Design and validation of a controller

Laure Petrucci

LSV, CNRS UMR 8643, ENS de Cachan
61, avenue du Président Wilson, 94235 Cachan Cedex, France

ABSTRACT

In this paper[1], we consider automated manufacturing systems modeled with Petri nets. Some of the transitions are controllable while the others are not. We also suppose that forbidden states are specified. The goal is to design a controller which will act on the controllable transitions only, such that no forbidden state can be reached. Moreover, the controller should be as permissive as possible, i.e. should allow to reach as many authorized states as possible.

Here, we present a methodology to find the states in which the controller has to be used, and to test the optimality of the controller. We exemplify this method with the five AGVs problem. We also show how this example could be treated using a combination of existing tools, one based on enumerative approach, and the other on symbolic calculus. Finally we exhibit a controller which we prove optimal.

Keywords: Control, Petri nets, Reachability, Symbolic analysis.

1. THE CONTROL PROBLEM

In this paper, we present a methodology to investigate the control problem for automated manufacturing systems. We assume that the system to control is modeled by means of a Petri net ([7]), where two kinds of transitions are distinguished: those that can be controlled (by an operator or a machine) and the other ones on which no external event can operate. Moreover, we suppose that a specification of the undesired states, i.e. the situations that should never happen, is also provided.

The goal is to check that the system never enters the undesired states and if it does, to model a controller, acting on the controllable transitions only, such that the obtained behavior becomes correct. Hence, the controller restricts the set of possible behaviors of the system. An important issue is also to be able to determine whether the controller is optimal, i.e. it is impossible to find another controller

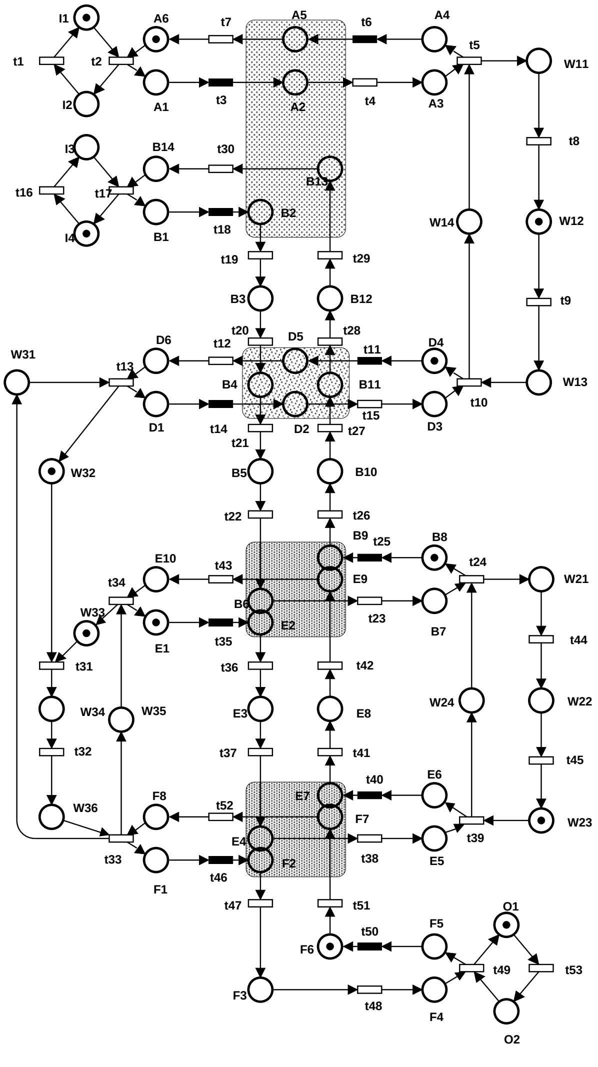

FIGURE 1. The five AGVs problem.

[1]This work has beeen done within the french INRIA project MARS.

admitting more correct behaviors. This is a very difficult problem since the existence and unicity of an optimal controller cannot be guaranteed.

Throughout this paper, each step of our method will be described and then applied to the five automated guided vehicles (AGVs) problem from [6].

The example consists of three workstations which have to operate on parts, two input and one output stations, and five AGVs which move parts from one station to another. The Petri net of figure 1 describes the system without any control. The full transitions are those that can be controlled while the other ones are uncontrollable (but can provide some feedback). The grayed boxes represent dangerous zones, i.e. locations where AGVs will eventually collide. A controller must be designed in order to avoid collisions, so there must always be at most one token in the dangerous zones.

We use DESIGN/CPN, a tool dedicated to Petri nets ([1]) to make reachability analysis, and also as an editor to draw Petri nets[2].

The structure of the paper follows our methodology. In section 2, we describe how a large model can be reduced, without loss of information, in order to improve the efficiency of analysis. Then, section 3 explains how to find "dangerous" states that will eventually lead to forbidden states, using backward reachability. We refine this set of states in section 4, using forward analysis. Finally, we show, in section 5, that these combined results of backward and forward reachability analysis give criteria for the design and validation of a controller.

2. REDUCING THE SIZE OF THE PROBLEM

At first, we can see that it is impossible to generate the full occurrence graph of the system because it is too large. Effectively, if we try to generate this graph, using DESIGN/CPN, the tool eventually stops after a few days, when no memory is left. Hence, we will reduce the size of the model to analyze. This can be done using the classical Petri nets reduction techniques, such as the agglomerations, implicit places,... defined in [4]. But we have to apply these with care, i.e. without changing the global structure of the net, the meaning of the different parts, nor the controllable transitions and critical zones.

Let us explain the reductions applied on our example. Places W11, resp. W13, are substitutable, therefore transitions t5 and t8, resp. t9 and t10, can be agglomerated, leading to a new cycle for the first workstation with only places W12, W14 and transitions t5, t10, as on figure 2. A similar reduction is performed for the two other workstations.

[2] In fact, the tool operates on high-level nets, namely coloured Petri nets ([5]). Here, we consider place/transition nets, where all tokens have the same elementary color e.

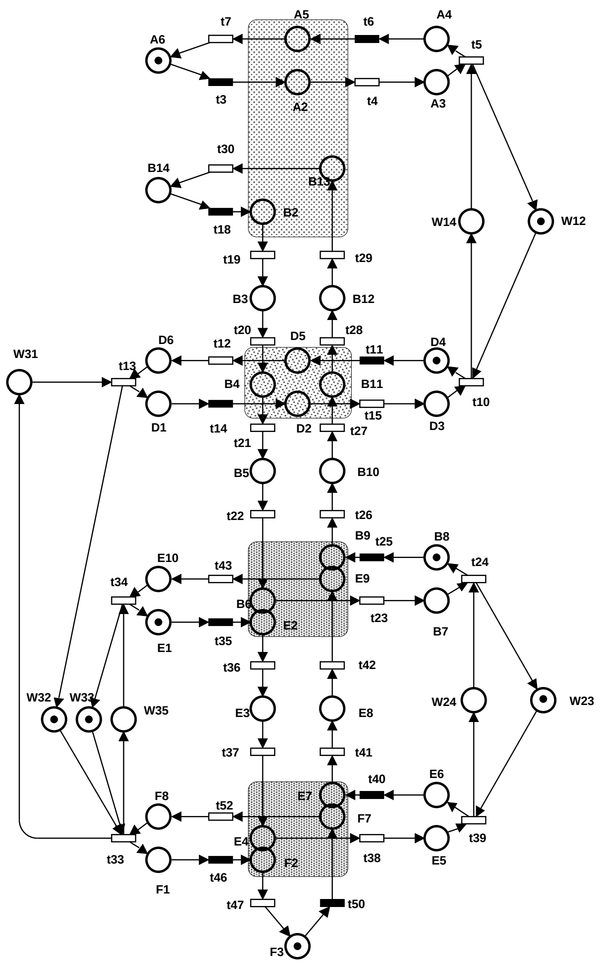

FIGURE 2. The reduced model.

In the first input station, place I2 can be removed, gluing transitions t1 and t2 together, in the same manner as before. Then place I1 is linked only to transition t1 by a self-loop. Hence, it is an implicit place ([4]), and it can be removed. The same operation can be applied to the other input/output stations. Finally, using these reductions again, places A1, B1, F4 and F5 as well as transitions t2, t17, t48 and t49 can be removed.

All these reductions lead to the net in figure 2. If the net was further reduced, some crucial information about the net would be lost. E.g., concerning workstations, removing place W14 would also mean fusing transitions t5 and t10, thus forcing a synchronization between AGVs A and B. Other reductions on the AGVs themselves are impossible. For example, removing B3 and fusing transitions t19 and t20 would produce a net without uncritical place between B2 and B4, which are both in different dangerous zones. Such a reduction would lead to removing an escape place, and thus should not be done.

When we examine the reduced system (figure 2), we can see that the number of reachable states is at most the product of the number of reachable states of the individual components, i.e. $5 \times 13 \times 6 \times 10 \times 5 \times 2 \times 2 \times 4 = 312,000$. The generation of the occurrence graph with DESIGN/CPN shows that there are exactly 312,000 reachable states.

Thus, we can conclude that all the combinations of states of the individual components can be attained. The reductions we applied do not affect this sort of behavior, since are not involving transitions which synchronize the different components. Hence, we can deduce that the whole system would have as many reachable states as the product of the individual components, i.e. $6 \times 14 \times 6 \times 10 \times 8 \times 4 \times 4 \times 6 \times 2 \times 2 \times 2 = 30,965,760$ and all the forbidden states are reachable.

3. COMPUTING THE DANGEROUS STATES

In order to avoid the forbidden states, we will look for the transitions that lead to these, and more precisely for the last controllable one that will eventually allow to fire uncontrollable transitions and finally reach a forbidden state, as depicted in figure 3. We will say that the states enabling such a controllable transition are *dangerous*. The transition must then be disabled in the dangerous state, otherwise there would be no way to avoid reaching the forbidden state.

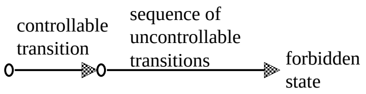

FIGURE 3. Backward reachability from forbidden state.

Finding the set of dangerous states can be done efficiently by performing backward reachability, using symbolic methods and computations in real arithmetic, features which are provided by HYTECH ([2]). This tool was originally designed for hybrid automata, but it can be used for computations in Petri nets as shown in [3]). A small program we wrote allows to automatically generate an HYTECH automaton from a DESIGN/CPN Petri net, according to the description in [3]. The automaton has only one state, there is one variable per place of the net, and all transitions loop on the state, updating the values of variables as in the Petri net. An optimization consists in using one variable per state machine instead of one per place of the state machine. For example, instead of describing the AGV A with 5 variables A2, A3, A4, A5 and A6, each having value 0 or 1, and such that exactly one of them is not null, we use a single variable A with values 2, 3, 4, 5 or 6, indicating the place in which the AGV is.
Part of the automaton corresponding to the Petri net is given in figure 4 for transitions t3 (controllable) and t5 (uncontrollable).

```
loc netstate: while A>=1 & B>=1 & W1>=0 &
    D>=1 & W2>=0 & E>=1 & F>=1 &
    W33>=0 & W31>=0 & W35>=0 & W32>=0
    wait{ }
-- t3
    when A = 6 sync t3
        do {A'=2} goto netstate;
-- t5
    when A = 3  &  W1 = 1 sync uncont
        do {A'=4, W1'=0} goto netstate;
```

FIGURE 4. Part of the HYTECH net automaton.

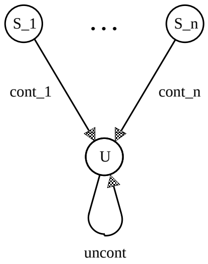

FIGURE 5. The observer automaton.

The forbidden states are specified with the variables. For our purposes, we need to distinguish between controllable and uncontrollable transitions. To do so, we label the controllable transitions with their name, and all the other ones with the same label, e.g. `uncont`. It is easy to do this in HYTECH by synchronizing the automata of the Petri net with another one, designed to observe the behavior, sketched in figure 5. It has one state (`S_1`,..., `S_n`) per controllable transition (`cont_1`,..., `cont_n`), and this transition is the only one exiting the state, leading for all transitions to the same additional state (`U`) where only uncontrollable transitions (`uncont`) can be fired. The observer automaton for the five AGVs problem is given in figure 6.

When we analyze the synchronous product of the two automata, using backwards reachability from the forbidden states, with a code similar to the one in figure 7.

The sets of dangerous states obtained for our example is the following, sorted by outgoing controllable transition, and presented with the names of places of the Petri net:

```
automaton OBSERVE
synclabs : uncont, t3, t6, t11, t14,
           t18, t25, t35, t40, t46, t50;
initially firststate;

-- This simulates several initial states
loc firststate: while True wait{ }
    when True goto St3;
    when True goto St6;
    ...
    when True goto St50;
loc St3: while True wait{ }
-- t3
    when True sync t3
        do {} goto U;
loc St6: while True wait{ }
-- t6
    when True sync t6
        do {} goto U;
...
loc St50: while True wait{ }
-- t50
    when True sync t50
        do {} goto U;
loc U: while True wait{ }
-- uncont
    when True sync uncont
        do {} goto U;
end
```

FIGURE 6. Part of the HyTech observer automaton.

```
var pb, aux, auxunc, auxt3 : region;
-- The description of forbidden states
pb := ((A=5|A=2) & (B=2|B=13)) |
      ((D=2|D=5) & (B=4|B=11)) |
      ((E=2|E=9) & (B=6|B=9)) |
      ((E=4|E=7) & (F=2|F=7));
aux := reach backward from
           (pb & loc[OBSERVE]=U)
       endreach;
-- The uncontrollable parts
auxunc := hide  W1, W2, W31, W32, W33, W35
              in aux & loc[OBSERVE]=U
          endhide;
print auxunc;
-- The markings enabling t3 followed by an
-- uncontrollable sequence
auxt3 := hide W1, W2, W31, W32, W33, W35
             in aux & loc[OBSERVE]=St3
         endhide;
print auxt3;
```

FIGURE 7. A piece of the HyTech code computing dangerous states.

trans.	*states*
t3	A6 & (B2 | B9 | B10 | B11 | B12 | B13)
t6	A4 & (B2 | B9 | B10 | B11 | B12 | B13)
t11	D4 & (B2 | B3 | B4 | B9 | B10 | B11)
t14	D1 & (B2 | B3 | B4 | B9 | B10 | B11)
t18	B14 & (E2 | E7 | E8 | E9 | D2 | D5 | A2 | A5)
t25	B8 & (E2 | E7 | E8 | E9 | D2 | D5 | A2 | A5)
t35	E1 & (B2 | B3 | B4 | B5 | B6 | B9 | F2 | F7)
t40	E6 & (B2 | B3 | B4 | B5 | B6 | B9 | F2 | F7)
t46	F1 & (E2 | E3 | E4 | E7)
t50	F3 & (E2 | E3 | E4 | E7)

As HyTech performs symbolic computations, the result has a form similar to the table above.

4. REDUCTION AND ANALYSIS OF THE SET OF DANGEROUS STATES

We now have, for each controllable transition, a description of all the states in which it is dangerous to fire the transition, i.e. the states that we will potentially have to control. But we are not ensured that these states will really be reachable after adding a controller. We must now find those states that will effectively have to be controlled. This can be checked on a part of the occurrence graph of the system, where the dangerous firings are avoided.

To do so, we use Design/CPN again, explicitly saying that the successors of these states, using the corresponding controllable transition in the table, should not be further developed. This is easily done by evaluating a small piece of ML code within Design/CPN branching options:

The occurrence graph without the states which are not further developped, i.e. specified in the branching options, has 152, 384 nodes. We check which of the dangerous states, according to the table found with HyTech, are missing in this graph. In this case, we find none. Hence we can conclude that the transitions must be controlled in order not to be fired from the states specified in the table.

5. CONSTRUCTION AND VERIFICATION OF A CONTROLLER

For each controllable transition, we now have the set of states in which the transition must not be fired. This is very helpful in the process of designing a controller.

Moreover, when a controller is added, a comparison of its occurrence graph with the one previously obtained will tell about the optimality of the controller or show which markings are no more reachable and why.

We construct a controller for our example by adding new places corresponding to the four critical

```
OGSet.BranchingOptions{
TransInsts = 0,
Bindings = 0,
Predicate = fn n =>
(cf(e,Mark.AGV'A2 1 n))*     (* 1st line *)
  (cf(e,Mark.AGV'B2 1 n)+      (* of *)
   cf(e,Mark.AGV'B9 1 n)+    (* the table *)
   cf(e,Mark.AGV'B10 1 n)+
   cf(e,Mark.AGV'B11 1 n)+
   cf(e,Mark.AGV'B12 1 n)+
   cf(e,Mark.AGV'B13 1 n))+
(cf(e,Mark.AGV'A5 1 n))*     (* 2nd line *)
  (cf(e,Mark.AGV'B2 1 n)+
   cf(e,Mark.AGV'B9 1 n)+
   cf(e,Mark.AGV'B10 1 n)+
   cf(e,Mark.AGV'B11 1 n)+
   cf(e,Mark.AGV'B12 1 n)+
   cf(e,Mark.AGV'B13 1 n))+
(cf(e,Mark.AGV'D5 1 n))*     (* 3rd line *)
  (cf(e,Mark.AGV'B2 1 n)+
   cf(e,Mark.AGV'B9 1 n)+
   cf(e,Mark.AGV'B10 1 n)+
   cf(e,Mark.AGV'B11 1 n)+
   cf(e,Mark.AGV'B4 1 n)+
   cf(e,Mark.AGV'B3 1 n))+
(cf(e,Mark.AGV'D2 1 n))*     (* 4th line *)
  (cf(e,Mark.AGV'B2 1 n)+
   cf(e,Mark.AGV'B9 1 n)+
   cf(e,Mark.AGV'B10 1 n)+
   cf(e,Mark.AGV'B11 1 n)+
   cf(e,Mark.AGV'B4 1 n)+
   cf(e,Mark.AGV'B3 1 n))+
... =0 };
```

FIGURE 8. The branching options.

zones, that will constraint the firing of the controllable transitions:

New place	*input trans.*	*output trans.*
Z1	t4, t7, t19, t30	t3, t6, t18, t25
Z2	t12, t15, t21, t28	t11, t14, t18, t25
Z3	t23, t26, t36, t43	t18, t25, t35, t40
Z4	t38, t41, t47, t52	t35, t40, t46, t50

The occurrence graph contains 152,384 nodes. This is exactly the number of states found at the previous step. Thus, this controller is *optimal*. The occurrence graph analysis also shows that the controlled system behaves correctly, i.e. there is no deadlock, the system is live and re-initialisable.

6. CONCLUSION

We presented a methodology to design and validate a controller. When dealing with a large system, the corresponding model can first be reduced without any loss of information. Then, a combination of symbolic and enumerative methods, using successively backward and forward reachability, available in already existing tools provides, for each controllable transition, the set of states in which the control has to be applied. Finally, once a controller is designed, its behaviour can be checked and compared to the maximal behaviour expected.

REFERENCES

[1] DESIGN/CPN. http://www.daimi.au.dk/designCPN.

[2] *A user guide to* HYTECH. http://www.eecs.berkeley.edu/˜tah/HyTech.

[3] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In Springer, editor, *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 178–193, Eindhoven, August 1999.

[4] G. Berthelot. Checking properties of nets using transformations. In *Advances in Petri Nets 1985*, volume 222, pages 19–40. Springer, 1986.

[5] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and pratical use. Volume 1: basic concepts*. Monographs in Theoretical Computer Science. Springer, 1992.

[6] B. Krogh and L. Holloway. Synthesis of feedback control logic for discrete manufacturing systems. *automatica*, 27(4), 1991.

[7] W. Reisig. *Petri nets. An introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.