Published in Int J Softw Tools Technol Transfer (2019).
https://doi.org/10.1007/s10009-019-00519-1

Counting Petri Net Markings From Reduction Equations

Bernard Berthomieu · Didier Le Botlan · Silvano Dal Zilio

April 2019

Abstract We propose a method to count the number of reachable markings of a Petri net without having to enumerate these first. The method relies on a structural reduction system that reduces the number of places and transitions of the net in such a way that we can faithfully compute the number of reachable markings of the original net from the reduced net and the reduction history. The method has been implemented and computing experiments show that reductions are effective on a large benchmark of models.

Keywords model counting · model-checking

1 Introduction

Structural reductions are an important class of optimization techniques for the analysis of Petri Nets (PN for short). The idea is to use a series of reduction rules that decrease the size of a net while preserving some given behavioral properties. These reductions are then applied iteratively until an irreducible PN is reached on which the desired properties are checked directly. This approach, pioneered for Petri nets by Berthelot [2,3], has been used to reduce the complexity of several problems, such as checking for boundedness of a net, for liveness analysis, for checking reachability properties [12] or for LTL model checking [7].

In this paper, we enrich the notion of structural reduction by keeping track of the relation between the markings of an (initial) Petri net, N_1, and its reduced (final) version, N_2. We use reductions of the form (N_1, Q, N_2), where Q is a system of linear equations that relates the (markings of) places in N_1 and N_2. We say that Q is a set of *reduction equations*.

In our approach, reductions are tailored so that the state space of N_1 (its set of reachable markings) can be faithfully reconstructed from that of N_2 and equations Q. In particular, when

Bernard Berthomieu
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
E-mail: bernard.berthomieu@laas.fr

Didier Le Botlan
LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
E-mail: didier.le-botlan@insa-toulouse.fr

Silvano Dal Zilio
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
E-mail: silvano.dal.zilio@laas.fr

N_1 is totally reduced (N_2 is then the empty net), the state space of N_1 corresponds with the set of non-negative integer solutions to Q. In some sense, Q acts as a symbolic, "equational" representation of the reachable markings, in much the same way one can use decision diagrams or SAT-based techniques to compute on sets of states.

In practice, reductions often lead to an irreducible non-empty residual net. In this case, we can still benefit from an hybrid representation combining the state space of the residual net (expressed, for instance, using a decision diagram) and the symbolic representation provided by linear equations.

This approach can provide a very compact representation of the state space of a net. Therefore it is suitable for checking *reachability properties*, that is whether some reachable marking satisfies a given set of linear constraints. However, checking reachability properties could benefit of more aggressive reductions since it is not generally required there that the full state space is available (see e.g. [12]). At the opposite, we focus on computing a (symbolic) representation of the full state space. A positive outcome of our choice is that we can derive a method to count the number of reachable markings of a net without having to enumerate them first.

Computing the cardinality of the reachability set has several applications. For instance, it is a straightforward way to assess the correctness of tools—all tools should obviously find the same results on the same models. This is the reason why this problem was chosen as the first category of examination in the recurring Model-Checking Contest (MCC) [8,9]. We have implemented our approach in the framework of the TINA toolbox [5] and used it on the large set of examples provided by the MCC (see Sect. 7). Our results are very encouraging, with numerous instances of models where our performances are several orders of magnitude better than what is observed with the best available tools.

Outline. We first define the notations used in the paper then describe the reduction system underlying our approach, in Sect. 3. After illustrating the approach on some full examples, in Sect. 4, we prove in Sect. 5 that the equations associated with reductions allow one to reconstruct the state space of the initial net from that of the reduced one. Section 6 discusses how to count markings from our representation of a state space while Sect. 7 details our experimental results. We conclude with a discussion on related works and possible future directions.

Many results and definitions were already presented in a shorter version of the paper [4]. This extended version contains several additions. We define an improved set of reductions rules that adds generalized versions of rules for agglomerating chains and cycles of places. These new rules allow us to reduce more instances of Petri nets and to reduce some instances further. (We show the impact of this new reduction strategy, called `compact+`, on our benchmarks in Sect. 7.) We also consider a specific rule for simplifying, under some conditions, a sub-part of a net that is a *Marked Graph* (see Sect. 2). To better understand the effect of this new strategy, we have extended the text that illustrates our approach with a new example, see Sect. 4.2, and we give more detailed proofs when studying the correctness of our approach. Finally, we have enriched our discussion on the methods used to compute the number of reachable markings from the set of reduction equations. Most particularly, we describe our own combinatorial method for computing the number of solutions to a set of reduction equations (see the discussion on *polycount* in Sect. 6.2).

2 Petri Nets

Some familiarity with Petri nets is assumed from the reader. We recall some basic terminology. Throughout the text, comparison ($=, \geq$) and arithmetic operations ($-, +$) are extended pointwise to functions.

A marked *Petri net* is a tuple $N = (P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$ in which P, T are disjoint finite sets, called the *places* and *transitions*, $\mathbf{Pre}, \mathbf{Post} : T \to (P \to \mathbb{N})$ are the *pre* and *post condition* functions. A *marking* is a function mapping a number of *tokens* to every place, $m_0 : P \to \mathbb{N}$ is the *initial marking*.

Figure 1 gives an example of Petri net, taken from [20], using a graphical syntax: places are pictured as circles, transitions as squares, there is an arc from place p to transition t if $\mathbf{Pre}(t)(p) > 0$, and one from transition t to place p if $\mathbf{Post}(t)(p) > 0$. The arcs are weighted by the values of the corresponding pre or post conditions (default weight is 1). The initial marking of the net associates integer 1 to place p_0 and 0 to all others.

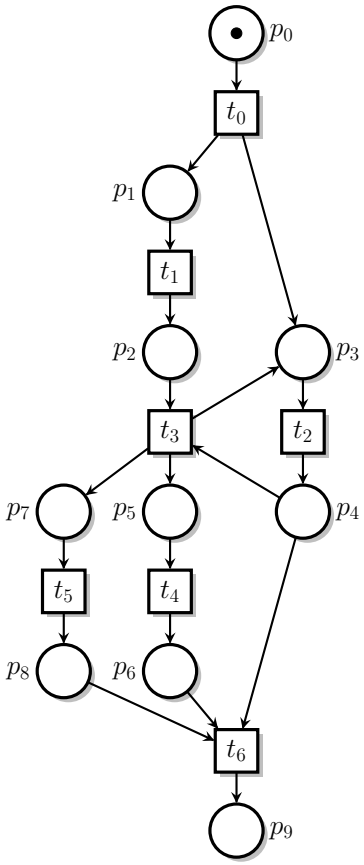


Fig. 1 An example Petri net

A transition t in T is said *enabled* at marking m if $m \geq \mathbf{Pre}(t)$. If enabled at m, transition t may *fire* yielding a marking $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$. This is written $m \xrightarrow{t} m'$, or simply $m \to m'$ when only markings are of interest. Intuitively, places hold integers and together encode the state (or marking) of a net; transitions define state changes.

The *reachability set*, or *state space*, of N is the set of markings $\mathcal{R}(N) = \{\, m \mid m_0 \xrightarrow{*} m \,\}$, where $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$.

A *firing sequence* σ over T is a sequence $t_1 \dots t_n$ of transitions in T such that there are some markings $m_1 \dots, m_{n+1}$ with $m_1 \xrightarrow{t_1} m_2 \wedge \dots \wedge m_n \xrightarrow{t_n} m_{n+1}$. This can be written $m_1 \xrightarrow{\sigma} m_{n+1}$. Its

displacement, or *marking change*, is $\Delta(\sigma) = \Sigma_{i=1}^{n}(\mathbf{Post}(t_i) - \mathbf{Pre}(t_i))$, where $\Delta : T^* \to P \to \mathbb{Z}$, and its *hurdle* $H(\sigma)$, where $H : T^* \to P \to \mathbb{N}$, is the smallest marking (pointwise) from which the sequence is firable. The hurdle of a sequence is ensured to exist and is unique [10]; its coordinates can be computed independently: for any $p \in P$, $t \in T$, $\sigma \in T^*$, we have $H(\lambda)(p) = 0$, $H(\sigma.t)(p) = \mathbf{max}(H(\sigma)(p), \mathbf{Pre}(t)(p) - \Delta(\sigma)(p))$ where λ is the empty sequence.

As an illustration, the displacement of sequence $t_2t_5t_3$ in the net of Fig. 1 is the vector $(0, 0, -1, 0, 0, 1, 0, 0, 0, 1)$, which can be written more compactly as $\{(p_2, -1), (p_5, 1), (p_9, 1)\}$. The hurdle for this sequence is $\{(p_2, 1), (p_3, 1), (p_7, 1)\}$.

The *postset* of a transition t is $t^\bullet = \{\, p \mid \mathbf{Post}(t)(p) > 0 \,\}$, its *preset* is ${}^\bullet t = \{\, p \mid \mathbf{Pre}(t)(p) > 0 \,\}$. Symmetrically for places, $p^\bullet = \{\, t \mid \mathbf{Pre}(t)(p) > 0 \,\}$ and ${}^\bullet p = \{\, t \mid \mathbf{Post}(t)(p) > 0 \,\}$.

A net is *ordinary* if all its arcs have weight one, meaning that for all transition t in T, and place p in P, we have $\mathbf{Pre}(t)(p) \leq 1$ and $\mathbf{Post}(t)(p) \leq 1$. Otherwise it is said *generalized*.

A net N is *bounded* if there is an (integer) bound b such that $m(p) \leq b$ for all $m \in \mathcal{R}(N)$ and $p \in P$. The net is said *safe* when the bound is 1. All nets considered in this paper are assumed bounded.

A net N is *live* if from any $m \in \mathcal{R}(N)$ and any transition $t \in T$ we can always reach a marking where t can fire. Liveness encompasses *pseudo-liveness* (absence of markings from which no transition is firable, also called deadlocks) and *quasi-liveness* (that any $t \in T$ is firable at least once).

For example, the net in Fig. 1 is ordinary and safe, but not live. Its state space holds 14 markings.

Two special cases of nets of interest. In the following, we will refer to two particular (structural) categories of nets. A net is a *State Machine* if it is ordinary and for each $t \in T$, transition t has exactly one input and one output place (${}^\bullet t$ and $t^\bullet$ are singleton sets). Symmetrically, a net is a *Marked Graph* if it is ordinary and for each $p \in P$, ${}^\bullet p$ and $p^\bullet$ are singletons.

Clearly, any State Machine can be seen as a directed graph in which the vertices are the places of the net and its edges are the net transitions, and symmetrically for Marked Graphs. We say that a State Machine (resp. Marked Graph) is strongly connected if its underlying graph is strongly connected.

These categories of nets exhibit some additional properties. For example, a State Machine is necessarily live when it is strongly connected and at least one of its places is marked. A Marking Graph is live iff it does not include a token-free circuit of places, and a live and connected marking graph is necessarily strongly connected [6].

Figure 2 below shows a state machine; a marked graph is represented in Figure 13. Both nets are bounded and live; the first admits 45 reachable markings and the second 25176065 markings.

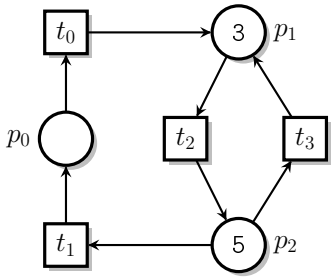


Fig. 2 An example State Machine

3 The Reduction System

We describe our set of reduction rules using three main categories. For each category, we give a property that can be used to recover the state space of a net, after reduction, from that of the reduced net. The set of rules described here enriches that presented in [4].

3.1 Removal of Redundant Transitions

A transition is redundant when its marking change can always be achieved by firing instead an alternative sequence of transitions. Our definition of redundant transitions slightly strengthens that of *bypass* transitions in [17]. It is not fully structural either, but makes it easier to identify special cases structurally.

Definition 1 (T: Redundant transition) Given a net $(P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$, a transition t in T is *redundant* if there is a firing sequence σ over $T \setminus \{t\}$ such that $\Delta(t) = \Delta(\sigma)$ and $H(t) \geq H(\sigma)$. ■

There are special cases that do not require to explore sequences of transitions. This includes *identity* transitions, such that $\Delta(t) = 0$, and *duplicate* transitions, such that for some other transition t' and integer k, $\Delta(t) = k.\Delta(t')$. Finding redundant transitions using Definition 1 can be convenient too, provided the candidate σ are restricted (e.g. in length). Clearly, removing a redundant transition from a net does not change its state space.

Figure 3 shows some examples of redundant transitions. Transitions a and b have null displacement, they obey Definition 1 taking as σ the empty firing sequence. For transition e, observe that $H(e) = H(c)+H(g)+H(d)$, meaning that whenever transition e is firable, any sequence firing once each of transitions c, g, d (in any order) is also firable. Further, since $\Delta(e) = \Delta(c)+\Delta(g)+\Delta(d)$, firing that sequence produces the same marking as firing e. Similarly, whevener transition c is enabled, the sequence $g.g$ is firable and produces the same marking as firing c. Finally, firing d has exactly the same effects on markings than firing sequence $h.f$, and that latter sequence is always firable when d is since $H(d) = H(h.f)$, hence d is redundant.

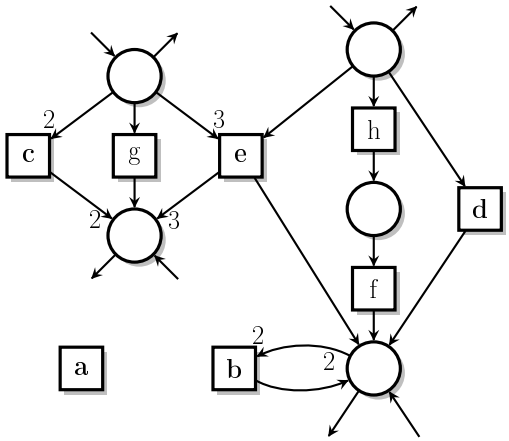
Theorem 1 *If net N' is the result of removing some redundant transition in net N then $\mathcal{R}(N) = \mathcal{R}(N')$*

3.2 Removal of Redundant Places

Intuitively, a place is redundant if removing it from the net would not change its language of firing sequences. But we wish to avoid enumerating marking for detecting such places, and further be able to recover the marking of a redundant place from those of the other places. For these reasons, our definition of redundant places is a slightly strengthened version of that of *structurally redundant* places in [3] (last clause is an equation).

Definition 2 (R: Redundant place) Given a net $(P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$, a place p in P is *redundant* if there is some set of places I from $P \setminus \{p\}$, some valuation $v : (I \cup \{p\}) \to (\mathbb{N} - \{0\})$, and some constant $b \in \mathbb{N}$ such that, for any $t \in T$:

1. The weighted initial marking of p is not smaller than that of I:
   $b = v(p).m_0(p) - \Sigma_{q \in I} v(q).m_0(q)$
2. To fire t, the difference between its weighted precondition on p and that on I may not be larger than b: $v(p).\mathbf{Pre}(t)(p) - \Sigma_{q \in I} v(q).\mathbf{Pre}(t)(q) \leq b$



identity (a,b), duplicate (c)
general redundant (d,e)

Fig. 3 Some examples of redundant transitions

3. When t fires, the weighted growth of the marking of p is equal to that of I:
   $v(p).\Delta(t)(p) = \Sigma_{q \in I} v(q).\Delta(t)(q)$ ■

This definition can be rephrased as an integer linear programming problem [19], convenient in practice for computing redundant places in reasonably sized nets (say when $|P| \leq 50$). Like with redundant transitions, there are special cases that lead to easily identifiable redundant places. These are *constant* places—those for which set I in the definition is empty—and *duplicated* places, when set I is a singleton.

Figure 4 gives some examples of such places. For each reachable marking m of the net in Figure 4, we have $m(a) = 1$, $m(b) = 2$, $m(c) = 2 * m(g) + 3$, $m(d) = m(h) + m(f)$ and $m(e) = 3 * m(g) + m(h) + m(f) + 3$. One will easily check that for each redundant place $r \in \{a, b, c, d, e\}$ and transition t we have $m \backslash r \geq (\mathbf{Pre}(t)) \backslash r \Rightarrow m \geq \mathbf{Pre}(t)$, where $f \backslash r$ is function $f : P \to \mathbb{N}$ restricted to the domain $P - \{r\}$. So removing places a, b, c, d and e does not change the set of firable firing sequences of the net.

From Definition 2, we can show that the marking of a redundant place p can always be computed from the markings of the places in I and the valuation function v. Indeed, for any marking m in $\mathcal{R}(N)$, we have $v(p).m(p) = \Sigma_{q \in I} v(q).m(q) + b$, where the constant b derives from the initial marking m_0. Hence we have a relation $k_p.m(p) = \rho_p(m)$, where $k_p = v(p)$ and ρ_p is some linear expression on the places of the net.

Theorem 2 *If N' is the result of removing some redundant place p from net N, then there is an integer constant $k \in \mathbb{N}^*$, and a linear expression ρ, such that, for all marking m: $m \cup \{(p, (1/k).\rho(m))\} \in \mathcal{R}(N) \Leftrightarrow m \in \mathcal{R}(N')$.*

3.3 Place Agglomerations

Conversely to the rules considered so far, place agglomerations do not preserve the number of markings of the nets they are applied to. They constitute the cornerstone of our reduction system; the purpose of the previous rules is merely to simplify the net so that agglomeration rules can be applied.

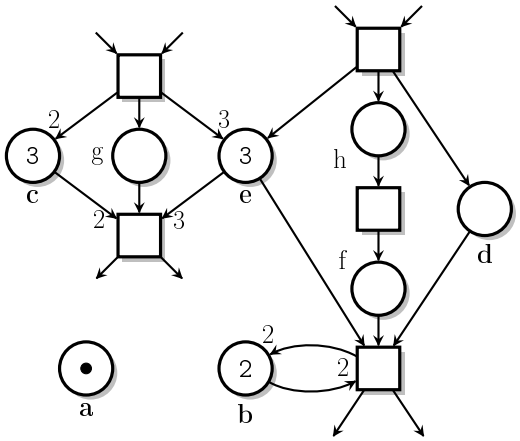


Fig. 4 Some examples of redundant places

We start by introducing a convenient notation.

Definition 3 (Sum of places) A place a is the sum of places p and q, written $a = p \boxplus q$, if: $m_0(a) = m_0(p) + m_0(q)$ and, for all transition t, $\mathbf{Pre}(t)(a) = \mathbf{Pre}(t)(p) + \mathbf{Pre}(t)(q)$ and $\mathbf{Post}(t)(a) = \mathbf{Post}(t)(p) + \mathbf{Post}(t)(q)$. ■

Clearly, operation $\boxplus$ is commutative and associative. Place agglomeration rules will all make use of the following net transformation.

Definition 4 (Substitution of a transition) Assume transition t of net N is such that:

- It has n input places $E = \{p_1, \ldots, p_n\}$ and m output places $F = \{q_1, \ldots, q_m\}$, with $n, m > 0$;
- $E \cap F = \emptyset$;
- All arcs adjacent to t have weight 1.

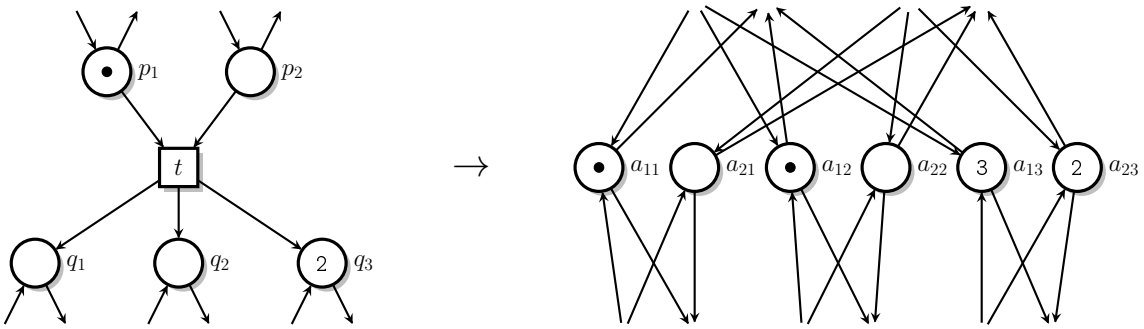
Then substituting transition t in net N consists of replacing the places in $E \cup F$ by those in set $A = \{p_i \boxplus q_j \mid p_i \in E \wedge q_j \in F\}$, and then removing transition t. ■

Witness equations: With each substitution of a transition we will associate an equation system, referred to as its *witness equation system*. That system relates the markings of the places introduced (the agglomeration places) to those of the places removed (the agglomerated places). For readability, the variables representing the markings of places bear the same name as the places they are referred to.

An example transition substitution, together with its witness equation system, is shown in Fig. 5. The place representing $p_i \boxplus q_j$ is named a_{ij}.

We are not interested in all transition substitutions, but only in those such that the reachable markings of the original net (before substitution) can be computed from those of the reduced net (after substitution).

We now define agglomeration rules. There are two categories of *place agglomeration* rules, called *chain agglomerations* and *loop agglomerations*, respectively. Each rule consists of the substitution of some transition(s). In addition, for technical reasons that will be made clear in Sect.



$$a_{11} = p_1 + q_1 \qquad a_{12} = p_1 + q_2 \qquad a_{13} = p_1 + q_3$$
$$a_{21} = p_2 + q_1 \qquad a_{22} = p_2 + q_2 \qquad a_{23} = p_2 + q_3$$

Fig. 5 Substitution of transition t (above) and witness equation system (below)

6, we define two rules in each category: one addressing a particular case, and another, more general, rule.

In all rule definitions we start from a net $N = (P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$. We start with the simpler agglomeration rules:

Definition 5 (SCA: Simple chain agglomeration) If net N contains a transition $t \in T$ and two places p, q in P such that:

1. ${}^\bullet t = \{p\}$, $t^\bullet = \{q\}$ and ${}^\bullet q = \{t\}$
2. $\mathbf{Pre}(t)(p) = \mathbf{Post}(t)(q) = 1$
3. $m_0(q) = 0$

Then transition t is substituted.
The transformation amounts to replace places p and q by a place a equal to their sum: $a = p \boxplus q$ and then removing transition t. ■

Definition 6 (SLA: Simple loop agglomeration) If there is in net N a sequence of n places $(\pi_i)_{i=0}^{n-1}$ such that:

$$(\forall i < n)(\exists t \in T)(\ \mathbf{Pre}(t) = \{(\pi_i, 1)\} \\ \wedge\ \mathbf{Post}(t) = \{(\pi_{(i+1)(\mathrm{mod}\ n)}, 1)\})\ .$$

Then all transitions having an input place and an output place in sequence $(\pi_i)_{i=0}^{n-1}$ are successively substituted. The transformation amounts to replace places $\pi_0, \ldots, \pi_{n-1}$ by a single place, a, defined as their sum: $a = \boxplus_{i=0}^{n-1} \pi_i$, and then removing the transitions linking them (the t transitions in the above condition). ■

Simple agglomerations are illustrated in Fig. 6.

Clearly, whenever some place a of a net obeys $a = p \boxplus q$ for some places p and q of the same net, then place a is redundant in the sense of definition 2. The effects of simple agglomerations on markings are stated by Theorem 3.

Theorem 3 *Let N and N' be the nets before and after simple agglomeration of some set of places A as place a. Then for all markings m over $(P \setminus A)$ and m' over A we have: $(m \cup m') \in \mathcal{R}(N) \Leftrightarrow m \cup \{(a, \Sigma_{p \in A} m'(p)\} \in \mathcal{R}(N')$.*

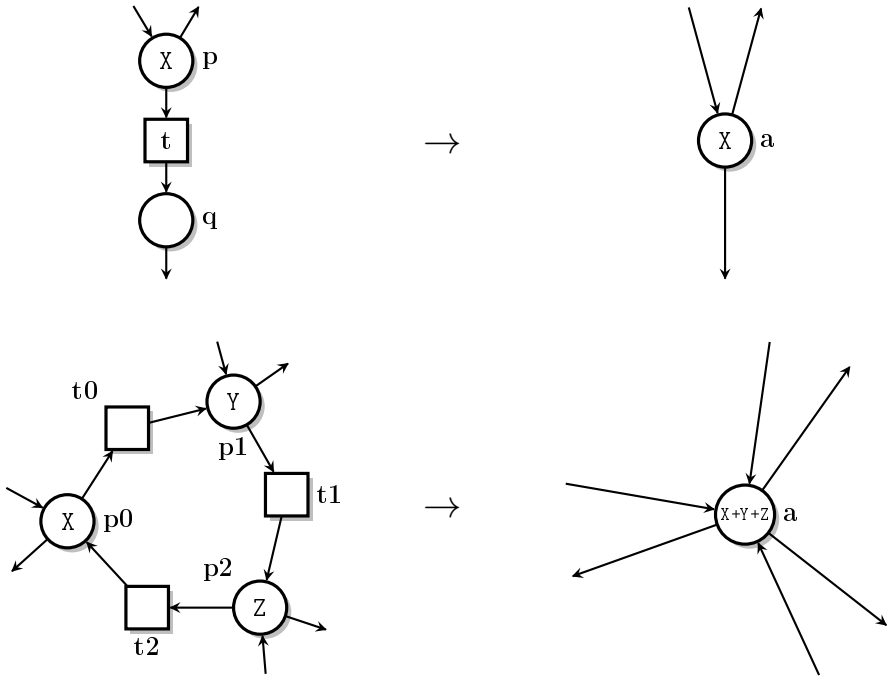


Fig. 6 Simple agglomeration examples: chain (top), loop (for $n = 3$, bottom)

Proof Assume N is a net with set of places P.
1. Let us first consider the case of SLA, the chain agglomeration rule in Fig. 6 (top). We have to prove that for all marking m of $P \setminus \{p, q\}$ and for all values x, y in $\mathbb{N}$:

$$m \cup \{(p, x), (q, y)\} \in \mathcal{R}(N) \Leftrightarrow m \cup \{(a, x + y)\} \in \mathcal{R}(N')$$

Left to right (L): Let N^+ be net N with place $a = p \boxplus q$ added. Clearly, a is redundant in N^+, with $v(a) = v(p) = v(q) = 1$. So N and N^+ admit the same firing sequences, and for any $m \in \mathcal{R}(N^+)$, we have $m(a) = m(p) + m(q)$. Next, removing places p and q from N^+ (the result is net N') can only relax firing constraints, hence any σ firable in N^+ (and thus in N) is also firable in N', which implies the goal.

Right to left (R): we use two intermediate properties ($\forall m, x, u, v$ implicit). We write $m \sim m'$ when m and m' agree on all places except p, q and a, and $m \approx m'$ when $m \sim m' \wedge m(p) = m'(a) \wedge m(q) = 0$.

Property (1): $m \cup \{(a, x)\} \in \mathcal{R}(N') \Rightarrow m \cup \{(p, x), (q, 0)\} \in \mathcal{R}(N)$.

Since $\Delta(t) = 0$, any marking reachable in N' is reachable by a sequence not containing t, call these sequences t-free. Property (1) follows from a simpler relation, namely (Z): *whenever* $m \approx m'$ *(*$m \in \mathcal{R}(N)$*,* $m' \in \mathcal{R}(N')$*) and* $m' \xrightarrow{\delta} w'$*, (*$\delta$ *t-free), then there is a sequence* ω *such that* $m \xrightarrow{\omega} w$ *and* $w \approx w'$.

Any t-free sequence firable in N' but not in N can be written $\sigma.t'.\gamma$, where σ is firable in N and transition t' is not firable in N after σ. Let w, w' be the markings reached by σ in N and N', respectively. Since σ is firable in N, we have $w \approx w'$, by (L) and the fact that σ is t-free (only

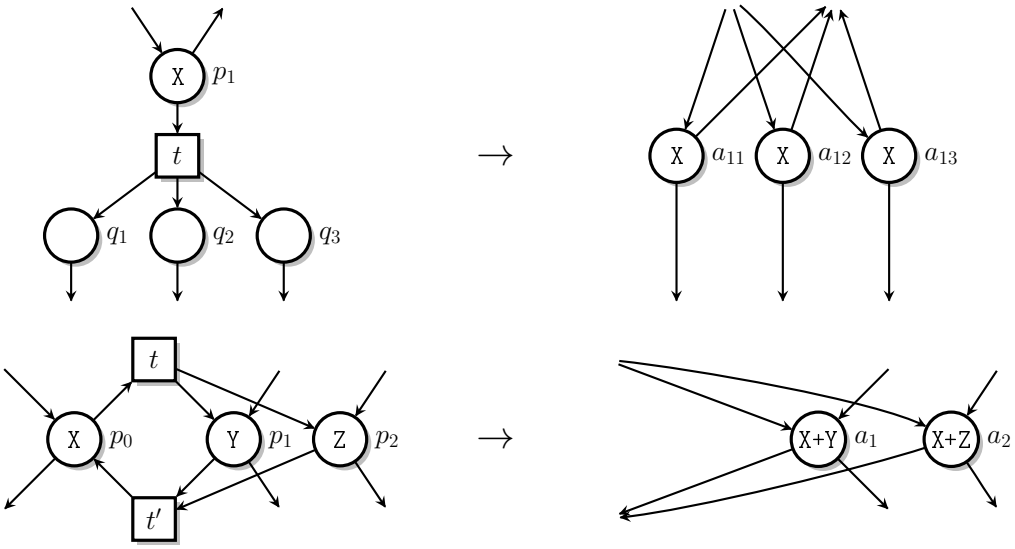


Fig. 7 General agglomeration examples: chain (top), loop (for $n = 2$, bottom)

t can put tokens in q). That t' is not firable at w but firable at w' is only possible if t' is some output transition of a since $w \sim w'$ and the preconditions of all other transitions of N' than a are identical in N and N'. That is, t' must be an output transition of either or both p or q in N. If t' has no precondition on q in N, then it ought to be firable at w in N since $w(p) = w'(a)$. So t' must have a precondition on q; we have $w(q) \not\geq \mathbf{Pre}(t')(q)$ in N and $w'(a) \geq \mathbf{Pre}'(t')(a)$ in N'. Therefore, we can fire transition t n times from w in N, where $n = \mathbf{Pre}(t')(q)$, since $w'(a) = w(p)$ and t' is enabled at w', and this leads to a marking enabling t'. Further, firing t' at that marking leaves place q in N empty since only transition t may put tokens in q. Then the proof of Property (1) follows from (Z) and the fact that Definition 5 ensures $m_0 \approx m'_0$.

Property (2): if $m \cup \{(p, x), (q, 0)\} \in \mathcal{R}(N)$ and $(u + v = x)$ then $m \cup \{(p, u), (q, v)\} \in \mathcal{R}(N)$.

Obvious from Definition 5: the tokens in place p can be moved one by one into place q by firing t in sequence v times.

Combining Property (1) and (2) is enough to prove (R), which completes the proof for chain agglomerations.

2. For loop agglomerations (Fig. 6 (bottom), for instance) observe that, in net N, tokens may freely flow in the places of A by firing only the transitions substituted in Defn. 6. So, if the set of places A is marked with k tokens and the other places are marked as in m (say), then any marking putting k tokens in the places in $(\pi_i)_{i=0}^{n-1}$ and leaving the other places unchanged (marked as in m) is reachable in N; which is exactly what we had to prove for the case SLA. □

We now define generalizations of both simple agglomeration rules, still starting from a net $N = (P, T, \mathbf{Pre}, \mathbf{Post}, m_0)$.

Definition 7 (GCA: Generalized chain agglomeration) If there is in net N a place p, a set $F \subseteq P$ and some $t \in T$ such that:

1. ${}^\bullet t = \{p\}$; $t^\bullet = F$; $p \notin F$, $\mathbf{Pre}(t)(p) = 1$;
2. for any $q \in F$: $\mathbf{Post}(t)(q) = 1$; ${}^\bullet q = \{t\}$ and $m_0(q) = 0$.

Then transition t is substituted. ■

Transition circuits will help defining the richer loop agglomerations.

Definition 8 (transition circuit) Let a *transition circuit* in net N be a sequence of transitions $(\tau_i)_{i=0}^{n-1}$ such that, for any $i, j < n$, $i \neq j$ and any $p \in P$:

- $\mathbf{Post}(\tau_i) = \mathbf{Pre}(\tau_{(i+1) \bmod n})$
- $\mathbf{Pre}(\tau_i)(p) \leq 1 \wedge \mathbf{Post}(\tau_i)(p) \leq 1$
- $\tau_i^\bullet$ and $\tau_j^\bullet$ are disjoint. ■

Clearly, if some transition τ_i in a transition circuit is enabled at some marking m, then the transition sequence $(\tau_j)_{j=i}^{(i+n)(\bmod n)}$, including exactly all transitions of the circuit, is firable at m and is cyclic.

Definition 9 (GLA: Generalized loop agglomeration) If there is in net N a sequence of n subsets of places $(\Pi_i)_{i=0}^{n-1}$ and a circuit $(\tau_i)_{i=0}^{n-1}$ of transitions such that:
$(\forall i < n)(\Pi_i = \mathbf{Pre}(\tau_i) \wedge \Pi_{(i+1) \bmod n} = \mathbf{Post}(\tau_i))$
Then all transitions in circuit $(\tau_i)_{i=0}^{n-1}$ are successively substituted.

The transformation amounts to add for each tuple $\{z_0, \cdots, z_{n-1}\} \in \Pi_0 \times \cdots \times \Pi_{n-1}$ a place equal to $z_0 \boxplus \cdots \boxplus z_{n-1}$ and then removing the places in $(\Pi_i)_{i=0}^{n-1}$ and the transitions in circuit $(\tau_i)_{i=0}^{n-1}$. ■

Theorem 4 *Let N and N' be the nets before and after application of rule GCA or GLA.*

Let E be the set of places in N' but not in N (the agglomeration places introduced by the rule), and $A : E \to P$ the function associating with each agglomeration place the set of places of N it agglomerates. Function A is directly derived from the witness equation system produced by the rule applied.

Let $AA = \bigcup_{i \in E}(A(i))$.

Then for all markings m over $(P \setminus AA)$ and m' over AA we have: $(m \cup m') \in \mathcal{R}(N) \Leftrightarrow m \cup (i, \Sigma_{p \in A(i)} m'(p))_{i \in E} \in \mathcal{R}(N')$.

Proof

1. case GCA

Generalized chain agglomerations differ from simple agglomerations by the fact that the transition substituted may have several output places. Their effects are that they simultaneously agglomerate each output place of the substituted transition with a copy of its input place.

The proof scheme for this case is essentially similar to that of case SLA in the proof of Theorem 3, except that we have to consider the several agglomerations simultaneously.

2. case GLA

Consider the subnet L of N constituted of the places in $(\Pi_i)_{i=0}^{n-1}$, the transitions in $(\tau_i)_{i=0}^{n-1}$ and the arcs connecting them. Clearly, the case to be proven in net N is true if it is true in subnet L; we prove it in subnet L.

Next, by construction, all places in each Π_i in subnet L have the save input transition and the same output transition. So in each Π_i, all places are redundant except one with the smallest initial marking. Without loss of generality, assume the non redundant places are the first place of each Π_i, let us denote them $\Pi_0^0, \cdots, \Pi_{n-1}^0$. Further, still by construction, all arcs of subnet L have weight 1. So the markings of all redundant places in each Π_i only differ from that of the non redundant one Π_i^0 by some added constant.

In the reduced subnet, there is by construction a place summing the non redundant places of each Π_i: $a_{0,\cdots,0} = \Pi_1^0 \boxplus \cdots \boxplus \Pi_{n-1}^0$. By Theorem 3 (case SLA), we have that for any marking k of that place, and any integers $k_0, \cdots, k_{n-1}$ such that their sum is k, there is a marking in the initial net such that place Π_i^0 is marked with k_i tokens, for each i.

Let us consider now the redundant places. It is easily seen that any place in the reduced subnet resulting from a sum of places of the initial net involving a redundant place of some Π_i is redundant versus place $a_{0,\cdots,0}$ in the reduced L and that its markings only differ from that of $a_{0,\cdots,0}$ by some added constant. So the proposition holds for the redundant places too since, in the initial subnet L and its reduction, their markings are uniquely determined from those of the non redundant places. □

3.4 The Reduction System

We say that a net is *totally reduced* when its set of places and transitions are empty ($P = T = \emptyset$).

The three categories of rules introduced in the previous sections constitute the core of our reduction system. Our implementation actually adds to those a few special purpose rules. We mention three examples of such rules here, because they play a significant role in the experimental results of Sect. 7, but without technical details. The first two are useful on nets generated from high level descriptions, that often exhibit translation artifacts like dead transitions or source places.

The first extra rule is the *dead transition removal* rule. It is sometimes possible to determine statically that some transitions of a net are never firable. A fairly general rule for identifying statically dead transitions is proposed in [7]. Removal of statically dead transitions from a net has no effects on its state space.

A second rule allows one to remove a transition t from a net N when t is the sole transition enabled at the initial marking and t is firable only once. Then, instead of counting the markings reachable from the initial marking of the net, we count those reachable from the output marking of t in N then add 1. Removing such transitions often yields structurally simpler nets.

The rules in the third group can be used to do away with nets or subnets for which an equational description of their reachable markings can be computed by other means than reductions.

The first handles some particular nets or subnets containing only a single place.

Definition 10 (SSP: Source-sink pair) A pair (p, t) in net N is a *source-sink pair* if ${}^\bullet p = \emptyset$, $p^\bullet = \{t\}$, $\mathbf{Pre}(t) = \{(p, 1)\}$ and $\mathbf{Post}(t) = \emptyset$.

Remove any source-sink pair from the net and add to the inequality system the inequality $p \leq m_0(p)$. ■

Theorem 5 (Source-sink pairs) *If N' is the result of removing a source-sink pair (p, t) in net N then $(\forall z \leq m_0(p))(\forall m)(m \cup \{(p, z)\} \in \mathcal{R}(N) \Leftrightarrow m \in \mathcal{R}(N'))$.*

Besides trivial nets, there are at least two subclasses of nets for which one can compute an equational description of their reachability set, these are the live state machines and the live marked graphs.

Concerning live state machines, note that they are already totally reduced by rule SLA (Defn. 6) and removal of a constant place, so no additional rule is needed to handle them.

On the other hand live marked graphs are not generally totally reduced by our system. But it is known that the reachability set of a live marked graph is the solution set of the equation system constituting its basis of place-invariants. Those equations can be efficiently obtained from a basis of circuits of the underlying graph and the initial marking of the net. Hence we added a rule capturing the case where the residual net is a live marked graph.

Definition 11 (LMG: Live Marked Graph) When net N is a live marked graph, add to the equation system its basis of place invariants, then remove the whole net. ■

Omitting for the sake of clarity the first two extra rules mentioned above, our final reduction system resumes to removal of redundant transitions (referred to as the T rule), removal of redundant places (R rule), agglomerations of places, and application of the above two "do-away" rules.

Rule T has no effects on markings. For the other three rules, the effect on the markings is captured by a system of equations or inequalities. In these systems, the variables are marking variables. For readability, we will typically use the name of the place instead of its associated marking variable. For instance, the marking equation $2.m(p) = 3.m(q) + 4$, resulting from application of rule (R), would be simply written $2.p = 3.q + 4$.

We will refer in the following sections to three particular subsets of the reduction rules, defined as follows:

- The `clean` rule set, constituted of rules R and T;
- The `compact` rule set, adding to the `clean` set rules SCA, SLA and SSP;
- The `compact+` rule set, adding to the `clean` set rules GCA, GLA, SSP and LMG.

We show in Sect. 5 that the state space of a net can be reconstructed from that of its reduced net and the set of inequalities collected when rules are applied. Before considering this result, we illustrate the effects of reductions on two full examples.

4 Two illustrative examples

We illustrate our approach on two examples of Petri net taken from the *Model Checking Contest* (MCC, `http://mcc.lip6.fr`), a recurring competition of model-checking tools [8].

4.1 HouseConstruction-10

Our first example is a variation of a Petri net model found in [16], which is itself derived from the PERT chart of the construction of a house found in [13]. The model found in the MCC collection, reproduced in Fig. 8, differs from that of [16] in that it omits time constraints and a final sink place. In addition, the net represents the house construction process for a number of houses simultaneously rather than a single one. The number of houses being built is represented by the marking of place p_1 of the net (10 in the net represented in Fig. 8).

This example is totally reduced by rule set `compact`. We list in Fig. 9 a possible reduction sequence for our example, where each line describes one application of a rule. To save space, we have omitted the removal of redundant transitions. For each reduction, we give an indication of its kind (the same names given to the reduction rules in Sect. 3), the resulting marking equation witnessing the reduction, and a short description. The first reduction, for instance, states that place p_{19} is removed, being a duplicate of place p_{20}. At the second step, places $p11$ and $p7$ are agglomerated into a "fresh" place $a1$.

Each reduction is associated with an equation or inequality linking the markings of the net before and after application of a rule. The system of inequalities gathered is shown in Fig. 10, with agglomeration places a_i eliminated. We show in the next section that the set of solutions of this system, taken as markings, is exactly the set of reachable markings of the net.

4.2 SmallOperatingSystem-8192-4096

Our second example is shown in Fig. 11. It abstracts the lifecycle of a task in a simplified operating system handling the execution of tasks on a machine with several memory segments,

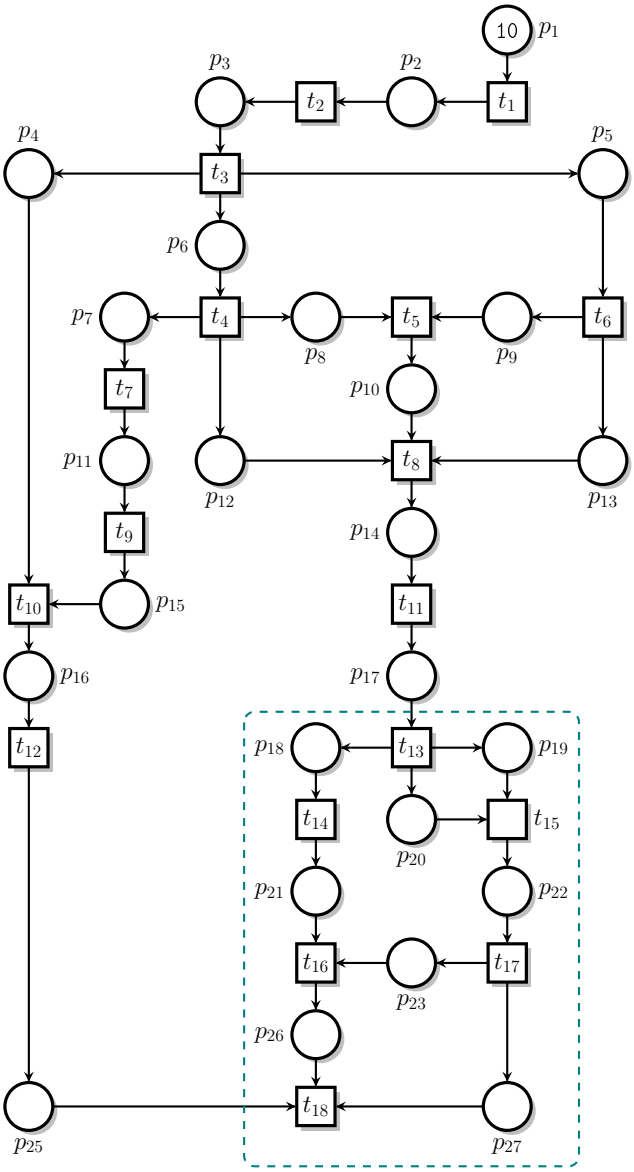


Fig. 8 HouseConstruction-10 example net

.

disk controller units, and cores. The initial marking of the net gives the number of resources available (e.g. there are 8192 available memory segments in our example).

This net is totally reduced by the `compact+` strategy. The reduction traces are shown in Fig. 12.

There are two interesting examples of reductions in Fig. 12 that feature the effectiveness of strategy `compact+`.

A first example is the application of rule GLA that detected a (general) agglomeration loop of size 2 involving places $CPUUnit$, $TaskSuspended$ and $ExecutingTask$. (For added legibility, we have colored the places and transitions involved in the figure). In this context, transition

```
R   |- p19 = p20          p19 duplicate
SCA |- a1 = p11 + p7      agglomeration
SCA |- a2 = p17 + p14     agglomeration
SCA |- a3 = p2 + p1       agglomeration
SCA |- a4 = p21 + p18     agglomeration
SCA |- a5 = p22 + p20     agglomeration
SCA |- a6 = p25 + p16     agglomeration
SCA |- a7 = p15 + a1      agglomeration
SCA |- a8 = p3 + a3       agglomeration
R   |- p12 = p10 + p8     p12 redundant
R   |- p13 = p10 + p9     p13 redundant
R   |- a4 = a5 + p23      a4 redundant
R   |- p27 = p23 + p26    p27 redundant
R   |- p4 = p6 + a7       p4 redundant
SCA |- a9 = a2 + p10      agglomeration
SCA |- a10 = a6 + a7      agglomeration
SCA |- a11 = p23 + a5     agglomeration
SCA |- a12 = p9 + p5      agglomeration
SCA |- a13 = a11 + p26    agglomeration
SCA |- a14 = a13 + a9     agglomeration
R   |- a12 = p6 + p8      a12 redundant
R   |- a10 = a14 + p8     a10 redundant
SCA |- a15 = a14 + p8     agglomeration
SCA |- a16 = p6 + a8      agglomeration
SCA |- a17 = a15 + a16    agglomeration
SSP |- a17 <= 10          a17 source
```

Fig. 9 Reduction traces for net HouseConstruction-10

$$
\begin{aligned}
p19 &= p20\\
p4 &= p6 + p15 + p11 + p7\\
p12 &= p10 + p8\\
p9 + p5 &= p6 + p8\\
p13 &= p10 + p9\\
p21 + p18 &= p22 + p20 + p23\\
p27 &= p23 + p26\\
p25 + p16 + p15 + p11 + p7 &=\\
p26 + p23 + p22 + p20 + p17 + p14 + p10 + p8 &\\
p26 + p23 + p22 + p20 + p17 + p14 + p10 + p8 + &\\
p6 + p3 + p2 + p1 &\leq 10
\end{aligned}
$$

Fig. 10 HouseConstruction-10 inequality system.

startNext plays the role of transition t in the GLA rule depicted in Fig. 7 and *suspend* plays the role of t'. After reduction, we introduce two new places, a_1 and a_2, in place of the initial three.

Our second example is the last rule of the reduction; an occurrence of rule LMG (see Definition 11). The reduced net obtained just before the last reduction step is shown in Fig. 13. This residual net is clearly a live marked graph. Its basis of marking invariants is added to the equations by rule LMG. The final simplified equation system is shown in Fig. 14.

These examples are instances of totally reducible nets. We have found many other examples of totally reducible nets in the MCC benchmarks. In the general case, our reduction system is not complete however; some nets may be only partially reduced, or not at all. When a net is only partially reducible, the inequalities, together with an explicit or logic-based symbolic description of the reachability set of the residual net, yield a hybrid representation of the state space of the initial net. Such hybrid representations are certainly less convenient than a full equational representation but are nonetheless suitable for model checking reachability properties or counting markings.

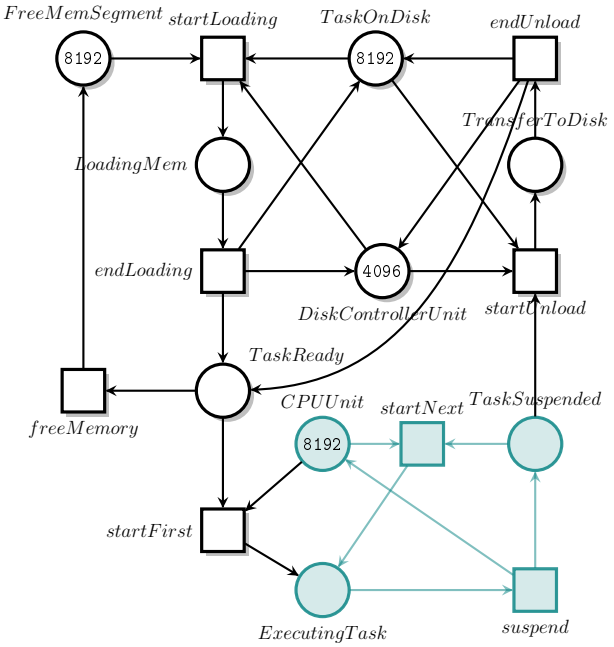


Fig. 11 Small Operating System Petri net

```
R   |- TaskOnDisk = DiskControllerUnit + 4096
GLA |- a1 = CPUUnit + ExecutingTask
       a2 = TaskSuspended + ExecutingTask
R   |- a1 = 8192
SCA |- a3 = a2 + TaskReady
GLA |- a4 = DiskControllerUnit + TransferToDisk
       a5 = a3 + TransferToDisk
LMG |- a4 + LoadingMem = 4096
       FreeMemSegment + LoadingMem + a5 = 8192
```

Fig. 12 Reduction traces for net SmallOperatingSystem

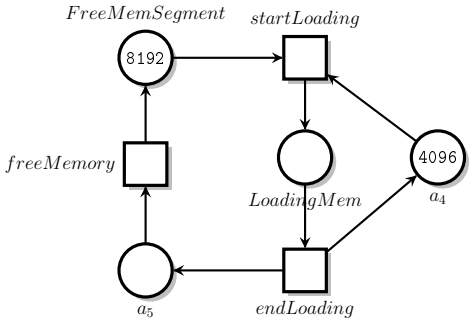


Fig. 13 Residual net before application of rule LMG

$$
\begin{aligned}
DiskControllerUnit + TransferToDisk + \\
LoadingMem &= 4096 \\
FreeMemSegment + LoadingMem + \\
TaskSuspended + ExecutingTask + \\
TaskReady + TransferToDisk &= 8192 \\
CPUUnit + ExecutingTask &= 8192 \\
TaskOnDisk - DiskControllerUnit &= 4096
\end{aligned}
$$

Fig. 14 SmallOperatingSystem equation system.

5 Correctness of Markings Reconstruction

5.1 Net-abstractions and correctness

We prove that we can reconstruct the markings of an (initial) net, before application of a rule, from that of the reduced net. This property ensues from the definition of a new relation, the *net-abstraction* relation, which we define below. For readability, we prove this result for the `compact` rule set; the proof for rule set `compact+` would be along the same lines.

We start by defining some notations useful in our proofs. We use $\mathcal{U}, \mathcal{V}, \ldots$ for finite sets of non-negative integer variables. We use Q, Q' for systems of linear equations (and inequalities) and the notation $\mathbf{V}(Q)$ for the set of variables occurring in Q. The system obtained by concatenating the relations in Q_1 and Q_2 is denoted $(Q_1; Q_2)$ and the "empty system" is denoted $\emptyset$.

Let $\mathcal{V}$ be the set of variables occurring in Q, that is $\mathcal{V} = \mathbf{V}(Q)$. A valuation e of $\mathbb{N}^{\mathcal{V}}$ is a function from $\mathcal{V}$ to natural numbers. It is a solution of Q if all the relations in Q are (trivially) valid when replacing all variables x in $\mathcal{V}$ by their value $e(x)$. We denote $\langle Q \rangle$ the subset of $\mathbb{N}^{\mathcal{V}}$ composed of all the solutions of Q.

If $E \subseteq \mathbb{N}^{\mathcal{V}}$ and $\mathcal{U} \subseteq \mathcal{V}$, then $E \downarrow \mathcal{U}$ is the projection of E over variables $\mathcal{U}$, that is the subset of $\mathbb{N}^{\mathcal{U}}$ obtained from E by restricting the domain of its elements to $\mathcal{U}$. Conversely, if $\mathcal{U} \supseteq \mathcal{V}$, we use $E \uparrow \mathcal{U}$ to denote the lifting of E to $\mathcal{U}$, that is the largest subset E' of $\mathbb{N}^{\mathcal{U}}$ such that $E' \downarrow \mathcal{V} = E$.

Definition 12 (Net-abstraction) A triple (N_1, Q, N_2) is a *net-abstraction*, or simply an abstraction, if N_1, N_2 are nets with respective sets of places P_1, P_2 (we may have $P_1 \cap P_2 \neq \emptyset$), Q is a linear system of equations, and:

$$
\begin{aligned}
\mathcal{R}(N_1) = ((\mathcal{R}(N_2) \uparrow \mathcal{V}) \cap (\langle Q \rangle \uparrow \mathcal{V})) \downarrow P_1 \\
\text{where } \mathcal{V} = \mathbf{V}(Q) \cup P_1 \cup P_2 \ .
\end{aligned}
$$

Intuitively, N_2 is an abstraction of N_1 (through Q) if, from every reachable marking $m \in \mathcal{R}(N_2)$, the markings obtained from solutions of Q—restricted to those solutions such that $x = m(x)$ for all "place variable" x in P_2—are always reachable in N_1. The definition also entails that all the markings in $\mathcal{R}(N_1)$ can be obtained this way.

Theorem 6 (Net-abstractions from reductions) *For any nets N, N_1, N_2:*

1. *$(N, \emptyset, N)$ is an abstraction;*
2. *If (N_1, Q, N_2) is an abstraction then (N_1, Q', N_3) is an abstraction if either:*
   - *(T) $Q' = Q$ and N_3 is obtained from N_2 by removing a redundant transition (see Sect. 3.1);*
   - *(R) $Q' = (Q; k.p = l)$ and N_3 is obtained from N_2 by removing a redundant place p and $k.p = l$ is the associated marking equation (see Sect. 3.2);*
   - *(A) $Q' = (Q; a = \Sigma_{p \in A}(p))$, where $a \notin \mathbf{V}(Q)$ and N_3 is obtained from N_2 by agglomerating the places in A as a new place, a (see Sect. 3.3);*
   - *(L) $Q' = (Q; p \leq k)$ and N_3 is obtained from N_2 by removal of a source-sink pair (p, t) with $m_0(p) = k$ (see Sect. 3.4).*

Proof Property (1) is obvious from Definition 12. Property (2) is proved by case analysis. First, let $\mathcal{V} = \mathbf{V}(Q) \cup P_1 \cup P_2$ and $\mathcal{U} = \mathcal{V} \cup P_3$ and notice that for all candidate (N_1, Q', N_3) we have $\mathbf{V}(Q') \cup P_1 \cup P_3 = \mathcal{U}$. Then, in each case, we know $(H) : \mathcal{R}(N_1) = (\mathcal{R}(N_2) \uparrow \mathcal{V} \cap \langle Q \rangle \uparrow \mathcal{V}) \downarrow P_1$ and we must prove $(G) : \mathcal{R}(N_1) = (\mathcal{R}(N_3) \uparrow \mathcal{U} \cap \langle Q' \rangle \uparrow \mathcal{U}) \downarrow P_1$.

Case (T) : $Q' = Q$. By Th. 1, we have $P_3 = P_2$, hence $\mathcal{V} = \mathcal{U}$, and $\mathcal{R}(N_3) = \mathcal{R}(N_2)$. Replacing $\mathcal{R}(N_2)$ by $\mathcal{R}(N_3)$ and $\mathcal{V}$ by $\mathcal{U}$ in (H) yields (G).

Case (R) : By Th. 2 we have : $\mathcal{R}(N_2) = \mathcal{R}(N_3) \uparrow P_2 \cap \langle k.p = l \rangle \uparrow P_2$. replacing $\mathcal{R}(N_2)$ by this value in (H) yields $\mathcal{R}(N_1) = ((\mathcal{R}(N_3) \uparrow P_2 \cap \langle k.p = l \rangle \uparrow P_2) \uparrow \mathcal{V} \cap \langle Q \rangle \uparrow \mathcal{V}) \downarrow P_1$. Since $P_2 \subseteq \mathcal{V}$, we may safely lift to $\mathcal{V}$ instead of P_2, so: $\mathcal{R}(N_1) = (\mathcal{R}(N_3) \uparrow \mathcal{V} \cap \langle k.p = l \rangle \uparrow \mathcal{V} \cap \langle Q \rangle \uparrow \mathcal{V}) \downarrow P_1$. Which is equivalent to: $\mathcal{R}(N_1) = (\mathcal{R}(N_3) \uparrow \mathcal{V} \cap \langle Q; k.p = l \rangle \uparrow \mathcal{V}) \downarrow P_1$, and equal to (G) since $P_3 \subseteq \mathcal{V}$ and $Q' = (Q; k.p = l)$.

Case (A): Let S_p denotes the value $\Sigma_{p \in A}(p)$. By Th. 3 we have: $\mathcal{R}(N_2) = (\mathcal{R}(N_3) \uparrow (P_2 \cup P_3) \cap \langle a = S_p \rangle \uparrow (P_2 \cup P_3)) \downarrow P_2$. Replacing $\mathcal{R}(N_2)$ by this value in (H) yields: $\mathcal{R}(N_1) = (((\mathcal{R}(N_3) \uparrow (P_2 \cup P_3) \cap \langle a = S_p \rangle \uparrow (P_2 \cup P_3)) \downarrow P_2) \uparrow \mathcal{V} \cap \langle Q \rangle \uparrow \mathcal{V}) \downarrow P_1$. Instead of $\mathcal{V}$, we may lift to $\mathcal{U}$ since $\mathcal{U} = \mathcal{V} \cup \{a\}$, $a \notin \mathbf{V}(Q)$ and $a \notin P_1$, so: $\mathcal{R}(N_1) = (((\mathcal{R}(N_3) \uparrow (P_2 \cup P_3) \cap \langle a = S_p \rangle \uparrow (P_2 \cup P_3)) \downarrow P_2) \uparrow \mathcal{U} \cap \langle Q \rangle \uparrow \mathcal{U}) \downarrow P_1$. Projection on P_2 may be omitted since $P_2 \cup P_3 = P_2 \cup \{a\}$ and $a \notin \mathbf{V}(Q)$, leading to:

$\mathcal{R}(N_1) = ((\mathcal{R}(N_3) \uparrow (P_2 \cup P_3) \cap \langle a = S_p \rangle \uparrow (P_2 \cup P_3)) \uparrow \mathcal{U} \cap \langle Q \rangle \uparrow \mathcal{U}) \downarrow P_1$.

Since $P_2 \cup P_3 \subseteq \mathcal{U}$, this is equivalent to: $\mathcal{R}(N_1) = (\mathcal{R}(N_3) \uparrow \mathcal{U} \cap \langle a = S_p \rangle \uparrow \mathcal{U} \cap \langle Q \rangle \uparrow \mathcal{U}) \downarrow P_1$. Grouping equations yields: $\mathcal{R}(N_1) = (\mathcal{R}(N_3) \uparrow \mathcal{U} \cap \langle Q; a = S_p \rangle \uparrow \mathcal{U}) \downarrow P_1$, which is equal to (G) since $Q' = (Q; a = S_p)$.

case (L): The proof is similar to that of case (R) and is based on the relation $\mathcal{R}(N_2) = \mathcal{R}(N_3) \uparrow P_2 \cap \langle p \leq k \rangle \uparrow P_2$, obtained from Th. 5. □

Theorem 6 states the correctness of our reduction systems, since we can compose reductions sequentially and always obtain a net-abstraction. In particular, if a net N is fully reducible, then we can derive a system of linear equations Q such that $(N, Q, \emptyset)$ is a net-abstraction. In this case the reachable markings of N are exactly the solutions of Q, projected on the places of N. If the reduced net, say N_r, is not empty then each marking $m \in \mathcal{R}(N_r)$ represents a set of markings $\langle Q \rangle_m \subset \mathcal{R}(N)$: the solution set of Q in which the places of the residual net are constrained as in m, and then projected on the places of N. Moreover the family of sets $\{ \langle Q \rangle_m \mid m \in \mathcal{R}(N_r) \}$ is a partition of $\mathcal{R}(N)$.

5.2 Order of application of rules and confluence

Our reduction system does not constrain the order in which reductions are applied. Instead, our tool attempts to apply them in an order that minimizes reduction costs.

Rules can be classified into "local" rules, detecting some structural patterns on the net and transforming them (like removal of duplicate transitions or places, or chain agglomerations), and ''non-local" rules, like removal of redundant places in the general case (using integer programming). Our implementation defers the application of the non-local rules until no more local rule can be applied. This decreases the cost of non-local reductions as they are applied to smaller nets.

Another issue is the confluence of the rules. Our reduction systems `compact` and `compact+` are not confluent: different reduction sequences for the same net could yield different residual nets. For rule set `compact` this follows from the fact that agglomeration rules do not preserve in general the *ordinary* character of the net (that all arcs have weight 1), while agglomeration rules require that the candidate places are connected by arcs of weight 1 to the same transition.

An example net exhibiting the problem is shown in Fig. 15(a). Agglomeration of places $p3$ and $p4$ in this net by rule SLA yields the net in Fig. 15(b). Place $a1$ in the reduced net is the result of agglomerating $p3$ and $p4$; this is witnessed by equation $a1 = p3 + p4$. Note that the arcs connecting place $a1$ to transitions $t0$ and $t1$ both have weight 2.

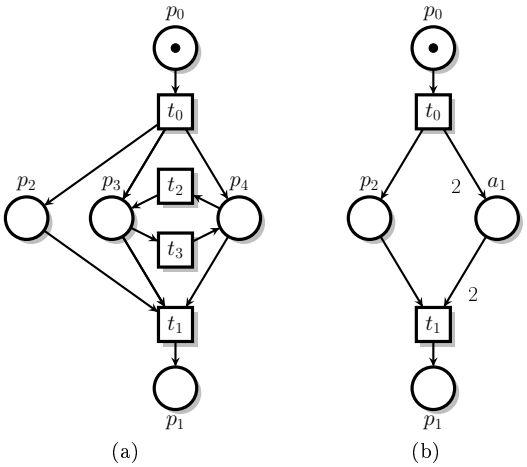


Fig. 15 Non confluence of rule set `compact`

Next, place $p2$ in the reduced net is a duplicate of place $a1$, according to the definitions of Sect. 3.2, the corresponding equation is $2.p2 = a1$. But, from the same equation, $a1$ is a duplicate of $p2$ as well. But removing $p2$ or $a1$ have different effects:

- If $a1$ is removed, then we can fully reduce the net by the following sequence of reductions:

```
SCA |- a2 = p1 + p2  Simple chain agglomeration
SCA |- a3 = a2 + p0  Simple chain agglomeration
R   |- a3 = 1        constant place
```

- If $p2$ is removed instead, then the resulting net cannot be reduced further: places $p0$, $a1$ and $p1$ cannot be agglomerated because of the presence of arcs with weight larger than 1.

So, rule set `compact` is not confluent. But the lack of confluence in this case is due to a self-imposed limit of our agglomeration reductions. This constraint could be slightly relaxed, taking advantage of the fact that multiplying or dividing by some constant the edges adjacent to a place and its marking would not change the number of markings. Handling some patterns of weighted arcs in agglomerations is indeed a scheduled extension to our agglomeration rules.

But even if agglomerations were not introducing weighted arcs, rule set `compact+` would not be confluent, as illustrated by the next example.

By the following reduction sequence, the net in
Fig. 16(a) reduces to the net of Fig. 16(b), which is irreducible by our strongest system `compact+`.

```
SCA |- a1 = p2 + p3  Simple chain agglomeration
SCA |- a2 = p1 + a1  Simple chain agglomeration
GCA |- a3 = p4 + a2  General chain agglomeration
       a4 = p5 + a2
```

But by the following alternative reduction sequence, the same net reduces to the net of Fig. 16(c), also irreducible and different from the net of Fig. 16(b).

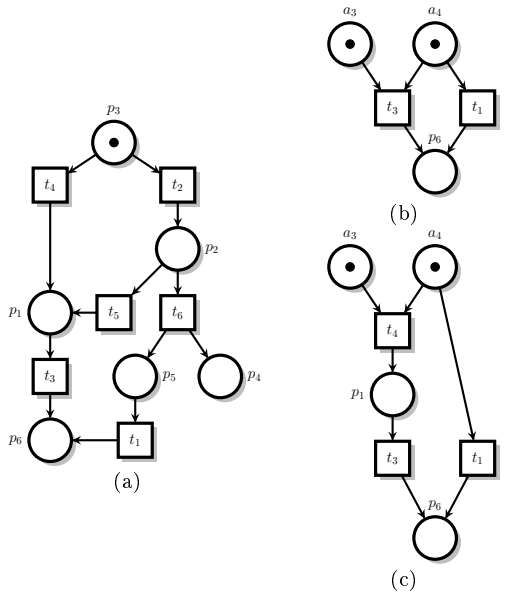


Fig. 16 Non confluence of rule set `compact+`

```
GCA |- a1 = p4 + p2  General chain agglomeration
       a2 = p5 + p2
GCA |- a3 = a1 + p3  General chain agglomeration
       a4 = a2 + p3
```

The non-confluence of these reductions introduces two directions for future works. First, the cases of non-confluence should be investigated further, and may inspire new reductions that would make them confluent. Second, it would be worth finding an algorithm which determines if a net is totally reducible, and finds the correct reduction strategy.

6 Counting Markings

We consider the problem of counting the number of markings of a net N from the set of markings of the residual net N_r and the (collected) system of linear equations Q. For totally reduced nets, the problem of counting the markings of N translates to that of counting the number of non-negative integer solutions to Q. For partially reduced nets, a similar process must be iterated over all markings m reachable in N_r (we discuss a better implementation later).

6.1 Off the shelf methods

Counting the number of integer solutions of a linear system of equations (inequalities can always be represented by equations through the addition of slack variables) is an active area of research.

A method was proposed in [1], implemented in the tool azove, for the particular case where variables take their values in $\{0, 1\}$. The method consists of building a Binary Decision Diagram for each equation, using Shannon expansion, and then to compute their conjunction (this is done with a specially tailored algorithm). The number of paths of the BDD gives the expected result. Our experiments with azove show that its performances heavily depend on the ordering chosen

for the BDD variables; this is typical of decision diagram based techniques. In any case, its usage in our context would be limited to safe nets.

For the general case, the current state of the art can be found in the work of De Loera et al. [14,15] on counting lattice points in convex polytopes. Their approach is implemented in a tool called LattE; it relies on algebraic and geometric methods; namely the use of rational functions and the decomposition of cones into unimodular cones. Our experiments with LattE show that it can be conveniently used on systems with, say, less than 50 variables. For instance, LattE is strikingly fast (less than 1s) at counting the number of solutions of the systems computed in Sect. 4. Moreover, its running time does not generally depend on the constants found in the system. As a consequence, computing the reachability count for 10 or, say, 10^{12} houses in the HouseConstruction net, takes exactly the same time.

An alternative to LattE is the barvinok tool, that provides a library for counting the number of integer points in parametric and non-parametric polytopes. The underlying methods behind this tool are presented in [22].

On totally reduced nets, we found LattE and barvinok equally convenient, though barvinok seems able to handle larger systems (up to a few hundred variables). Performances of both LattE and barvinok may slightly vary depending on the options selected when running the tools (many are available), which makes a rigorous comparison difficult.

On partially reduced net, that is in the case where we need to count the solutions of many instances of the same linear system differing only by some constants, LattE does not provide any builtin support. On the other hand barvinok supports so-called *parameterized polytopes*: Given an equation system in which some variables are declared as parameters, it is able to compute an expression (a quasi-polynomial) computing the number of solutions of the system given values for the parameters. This should be exactly what we need for handling partially reduced nets, but our preliminary experiments suggest that the method is limited to a small number of parameters, otherwise the expressions computed tend to be huge. It is interesting however, for further investigations, that a theory is available to solve such problems.

Though our experiments with LattE and barvinok suffice to show that these approaches are practicable, we implemented our own counting method in a library called Polycount. The methods underlying Polycount are described in the next section.

Polycount takes advantage of the stratified structure of the systems obtained from reductions, and it relies on combinatorial rather than geometric methods. Its main benefits over LattE and barvinok, important for practical purposes, are that it can handle systems with many variables (say thousands), though it can be slower than those on small systems. For partially reduced nets (i.e. for parameterized equation systems), Polycount can compute efficiently small symbolic expressions or multivariate polynomials acting as generating functions. But these benefits come at a cost: Polycount cannot handle the full rule set `compact+`, its usage is limited to the simpler set `compact` (see their definitions at the end of Sect. 3).

6.2 Polycount – an adhoc counting method

Polycount is a library which takes as input the system of linear equations Q generated by `compact`, and produces a multivariate polynomial $g(Q)$. The variables of the resulting polynomial correspond to the places of the residual net N_r. At the moment, polycount is not able to handle all the constraints generated using the `compact+` strategy.

Given a reduction (N, Q, N_r) and a marking m of the residual net, that is a valuation of the variables in $g(Q)$, the expression $g(Q)(m)$ evaluates to the number of markings in N that correspond to the marking m in N_r. As a particular case, when N is totally reduced, N_r has

no places, which implies that the resulting polynomial is a constant (a zero degree polynomial) equal to the number of reachable marking in N.

We start by giving a syntactic definition of the possible set of equations Q generated with `compact`, and define $\text{inp}(Q)$ as the set of variables remaining in the residual net N_r. The system of equations (Q) generated by the reductions in `compact` can be described by the following grammar, where, for the sake of conciseness, we write $A \vdash$ instead of $SCA \vdash$ or $SLA \vdash$ for constraints originating from agglomerations.

$$Q ::= \emptyset \mid Q, A \vdash x = y + z \mid Q, R \vdash x = L(y_1, .., y_k)$$

where x, y, z, y_i are variables and $L(y_1, .., y_k)$ is a linear combination of $y_1, .. y_k$. Basically, Q is a (comma-separated) list of equations of the form $A \vdash ...$ or $R \vdash ....$

We assume that P is the set of places of N in the reduction (N, Q, N_r). Then, the set of places of N_r only depends on N and Q, and is written $\text{inp}_P(Q)$, or simply $\text{inp}(Q)$ when P is obvious from the context. We call $\text{inp}(Q)$ the set of *input variables* of Q. It can be defined by induction on the list Q.

$$\begin{aligned}
&\text{inp}_P(\emptyset) = P \\
&\text{inp}_P(Q, A \vdash x = y + z) = \{x\} \cup (\text{inp}_P(Q) \setminus \{y, z\}) \\
&\text{inp}_P(Q, R \vdash x = L(y_1, .., y_k)) = \text{inp}_P(Q) \setminus \{x\}
\end{aligned}$$

The empty set of equations $\emptyset$ implies that no reduction has been applied to N, hence $N_r = N$, and $\text{inp}_P(\emptyset) = P$. In the agglomeration case $A \vdash x = y + z$, a new place x is introduced, whereas two places y and z are removed from the net. The definition of $\text{inp}(Q, A \vdash x = y + z)$ reflects the introduction of x and the removal of y and z. Similarly, the definition of $\text{inp}(Q, R \vdash x = L(y_1, .., y_k))$ reflects the removal of the redundant place x.

As a particular case, if N is totally reduced by an abstraction $(N, Q, \emptyset)$, then $\text{inp}(Q)$ is empty.

Counting the number of solutions. In order to count the number of reachable markings of N, we define a function g parameterized by a system of equations Q. The function $g(Q) : \mathbb{N}^{\text{inp}(Q)} \to \mathbb{N}$, applied to a marking of the residual net N_r, computes the number of associated markings in the original net N. It will be shown that $g(Q)$ is actually a polynomial in the variables of $\text{inp}(Q)$. We build the parameterized function $g(Q)$ incrementally by providing a (computable) term whose free variables are included in $\text{inp}(Q)$. Then, we show that this term is actually a multivariate polynomial.

$$\begin{aligned}
&g(\emptyset) = 1 \\
&g(Q, A \vdash x = y + z) = \textstyle\sum_{y=0}^{x} g(Q)\,[z := x - y] \\
&g(Q, R \vdash x = L(y_1, .., y_k)) = g(Q)\,[x := L(y_1, .., y_k)]
\end{aligned}$$

where $g(Q)\,[x := \alpha]$ is the substitution of variable x by α in $g(Q)$.

The empty case $g(\emptyset) = 1$ holds for an unreduced net N. It returns the term 1, to be considered as a constant function with domain $\mathbb{N}^{\text{inp}(\emptyset)}$, that is $\mathbb{N}^P$. This means that, given one marking of N (that is an element of $\mathbb{N}^P$), the number of associated markings in N is, obviously, 1. In contrast, the term $g(Q, A \vdash x = y + z)$ is obtained by counting all markings where y and z are such that $y + z = x$, that is y takes all values in $[0; x]$ (hence $\sum_{y=0}^{x}$) and z must be equal to $x - y$ (hence the substitution). Similarly, the term $g(Q, R \vdash x = L(y_1, .., y_k))$ reflects the redundancy between x and the places (y_i) with the substitution of x by $L(y_1, .., y_k)$. As an example, consider $g(R \vdash x = L(y_1, .., y_k))$, that is a redundancy occurring in the initial net N. It is equal to $g(\emptyset)\,[x := L(y_1, .., y_k)]$, that is 1. This shows that a redundant place in the initial net does not contribute to additional markings.

We illustrate the computation of the number of markings by considering the subset of the HouseConstruction example inside the dashed zone (see fig. 8). This net consists of places in the range p_{18}—p_{27} and is reduced by our system to a single place, a_{13}. The subset of equations (Q) related to this subnet is:

$$\begin{array}{ll} (i)\ R \vdash p_{19} = p_{20} & (v)\ R \vdash p_{27} = p_{23} + p_{26} \\ (ii)\ A \vdash a_4 = p_{21} + p_{18} & (vi)\ A \vdash a_{11} = a_5 + p_{23} \\ (iii)\ A \vdash a_5 = p_{22} + p_{20} & (vii)\ A \vdash a_{13} = a_{11} + p_{26} \\ (iv)\ R \vdash a_4 = a_5 + p_{23} & \end{array}$$

One checks that $\mathrm{inp}(Q) = \{a_{13}\}$, as expected. Let us compute the term $g(Q)$ in an incremental way. We will define g_1 as the expression $g(\,(i)\,)$ —the system consisting of equation (i) alone—g_2 as $g(\,(i),(ii)\,)$; and so on, ending with $g_7 = g(Q)$.

We introduce a useful notation: let $((k))(x)$ be the expression $\binom{x+k-1}{k-1}$, which denotes the number of ways to put x tokens into k slots. It is also the number of positive solutions to the equation $y_1 + \cdots + y_k = x$. We observe that $((1))(x) = 1$, and for any $x > 0$, $\sum_{y=0}^{x} ((k))(y) = ((k+1))(x)$ (the proof is left as a combinatorial exercise).

- By definition, g_1 is $g(\,(i)\,) = g(\emptyset)\,[p_{19} := p_{20}]$, that is $g_1 = 1$.
- g_2 is $\sum_{p_{21}=0}^{a_4} g_1\,[p_{18} := a_4 - p_{21}]$, that is $g_2 = \sum_{p_{21}=0}^{a_4} 1$, which happens to be $a_4 + 1$ or, using our notation, $g_2 = ((2))(a_4)$.
- g_3 is $\sum_{p_{22}=0}^{a_5} g_2\,[p_{20} := a_5 - p_{22}]$, that is $g_3 = \sum_{p_{22}=0}^{a_5} ((2))(a_4) = ((2))(a_4) \times ((2))(a_5)$.
- $g_4 = ((2))(a_5 + p_{23}) \times ((2))(a_5)$.
- Equation (v) has no effect since p_{27} does not occur in g_4, hence $g_5 = g_4$.
- g_6 is $\sum_{a_5=0}^{a_{11}} g_5\,[p_{23} := a_{11} - a_5]$, that is $g_6 = \sum_{a_5=0}^{a_{11}} ((2))(a_{11}) \times ((2))(a_5)$, which is simplified into $g_6 = ((2))(a_{11}) \times ((3))(a_{11})$.
- The last equation leads to $g_7 = \sum_{a_{11}=0}^{a_{13}} g_6\,[p_{26} := a_{13} - a_{11}]$, that is $g_7 = \sum_{a_{11}=0}^{a_{13}} g_6$.

Hence, we finally get $g = \sum_{a_{11}=0}^{a_{13}} ((2))(a_{11}) \times ((3))(a_{11})$, which, unfortunately, cannot be easily simplified. As explained next, this expression is actually a polynomial of degree 4 in the variable a_{13}. With some extra computations we can actually show that g is equivalent to $\frac{1}{8}a_{13}^4 + \frac{11}{12}a_{13}^3 + \frac{19}{8}a_{13}^2 + \frac{31}{12}a_{13} + 1$.

Theorem 7 *Given a system of reduction equations Q obtained with strategy* `compact`*, the term $g(Q)$ is equal to a polynomial with variables included in* $\mathrm{inp}(Q)$.

Proof This is shown by structural induction on Q. The case $Q = \emptyset$ is immediate. The case $Q = (Q', R \vdash x = L(y_1, .., y_k))$ is shown with no difficulty by induction hypothesis. The case $Q = (Q', A \vdash x = y + z)$ introduces the term $\sum_{y=0}^{x} g(Q')\,[z := x - y]$ whose variables are included in the set $\{x\} \cup (\mathrm{inp}(Q') \setminus \{y, z\}) = \mathrm{inp}(Q)$. Besides, this term happens to be a polynomial, as a consequence of the following: first notice that by induction hypothesis, $g(Q')\,[z := x - y]$ is a polynomial with variables in $\mathrm{inp}(Q) \setminus \{z\}$. Therefore it can be decomposed into a finite sum $\sum_{i \in I} \alpha_i y^i$, where α_i are polynomials with variable in $\mathrm{inp}(Q) \setminus \{z, y\}$. The result follows from the fact that the term $S_i = \sum_{y=0}^{x} y^i$ is equal to a polynomial in x. Actually we have that S_i is a polynomial in x of degree $i+1$ since, thanks to a well-known induction formula, it is known that $(i+1)S_i = (x+1)^{i+1} - \sum_{k=0}^{i-1} \binom{i+1}{k} S_k$. □

Polynomial optimizations. By applying some local rewriting rules, we are sometimes able to simplify the expression of $g(Q)$ (e.g. factorize a subterm out of a sum, when it does not depend on the sum variable). In the most favorable cases, we may factorize the polynomial $g(Q)$ into a product of the form $\Pi_{x \in \mathrm{inp}(Q)} g_x$, where g_x is a polynomial on the single variable x. We call this

a *partitioned form* of $g(Q)$. Such a form exists, for instance, when the initial net is composed of independent subnets. Partitioned forms are efficiently computed in the context of SDDs (see next).

Reducible and non-reducible nets. In practice, even when the input net is fully reducible, that is when $g(Q)$ is a constant, we may have to compute very complex, intermediate polynomial expressions g_1, g_2, ... With polycount, when polynomial optimizations apply gracefully, we are able to compute $g(Q)$ efficiently for nets having up to a few thousands places. On the contrary, if intermediate polynomials cannot be optimized, computation can quickly become untractable: it requires to repeatedly multiply polynomials having hundreds of variables and a degree over a few hundreds.

When the reduction (N, Q, N_r) is only partial, the computation of the number of markings can be performed by iterating over all markings m_r of N_r and accumulating, for each reduced marking, the value of $g(Q)(m_r)$. This approach is obviously very inefficient when $\mathcal{R}(N_r)$ is large.

When the set of markings of N_r is represented by a decision diagram, each level in the diagram concerns only a given place of N_r. The computation of $g(Q)(m_r)$ for all markings m_r still requires to iterate over all markings, that is all paths of the diagram, which is expensive and inefficient in general. However, if $g(Q)$ is in partitioned form (as defined above), then each subterm of the partitioned polynomial $g(Q)$ can be computed locally, at each level of the diagram. Hence, the computation of $g(Q)$ directly matches the structure of the diagram and the number of markings of N can be computed very efficiently in a single recursive traversal of the diagram. Polycount tries to return its result, as much as possible, in a partitioned form.

7 Computing Experiments

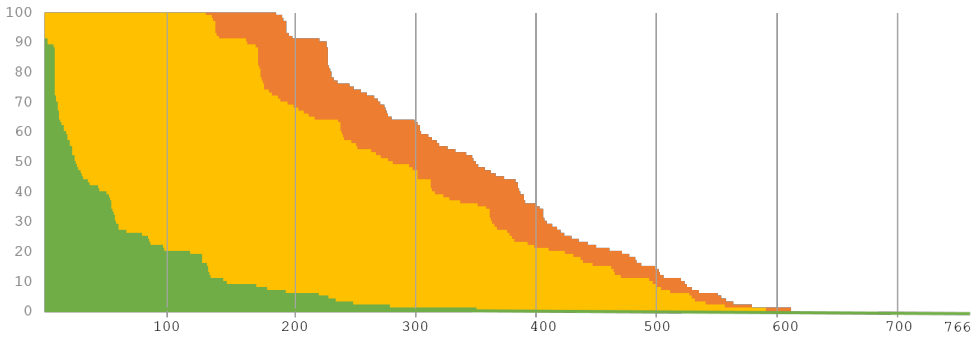


Fig. 17 Distribution of reduction ratios (place count) over the 766 PN instances.

We have integrated our reduction system and counting method inside a state space generation tool called *tedd*. The tool is part of our Petri nets analysis toolbox called *TINA* [5] (`www.laas.fr/tina`). Tool *tedd* makes use of symbolic exploration and stores markings in a Set Decision Diagram [21]. For counting markings in presence of agglomerations, one has the choice between using the external tool LattE or using our native counting method (Polycount) discussed in Sect. 6.

Benchmarks. Our benchmark is made of the full collection of Petri nets used in the Model Checking Contest [8,11]. It includes 766 instances of Petri nets, organized into 82 classes (simply

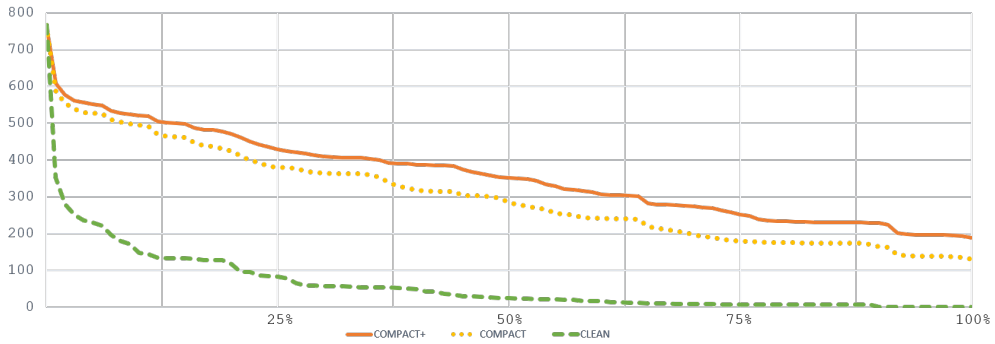


Fig. 18 Cumulative reduction ratios (place count) over the 766 PN instances.

called models). Each class includes several nets (called instances) that typically differ by their initial marking or by the number of components constituting the net. The size of the nets vary widely, from 9 to 50 000 places, 7 to 200 000 transitions, and 20 to 1 000 000 arcs. Most nets are ordinary (arcs have weight 1) but a significant number are generalized nets. Overall, the collection provides a large number of PN with various structural and behavioral characteristics, covering a large variety of use cases.

Reduction ratio and prevalence. Our first results are about how well the reductions perform. We provide three different reduction strategies:

- `clean`, that only applies removal of redundant places and transitions (rules R and T in Sect. 3);
- `compact`, that extends strategy `clean` with rules for reduction of simple loop and chain agglomerations (rules SLA and SCA from Definitions 5 and 6) as well as simple source-sink pairs elimination (rule SSP);
- `compact+`, that extends strategy `compact` with generalized versions of loop and chain agglomeration (rules GLA and GCA from Def. 9 and 7) as well as the simplification of Live Marked Graphs (rule LMG).

We give in Fig. 17 the reduction ratios obtained with our three strategies. The reduction ratio is expressed in terms of number of places (before and after reduction) for each of the MCC instances, sorted in descending order. We overlay the results for the different strategies (the lower, in dark/green color, for `clean`; the middle, in light/yellow color for `compact`; and the upper, in orange, for `compact+`), which means that values at the same instance count (points with the same abscissa but on different curves) may not correspond to the same benchmark instance.

With this figure, we see that there is a surprisingly high number of models that are totally reducible by our approach, since about a quarter of the models (188 instances out of 766) are fully reducible. We also observe that the impact of strategy `clean` alone is minor compared to `compact`, and that `compact+` allows us to completely reduce about 50 more instances than `compact` alone.

In Fig. 18, we display a "cumulative version" of the same results. For a given reduction ratio, in abscissa, we give the total number of instances that are reduced by at least this amount. (Figure 18 can be interpreted as an alternative presentation of the data in Fig. 17 where we have switched the axis.) With this figure, we can easily answer questions of the form "how many instances are reduced by X% or more with strategy Y?". Globally, our results show that reductions have a significant impact on about half the instances, with a very high impact on about a quarter of them. For instance, we can observe that about half the instances are reduced

by a factor of more than 25% with strategy `compact+`. Also, there are about 150 instances that cannot be reduced.

Computing time of reductions. Many of the reduction rules implemented have a cost polynomial in the size of the net. The rule removing redundant places in the general case is more complex, since it requires to compute invariants on the net and therefore may require to solve an integer programming problem. For this reason we limit its application to nets with less than 50 places. With this restriction, reductions are computed in a few seconds in most cases, and in about 3 minutes for the largest nets. The restriction is necessary but, because of it, we do not reduce some nets that would be fully reducible otherwise.

Impact on the marking count problem. In our benchmark, there are 218 models (out of 766) for which no tool was ever able to compute a marking count in the condition of the MCC (1 hour of computation with a cap of 16 Gb on active memory). Those are the most difficult instances in the contest. With our method, we can count the markings of at least 22 of these "most difficult" instances.

Net instance	size		MCC best	*tedd* native	*tedd* LattE	speed-up
	♯ places	♯ states				
DLCround-13a	463	2.40e17	9	0.33	-	27
FlexibleBarrier-22a	267	5.52e23	5	0.25	-	20
NeighborGrid-d4n3m2c23	81	2.70e65	330	0.21	44	1571
NeighborGrid-d5n4m1t35	1 024	2.85e614	-	340	-	∞
Referendum-1000	3 001	1.32e477	29	12	-	2
RobotManipulation-00050	15	8.53e12	94	0.1	0.17	940
RobotManipulation-10000	15	2.83e33	-	102	0.17	∞
Diffusion2D-50N050	2 500	4.22e105	1 900	5.84	-	325
Diffusion2D-50N150	2 500	2.67e36	-	5.86	-	∞
DLCshifumi-6a	3 568	4.50e160	950	6.54	-	145
Kanban-1000	16	1.42e30	240	0.11	0.24	2182
HouseConstruction-100	26	1.58e24	630	0.4	0.85	1575
HouseConstruction-500	26	2.67e36	-	30	0.85	∞
ERK-1000	11	1.41e16	550	-	0.13	4231
ERK-100000	11	1.39e28	-	-	0.13	∞
SmallOS-2048-512	9	1.04e14	1500	-	0.15	10000
SmallOS-8192-4096	9	2.50e17	-	-	0.14	∞
Airplane-4000	28 019	2.18e12	2520	102	-	25
AutoFlight-48a	1127	1.61e51	19	3.57	-	5
DES-60b	519	8.35e22	2300	364	-	6
Peterson-4	480	6.30e8	470	35.5	-	13
Peterson-5	834	1.37e11	-	1200	-	∞

Table 1 Times in seconds and speed-up for counting markings on some totally (top) and partially (bottom) reduced nets

If we concentrate on *tractable nets*—instances managed by at least one tool in the MCC 2018—our approach yields generally large improvements on the time taken to count markings; sometimes orders of magnitude faster.

Table 1 (top) lists the CPU time (in seconds) for counting the markings on a selection of fully reducible instances. We give the best time obtained by a tool during the last MCC (third column) and compare it with the time obtained with *tedd*, using two different ways of counting solutions (first with our own, Polycount, method then with LattE). We also give the resulting speed-up. These times also include parsing and applying reductions. An absent value (−) means that it cannot be computed in less than 1 hour with 16 Gb of storage, or that particular method is not

applicable to that model (e.g. the last four totally reducible models require strategy `compact+`, which is not supported by Polycount).

Concerning partially reducible nets, the improvements are less spectacular in general, though still significant. Counting markings in this case is more expensive than for totally reduced nets. But, more importantly, we have to build in that case a representation of the state space of the residual net, which is typically much more expensive than counting markings. Furthermore, if using symbolic methods for that purpose, several other parameters come into play that may impact the results, like the choice of an order on decision diagram variables or the particular kind of diagrams used. Nevertheless, improvements are clearly visible on a number of example models; some speedups are shown in Table 1 (bottom). Also, to minimize such side issues, instead of comparing *tedd* with `compact` reductions with the best tool performing at the MCC, we compared it with *tedd* without reductions or with the weaker `clean` strategy. In that case, `compact` reductions are almost always effective at reducing computing times.

Finally, there are also a few cases where applying reductions lower performances, typically when the reduction ratio is very small. For such quasi-irreducible nets, the time spent computing reductions is obviously wasted.

8 Related Work and Conclusion

Our work relies on well understood structural reduction methods, adapted here for the purpose of abstracting the state space of a net. This is done by representing the effects of reductions by a system of linear equations. To the best of our knowledge, reductions have never been used for that purpose before.

Linear algebraic techniques are widely used in Petri net theory but, again, not with our exact goals. It is well known, for instance, that the state space of a net is included in the solution set of its so-called "state equation", or from a basis of marking invariants. But these solutions, though exact in the special case of live marked graphs, yield approximations that are too coarse. Other works take advantage of marking invariants obtained from semiflows on places, but typically for optimizing the representation of markings in explicit or symbolic enumeration methods rather than for helping their enumeration, see e.g. [18,23]. Finally, these methods are only remotely related to our.

Another set of related work concerns symbolic methods based on the use of decision diagrams. Indeed they can be used to compute the state space size. In such methods, markings are computed symbolically and represented by the paths of some directed acyclic graph, which can be counted efficiently. Crucial for the applicability of these methods is determining a "good" variable ordering for decision diagram variables, one that maximizes sharing among the paths. Unfortunately, finding a convenient variable ordering may be an issue, and some models are inherently without sharing. For example, the best symbolic tools participating to the MCC 2018 can solve our illustrative example only for $p_1 \leq 200$, at a high cost, while we compute the result in a fraction of a second for virtually any possible initial marking of p_1.

Finally, though not aimed at counting markings nor relying on reductions, the work reported in [20] is certainly the closest to our. It defines a method for decomposing the state space of a net into the product of "independent sets of submarkings". The ideas discussed in the paper resemble what we achieved with agglomeration. In fact, the running example in [20], reproduced here in Fig. 1, is a fully reducible net in our approach. But no effective methods are proposed to compute decompositions.

Concluding remarks. We propose a new symbolic approach for representing the state space of a PN relying on systems of linear equations. Our results show that the method is almost always effective at reducing computing times and memory consumption for counting markings. Even more interesting is that our methods can be used together with traditional explicit and symbolic enumeration methods, as well as with other abstraction techniques like symmetry reductions for example. They can also help for other problems, like reachability analysis.

There are many opportunities for further research. For the close future, we are investigating richer sets of reductions for counting markings and application of the method to count not only the markings, but also the number of transitions of the reachability graph. Model-checking of linear reachability properties is another obvious prospective application of our methods. On the long term, we also plan to find efficient methods for describing the state spaces of bounded Petri nets using only sets of solutions to systems of linear equations; that is a method for computing a fully equational descriptions of its state space.

References

1. Markus Behle and Friedrich Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *9th Workshop on Algorithm Engineering and Experiments*. SIAM, 2007.
2. Gérard Berthelot. Checking properties of nets using transformations. In *European Workshop on Applications and Theory in Petri Nets*, pages 19–40. Springer, 1985.
3. Gérard Berthelot. Transformations and decompositions of nets. In *Advanced Course on Petri Nets*, pages 359–376. Springer, 1986.
4. Bernard Berthomieu, Didier Le Botlan, and Silvano Dal Zilio. Petri net reductions for counting markings. In *International Symposium on Model Checking Software*, volume 10869 of *LNCS*, pages 65–84. Springer, June 2018.
5. Bernard Berthomieu, Pierre-Olivier Ribet, and François Vernadat. The tool TINA–construction of abstract state spaces for Petri nets and Time Petri nets. *International journal of production research*, 42(14):2741–2756, 2004.
6. Frederic Commoner, Anatol W. Holt, Shimon Even, and Amir Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5(5):511–523, 1971.
7. Javier Esparza and Claus Schröter. Net reductions for LTL model-checking. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 310–324. Springer, 2001.
8. Fabrice Kordon et al. Complete Results for the 2018 Edition of the Model Checking Contest. http://mcc.lip6.fr/2018/results.php, June 2018.
9. Fabrice Kordon et al. MCC'2017–The Seventh Model Checking Contest. In *Transactions on Petri Nets and Other Models of Concurrency XIII*, pages 181–209. Springer, 2018.
10. Michel Hack. *Decidability questions for Petri Nets.* PhD thesis, Massachusetts Institute of Technology, 1976.
11. Lom M. Hillah and Fabrice Kordon. Petri Nets Repository: a tool to benchmark and debug Petri Net tools. In *38th International Conference on Petri Nets and Other Models of Concurrency (Petri Nets)*, volume 10258 of *LNCS*. Springer, June 2017.
12. Jonas F Jensen, Thomas Nielsen, Lars K Oestergaard, and Jiří Srba. TAPAAL and reachability analysis of P/T nets. In *Transactions on Petri Nets and Other Models of Concurrency XI*, pages 307–318. Springer, 2016.
13. Ferdinand K Levy, Gerald L Thompson, and JD Wiest. Introduction to the critical-path method. *Industrial Scheduling, Prentice-Hall, Englewood Cliffs (NJ)*, 1963.
14. Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*. SIAM, 2013.
15. Jesús A. De Loera, Raymond Hemmecke, Jeremiah Tauzer, and Ruriko Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4), 2004.
16. James Lyle Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
17. Laura Recalde, Enrique Teruel, and Manuel Silva. Improving the decision power of rank theorems. In *1997 IEEE Int. Conf. on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation*, volume 4, pages 3768–3773, 1997.
18. Karsten Schmidt. Using Petri net invariants in state space construction. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 473–488, 2003.

19. Manuel Silva, Enrique Teruel, and José Manuel Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *Advanced Course on Petri Nets*, pages 309–373. Springer, 1996.
20. Christian Stahl. Decomposing Petri net State Spaces. In *18th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2011), Hagen, Germany*, Sep 2011.
21. Yann Thierry-Mieg, Denis Poitrenaud, Alexandre Hamez, and Fabrice Kordon. Hierarchical set decision diagrams and regular models. In *TACAS–Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–15, 2009.
22. Sven Verdoolaege, Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. Counting integer points in parametric polytopes using barvinok's rational functions. *Algorithmica*, 48(1):37–66, 2007.
23. Karsten Wolf. Generating Petri net state spaces. *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 29–42, 2007.