Osaka, June ’96, pp. 177-196.

[3] HEINER, M.; POPOVA-ZEUGMANN, P.: Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets; BTU Cottbus, Dep. of CS, Techn. Report I-02/1996, available on http://www.informatik.tu-cottbus.de.

[4] Melzer, S.; Esparza, J.: Checking System Properties via Integer Programming; ESOP ’96, Linköping, LNCS 1058, pp. 250-264.

[5] PED: http://www-dssz.Informatik.TU-Cottbus.De/~wwwdssz/ped.html.

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

[7] RAUSCH, M.; LÜDER; A.; HANISCH, H.-M.: Combined Synthesis of Locking and Sequential Controllers; Proc. WODES ’96, Edinburgh/ UK, Aug. 1996, pp. 133-138.

[8] SEMENOV, A.; YAKOVLEV, A.: Verification of Asynchronous Circuits Using Petri Net Unfolding; Proc. DAC ‘96, Las Vegas, June 1996, pp. 59-63.

[9] STARKE, P. H.: INA - Integrated Net Analyzer; Manual, Berlin 1992.

[10] VARPAANIEMI, K. et al.: PROD Reference Manual; Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, August 1995.