arXiv:2106.12813v1 [cs.LO] 24 Jun 2021

Accelerating the Computation of Dead and Concurrent Places using Reductions

Nicolas Amat[1], Silvano Dal Zilio[1], and Didier Le Botlan[1]

[1]LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France

Abstract

We propose a new method for accelerating the computation of a concurrency relation, that is all pairs of places in a Petri net that can be marked together. Our approach relies on a state space abstraction, that involves a mix between structural reductions and linear algebra, and a new data-structure that is specifically designed for our task. Our algorithms are implemented in a tool, called Kong, that we test on a large collection of models used during the 2020 edition of the Model Checking Contest. Our experiments show that the approach works well, even when a moderate amount of reductions applies.

1 Introduction

We propose a new approach for computing the *concurrency relation* of a Petri net, that is all pairs of places that can be marked together in some reachable states. This problem has practical applications, for instance because of its use for decomposing a Petri net into the product of concurrent processes [9, 10]. It also provides an interesting example of safety property that nicely extends the notion of *dead places*. These problems raise difficult technical challenges and provide an opportunity to test and improve new model checking techniques [11].

Naturally, it is possible to compute the concurrency relation by first computing the complete state space of a system and then checking, individually, the reachability of each pair of places. But this amounts to solving a quadratic number of reachability properties—where the parameter is the number of places in the net—and one would expect to find smarter solutions, even if it is only for some specific cases. We are also interested in partial solutions, where computing the whole state space is not feasible.

We recently became interested in this problem because we see it as a good testbed for a new model checking technique that we are actively developing [1, 4, 5]. It is an abstraction technique, based on the use of structural reductions [3], that we successfully implemented into a symbolic model checker called Tedd. The idea is to compute reductions of the form $N_1 \rhd_E N_2$, where: N_1 is an initial Petri net (that we want to analyse); N_2 is a residual net (hopefully simpler than N_1); and E is a system of linear equations. The goal is to preserve enough information in E so that we can rebuild the reachable markings of N_1 knowing only those of N_2. While there are many examples of the benefits of structural

reductions when model checking Petri nets, the use of an equation system (E) for tracing back the effect of reductions is new, and we are hopeful that this approach can be applied to other problems. For example, we proved recently [1] that this approach also works well when combined with SMT.

In this paper, we confirm that the same holds true when we tackle the concurrent places problem. In practice, we can often reduce a net N_1 into another net N_2 with far fewer places. We show that we can reconstruct the concurrency relation of N_1 from the one of N_2, using a surprising and very efficient "inverse transform" that depends only on E and does not involve computing reachable markings. (This is a model checking paper where no transitions are fired!) This is useful since the number of places is a predominant parameter when computing the concurrency relation. Note that we are not concerned with how to compute the relation on N_2, but only by how we can *accelerate* its calculation on N_1.

Related Work. Several works address the problem of finding or characterizing the concurrent places of a Petri net. This notion is mentioned under various names, such as *coexistency defined by markings* [18], *concurrency graph* [27] or *concurrency relation* [12, 20, 19, 24, 28]. The main motivation is that the concurrency relation characterizes the sub-parts, in a net, that can be simultaneously active. Therefore it plays a useful role when decomposing a net into a collection of independent components. This is the case in [28], where the authors draw a connection between concurrent places and the presence of "sequential modules (state machines)". Another example is the decomposition of nets into unit-safe NUPNs (Nested-Unit Petri Nets) [9, 10], for which the computation of the concurrency relation is one of the main bottlenecks.

We know only a couple of tools that support the computation of the concurrency relation. A recent tool is part of the Hippo platform [28], available online. Our reference tool is CÆSAR.BDD, from the CADP toolbox [8, 17], that uses BDD techniques to explore the state space of a net and find concurrent places. It supports the computation of a partial relation and can output the "concurrency matrix" of a net using a specific textual format [11]. We adopt the same format since we use CÆSAR.BDD to compute the concurrency relation on the residual net, N_2, and as a yardstick in our benchmarks.

Concerning our use of structural reductions, our main result can be interpreted as an example of *reduction theorem* [22], that allows to deduce properties of an initial model (N_1) from properties of a simpler, coarser-grained version (N_2). But our notion of reduction is more complex and corresponds to the one pioneered by Berthelot [3] (with the equations added). Several tools use reductions for checking reachability properties but none specializes in computing the concurrency relation. We can mention TAPAAL [7], an explicit-state model checker that combines partial-order reduction techniques and structural reductions or, more recently, ITS Tools [26], which combines several techniques, including structural reductions and the use of SAT and SMT solvers.

Outline and Contributions. We define the semantics of Petri nets and the notion of concurrent places in Sect. 2. This section also introduces a simplified notion of "reachability equivalence", called *polyhedral abstraction*, that gives a formal definition to the relation $N_1 \rhd_E N_2$. Section 3 contains our main contributions. We describe a new data-structure, called Token Flow Graph (TFG),

that captures the particular structure of the equation system generated with our approach. We prove several results on TFGs that allow us to reason about the reachable places of a net by playing a token game on this graph. We use TFGs (Sect. 4) to define an algorithm that implements our "inverse transform" and show how to adapt it to situations where we only have partial knowledge of the residual concurrency relation. Our approach has been implemented and computing experiments (Sect. 5) show that reductions are effective on a large set of models. We perform our experiments on an independently managed collection of Petri nets (588 instances) corresponding to the safe nets used during the 2020 edition of the Model Checking Contest [2]. We observe that, even with a moderate amount of reductions (say we can remove 25% of the places), we can compute complete results much faster with reductions than without (often with speed-ups greater than $\times 100$). We also show that we perform well with incomplete relations, where we are both faster and more accurate. We include the proofs of all our results in the appendix.

2 Petri Nets and Polyhedral Abstraction

A *Petri net* N is a tuple $(P, T, \mathbf{pre}, \mathbf{post})$ where $P = \{p_1, \ldots, p_n\}$ is a finite set of places, $T = \{t_1, \ldots, t_k\}$ is a finite set of transitions (disjoint from P), and $\mathbf{pre} : T \to (P \to \mathbb{N})$ and $\mathbf{post} : T \to (P \to \mathbb{N})$ are the pre- and post-condition functions (also called the flow functions of N). We often simply write that p is a place of N when $p \in P$. A state m of a net, also called a *marking*, is a total mapping $m : P \to \mathbb{N}$ which assigns a number of *tokens*, $m(p)$, to each place of N. A marked net (N, m_0) is a pair composed of a net and its initial marking m_0.

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m(p) \geqslant \mathbf{pre}(t, p)$ for all places p in P. (We can also simply write $m \geqslant \mathbf{pre}(t)$, where $\geqslant$ stands for the component-wise comparison of markings.) A marking m' is reachable from a marking m by firing transition t, denoted $m \xrightarrow{t} m'$, if: (1) transition t is enabled at m; and (2) $m' = m - \mathbf{pre}(t) + \mathbf{post}(t)$. When the identity of the transition is unimportant, we simply write this relation $m \to m'$. More generally, marking m' is reachable from m in N, denoted $m \to^\star m'$ if there is a (possibly empty) sequence of reductions such that $m \to \cdots \to m'$. We denote $R(N, m_0)$ the set of markings reachable from m_0 in N.

A marking m is k-bounded when each place has at most k tokens and a marked Petri net (N, m_0) is bounded when there is a constant k such that all reachable markings are k-bounded. While most of our results are valid in the general case—with nets that are not necessarily bounded and without any restrictions on the flow functions (the weights of the arcs)—our tool and our experiments focus on the class of 1-bounded nets, also called *safe* nets.

Given a marked net (N, m_0), we say that places p, q of N are concurrent when there exists a reachable marking m with both p and q marked. The *Concurrent Places* problem consists in enumerating all such pairs of places.

Definition 2.1 (Dead and Concurrent places)**.** *We say that a place p of (N, m_0) is* not-dead *if there is m in $R(N, m_0)$ such that $m(p) > 0$. In a similar way, we say that places p, q are* concurrent*, denoted $p \parallel q$, if there is m in $R(N, m_0)$ such that both $m(p) > 0$ and $m(q) > 0$. By extension, we use the notation $p \parallel p$*

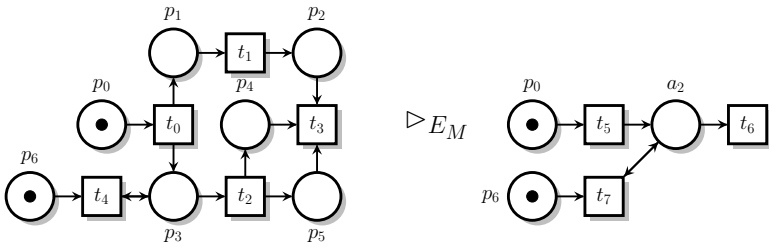

Figure 1: An example of Petri net, M_1 (left), and one of its polyhedral abstraction, M_2 (right), with $E_M \triangleq (p_5 = p_4), (a_1 = p_1 + p_2), (a_2 = p_3 + p_4), (a_1 = a_2)$.

when p is not-dead. We say that p, q are nonconcurrent, denoted $p \# q$, when they are not concurrent.

Relation with Linear Arithmetic Constraints. Many results in Petri net theory are based on a relation with linear algebra and linear programming techniques [23, 25]. A celebrated example is that the potentially reachable markings of a net (N, m_0) are non-negative, integer solutions to the *state equation* problem, $m = I \cdot \sigma + m_0$, with I an integer matrix defined from the flow functions of N and σ a vector in $\mathbb{N}^k$. It is known that solutions to the system of linear equations $\sigma^T \cdot I = \vec{0}$ lead to *place invariants*, $\sigma^T \cdot m = \sigma^T \cdot m_0$, that can provide some information on the decomposition of a net into blocks of nonconcurrent places, and therefore information on the concurrency relation.

For example, for net M_1 (Fig. 1), we can compute invariant $p_4 - p_5 = 0$. This is enough to prove that places p_4 and p_5 are concurrent, if we can prove that at least one of them is not-dead. Likewise, an invariant of the form $p + q = 1$ is enough to prove that p and q are 1-bounded and cannot be concurrent. Unfortunately, invariants provide only an over-approximation of the set of reachable markings, and it may be difficult to find whether a net is part of the few known classes where the set of reachable markings equals the set of potentially reachable ones [16].

Our approach shares some similarities with this kind of reasoning. A main difference is that we will use equation systems to draw a relation between the reachable markings of two nets; not to express constraints about (potentially) reachable markings inside one net. Like with invariants, this will allow us, in many cases, to retrieve information about the concurrency relation without "firing any transition", that is without exploring the state space.

In the following, we will often use place names as variables, and markings $m : P \to \mathbb{N}$ as partial solutions to a set of linear equations. For the sake of simplicity, all our equations will be of the form $x = y_1 + \cdots + y_l$ or $y_1 + \cdots + y_l = k$ (with k a constant in $\mathbb{N}$).

Given a system of linear equations E, we denote $fv(E)$ the set of all its variables. We are only interested in the non-negative integer solutions of E. Hence, in our case, a *solution* to E is a total mapping from variables in $fv(E)$ to $\mathbb{N}$ such that all the equations in E are satisfied. We say that E is *consistent* when there is at least one such solution. Given these definitions, we say that the mapping $m : \{p_1, \ldots, p_n\} \to \mathbb{N}$ is a (partial) solution of E if the system $E \,, \lfloor m \rfloor$ is consistent, with $\lfloor m \rfloor$ the sequence of equations $p_1 = m(p_1) \,, \cdots \,, p_n = m(p_n)$.

(In some sense, we use $\lfloor m \rfloor$ as a substitution.) For instance, places p, q are concurrent if the system $p = 1 + x \,,\, q = 1 + y \,,\, \lfloor m \rfloor$ is consistent, where m is a reachable marking and x, y are some fresh (slack) variables.

Given two markings $m_1 : P_1 \to \mathbb{N}$ and $m_2 : P_2 \to \mathbb{N}$, from possibly different nets, we say that m_1 and m_2 are *compatible*, denoted $m_1 \equiv m_2$, if they have equal marking on their shared places: $m_1(p) = m_2(p)$ for all p in $P_1 \cap P_2$. This is a necessary and sufficient condition for the system $\lfloor m_1 \rfloor \,,\, \lfloor m_2 \rfloor$ to be consistent.

Polyhedral Abstraction. We recently defined a notion of *polyhedral abstraction* based on our previous work applying structural reductions to model counting [1, 5]. We only need a simplified version of this notion here, which entails an equivalence between the state space of two nets, (N_1, m_1) and (N_2, m_2), "up-to" a system E of linear equations.

Definition 2.2 (E-equivalence)**.** *We say that* (N_1, m_1) *is* E*-equivalent to* (N_2, m_2)*, denoted* $(N_1, m_1) \rhd_E (N_2, m_2)$*, if and only if:*

(A1) $E \,,\, \lfloor m \rfloor$ *is consistent for all markings* m *in* $R(N_1, m_1)$ *and* $R(N_2, m_2)$*;*

(A2) *initial markings are* compatible*, meaning* $E \,,\, \lfloor m_1 \rfloor \,,\, \lfloor m_2 \rfloor$ *is consistent;*

(A3) *assume* m'_1, m'_2 *are markings of* N_1, N_2*, respectively, such that* $E \,,\, \lfloor m'_1 \rfloor \,,\, \lfloor m'_2 \rfloor$ *is consistent, then* m'_1 *is reachable if and only if* m'_2 *is reachable:*
$m'_1 \in R(N_1, m_1) \iff m'_2 \in R(N_2, m_2)$.

By definition, relation $\rhd_E$ is symmetric. We deliberately use a symbol oriented from left to right to stress the fact that N_2 should be a reduced version of N_1. In particular, we expect to have less places in N_2 than in N_1.

Given a relation $(N_1, m_1) \rhd_E (N_2, m_2)$, each marking m'_2 reachable in N_2 can be associated to a unique subset of markings in N_1, defined from the solutions to $E \,,\, \lfloor m'_2 \rfloor$ (by condition A1 and A3). We can show that this gives a partition of the reachable markings of (N_1, m_1) into "convex sets"—hence the name polyhedral abstraction—each associated to a reachable marking in N_2. Our approach is particularly useful when the state space of N_2 is very small compared to the one of N_1. In the extreme case, we can even find examples where N_2 is the "empty" net (a net with zero places, and therefore a unique marking), but this condition is not a requisite in our approach.

We can illustrate this result using the two marked nets M_1, M_2 in Fig. 1, for which we can prove that $M_1 \rhd_{E_M} M_2$. We have that $m'_2 \triangleq a_2 = 1 \,,\, p_6 = 1$ is reachable in M_2, which means that every solution to the system $p_0 = 0, p_1 + p_2 = 1 \,,\, p_3 + p_4 = 1 \,,\, p_4 = p_5 \,,\, p_6 = 1$ gives a reachable marking of M_1. Moreover, every solution such that $p_i \geqslant 1$ and $p_j \geqslant 1$ gives a witness that $p_i \parallel p_j$. For instance, p_1, p_4, p_5 and p_6 are certainly concurrent together. We should exploit the fact that, under some assumptions about E, we can find all such "pairs of variables" without the need to explicitly solve systems of the form $E \,,\, \lfloor m \rfloor$; just by looking at the structure of E.

For this current work, we do not need to explain how to derive or check that an equivalence statement is correct in order to describe our method. In practice, we start from an initial net, (N_1, m_1), and derive (N_2, m_2) and E using a combination of several structural reduction rules. You can find a precise description of our set of rules in [5] and a proof that the result of these reductions

always leads to a valid E-equivalence in [1]. In most cases, the system of linear equations obtained using this process exhibits a graph-like structure. In the next section, we describe a set of constraints that formalizes this observation. This is one of the contributions of this paper, since we never defined something equivalent in our previous works. We show with our benchmarks (Sect. 5) that these constraints are general enough to give good results on a large set of models.

3 Token Flow Graphs

We introduce a set of structural constraints on the equations occurring in an equivalence statement $(N_1, m_1) \rhd_E (N_2, m_2)$. The goal is to define an algorithm that is able to easily compute information on the concurrency relation of N_1, given the concurrency relation on N_2, by taking advantage of the structure of the equations in E.

We define the *Token Flow Graph* (TFG) of a system E of linear equations as a Directed Acyclic Graph (DAG) with one vertex for each variable occurring in E. Arcs in the TFG are used to depict the relation induced by equations in E. We consider two kinds of arcs. Arcs for *redundancy equations*, $q \twoheadrightarrow\!\!\bullet\ p$, to represent equations of the form $p = q$ (or $p = q + r + \dots$), expressing that the marking of place p can be reconstructed from the marking of $q, r, \dots$ In this case, we say that place p is *removed* by arc $q \twoheadrightarrow\!\!\bullet\ p$, because the marking of q may influence the marking of p, but not necessarily the other way round.

The second kind of arcs, $a \circ\!\!\rightarrow p$, is for *agglomeration equations*. It represents equations of the form $a = p + q$, generated when we agglomerate several places into a new one. In this case, we expect that if we can reach a marking with k tokens in a, then we can certainly reach a marking with k_1 tokens in p and k_2 tokens in q when $k = k_1 + k_2$ (see property Agglomeration in Lemma 3.2). Hence information flows in reverse order compared to the case of redundancy equations. This is why, in this case, we say that places/nodes p and q are removed. We also say that node a is *inserted*; it does not appear in N_1 but may appear as a new place in N_2. We can have more than two places in an agglomeration.

A TFG can also include nodes for *constants*, used to express invariant statements on the markings of the form $p + q = k$. To this end, we assume that we have a family of disjoint sets $K(n)$ (also disjoint from place and variable names), for each n in $\mathbb{N}$, such that the "valuation" of a node $v \in K(n)$ will always be n. We use K to denote the set of all constants.

Definition 3.1 (Token Flow Graph)**.** *A TFG with set of places P is a directed (bi)graph (V, R, A) such that: $V = P \cup S$ is a set of vertices (or nodes) with $S \subset K$ a finite set of constants; $R \in V \times V$ is a set of* redundancy arcs*, $v \twoheadrightarrow\!\!\bullet\ v'$; and $A \in V \times V$ is a set of* agglomeration arcs*, $v \circ\!\!\rightarrow v'$, disjoint from R.*

The main source of complexity in our approach arises from the need to manage interdependencies between A and R nodes, that is situations where redundancies and agglomerations alternate. This is not something that can be easily achieved by looking only at the equations in E and what motivates the need to define a specific data-structure.

We define several notations that will be useful in the following. We use the notation $v \to v'$ when we have $(v \twoheadrightarrow\!\!\bullet\ v')$ in R or $(v \circ\!\!\rightarrow v')$ in A. We say that a node

```
# R |- p5 = p4
# A |- a1 = p2 + p1
# A |- a2 = p4 + p3
# R |- a1 = a2
```

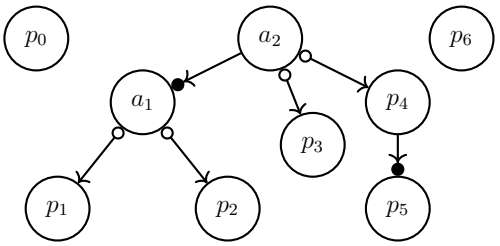

Figure 2: Equations generated from net M_1, in Fig.1, and associated TFG $[\![E_M]\!]$

v is a *root* if it is never the target of an arc. A sequence of nodes $(v_1, \ldots, v_n)$ in V^n is a *path* if we have $v_i \to v_{i+1}$ for all $i < n$. We use the notation $v \to^\star v'$ when there is a path from v to v' in the graph, or when $v = v'$. We write $v \circ\!\!\to X$ when X is the largest subset $\{v_1, \ldots, v_k\}$ of V such that $X \neq \emptyset$ and $v \circ\!\!\to v_i \in A$ for all $i \in 1..k$. Similarly, we write $X \to\!\!\bullet\, v$ when X is the largest, non-empty set of nodes $\{v_1, \ldots, v_k\}$ such that $v_i \to\!\!\bullet\, v \in R$ for all $i \in 1..k$.

We display an example of Token Flow Graphs in Fig. 2, where "black dot" arcs model edges in R and "white dot" arcs model edges in A. The idea is that each relation $X \to\!\!\bullet\, v$ or $v \circ\!\!\to X$ corresponds to one equation $v = \sum_{v_i \in X} v_i$ in E, and that all the equations in E should be reflected in the TFG. We want to avoid situations where the same place is removed more than once, or where some place occurs in the TFG but is never mentioned in N_1, N_2 or E. All these constraints can be expressed using a suitable notion of well-formed graph.

Definition 3.2 (Well-Formed TFG)**.** *A TFG $G = (V, R, A)$ for the equivalence statement $(N_1, m_1) \rhd_E (N_2, m_2)$ is* well-formed *when all the following constraints are met, where P_1 and P_2 stand for the set of places in N_1 and N_2:*

(T1) no unused names: $V \setminus K = P_1 \cup P_2 \cup fv(E)$,

(T2) nodes in K are roots: *if $v \in V \cap K$ then v is a root of G,*

(T3) nodes can be removed only once: *it is not possible to have $p \circ\!\!\to q$ and $p' \to q$ with $p \neq p'$, or to have both $p \to\!\!\bullet\, q$ and $p \circ\!\!\to q$,*

(T4) we have all and only the equations in E: *we have $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$ if and only if the equation $v = \sum_{v_i \in X} v_i$ is in E.*

Given a relation $(N_1, m_1) \rhd_E (N_2, m_2)$, the well-formedness conditions are enough to ensure the unicity of a TFG (up-to the choice of constant nodes) when we set each equation to be either in A or in R. In this case, we denote this TFG $[\![E]\!]$. In practice, we use a tool called Reduce to generate the E-equivalence from the initial net (N_1, m_1). This tool outputs a sequence of equations suitable to build a TFG and, for each equation, it adds a tag indicating if it is a Redundancy or an Agglomeration. We display in Fig. 2 the equations generated by Reduce for the net M_1 given in Fig. 1.

A consequence of condition (T3) is that a well-formed TFG is necessarily acyclic; once a place has been removed, it cannot be used to remove a place later. Moreover, in the case of reductions generated from structural reductions, the roots of the graph are exactly the constant nodes and the places that occur

in N_2 (since they are not removed by any equation). The constraints (T1)–(T4) are not artificial or arbitrary. In practice, we compute E-equivalences using multiple steps of structural reductions, and a TFG exactly records the constraints and information generated during these reductions. In some sense, equations E abstract a relation between the semantics of two nets, whereas a TFG records the structure of reductions between places during reductions.

Configurations of a Token Flow Graph. By construction, there is a strong connection between "systems of reduction equations", E, and their associated graph, $[\![E]\!]$. We show that a similar relation exists between solutions of E and "valuations" of the graph (what we call *configurations* thereafter).

A *configuration* c of a TFG (V, R, A) is a partial function from V to $\mathbb{N}$. We use the notation $c(v) = \bot$ when c is not defined on v and we always assume that $c(v) = n$ when v is a constant node in $K(n)$.

Configuration c is *total* when $c(v)$ is defined for all nodes v in V; otherwise it is said *partial*. We use the notation $c_{|N}$ for the configuration obtained from c by restricting its support to the set of places in the net N. We remark that when c is defined over all places of N then $c_{|N}$ can be viewed as a marking. By association with markings, we say that two configurations c and c' are *compatible*, denoted $c \equiv c'$, if they have same value on the nodes where they are both defined: $c(p) = c'(p)$ when $c(v) \neq \bot$ and $c'(v) \neq \bot$. We also use $\lfloor c \rfloor$ to represent the system $v_1 = c(v_1)\,, \cdots, v_k = c(v_k)$ where the $(v_i)_{i \in 1..k}$ are the nodes such that $c(v_i) \neq \bot$. We say that a configuration c is *well-defined* when the valuation of the nodes agrees with the equations associated with the A and R arcs of $[\![E]\!]$.

Definition 3.3 (Well-Defined Configurations)**.** *Configuration c is well-defined when for all nodes p the following two conditions hold:* **(CBot)** *if $v \to w$ then $c(v) = \bot$ if and only if $c(w) = \bot$; and* **(CEq)** *if $c(v) \neq \bot$ and $v \circ\!\!\to X$ or $X \to\!\!\bullet\, v$ then $c(v) = \sum_{v_i \in X} c(v_i)$.*

We prove that the well-defined configurations of a TFG $[\![E]\!]$ are partial solutions of E, and reciprocally. Therefore, because all the variables in E are nodes in the TFG (condition T1) we have an equivalence between solutions of E and total, well-defined configurations of $[\![E]\!]$.

Lemma 3.1 (Well-defined Configurations are Solutions)**.** *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$. If c is a well-defined configuration of $[\![E]\!]$ then $E\,,\, \lfloor c \rfloor$ is consistent. Conversely, if c is a total configuration of $[\![E]\!]$ such that $E\,,\, \lfloor c \rfloor$ is consistent then c is also well-defined.*

We can prove several properties related to how the structure of a TFG constrains possible values in well-formed configurations. These results can be thought of as the equivalent of a "token game", which explains how tokens can propagate along the arcs of a TFG. This is useful in our context since we can assess that two nodes are concurrent when we can mark them in the same configuration. (A similar result holds for finding pairs of nonconcurrent nodes.)

Our first result shows that we can always propagate tokens from a node to its children, meaning that if a node has a token, we can find one in its *successors* (possibly in a different well-defined configuration). In the following, we use the notation $\downarrow v$ for the set of successors of v, meaning: $\downarrow p \triangleq \bigcup \{q \in V \mid p \to^\star q\}$. Property (Backward) states a dual result; if a child node is marked then one of its parents must be marked.

Lemma 3.2 (Token Propagation)**.** *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$ and c a well-defined configuration of $[\![E]\!]$.*

(Forward) *if p, q are nodes such that $c(p) \neq \bot$ and $p \to^\star q$ then we can find a well-defined configuration c' such that $c'(q) \geqslant c'(p) = c(p)$ and $c'(v) = c(v)$ for every node v not in $\downarrow p$.*

(Backward) *if $c(p) > 0$ then there is a root v such that $v \to^\star p$ and $c(v) > 0$.*

(Agglomeration) *if $p \circ\!\!\to \{q_1, \ldots, q_k\}$ and $c(p) \neq \bot$ then for every sequence $(l_i)_{i \in 1..k}$ of $\mathbb{N}^k$, if $c(p) = \sum_{i \in 1..k} l_i$ then we can find a well-defined configuration c' such that $c'(p) = c(p)$, and $c'(q_i) = l_i$ for all $i \in 1..k$, and $c'(v) = c(v)$ for every node v not in $\downarrow p$.*

Until this point, none of our results rely on the properties of E-equivalence. We now prove that there is an equivalence between reachable markings and configurations of $[\![E]\!]$. More precisely, we prove (Th. 3.3) that every reachable marking in N_1 or N_2 can be extended into a well-defined configuration of $[\![E]\!]$. This entails that we can reconstruct all the reachable markings of N_1 by looking at well-defined configurations obtained from the reachable markings of N_2. Our algorithm (see next section) will be a bit smarter since we do not need to enumerate exhaustively all the markings of N_2. Instead, we only need to know which roots can be marked together.

Theorem 3.3 (Configuration Reachability)**.** *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$. If m is a marking in $R(N_1, m_1)$ or $R(N_2, m_2)$ then there exists a total, well-defined configuration c of $[\![E]\!]$ such that $c \equiv m$. Conversely, given a total, well-defined configuration c of $[\![E]\!]$, if marking $c_{|N_1}$ is reachable in (N_1, m_1) then $c_{|N_2}$ is reachable in (N_2, m_2).*

Proof (sketch). Take m a marking in $R(N_1, m_1)$. By property of E-abstraction, there is a reachable marking m_2' in $R(N_2, m_2)$ such that $E \,,\, \lfloor m \rfloor \,,\, \lfloor m_2' \rfloor$ is consistent. Therefore we can find a non-negative integer solution c to the system $E \,,\, \lfloor m \rfloor \,,\, \lfloor m_2' \rfloor$. And c is total because of condition (T1). For the converse property, we assume that c is a total and well-defined configuration of $[\![E]\!]$ and that $c_{|N_1}$ is a marking of $R(N_1, m_1)$. By Lemma 3.1, since c is well-defined, we have that $E \,,\, \lfloor c \rfloor$ is consistent, and therefore so is $E \,,\, \lfloor c_{|N_1} \rfloor \,,\, \lfloor c_{|N_2} \rfloor$. This entails $c_{|N_2}$ in $R(N_2, m_2)$ by condition (A3), as needed. □

In the following, we will often consider that nets are safe. This is not a problem in practice since our reduction rules preserve safeness. Hence we do not need to check if (N_2, m_2) is safe when (N_1, m_1) is. The fact that the nets are safe has consequences. In particular, as a direct corollary of Th. 3.3, we can assume that, for any well-defined configuration c, if $c_{|N_2}$ is reachable in (N_2, m_2) then $c(v) \in \{0, 1\}$.

By Th. 3.3, if we take reachable markings in N_2—meaning we fix the values of roots in $[\![E]\!]$—we can find places of N_1 that are marked together by propagating tokens from the roots to the leaves (Lemma 3.2). In our algorithm, next, we show that we can compute the concurrency relation of N_1 by looking at just two cases: (1) we start with a token in a single root p, with p not dead, and propagate this token forward until we find a configuration with two places of N_1 marked together; or (2) we do the same but placing a token in two separate roots, p_1, p_2, such that $p_1 \parallel p_2$. We base our approach on the fact that we

can extend the notion of concurrent places (in a marked net), to the notion of concurrent nodes in a TFG, meaning nodes that can be marked together in a "reachable configuration".

4 Dimensionality Reduction Algorithm

We define an algorithm that takes as inputs a well-formed TFG $[\![E]\!]$ plus the concurrency relation for the net (N_2, m_2), say $\|_2$, and outputs the concurrency relation for (N_1, m_1), say $\|_1$. Actually, our algorithm computes a *concurrency matrix*, C, that is a matrix such that $\mathrm{C}[v, w] = 1$ when the nodes v, w can be marked together in a "reachable configuration", and 0 otherwise. We prove (Th. 4.1) that the relation induced by C matches with $\|_1$ on N_1. For the case of "partial relations", we use $\mathrm{C}[v, w] = \bullet$ to mean that the relation is undecided. In this case we say that matrix C is *incomplete*.

The complexity of computing the concurrency relation is highly dependent on the number of places in the net. For this reason, we say that our algorithm performs some sort of a "dimensionality reduction", because it allows us to solve a problem in a high-dimension space (the number of places in N_1) by solving it first on a lower dimension space (since N_2 may have far fewer places) and then transporting back the result to the original net. In practice, we compute the concurrency relation on (N_2, m_2) using the tool CÆSAR.BDD from the CADP toolbox; but we can rely on any kind of "oracle" to compute this relation for us. This step is not necessary when the initial net is fully reducible, in which case the concurrency relation for N_2 is trivial and all the roots in $[\![E]\!]$ are constants.

We assume that $[\![E]\!]$ is a well-formed TFG for the relation $(N_1, m_1) \rhd_E (N_2, m_2)$; that both nets are safe; and that all the roots in $[\![E]\!]$ are either constants (in $K(0) \cup K(1)$) or places in N_2. We use symbol $\|_2$ for the concurrency relation on (N_2, m_2) and $\|_1$ on (N_1, m_1). To simplify our notations, we assume that $v \|_2 w$ when v is a constant node in $K(1)$ and w is not-dead. On the opposite, $v \#_2 w$ when $v \in K(0)$ or w is dead.

Our algorithm is divided into two main functions, Matrix and Propagate. In the main function, `Matrix`, we iterate over the non-dead roots of $[\![E]\!]$ and recursively propagates a "token" to its successors (the call to `Propagate` in line 4). After this step, we know all the live nodes in C. The call to `Propagate` has two effects. First, we retrieve the list of successors of the live roots. Second, as a side-effect, we update the concurrency matrix C by finding all the concurrent nodes that arise from a unique root. We can prove all such cases arise from redundancy arcs that are "under node v". Actually, we can prove that if $v \to w_1$ and $v \to\!\!\!\!\bullet\, w_2$ (with $w_1 \neq w_2$) then the nodes in the set $\downarrow v \setminus \downarrow w_2$ are concurrent to all the nodes in $\downarrow w_2$. Next, in the second **foreach** loop of `Matrix`, we compute the concurrent nodes that arise from two distinct live roots (v, w). In this case, we can prove that all the successors of v are concurrent with successors of w: all the pairs in $\downarrow v \times \downarrow w$ are concurrent.

We can prove that our algorithm is sound and complete using the theory that we developed on TFGs and configurations.

Theorem 4.1. *If* C *is the matrix returned by a call to* `Matrix`$([\![E]\!], \|_2)$ *then for all places* p, q *in* N_1 *we have* $p \|_1 q$ *if and only if either* $\mathrm{C}[p, q] = 1$ *or* $\mathrm{C}[q, p] = 1$.

We can perform a cursory analysis of the complexity of our algorithm. By

Function Matrix($[\![E]\!]$: TFG, $\|_2$: concurrency relation on (N_2, m_2))

Result: the concurrency matrix C

```
1 C ← 0⃗                    /* the matrix is initialized with zeros */
2 foreach root v in [[E]] do
3 |  if v ||_2 v then
4 |  |  succs[v] ← Propagate([[E]], C, v)
5 foreach pair of roots (v, w) in [[E]] do
6 |  if v ||_2 w then
7 |  |  foreach (v', w') ∈ succs[v] × succs[w] do C[v', w'] ← 1
8 return C
```

Function Propagate($[\![E]\!]$: TFG, C : concurrency matrix, v : node)

Result: the successors of v in $[\![E]\!]$. As a side-effect, we add to C all the relations that stem from knowing v not-dead.

```
1 C[v, v] ← 1
2 succs ← {v}                    /* succs collects the nodes in ↓v */
3 succr ← {}          /* auxiliary variable used to store ↓w when
    v →• w */
4 foreach w such that v o→ w do succs ← succs ∪ Propagate([[E]], C, w)
5 foreach w such that v →• w do
6 |  succr ← Propagate([[E]], C, w)
7 |  foreach (v', w') ∈ succs × succr do C[v', w'] ← 1
8 |  succs ← succs ∪ succr
9 return succs
```

construction, we update the matrix by following the edges of $[\![E]\!]$, starting from the roots. Since a TFG is a DAG, it means that we could call function `Propagate` several times on the same node. However, a call to `Propagate`($[\![E]\!]$, C, v) can only update C by adding a 1 between nodes that are successors of v (information only flows in the direction of $\rightarrow$). It means that `Propagate` is idempotent; a subsequent call to `Propagate`($[\![E]\!]$, C, v) will never change the values in C. As a consequence, we can safely memoize the result of this call and we only need to go through a node at most once. More precisely, we need to call `Propagate` only on the nodes that are not-dead in $[\![E]\!]$. During each call to `Propagate`, we may update at most $O(N^2)$ values in C, where N is the number of nodes in $[\![E]\!]$, which is also $O(|\mathrm{C}|)$, the size of our output. In conclusion, our algorithm has a linear time complexity (in the number of live nodes) if we count the numbers of function calls and a linear complexity, in the size of the output, if we count the number of updates to C. This has to be compared with the complexity of building then checking the state space of the net, which is PSPACE.

In practice, our algorithm is very efficient, highly parallelizable, and its execution time is often negligible when compared to the other tasks involved when computing the concurrency relation. We give some results on our performances in the next section.

Extensions to Incomplete Concurrency relations. With our approach, we only ever writes 1s into the concurrency matrix C. This is enough since we know relation $\|_2$ exactly and, in this case, relation $\|_1$ must also be complete (we can have only 0s or 1s in C). This is made clear by the fact that C is initialized with 0s everywhere. We can extend our algorithm to support the case where we only have a partial knowledge of $\|_2$. This is achieved by initializing C with the special value $\bullet$ (undefined) and adding rules that let us "propagate 0s" on the TFG, in the same way that our total algorithm only propagates 1s. For example, we know that if $\mathrm{C}[v, w] = 0$ (v, w are nonconcurrent) and $v \circ\!\!\rightarrow w'$ (we know that always $c(v) \geqslant c(w')$ on reachable configurations) then certainly $\mathrm{C}[w', w] = 0$. Likewise, we can prove that following rule for propagating "dead nodes" is sound: if $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, w] = 0$ (node w is dead) for all $w \in X$ then $\mathrm{C}[v, v] = 0$.

Partial knowledge on the concurrency relation can be useful. Indeed, many use cases can deal with partial knowledge or only rely on the nonconcurrency relation (a 0 on the concurrency matrix). This is the case, for instance, when computing NUPN partitions, where it is always safe to replace a $\bullet$ with a 1. It also means that knowing that two places are nonconcurrent is often more valuable than knowing that they are concurrent; 0s are better than 1s.

We have implemented an extension of our algorithm for the case of incomplete matrices using this idea and we report some results obtained with it in the next section. Unfortunately, we do not have enough space to describe the full algorithm here. It is slightly more involved than for the complete case and is based on a collection of six additional axioms:

- If $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[v, w] = 0$ for all node w in $[\![E]\!]$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v] = 0$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ then $\mathrm{C}[w, w'] = 0$ for all pairs of nodes $w, w' \in X$ such that $w \neq w'$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, v'] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v'] = 0$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[v, v'] = 0$ then $\mathrm{C}[w, v'] = 0$ for all nodes w in X.

While we can show that the algorithm is sound, completeness takes a different meaning: we show that when nodes p and q are successors of roots v_1 and v_2 such that $\mathrm{C}[v_i, v_i] \neq \bullet$ for all $i \in 1..2$ then necessarily $\mathrm{C}[p, q] \neq \bullet$.

5 Experimental Results

We have implemented our algorithm in a new tool, called Kong (for Koncurrent places Grinder). The tool is open-source, under the GPLv3 license, and is freely available on GitHub (https://github.com/nicolasAmat/Kong). We have used the extensive database of models provided by the Model Checking Contest (MCC) [2, 14] to experiment with our approach. Kong takes as inputs safe Petri nets defined using the Petri Net Markup Language (PNML) [15]. The tool does

not compute net reductions directly but relies on another tool, called Reduce, that is developed inside the Tina toolbox [6, 21]. For our experiments, we also need to compute the concurrency matrix of reduced nets. This is done using the tool CÆSAR.BDD (version 3.4, published in August 2020), that is part of the CADP toolbox [8, 17], but we could adopt any other technology here[1].

Benchmarks and Distribution of Reduction Ratios. Our benchmark is built from a collection of 588 instances of safe Petri nets used in the MCC 2020 competition. Since we rely on how much reduction we can find in nets, we computed the reduction ratio (r), obtained using Reduce, on all the instances (see Fig. 3). The ratio is calculated as the quotient between how many places can be removed and the number of places in the initial net. A ratio of 100% ($r = 1$) means that the net is *fully reduced*; the residual net has no places and all the roots are constants. We see that there is a surprisingly high number of models whose size is more than halved with our approach (about 25% of the instances have a ratio $r \geqslant 0.5$), with approximately half of the instances that can be reduced by a ratio of 30% or more. We consider two values for the reduction ratio: one for reductions leading to a well-formed TFG (in dark blue), the other for the best possible reduction with Reduce (in light orange).

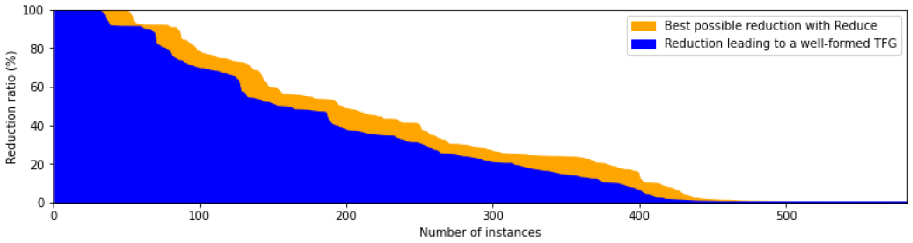

Figure 3: Distribution of reduction ratios over the safe instances in the MCC

We observe that we lose few opportunities to reduce a net due to our well-formedness constraint. Actually, we mostly lose the ability to simplify some instances of "partial" marking graphs that could be reduced using inhibitor arcs (a feature not supported by CÆSAR.BDD). We evaluated the performance of Kong on the 424 instances of safe Petri nets with a reduction ratio greater than 1%. We ran Kong and CÆSAR.BDD on each of those instances, in two main modes: first with a time limit of 1 h to compare the number of totally solved instances (when the tool compute a complete concurrency matrix); next with a timeout of 60 s to compare the number of values (the filling ratios) computed in the partial matrices. Computation of a partial concurrency matrix with CÆSAR.BDD is done in two phases: first a "BDD exploration" phase that can be stopped by the user; then a post-processing phase that cannot be stopped. In practice this means that the execution time is often longer (because of the post-processing phase) when we do not use Kong: the mean computation time for CÆSAR.BDD alone is about 62 s, while it is less than 21 s when we use Kong and CÆSAR.BDD together. In each test, we compared the output of Kong with the values obtained on the initial net with CÆSAR.BDD and achieved 100% reliability.

[1] we used version v3.4 of CÆSAR.BDD, part of CADP version 2020-h "Aalborg", published in August 2020.

Results for Totally Computed Matrices. We report our results on the computation of complete matrices and a timeout of 1 h in the table below. We report the number of computed matrices for three different categories of instances, *Low*/*Fair*/*High*, associated with different ratio ranges. We observe that we can compute more results with reductions than without (+25%). As could be expected, the gain is greater on category *High* (+53%), but it is still significant with the *Fair* instances (+32%).

Reduction Ratio (r)		# Test Cases	# Computed Matrices		
			Kong	cæsar.bdd	
Low	$r \in]0, 0.25[$	160	90	88	$\times 1.02$
Fair	$r \in [0.25, 0.5[$	112	53	40	$\times 1.32$
High	$r \in [0.5, 1]$	152	97	63	$\times 1.53$
Total	$r \in]0, 1]$	424	240	191	$\times 1.25$

To understand the impact of reductions on the computation time, we compare cæsar.bdd alone, on the initial net, and Kong + Reduce + cæsar.bdd on the reduced net. We display the result in a scatter plot, using a logarithmic scale (Fig. 4, left), with one point for each instance: time using reductions on the y-axis, and without on the x-axis. We use colours to differentiate between *Fair* instances (light orange) and *High* ones (dark blue), and fix a value of 3600 s when one of the computation timeout. Hence the cluster of points on the right part of the plots are when cæsar.bdd alone timeouts. We observe that the reduction ratio has a clear impact on the speed-up and that almost all the data points are below the diagonal, meaning reductions accelerate the computation in almost all cases, with many test cases exhibiting speeds-up larger than $\times 10$ or $\times 100$ (materialized by dashed lines under the diagonal).

Results with Partial Matrices. We can also compare the "accuracy" of our approach when we have incomplete results. To this end, we compute the concurrency relation with a timeout of 60 s on cæsar.bdd. We compare the *filling ratio* obtained with and without reductions. For a net with n places, this ratio is given by the formula $2\,|\mathrm{C}|/(n^2 + n)$, where $|\mathrm{C}|$ is the number of 0s and 1s in the matrix. We display our results using a scatter plot with linear scale, see Fig. 4 (right). Again, we observe that almost all the data points are on one side of the diagonal, meaning in this case that reductions increase the number of computed values, with many examples (top line of the plot) where we can compute the complete relation in 60 s only using reductions. The graphic does not discriminate between the number of 1s and 0s, but we obtain similar good results when we consider the filling ratio for only the concurrent places (the 1s) or only the nonconcurrent places (the 0s).

6 Conclusion and Further Work

The concurrency problem is difficult, especially when we cannot compute the complete state space of a net. We propose a method for transporting this problem from an initial "high-dimensionality" domain (the set of places in the net) into a smaller one (the set of places in the residual net). Our experiments

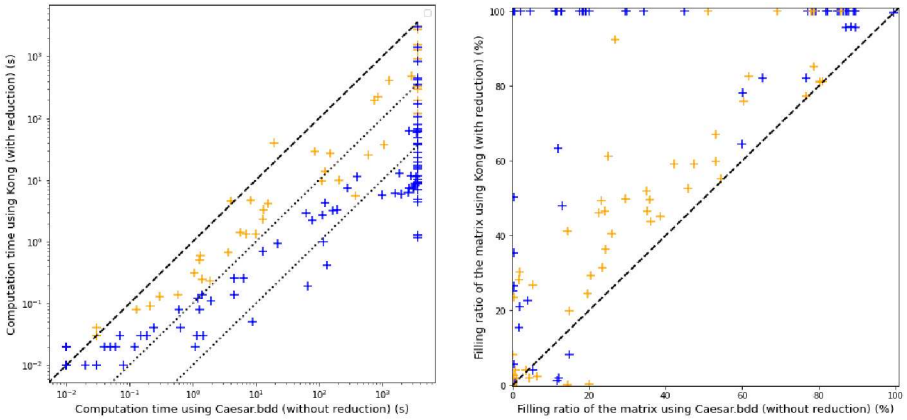

Figure 4: Comparing Kong (y-axis) and CÆSAR.BDD (x-axis) for instances with $r \in [0.25, 0.5[$ (light orange) and $r \in [0.5, 1]$ (dark blue). One diagram (left) compares the computation time for complete matrices; the other (right) compares the filling ratio for partial matrices with a timeout of 60 s.

confirm our intuition that the concurrency relation is much easier to compute after reductions (if the net can be reduced) and we provide an easy way to map back the result into the original net.

Our approach is based on a combination of structural reductions with linear equations first proposed in [4, 5]. Our main contribution, in the current work, is the definition of a new data-structure that precisely captures the structure of these linear equations, what we call the Token Flow Graph (TFG). We use the TFGs to accelerate the computation of the concurrency relation, both in the complete and partial cases. We have many ideas on how to apply TFGs to other problems and how to extend them. A natural application would be for model counting (our original goal in [4]), where the TFG could lead to new algorithms for counting the number of (integer) solutions in the systems of linear equations that we manage. Another possible application is the *max-marking* problem, which means finding the maximum of the expression $\sum_{p \in P} m(p)$ over all reachable markings. On safe nets, this amounts to finding the maximal number of places that can be marked together. We can easily adapt our algorithm to compute this value and could even adapt it to compute the result when the net is not safe.

We can even manage a more general problem, related to the notion of *max-concurrent* sets of places. We say that the set S is concurrent if there is a reachable m such that $m(p) > 0$ for all places p in S. (This subsume the case of pairs and singleton of places.) The set S is *max-concurrent* if no superset $S' \supsetneq S$ is concurrent. Computing the max-concurrent sets of a net is interesting for several reasons. First, it gives an alternative representation of the concurrency relation that can sometimes be more space efficient: (1) the max-concurrent sets provide a unique cover of the set of places of a net, and (2) we have $p \parallel q$ if and only if there is S max-concurrent such that $\{p, q\} \subset S$. Obviously, on safe nets, the size of the biggest max-concurrent set is the answer to the *max-marking* problem.

For future work, we would like to answer even more difficult questions, such

as proofs of Generalized Mutual Exclusion Constraints [13], that requires checking invariants involving a weighted sums over the marking of places, of the form $\sum_{p \in P} w_p.m(p)$. Another possible extension will be to support non-ordinary nets (which would require adding weights on the arcs of the TFG) and nets that are not safe (which can already be done with our current approach, but require changing some of the "axioms" used in our algorithm). Finally, another interesting direction for works would be to find reductions that preserve the concurrency relation (but not necessarily reachable states). As you can see, there is a lot to be done, which underlines the interest of studying TFGs.

Acknowledgements.

We would like to thank Pierre Bouvier and Hubert Garavel for their insightful suggestions that helped improve the quality of this paper.

References

[1] N. Amat, B. Berthomieu, and S. Dal Zilio. On the combination of polyhedral abstraction and SMT-based model checking for Petri nets. In *International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets)*, volume 12734 of *LNCS*. Springer, 2021.

[2] E. Amparore, B. Berthomieu, G. Ciardo, S. Dal Zilio, F. Gallà, L. M. Hillah, F. Hulin-Hubard, P. G. Jensen, L. Jezequel, F. Kordon, D. Le Botlan, T. Liebke, J. Meijer, A. Miner, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, T. van Dijk, and K. Wolf. Presentation of the 9th edition of the model checking contest. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2019.

[3] G. Berthelot. Transformations and Decompositions of Nets. In *Petri Nets: Central Models and their Properties*, LNCS, pages 359–376. Springer, 1987.

[4] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. Petri net reductions for counting markings. In *International Symposium on Model Checking Software (SPIN)*, volume 10869 of *LNCS*, pages 65–84. Springer, 2018.

[5] B. Berthomieu, D. Le Botlan, and S. Dal Zilio. Counting Petri net markings from reduction equations. *International Journal on Software Tools for Technology Transfer*, 2019.

[6] B. Berthomieu, P.-O. Ribet, and F. 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, July 2004.

[7] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba. Stubborn versus structural reductions for petri nets. *Journal of Logical and Algebraic Methods in Programming*, 102:46–63, 2019.

[8] P. Bouvier and H. Garavel. Efficient algorithms for three reachability problems in safe Petri nets. In *International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets)*, volume 12734 of *LNCS*. Springer, 2021.

[9] P. Bouvier, H. Garavel, and H. Ponce-de León. Automatic decomposition of Petri nets into automata networks – a synthetic account. In *Application and Theory of Petri Nets and Concurrency*, volume 12152. Springer, 2020.

[10] H. Garavel. Nested-unit Petri nets. *Journal of Logical and Algebraic Methods in Programming*, 104:60–85, Apr. 2019.

[11] H. Garavel. Proposal for Adding Useful Features to Petri-Net Model Checkers. Research Report 03087421, Inria Grenoble - Rhône-Alpes, Dec. 2020.

[12] H. Garavel and W. Serwe. State Space Reduction for Process Algebra Specifications. In *Algebraic Methodology and Software Technology*, LNCS. Springer, 2004.

[13] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion contraints on nets with uncontrollable transitions. In *IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 1992.

[14] L. Hillah and F. Kordon. Petri Nets Repository: A tool to benchmark and debug Petri net tools. In *Application and Theory of Petri Nets and Concurrency*, volume 10258 of *LNCS*. Springer, 2017.

[15] L.-M. Hillah, F. Kordon, L. Petrucci, and N. Treves. PNML framework: an extendable reference implementation of the Petri Net Markup Language. In *International Conference on Applications and Theory of Petri Nets*. Springer, 2010.

[16] T. Hujsa, B. Berthomieu, S. Dal Zilio, and D. Le Botlan. Checking marking reachability with the state equation in Petri net subclasses. 44 pages, Nov. 2020.

[17] INRIA. CADP. https://cadp.inria.fr/, 2020.

[18] R. Janicki. Nets, sequential components and concurrency relations. *Theoretical Computer Science*, 29(1-2), 1984.

[19] A. Kovalyov. A Polynomial Algorithm to Compute the Concurrency Relation of a Regular STG. In *Hardware Design and Petri Nets*. Springer, Boston, MA, 2000.

[20] A. V. Kovalyov. Concurrency relations and the safety problem for Petri nets. In *Application and Theory of Petri Nets 1992*, LNCS, Berlin, Heidelberg, 1992. Springer.

[21] LAAS-CNRS. Tina Toolbox. http://projects.laas.fr/tina, 2020.

[22] R. J. Lipton. Reduction: a method of proving properties of parallel programs. *Communications of the ACM*, 18(12), 1975.

[23] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[24] A. Semenov and A. Yakovlev. Combining partial orders and symbolic traversal for efficient verification of asynchronous circuits. In *Proceedings of ASP-DAC'95/CHDL'95/VLSI'95 with EDA Technofair*, 1995.

[25] M. Silva, E. Terue, and J. M. 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.

[26] Y. Thierry-Mieg. Structural reductions revisited. In *Application and Theory of Petri Nets and Concurrency*, volume 12152 of *LNCS*, pages 303–323. Springer, 2020.

[27] R. Wisniewski, A. Karatkevich, M. Adamski, A. Costa, and L. Gomes. Prototyping of Concurrent Control Systems with Application of Petri Nets and Comparability Graphs. *IEEE Transactions on Control Systems Technology*, 26(2), 2018.

[28] R. Wiśniewski, M. Wiśniewska, and M. Jarnut. C-exact hypergraphs in concurrency and sequentiality analyses of cyber-physical systems specified by safe Petri nets. *IEEE Access*, 7, 2019.

A Proofs

A.1 Proof of Lemma 3.1: Well-defined Configurations are Solutions

Lemma. *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$. If c is a well-defined configuration of $[\![E]\!]$ then $E \,,\, \lfloor c \rfloor$ is consistent. Conversely, if c is a total configuration of $[\![E]\!]$ such that $E \,,\, \lfloor c \rfloor$ is consistent then c is also well-defined.*

Proof. We prove each property separately.

Assume c is a well-defined configuration of $[\![E]\!]$. Since E is a system of reduction equations, it is a sequence of equalities $\phi_1, \dots, \phi_k$ where each equation ϕ_i has the form $x_i = y_1 + \cdots + y_n$. Also, since $[\![E]\!]$ is well-formed we have that $X_i \twoheadrightarrow\!\bullet\, v_i$ or $x_i \circ\!\!\rightarrow X_i$ (only one case is possible) with $X_i = \{y_1, \dots, y_n\}$ for all indices $i \in 1..k$. We define I the subset of indices in $1..k$ such that $c(x_i)$ is defined. By condition (CBot) we have $c(x_i) \neq \bot$ if and only if $c(v) \neq \bot$ for all $v \in X_i$. Therefore, if $c(x_i) \neq \bot$, we have by condition (CEq) that $\phi_i \,,\, \lfloor c \rfloor$ is consistent. Moreover the values of all the variables in ϕ_i are determined by $\lfloor c \rfloor$ (these variables have the same value in every solution). As a consequence, the system combining $\lfloor c \rfloor$ and the $(\phi_i)_{i \in I}$ has a unique solution. On the opposite, if $c(x_i) = \bot$ then no variables in ϕ_i are defined by $\lfloor c \rfloor$. Nonetheless, we know that system E is consistent. Indeed, by property of E-equivalence, we know that $E \,,\, \lfloor m_1 \rfloor$ has solutions, so it is also the case with E. Therefore the system combining the equations in $(\phi_i)_{i \notin I}$ is consistent. Since this system shares no variables with the equations in $(\phi_i)_{i \in I}$, we have that $E \,,\, \lfloor c \rfloor$ is consistent.

For the second case, we assume c total and $E \,,\, \lfloor c \rfloor$ consistent. Since c is total, condition (CBot) is true ($c(v) \neq \bot$ for all nodes in $[\![E]\!]$). Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. For condition (CEq), we rely on the fact that $[\![E]\!]$ is well-formed. Indeed, for all equations in E we have a corresponding relation $X \twoheadrightarrow\!\bullet\, v$ or $v \circ\!\!\rightarrow X$. Hence $E \,,\, \lfloor c \rfloor$ consistent implies that $c(v) = \sum_{w \in X} c(w)$. □

A.2 Proof of Lemma 3.2: Token Propagation

Lemma. *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$ and c a well-defined configuration of $[\![E]\!]$.*

(Agglomeration) if $p \circ\!\!\rightarrow \{q_1, \dots, q_k\}$ and $c(p) \neq \bot$ then for every sequence $(l_1, \dots, l_k)$ in $\mathbb{N}^k$ such that $c(p) = \sum_{i \in 1..k} l_i$ we can find a well-defined configuration c' such that $c'(p) = c(p)$, and $c'(q_i) = l_i$ for all $i \in 1..k$, and $c'(v) = c(v)$ for every node v not in $\downarrow p$.

(Forward) for every pair (p, q) of nodes such that $c(p) \neq \bot$ and $p \rightarrow^\star q$ we can find a well-defined configuration c' such that $c'(q) \geqslant c'(p) = c(p)$ and $c'(v) = c(v)$ for every node v not in $\downarrow p$.

(Backward) if $c(p) > 0$ then there is a root v such that $v \rightarrow^\star p$ and $c(v) > 0$.

Proof. We prove each property separately.

(Agglomeration Propagation): we prove that we can update the successors of p by following the order induced by the tree-like structure of the TFG. To

this end, we introduce the notion of level of a node. The level of a node v in $[\![E]\!]$, denoted $\mathrm{lvl}(v)$, is the length of longest path $r \to^\star v$ from a root r of $[\![E]\!]$ to v. The level can only increase when we follow an arc and is always defined since we have no cycles in the TFG.

Take a node p at level l such that $p \circ\!\!\to X$, with $X = \{q_1, \ldots, q_k\}$, and a sequence $(l_1, \ldots, l_k) \in \mathbb{N}^k$ such that $c(p) = \sum_{i \in 1..k} l_i$. We define configuration c' as follows. Take $c'(q_i) = l_i$ for all $i \in 1..k$, and $c'(p) = c(p)$, and $c'(v) = c(v)$ for all the nodes v such that $v \notin {\downarrow}p$ or $\mathrm{lvl}(v) \leqslant l$. We still need to find suitable values, $c'(w)$, for all the nodes w that are successors of the nodes in X. We proceed "levels after levels". Note that, by construction, we have that $\mathrm{lvl}(w) \geqslant l + 1$. If $c'(w)$ is in the last defined level and $w \circ\!\!\to Y$, then c' cannot be already defined over Y (otherwise it would mean that these nodes can be removed twice). In this case we are free to choose any possible valuation such that $\sum_{w' \in Y} c'(w') = c'(w)$ and we continue recursively with the successors of Y. If $Y \twoheadrightarrow\!\!\bullet\, w$ then all the nodes in Y are in a level smaller than w and therefore c' is defined over Y. In this case we choose $c'(w) = \sum_{w' \in Y} c'(w')$. Since we have a finite DAG, this process terminates with c' a total configuration. The proof proceeds by showing that c' is a well-defined configuration, which is obvious.
(Forward Propagation): take a well-defined configuration c of $[\![E]\!]$ and assume we have two nodes p, q such that $c(p) \neq \bot$ and $p \to^\star q$. The proof is by induction on the length of the path from p to q. The initial case is when $p = q$, which is trivial. Otherwise, assume $p \to r \to^\star q$. It is enough to find a well-defined configuration c' such that $c'(r) \geqslant c'(p) = c(p)$. Since the nodes not in ${\downarrow}p$ are not in the paths from p to q, we can ensure $c'(v) = c(v)$ for any node v not in ${\downarrow}p$. The proof proceeds by a case analysis on $p \to r$.

(Case R) assume $p \to r$ is a R-arc, meaning $p \twoheadrightarrow\!\!\bullet\, r$. More generally, it follows that $X \twoheadrightarrow\!\!\bullet\, r$ with $p \in X$. Then by (CEq) we have $c(r) = c(p) + \sum_{v \in X, v \neq p} c(v) \geqslant c(p)$ and we can choose $c' = c$.

(Case A) in this case we have $p \circ\!\!\to X$ with $r \in X$. By (Agglomeration Propagation) we can find a well-defined configuration c' such that $c'(r) = c'(p) = c(p)$ (and also $c'(v) = 0$ for all $v \in X \setminus \{r\}$).

(Backward Propagation): take a well-defined configuration c of $[\![E]\!]$ and assume we have $c(p) > 0$. The proof is by induction on the longest possible path from a root to node p. We re-use the notion of levels introduced previously. The initial case is when p is itself a root, $\mathrm{lvl}(p) = 0$, and is trivial. Otherwise there must be at least one predecessor node q such that $q \to p$. Like in the previous proof, we proceed by case analysis.

(Case R) we have $X \twoheadrightarrow\!\!\bullet\, p$ with $X \neq \emptyset$. By (CEq) we have $c(p) = \sum_{v \in X} c(v) > 0$. Hence there must be at least one node q in X such that $c(q) > 0$ and necessarily $\mathrm{lvl}(p) \geqslant \mathrm{lvl}(q) + 1$.

(Case A) in this case we have $q \circ\!\!\to X$ with $p \in X$. By (CEq) we have $c(q) = c(p) + \sum_{v \in X \setminus \{p\}} c(v) \geqslant c(p)$ as needed, with $\mathrm{lvl}(p) = \mathrm{lvl}(q) + 1$.

□

A.3 Proof of Theorem 3.3: Configuration Reachability

Theorem. *Assume $[\![E]\!]$ is a well-formed TFG for the equivalence $(N_1, m_1) \rhd_E (N_2, m_2)$. If m is a marking in $R(N_1, m_1) \cup R(N_2, m_2)$ then there exists a total, well-defined configuration c of $[\![E]\!]$ such that $c \equiv m$. Conversely, if c is a total, well-defined configuration of $[\![E]\!]$ then marking $c_{|N_1}$ is reachable in (N_1, m_1) if and only if $c_{|N_2}$ is reachable in (N_2, m_2).*

Proof. Take m a marking in $R(N_1, m_1)$. The other case is totally symmetric. By property of E-abstraction, there exists a reachable marking m'_2 in $R(N_2, m_2)$ such that $E \,,\, \lfloor m \rfloor \,,\, \lfloor m'_2 \rfloor$ is consistent. Therefore we can find a non-negative integer solution c to the system $E \,,\, \lfloor m \rfloor \,,\, \lfloor m'_2 \rfloor$, meaning a valuation for all the variables and places in $fv(E), N_1$ and N_2 such that $E \,,\, \lfloor c \rfloor$ is consistent and $c(p) = m(p)$ if $p \in N_1$ and $c(p) = m'_2(p)$ if $p \in N_2$. Because of condition (T1), this solution is total over all the nodes of $[\![E]\!]$ (the only other possible case is for constants, whose values are fixed).

For the converse property, we assume that c is a total and well-defined configuration of $[\![E]\!]$ and that $c_{|N_1}$ is a marking of $R(N_1, m_1)$. Since c is a well-defined configuration, from Lemma 3.1 we have that $E \,,\, \lfloor c \rfloor$ is consistent. Therefore we have that $E \,,\, \lfloor c_{|N_1} \rfloor \,,\, \lfloor c_{|N_2} \rfloor$ is consistent. By definition of the E-abstraction, condition (A3), we have $c_{|N_2}$ in $R(N_2, m_2)$, as needed. □

A.4 Safe Configurations

For the sake of simplicity, we can assume that all the leaf nodes in $[\![E]\!]$ are places in N_1. This is true for TFGs computed from structural reductions and this will simplify our proofs: for every node v we can always assume that there is p in N_1 such that $v \to^\star p$.

In our proof, we also implicitly assume that all the constants in E are either 0 or 1. We could relax this last constraint, but this would needlessly complicate our algorithm.

Lemma A.1 (Safe Configurations)**.** *Assume $[\![E]\!]$ is a well-formed TFG for $(N_1, m_1) \rhd_E (N_2, m_2)$ with (N_1, m_1) and (N_2, m_2) safe Petri nets. Then for every total, well-defined configuration c of $[\![E]\!]$ such that $c_{|N_2}$ reachable in (N_2, m_2), and every node v, we have $c(v) \in \{0, 1\}$.*

Proof. We prove the result by contradiction. Take a total and well-defined configuration c such that $c_{|N_2}$ is reachable in (N_2, m_2) and a node v such that $c(v) > 1$ and $v \to^\star p$, with p a place of N_1. By Lemma 3.2, we can find a well-defined configuration c' of $[\![E]\!]$ such that $c'(p) \geqslant c'(v) = c(v)$ and $c'(w) = c(w)$ for every node w not in $\downarrow v$. Therefore $c'_{|N_2}$ is also reachable in (N_2, m_2). This contradicts the fact that the nets are safe since, by Th. 3.3, we would have a reachable marking that is not 1-bounded. □

A.5 Checking Dead Places using Configurations

By our configuration reachability theorem (Th. 3.3), if we take reachable markings in N_2—meaning we fix the values of roots in $[\![E]\!]$—we can find places of N_1 that are marked together by propagating tokens from the roots to the leaves (Lemma 3.2). We prove that we can compute the concurrency relation of N_1 by looking at just two cases: (1) we start with a token in a single root p, with

p not dead, and propagate this token forward until we find a configuration with two places of N_1 marked together; or (2) we do the same but placing a token in two separate roots, p_1, p_2, such that $p_1 \parallel p_2$. We base our approach on the fact that we can extend the notion of concurrent places (in a marked net), to the notion of concurrent nodes in a TFG. Those are the nodes that can be marked together in a "reachable configuration".

Definition A.1 (Concurrent Nodes)**.** *The concurrency relation of $[\![E]\!]$, denoted $\mathcal{C}$, is the relation between pairs of nodes in $[\![E]\!]$ such that $v \mathcal{C} w$ if and only if there is a total, well-defined configuration c where: (1) c is reachable, meaning $c_{|N_2} \in R(N_2, m_2)$; and (2) $c(v) > 0$ and $c(w) > 0$.*

Like with the concurrency relation on nets, we have that $\mathcal{C}$ is symmetric and $v \mathcal{C} v$ means that v is not-dead (there is a valuation with $c(v) > 0$). We can also extend this relation to define a notion of max-concurrent sets of nodes.

By definition, if p, q are places in N_2 then $p \mathcal{C} q$ only if $p \parallel q$ in (N_2, m_2). We say in this case that p, q are *concurrent roots*. We can extend this notion to constants. We say that two roots v_1, v_2 are concurrent when $v_1 \mathcal{C} v_2$ and that root v_1 is not-dead when $v_1 \mathcal{C} v_1$. This include cases where v_1 or v_2 are in $K(1)$ (they are constants with value 1).

Since the places of N_1 are nodes in $[\![E]\!]$, we also have that $p \mathcal{C} q$ if and only if $p \parallel q$ in (N_1, m_1). This is the relation we use in our algorithm of Sect. 4.

We prove some properties about the relation $\mathcal{C}$ that are direct corollaries of our token propagation properties. For all the following results, we implicitly assume that $[\![E]\!]$ is a well-formed TFG for the relation $(N_1, m_1) \rhd_E (N_2, m_2)$, that both marked nets are safe, that all the roots in $[\![E]\!]$ are either constants or places in N_2; and that $\mathcal{C}$ is the concurrency relation of $[\![E]\!]$.

We start with a property (Lemma A.2) stating that the successors of a "live node" must also be not-dead. Lemma A.3 provides a dual result, useful to prove the completeness of our approach; it states that it is enough to explore the live roots to find all the live nodes.

Lemma A.2 (Propagation of Live Nodes)**.** *If $v \mathcal{C} v$ and $v \to^\star w$ then $w \mathcal{C} w$.*

Proof. Assume $v \mathcal{C} v$. This means that there is a total, well-defined configuration c such that $c(v) > 0$ and $c_{|N_2} \in R(N_2, m_2)$. Now take a successor node of v, say $v \to^\star w$. By Lemma 3.2, we can find another reachable configuration c' such that $c'(w) \geqslant c'(v) = c(v)$ and $c'(x) = c(x)$ for all nodes x not in $\downarrow v$. Therefore $w \mathcal{C} w$. □

Lemma A.3 (Live Nodes Come from Live Roots)**.** *If $v \mathcal{C} v$ then there is a root v_0 such that $v_0 \mathcal{C} v_0$ and $v_0 \to^\star v$.*

Proof. Assume $v \mathcal{C} v$. Then there is a total, well-defined configuration c such that $c_{|N_2} \in R(N_2, m_2)$ and $c(v) > 0$. By the backward propagation property of Lemma 3.2 we know that there is a root, say v_0, such that $c(v_0) \geqslant c(v)$ and $v_0 \to^\star v$. Hence v_0 is not-dead in $[\![E]\!]$. □

A.6 Checking Concurrent Places using Configurations

We can prove similar results for concurrent nodes instead of live ones. We consider the two cases mentioned at the beginning of the section: when concurrent nodes are obtained from two concurrent roots (Lemma A.4); or when they

are obtained from a single live root (Lemma A.5), because of redundancy arcs. Finally Lemma A.6 provides the associated completeness result.

Lemma A.4. *Assume v, w are two nodes in $[\![E]\!]$ such that $v \notin \downarrow w$ and $w \notin \downarrow v$. If $v \mathcal{C} w$ then $v' \mathcal{C} w'$ for all pairs of nodes $(v', w') \in \downarrow v \times \downarrow w$.*

Proof. Assume $v \mathcal{C} w$, $v \notin \downarrow w$ and $w \notin \downarrow v$. There must exist a total and well-defined configuration c such that $c(v), c(w) > 0$ and $c_{|N_2} \in R(N_2, m_2)$.

Take a successor v' in $\downarrow v$, by applying the token propagation from Lemma 3.2 we can construct a total and well-defined configuration c' of $[\![E]\!]$ such that $c'(v') \geqslant c'(v) = c(v)$ and $c'(x) = c(x)$ for any node x not in a $\downarrow v$. This is the case of w, hence $c'(w) = c(w) > 0$.

We can use the token propagation property again, on c'. This gives a total and well-defined configuration c'' such that $c''(w') \geqslant c''(w) = c'(w) = c(w)$ and $c''(x) = c'(x)$ for any node x not in $\downarrow w$.

If we prove that $v' \notin \downarrow w$ we will then have $c''(v') = c'(v') \geqslant c(v)$, and therefore $v' \mathcal{C} w'$ as needed. We prove this result by contradiction. Indeed, assume $v' \in \downarrow w$. Hence, $\downarrow v \cap \downarrow w \neq \emptyset$. Moreover, since E is a well-formed TFG, there must exists (condition T3) three nodes p, q, r such that $X \twoheadrightarrow r$, $p \in \downarrow v \cap X$ and $q \in \downarrow w \cap X$. In a similar way than the proof of Lemma 3.2 we can propagate the tokens contained in v, w to p, q, and obtain $c''(r) > 1$ from (CEq), which contradicts our assumption that the nets are safe. □

Lemma A.5. *If $v \mathcal{C} v$ and $v \twoheadrightarrow w$ then $v' \mathcal{C} w'$ for every pair of nodes (v', w') such that $v' \in (\downarrow(v) \setminus \downarrow w)$ and $w' \in \downarrow w$.*

Proof. Assume $v \mathcal{C} v$ and $v \twoheadrightarrow w$. Hence there is a total, well-defined configuration c such that $c(v) > 0$ and $c_{|N_2} \in R(N_2, m_2)$. Furthermore, since $v \twoheadrightarrow w$, we must have $c(w) > 0$ (condition CEq).

Take w' in $\downarrow w$. From Lemma 3.2 we can find a total, well-defined configuration c' such that $c'(w') \geqslant c'(w) = c(w) > 0$ and $c'(x) = c(x)$ for any node x not in $\downarrow w$. Since v is not in $\downarrow w$ we have $c'(v) = c(v)$. Likewise, places from N_2 are roots and therefore cannot be in $\downarrow w$. So we have $c'_{|N_2} \equiv c_{|N_2}$, which means $c'_{|N_2}$ is reachable in (N_2, m_2). At this point we have $v \mathcal{C} w'$.

Now, consider $v' \neq w$ such that $v \to v'$. We can use the forward propagation lemma a second time on c' to find a total and well-defined configuration c'' such that $c''(v') \geqslant c''(v) = c'(v)$ and $c''(x) = c(x)$ for all nodes x not in $\downarrow v$, and so, $c''_{|N_2}$ is reachable in (N_2, m_2). Since configuration c'' is well-defined we have (condition CEq) that $c''(v) = c''(w)$. We also have $v' \notin \downarrow w$ and $w \notin \downarrow v'$ since $v \to w$, $v \to v'$ and $[\![E]\!]$ is a well-formed TFG that must satisfy (T3). Finally, using Lemma A.4 is enough to prove that $v'' \mathcal{C} w'$ for every node $v'' \in \downarrow v'$. □

Lemma A.6. *If $v \mathcal{C} w$ and $v \neq w$ then one of the following two conditions is true.*

(Redundancy) *There is a live node v_0 such that $v_0 \twoheadrightarrow w_0$ and either (w, w) or (w, v) are in $(\downarrow v_0 \setminus \downarrow w_0) \times \downarrow w_0$.*

(Agglomeration) *There is a pair of distinct roots (v_0, w_0) such that $v_0 \mathcal{C} w_0$ with $v \in \downarrow v_0$ and $w \in \downarrow w_0$.*

Proof. Assume $v \mathcal{C} w$. Then there is a total, well-defined configuration c such that $c_{|N_2} \in R(N_2, m_2)$ and $c(u), c(v) = 1$ (the nets are safe). By the backward-propagation property in Lemma 3.2 there exists two roots v_0 and w_0 such that $c(v_0) = c(w_0) = 1$ with $v \in \downarrow v_0$ and $w \in \downarrow w_0$. We need to consider two cases, either $v_0 \neq w_0$ or $v_0 = w_0$.

The case where $v_0 \neq w_0$ corresponds to condition (Agglomeration).

In the case where $v_0 = w_0$, we prove that there must be a node v_1 such that $v_0 \rightarrow^\star v_1$ and $v_1 \rightarrow\!\!\bullet\, w_1$ with either (v, w) or (w, v) in $(\downarrow v_1 \setminus \downarrow w_1) \times \downarrow w_1$. We prove this result by contradiction. Indeed, if no such node exists then both v and w can be reached from v_0 by following only edges in A. Using the backward propagation property twice, and since $v \neq w$, this means that we can find a configuration c' such that $c'_{|N_2} \equiv c_{|N_2}$ and $c'(v_0) \geqslant c(v) + c(w) \geqslant 2$, which contradicts our hypothesis that the nets are safe. □

A.7 Proof of Theorem 4.1: our Algorithm is Sound and Complete

We prove a slightly different property that entails Th. 4.1. The following property makes use of the notations introduced in the previous section and proves an equivalent result but for all the nodes in $[\![E]\!]$, not only for the places in N_1.

Theorem. *If* C *is the matrix returned by a call to* `Matrix`$([\![E]\!], \|)$*, with* $\|$ *the concurrency relation between roots of* $[\![E]\!]$ *(meaning* N_2 *augmented with the constants), then for all nodes* v, w *we have* $v \mathcal{C} w$ *if and only if either* $\mathrm{C}[v, w] = 1$ *or* $\mathrm{C}[w, v] = 1$.

Proof. We can first remark that the call to `Matrix`$([\![E]\!], \|)$ will always terminate. We divide the proof into two different cases: first we prove that the computation of live nodes (the diagonal of C and the live nodes of $\mathcal{C}$) is sound and complete. Next, we prove the same result for concurrent nodes.
(Non-dead Places) The result is a direct consequence of Lemmas A.2 and A.3.
(Concurrent Places) We need to consider the two cases describe in Lemma A.6. The second **foreach** loop in the code of `Matrix` takes care of the cases where concurrency is a consequence of two distinct live roots. The second case corresponds to the **foreach** loop at line 7 in the code of `Propagate`, for the matrix, and Lemma A.4 for $\mathcal{C}$. Finally, Lemma A.6 implies that this phase of the computation is complete. □

A.8 Axioms for Computing Incomplete Concurrency Matrices

Our algorithm for the case of incomplete matrices is based on a collection of six additional axioms used to "propagate 0s" in the matrix C. We state each axiom separately and, in each case, we prove a property that states that the axiom is sound. Completeness takes a different meaning in this case. Indeed, we cannot prove that we find all the pairs of nonconcurrent nodes. But we can prove a result about the accuracy, meaning that all verdicts $\mathrm{C}[v, w] \neq \bullet$ must originate from some roots p, q such that $\mathrm{C}[p, q]$ is defined. More precisely, two concurrent nodes ($\mathrm{C}[v, w] = 1$) must come from concurrent roots (or one live root), and two nonconcurrent nodes ($\mathrm{C}[v, w] = 0$) imply that roots leading to v must all be nonconcurrent from roots leading to w.

In the following, we use the notation $v\,\bar{\mathcal{C}}\,w$ to say $\neg(v\mathrm{C}w)$; meaning v, w are nonconcurrent according to C. With our notations, $v\,\bar{\mathcal{C}}\,v$ means that v is dead: there is no well-defined, reachable configuration c with $c(v) > 0$.

A.8.1 Propagation of Dead Nodes.

We prove that a dead node, v, is necessarily nonconcurrent from all the other nodes. Also, if all the "direct successors" of a node are dead then also is the node.

Lemma A.7. *Assume v a node in $[\![E]\!]$. If $v\,\bar{\mathcal{C}}\,v$ then for all nodes w in $[\![E]\!]$ we have $v\,\bar{\mathcal{C}}\,w$.*

Proof. Assume $v\,\bar{\mathcal{C}}\,v$. Then for any total and well-defined configuration c such that $c_{|N_2}$ is reachable in (N_2, m_2) we have $c(v) = 0$. By definition of the concurrency relation $\mathcal{C}$, v cannot be concurrent to any node. □

Lemma A.8. *Assume v a node in $[\![E]\!]$ such that $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$. Then $v\,\bar{\mathcal{C}}\,v$ if and only if $w\,\bar{\mathcal{C}}\,w$ for all nodes w in X.*

Proof. We prove by contradiction both directions.

Assume $v\,\bar{\mathcal{C}}\,v$ and take $w \in X$ such that $w\,\mathcal{C}\,w$. Then there is a total, well-defined configuration c such that $c(w) > 0$. Necessarily, since $v\,\bar{\mathcal{C}}\,v$ we have $c(v) = 0$, which contradicts (CEq).

Next, assume $v\,\mathcal{C}\,v$ and $w\,\bar{\mathcal{C}}\,w$ for every node $w \in X$. Then there is a total, well-defined configuration c such that $c(v) > 0$. Necessarily, for all nodes $w \in X$ we have $c(w) = 0$, which also contradicts (CEq). □

These properties imply the soundness of the following three axioms:

- If $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[v, w] = 0$ for all node w in $[\![E]\!]$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v] = 0$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[v, v] = 0$ then $\mathrm{C}[w, w] = 0$ for all nodes $w \in X$.

A.8.2 Independency between Siblings.

We prove that direct successors of a node are nonconcurrent from each other (in the case of safe nets). This is basically a consequence of the fact that $c(v) = c(w) + c(w') + \ldots$ and $c(v) \leqslant 1$ implies that at most one of $c(w)$ and $c(w')$ can be equal to 1 when the configuration is fixed.

Like with our "safeness property", we assume for the sake of simplicity that all the leaves in $[\![E]\!]$ are places in N_1.

Lemma A.9. *Assume v a node in $[\![E]\!]$ such that $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$. For every pair of nodes w, w' in X, we have that $w \neq w'$ implies $w\bar{C}w'$.*

Proof. The proof is by contradiction. Take a pair of distinct nodes w, w' in X and assume $w\,\mathcal{C}\,w'$. Then there exists a total and well-defined configuration c such that $c(w) \geqslant 1$ and $c(w') \geqslant 1$, with $c_{|N_2}$ reachable in (N_2, m_2). Since c must satisfy (CEq) we have $c(v) \geqslant 2$, which contradicts the fact that our nets are safe, see Lemma A.1. □

This property implies the soundness of the following axiom:

- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ then $\mathrm{C}[w, w'] = 0$ for all pairs of nodes $w, w' \in X$ such that $w \neq w'$.

A.8.3 Heredity and Independency.

We prove that if v and v' are nonconcurrent, then v' must be nonconcurrent from all the direct successors of v (and reciprocally). This is basically a consequence of the fact that $c(v) = c(w) + \ldots$ and $c(v) + c(v') \leqslant 1$ implies that $c(w) + c(v') \leqslant 1$.

Lemma A.10. *Assume v a node in $[\![E]\!]$ such that $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$. Then for every node v' such that $v \,\bar{\mathcal{C}}\, v'$ we also have $w \,\bar{\mathcal{C}}\, v'$ for every node w in X. Conversely, if $w \,\bar{\mathcal{C}}\, v'$ for every node w in X then $v \,\bar{\mathcal{C}}\, v'$.*

Proof. We prove by contradiction each property separately.

Assume $v \,\bar{\mathcal{C}}\, v'$ and take $w \in X$ such that $w \,\mathcal{C}\, v'$. Then there is a total, well-defined configuration c such that $c(w), c(v') > 0$. Necessarily, since $v \,\bar{\mathcal{C}}\, v'$ we must have $c(v) = 0$ or $c(v') = 0$. We already know that $c(v') > 0$, so $c(v) = 0$, which contradicts (CEq) since $w \in X$.

Next, assume $w \,\bar{\mathcal{C}}\, v'$ for all nodes $w \in X$ and we have $v \,\mathcal{C}\, v'$. Then there is a total, well-defined configuration c such that $c(v), c(v') > 0$. Necessarily, for all nodes $w \in X$ we have $c(w) = 0$ or $c(v') = 0$. We already know that $c(v') > 0$, so $c(w) = 0$ for all nodes $w \in X$, which also contradicts (CEq). □

These properties imply the soundness of the following two axioms:

- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[w, v'] = 0$ for all nodes $w \in X$ then $\mathrm{C}[v, v'] = 0$.
- If $v \circ\!\!\rightarrow X$ or $X \rightarrow\!\!\bullet\, v$ and $\mathrm{C}[v, v'] = 0$ then $\mathrm{C}[w, v'] = 0$ for all nodes w in X.