Complexity of Checking Output-Determinacy in General Petri Nets*

Petr Jančar[1] and Victor Khomenko[2]

[1] Faculty of Science, Palacký University Olomouc, Czech Republic
pj.jancar@gmail.com
[2] School of Computing, Newcastle University, Newcastle upon Tyne, UK
Victor.Khomenko@ncl.ac.uk

Abstract. Output-determinacy is an important soundness notion in the theory of non-deterministic asynchronous concurrent systems whose alphabet is partitioned into input and output actions. Intuitively, it postulates that the set of enabled outputs of a system must be fully determined by the visible history of its interactions with the environment, as otherwise the specification is contradictory in the sense that it simultaneously requires and forbids some output.
In the case of safe or k-bounded Petri nets checking output-determinacy was known to be PSPACE-complete, but the complexity (and even decidability) of this problem for general Petri nets was open. In this paper we show that the problem of checking whether output-determinacy is violated is decidable and equivalent to reachability in general Petri nets.

Keywords: Output-determinacy · Petri nets · Reachability · Computational complexity

1 Introduction

Labelled Petri nets (LPNs) with the alphabet (of transition labels) partitioned into input and output actions are often used for specifying concurrent systems (see, e.g., [2, 3, 7–9]). When a specification is deterministic (in the sense of automata and formal language theory), its semantics could be the set of its possible traces, i.e. its *language*. As the final implementation must be deterministic, it may seem reasonable to confine oneself to deterministic specifications only. However, often this turns out to be too restrictive in practice. There are several situations which naturally give rise to non-deterministic specifications which still can be implemented:

Silent transitions For convenience of modelling, the designers often use *silent* transitions, which are not included into visible traces of the system. Such transitions make the Petri net non-deterministic.

* Devoted to Prof Maciej Koutny on the occasion of his 60^{th} birthday.

OR-causality When a safe LPN is used for modelling a situation where the system has to respond to any of several possible stimuli in the same way, non-determinism naturally arises. (OR-causality can also be modelled as a non-safe (2-bounded) LPN without non-determinism [3, 9], but in practice safe Petri nets are preferable as they are much easier to analyse.)

Hiding signals Non-determinism naturally arises when in a deterministic LPN some of the signals are hidden (by turning some visible transitions into silent ones) – hiding signals is essential in many applications, e.g. the decomposition algorithm of [2, 7, 8].

It is thus natural to consider a non-deterministic model for which we could verify that there is a deterministic implementation.

We use the following formal model. An LPN $N = (P, T, F, I, O, \ell, M_N)$ is a structure comprising finite disjoint sets P, T of *places* and *transitions*, respectively; the *flow relation* $F \subseteq (P \times T) \cup (T \times P)$; disjoint sets I, O of *input actions* and *output actions*, respectively; the labelling function $\ell : T \to I \cup O \cup \{\varepsilon\}$ mapping each transition either to an action or to the empty word ε – such transitions are called *silent*; and the *initial marking* $M_N \in \mathbb{N}^P$ (for $\mathbb{N} = \{0, 1, 2, \ldots\}$).

We use the usual notation $M[\sigma\rangle$ to denote that marking M enables a sequence of transitions σ, and write $M[\sigma\rangle M'$ if σ is finite and enabled by M, and firing σ from M yields marking M'. We lift this notation to labels as follows, for ℓ extended to sequences of transitions. For a sequence of actions ν we write $M[\nu\rangle\rangle$ (resp. $M[\nu\rangle\rangle M'$) iff $M[\sigma\rangle$ (resp. $M[\sigma\rangle M'$) for some sequence σ of transitions such that $\ell(\sigma) = \nu$. In particular, a label l is considered enabled by a marking M, written $M[l\rangle\rangle$, if one can fire a finite sequence of silent transitions to directly enable a transition labelled by l, i.e. we have $M[\sigma\rangle M'[t\rangle$ where σ contains only silent transitions and $\ell(t) = l$.

In our constructions we often use pairs of arcs (p, t) and (t, p) (i.e. $(p, t) \in F$ and $(t, p) \in F$) for some place p and transition t. Such a pair will be called a *read arc* and depicted by a line without arrowheads (between a place-circle and a transition-box).

An LPN N specifies the behaviour of a system in the sense that the system must provide *all and only* the specified outputs and that it must allow *at least* the specified inputs. As a consequence, the system must be able to perform at least all traces of N. In fact, N also describes assumptions about the environment the system will interact with; namely, the environment will only produce the inputs specified by N. A correct implementation of N may allow additional input events (and traces), but these events and subsequent behaviour will never occur in the envisaged environment. In other words, when the system is running in a proper environment, only traces of N can occur.

The implementation may actually have fewer input signals than N, keeping only those that are relevant for producing the required outputs. In this case, the environment may provide irrelevant inputs, but the implementation simply ignores them — and in this sense, they are always allowed.

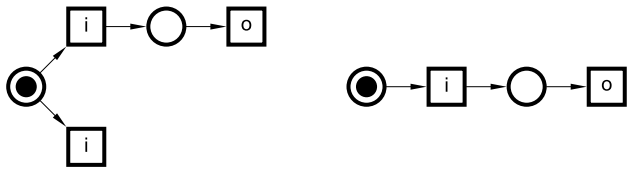

Fig. 1. The LPN on the left is not output-determinate; the result of determinisation is shown on the right. The latter LPN, though implementable and having the same language, is not a correct implementation of the original LPN: it can break the environment by producing o when the environment does not expect it.

Intuitively, assuming a deterministic implementation and possibly non-deterministic specification, the correctness of the implementation can be defined as follows (see [2] for a formal definition):

- the implementation must be able to perform all traces of the specification, maybe dropping some irrelevant input signals;
- after any trace, all the inputs allowed by the specification must be allowed (or ignored) by the implementation;
- after any trace, the implementation must enable exactly the specified outputs.

A non-deterministic specification can perform the same trace in two different ways, reaching different states (markings) M_1 and M_2. Assuming that the only information available to the system is the execution history, i.e. the trace performed, an implementation cannot determine whether its current state corresponds to state M_1 or M_2 of the specification. Hence, a deterministic implementation must behave consistently with the specification *no matter in which of these states the specification is*.

The above definition of correctness requires that the implementation must allow *at least* the inputs enabled by M_1 and *at least* the inputs enabled by M_2; this is easy to achieve even if these sets of inputs differ – i.e. the implementation may allow the union of these sets (or any superset thereof). However, the situation with outputs is different: The implementation must provide *exactly* the outputs enabled by M_1 and *exactly* the outputs enabled by M_2. This is only possible if M_1 and M_2 enable the same outputs.

This in particular implies that *the language is not an adequate semantics for non-deterministic specifications:* M_1 and M_2 may enable different sets of outputs, but the language cannot distinguish between such a system and its determinised version, yet the former has no deterministic implementations whereas the latter has (e.g. itself). For example, Fig. 1 illustrates a dangerous scenario when determinisation hides an error.

One possibility would be to define semantics based on some notion of bisimulation. However, it turns out that *bisimulation is unnecessarily strong* – it is

possible for two non-bisimilar systems to have exactly the same deterministic implementations. This has negative consequences for applications, e.g. some useful transformations preserving the possible implementations would be rejected if they do not yield a bisimilar system. For example, the decomposition algorithm of [8, 7] used semantics based on a variant of bisimulation (called 'angelic bisimulation'), and the semantics based on the notion of output-determinacy helped to improve it significantly [2].

In [2] a formal semantics of non-deterministic LPNs was proposed and justified. For this, the concept of *output-determinacy* (OD), which is a relaxation of determinism, was introduced. In particular, it was shown that for OD LPNs the language *is a sufficient semantics*, and this also holds in the case of distributed LPNs. Moreover, it was proved that an LPN cannot have deterministic implementations unless it is OD.

Definition 1 (Output-Determinacy). *An LPN N is called* output-determinate *(OD) if $M_N[\nu\rangle\rangle M_1$ and $M_N[\nu\rangle\rangle M_2$ implies for every output o that $M_1[o\rangle\rangle$ iff $M_2[o\rangle\rangle$.*

Therefore, OD is a useful correctness property that can be formally verified. Hence, the questions of decidability and computational complexity of this verification problem for various PN classes becomes relevant.

2 The complexity of checking output-determinacy

In [2] it was shown that the coverability problem is easily reducible to OD for safe/k-bounded/general PNs, which immediately yields the $\mathcal{PSPACE}$ lower bound for safe and for k-bounded PNs, as well as the $\mathcal{EXPSPACE}$ lower bound for general PNs. Moreover, for safe and k-bounded PNs an algorithm that runs in polynomial space was proposed. Hence checking OD is $\mathcal{PSPACE}$-complete for safe and k-bounded PNs. However, the upper bound and even decidability of this problem for general PNs were left open.

In this paper we show that finding a violation of OD for general Petri nets is polynomially equivalent to checking Reachability. Hence:

- The problem is decidable.
- The $\mathcal{EXPSPACE}$ lower bound in [2] is likely not tight, as Reachability is conjectured to be much harder.

We proceed by first establishing the improved lower bound, and then deriving a matching upper bound. Below, we denote by cOD the complement of OD, i.e. the problem of checking whether OD is violated.

2.1 The lower bound

We now show that Petri nets Reachability Problem (RP) is easily reducible to cOD, i.e. cOD is RP-hard. For simplicity, we use the special case of Zero

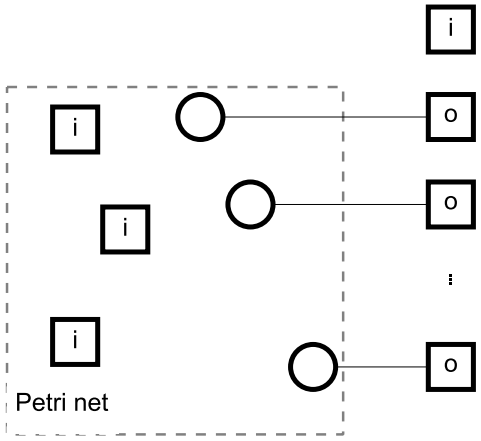

Fig. 2. Reduction from ZRP to OD.

Reachability Problem (ZRP), i.e. the reachability of the zero marking $\mathbf{0} = \{0\}^P$, that is known to be equivalent to RP [1].

Suppose one has to check whether the zero marking is reachable in a given (unlabelled) PN. We build an LPN as follows: If the initial marking of the PN is already $\mathbf{0}$, we return a fixed LPN violating OD. Otherwise we construct the LPN as follows, see Fig. 2:

- Each transition in the PN is labelled with the same input action i.
- For each place of the PN, a new o-labelled transition is created and connected to this place with a read arc; hence, o is always enabled in the resulting LPN as long as the marking is not $\mathbf{0}$.
- A new isolated transition $\hat{t}$ labelled with i is created; hence, i is always enabled in the resulting LPN.

Lemma 1. *Marking* $\mathbf{0}$ *is reachable in the original PN iff the constructed LPN violates OD. Moreover, the resulting LPN is safe/k-bounded/general if the original PN was safe/k-bounded/general.*

Proof. If the initial marking M_N of the PN is zero then the result is trivial, so we assume it is not and consider two cases:

(1) Suppose the zero marking is reachable in the original net via some non-empty execution σ, $M_N[\sigma\rangle\mathbf{0}$. The same execution can be performed in the constructed LPN yielding the trace terminating at the zero marking, $M_N[i^{|\sigma|}\rangle\rangle\mathbf{0}$, and by construction $\mathbf{0}$ does not enable o. Moreover, the LPN has another execution $M_N[\hat{t}^{|\sigma|}\rangle M_N$ yielding the same trace $i^{|\sigma|}$ but finishing at the initial marking that is not zero and thus enables o. Hence the LPN is not OD.

(2) Suppose the zero marking is not reachable in the PN, and hence in the LPN. This means that every reachable marking of LPN enables o, and so OD cannot be violated.

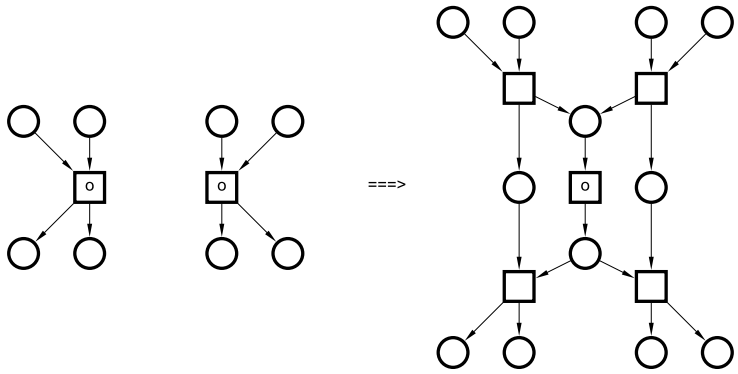

Fig. 3. Replacing multiple o-labelled transitions by a single one. Intuitively, the o-labelled transition is used as a sub-routine, with some silent transitions used to remove the respective input tokens before performing o, and to produce the respective output tokens after performing o.

The bound on the number of tokens follows from the fact that executing the added transitions does not alter the current marking. □

This result immediately implies that cOD is RP-hard (i.e., RP is polynomially reducible to cOD) for the corresponding class of PNs (safe/k-bounded/general).

2.2 The upper bound

OD can be checked by considering each output o in isolation, converting the other output labels to inputs: Indeed, OD holds iff each of these single-output LPNs is OD. Furthermore, multiple o-labelled transitions can be replaced by a single one using the transformation shown in Fig. 3. This transformation preserves the enabledness of o and thus the OD property. Hence, below we assume that the LPN has a single output o, and a single o-labelled transition t_o.

We follow the approach of [2] and compute the synchronous product of the LPN with itself (we thus get two copies of the net that evolve independently, except that any visible action in one copy can be only performed synchronously with the same action in the other copy). Then any violation of the OD is witnessed by the following trace of the product net: $(M_N, M_N)[\nu\rangle\rangle(M_1, M_2)$, with $M_1[t_o\rangle$ (i.e. t_o is immediately enabled by M_1) and $\neg M_2[o\rangle\rangle$ in the original LPN. Unfortunately, this property is difficult to decide for general PNs, since we must verify that M_2 does not enable o even via a very long sequence of silent transitions (on which we cannot bound the intermediate markings a priori). This is not a problem for safe and k-bounded Petri nets, as their markings are bounded and can be represented in polynomial space, but the decidability and complexity in case of general PNs was left open in [2].

We observe that the set En_o of markings of the LPN that enable o (perhaps, via a sequence of silent transitions), i.e.

$$En_o \stackrel{\text{df}}{=} \{M \in \mathbb{N}^P \mid M[o\rangle\rangle \text{ in the original LPN}\},$$

is upward closed, and so can be represented by the finite set of its minimal elements $E_1, E_2, \ldots, E_k \in \mathbb{N}^P$:

$$En_o = \bigcup_{i=1}^{k} \{M \in \mathbb{N}^P \mid M \geq E_i\}.$$

Furthermore, the natural numbers constituting vectors E_i are at most double-exponential in the size of the LPN. Indeed, consider the PN obtained from the LPN by removing all the non-silent transitions. Then these vectors are the minimal initial markings of this PN from which it is possible to cover the preset of t_o. Due to Rackoff's famous result [6], it is sufficient to consider firing sequences of double-exponential length, and they can consume only double-exponential number of tokens.

Here we need to consider the markings *not* enabling o (via silent transitions); hence we are interested in the complement of En_o, i.e. in the set

$$Dis_o \stackrel{\text{df}}{=} \overline{En_o} = \{M \in \mathbb{N}^P \mid \neg M[o\rangle\rangle \text{ in the original LPN}\}.$$

Hence Dis_o is a downward closed set, and it can be represented by the finite set of its maximal elements $D_1, D_2, \ldots, D_{k'}$ in the standard extension of $\mathbb{N}^P$ to the set $(\mathbb{N} \cup \{\omega\})^P$ of *generalised markings*; i.e.

$$Dis_o = \bigcup_{i=1}^{k'} \{M \in \mathbb{N}^P \mid M \leq D_i\}.$$

One can observe that the finite (i.e. non-ω) numbers in vectors D_i are smaller than the largest number occurring in vectors E_j. (Suppose some D_i contains $x \in \mathbb{N}$ that is greater than or equal to the largest number in vectors E_j. For every E_j we have $D_i \not\geq E_j$, i.e. some component of D_i is smaller than the corresponding component of E_j, which cannot be the component with x. But then D'_i arising from D_i by incrementing x also satisfies $D'_i \not\geq E_j$ for all E_j, which contradicts the maximality of D_i.)

Hence the non-ω elements of D_i are at most double-exponential in the size of the LPN, and so any D_i can be represented in exponential space using (e.g.) binary encoding of natural numbers, with a special code for ω.

We now define a non-deterministic algorithm that uses Reachability as an oracle and solves cOD. All operations except Reachability can be performed in exponential space; since Reachability is $\mathcal{EXPSPACE}$-hard, its complexity will dominate the overall complexity of the algorithm. The algorithm takes an LPN with single output o and a single o-labelled transition t_o and proceeds as follows.

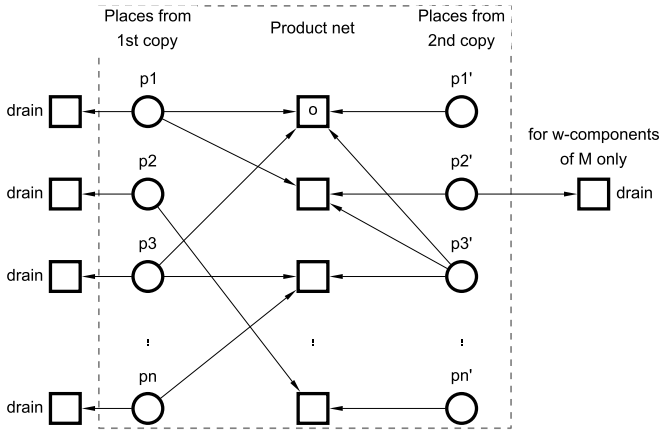

Fig. 4. Reduction from OD to RP.

- Non-deterministically generate a generalised marking M with either ω or at most double-exponential number of tokens per place (thus M can be stored in exponential space).
- Build a PN by removing all the non-silent transitions from the LPN, and set M as its initial marking while also removing the places corresponding to the ω-components of M. If the set of places corresponding to the preset of t_o (without the removed ones) is coverable in this PN then **reject**. (If the algorithm gets past this reject statement, then any finite marking $M' \leq M$ is in Dis_o. Moreover, in any finite $M' \in Dis_o$ we can possibly turn some components to ω and get a generalised M that satisfies the double-exponential restriction and is not rejected; this follows due to the bound on the size of maximal elements in $D_1, \ldots, D_{k'}$ derived above.)
- Build the instance of Reachability as follows (see Fig. 4):
  - Construct the product of the LPN with itself.
  - For each place in the first sub-net of the product, create a 'drain' transition consuming tokens from this place.
  - For each place in the second sub-net of the product that corresponds to an ω component of M, create a 'drain' transition consuming tokens from this place.
  - The marking to be reached puts a single token on each place corresponding to the preset of t_o in the first subnet, no tokens on the other places of the first subnet, and marking M on the places of the second subnet, with no tokens on the places corresponding to the ω components of M.
- If the constructed instance of Reachability is positive (as told by the oracle) then **accept** else **reject**.

Lemma 2. *The original LPN violates OD iff the above algorithm can accept the LPN.*

Proof. (1) Suppose the above algorithm has an accepting run; we fix one. Let M be the respective generalised marking that passed the test (of non-coverability of the preset of o). We consider an execution in the constructed PN that demonstrates the positive answer of Reachability; it yields the marking with tokens on the preset of t_o and no tokens elsewhere in the first subnet, and some tokens in the second subnet forming a finite marking $M' \leq M$. Removing all the 'drain' transitions from this execution still yields a valid execution of the PN, as these transitions can only remove tokens. This modified execution is an execution of the product net, still marks all the places in the preset of t_o in the first subnet, and in the second subnet yields some finite marking M'' that may have some extra tokens on places corresponding to the ω components of M; hence $M' \leq M'' \leq M$. The execution can thus be projected to two executions of the original LPN with the same visible trace, where one execution directly enables t_o and the other ends in a marking $M'' \in Dis_o$ (thus not enabling o even via a sequence of silent transitions); this constitutes a violation of OD.

(2) Suppose the LPN violates OD. Hence there are two executions with the same trace, one of them directly enabling t_o and the other ending in a finite marking M' not enabling o even via a sequence of silent transitions, i.e. $M' \in Dis_o$; thus $M' \leq D_i$ for some maximal element D_i of the extension of Dis_o. Let M be the marking that puts no tokens on places corresponding to the ω-entries of D_i, and coinciding with M' on other places. These two executions yield an execution of the product net leading to a marking enabling t_o in the first subnet and coinciding with M' in the second subnet of the product. By executing the 'drain' transitions as necessary, one can ensure that the marking of the first subnet has a single token on each of the places in the preset of t_o and no tokens elsewhere and the marking of the second subnet coincides with M — this marking corresponds to the one whose reachability is checked by the algorithm. Hence the algorithm can accept by first guessing M (that fits in exponential space due to the bound on the maximal non-ω elements of the extension of Dis_o) and then solving Reachability for the constructed reachable marking. □

A problem with the above construction is that the input to the Reachability sub-routine is large, as M might need exponential space (in the size of the original LPN) to be represented.

However, this problem can be solved by shifting the computation performed by the above algorithm into the constructed PN, thus polynomially reducing a cOD instance to a small instance of Reachability (in fact, to an instance of ZRP). A crucial ingredient is handled by Lipton's construction [4] (strengthened in [5]) enabling to simulate an exp-space-bounded automaton by a net of polynomial size. One can thus construct a polynomial-size PN (w.r.t. the size of the original LPN) which has the following behaviour (see Fig. 5):

– By a (polynomial-size) "Lipton module" it first non-deterministically generates a marking M where, moreover, each place p also gets its complementary place p' marked so that the sum of tokens in p and p' is $2^{2^{pol(n)}}$ for a suitable

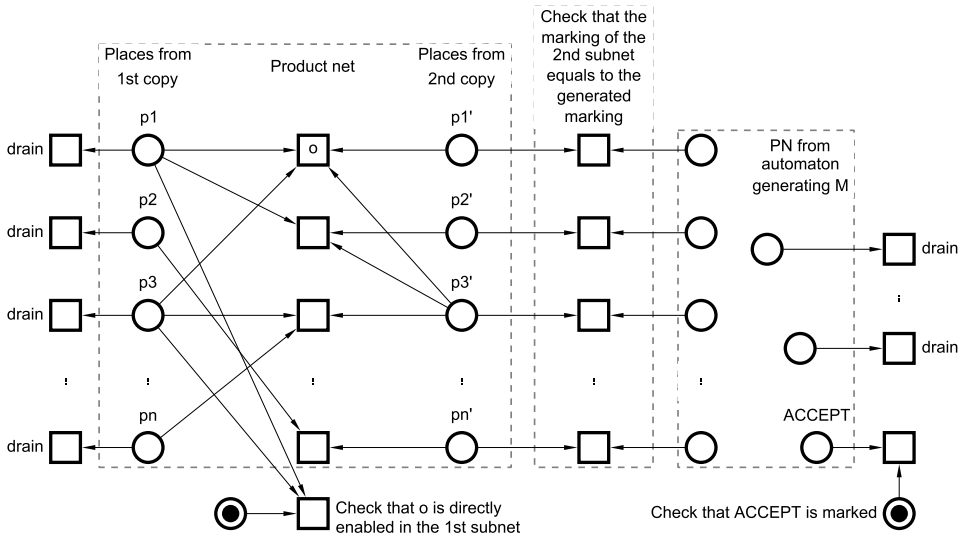

Fig. 5. Reduction from OD to ZRP.

polynomial *pol* (where n is the size of the original LPN). The polynomial *pol* is chosen so that the finite components in the maximal elements D_i of the extension of Dis_o are smaller than $2^{2^{pol(n)}}$; such polynomial *pol* follows from Rackoff's results [6]. We can view the value $2^{2^{pol(n)}}$ as ω. In fact, the module creates two copies of M, one copy (with complementary places) for the following test that $M \in Dis_o$, and the other copy for later comparing with the marking reached in the second part of the product net.

- Now another "Lipton module" checks that $M \in Dis_o$, i.e., the module has a possibility to get a token on a designated ACCEPT place precisely when $M \in Dis_o$. This module is again of polynomial size (it simulates checking if M can cover the preset of t_o within $2^{2^{pol(n)}}$ moves).
- (If there is a token on ACCEPT, then) an execution of the product net follows, reaching some marking (M_1, M_2).
- We want M_1 to enable o while M_2 to belong to Dis_o. The former condition can be handled by the drain transitions and asking that only one token is left in each place in the preset of t_o (in the final marking of the constructed Reachability instance). The other condition is established by comparing M_2 with (the stored copy of) M. This is achieved by special transitions consuming tokens from places of M_2 and corresponding places of M synchronously, asking that zero is reached for all of them. The only issue are "ω-places" in M, i.e. those having $2^{2^{pol(n)}}$ tokens. We first let such places (checked by a "Lipton module") to be adjusted anyhow by special transitions.

By adding several transitions and places and providing the possibility to drain any left over tokens from the "Lipton modules" we construct an instance of ZRP where the reachability of **0** marking guarantees that:

- the generation of M was successful, i.e. ACCEPT is marked;
- the marking reached in the second subnet equals to M (or to adjusted M which also belongs to Dis_o)
- the preset of t_o is marked.

Furthermore, guessing the output o for which OD is violated can also be implemented in the constructed PN in a straightforward way. Hence, we have a polynomial reduction from cOD to ZRP.

3 Conclusion

We have affirmatively answered the question of the decidability of Output-Determinacy in general Petri nets, and sketched the proof to determine its complexity: the complement of Output-Determinacy is polynomially interreducible with Reachability (or ZRP).

References

1. Hack, M.H.T.: Decidability Questions for Petri Nets. PhD thesis, MIT, 1975.
2. V. Khomenko, M. Schaefer and W. Vogler: *Output-Determinacy and Asynchronous Circuit Synthesis.* Special Issue on Best Papers from ACSD'07, IOS Press, *Fundamenta Informaticae*, 88(4) (2008) 541–579.
3. M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* John Wiley & Sons Ltd., 1994.
4. R. Lipton. *The Reachability Problem Requires Exponential Space.* Technical Report 62, Yale University, 1976.
5. E.W. Mayr and A.R. Meyer. *The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. Advances in Mathematics*, 46:305–329, 1982.
6. C. Rackoff. *The Covering and Boundedness Problems for Vector Addition Systems. Theoretical Computer Science*, 6:223–231, 1978.
7. W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. *Fundamenta Informaticae*, 76:161–197, 2006.
8. W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.
9. A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9:189–233, 1996.