Master of Science in Informatics at Grenoble
Master Informatique
Specialization High-confidence Embedded and Cyberphysical Systems

A New Approach for the Symbolic Model Checking of Petri Nets

Nicolas AMAT

June 23, 2020
(Revised version)

Research project performed at LAAS-CNRS x INRIA/LIG

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir

Under the supervision of:
Dr. Silvano DAL ZILIO
Dr. Hubert GARAVEL
Dr. Didier LE BOTLAN

Defended before a jury composed of:
Pr. Laurence PIERRE
Dr. Akram IDANI

June 2020

Abstract

The Vertics team at LAAS-CNRS has been developing a new approach for the symbolic model checking for years, i.e., a method to represent exactly the state-space of a system without enumerating it exhaustively. This study's originality consists in the verification of properties on a Petri net N from a reduced net N', as well as a system of linear equations E linking these two nets.

This study focuses on the adaptation of SAT methods such as BMC and PDR into SMT methods to verify efficiently different properties on non-safe Petri nets by taking advantage of these nets reductions.

After defining the reductions used, this study defines a new notion of equivalence to prove the soundness of these reduction rules.

This study is also interested in the decomposition of Petri nets into subnet called NUPNs, introduced by the Convecs team from INRIA. This decomposition requires the computation of concurrency relations between the places composing the net. This work proposes a promising method to answer this question on large nets by treating the reduction equations.

Résumé

L'équipe Vertics du LAAS-CNRS développe depuis quelques années une nouvelle approche pour le model checking symbolique, c'est-à-dire une méthode permettant de représenter exactement l'espace d'état d'un système sans devoir l'énumérer de manière exhaustive. L'originalité de cette étude consiste en la vérification, de propriétés sur un réseau de Petri N à partir d'un réseau réduit N', ainsi qu'un système d'équations linéaires E liant ces deux réseaux entre eux.

Cette étude se concentre sur l'adaptation de méthodes SAT telles que BMC et PDR en des méthodes SMT pour vérifier efficacement différentes propriétés sur des réseaux de Petri non saufs en tirant parti de ces réductions de réseaux.

Après avoir défini les réductions utilisées, cette étude définit une nouvelle notion d'équivalence afin de prouver la correction de ces règles de réduction.

Cette étude s'intéresse également à une question de décomposition de réseaux de Petri en sous-réseaux appelés NUPNs, introduite par l'équipe Convecs de l'INRIA. Cette décomposition nécessite le calcul de relations de concurrence entre les places composant le réseau. Ce travail introduit une méthode prometteuse afin de répondre à cette question sur de grands réseaux en traitant les équations de réduction.

Remerciements

Nous y voilà, ce rapport cloture un projet de recherche passionnant ainsi que cinq années d'études.

Je tiens tout d'abord à remercier Silvano Dal Zilio et Hubert Garavel, pour leur sujet proposé, qui était en parfaite adéquation avec les domaines que je souhaitais explorer.

Ce fut un grand plaisir de travailler avec Hubert Garavel, je le remercie pour nos nombreux échanges par mail, pour ses précieux conseils, pour son aide, pour sa confiance, et pour m'avoir donné la chance de reviewer un article pour la conférence MARS2020. Je tiens encore une fois à remercier Silvano Dal Zilio, sa passion communicante, sa gentillesse, ainsi que tout le temps qu'il m'a consacré m'ont permis d'aboutir à ce rapport. Il a été pour moi un mentor, et je fut honoré de "faire de la science" avec lui.

Je tiens également à remercier le cadre scientifique de l'équipe Vertics, Didier Le Botlan, François Vernadat et Pierre-Emmanuel Hladik, avec qui j'ai eu la chance de travailler, d'échanger mais aussi de rigoler. Je tiens à remercier Thomas Husja, pour ses conseils et sa culture scientifique qu'il a su me partager. Un grand merci également à Éric Lubat, pour son amitié et les liens que nous avons gardé durant le confinement.

Je souhaite remercier les nombreux enseignants-chercheurs que j'ai pu côtoyer durant mes études, qui m'ont donné l'envie de faire de la recherche.

Et pour finir, un immense merci à ceux, pour qui je serai éternellement reconnaissant, mes parents, qui m'ont permis de réaliser mes études et de vivre mes rêves.

Contents

Abstract i

Résumé i

Remerciements iii

1 Introduction 1
- 1.1 Model Checking Overview . . . 2
- 1.2 Petri Nets . . . 3
- 1.3 SAT/SMT Solving . . . 6

2 Motivations 9
- 2.1 Net Reductions . . . 9
- 2.2 Properties Verification . . . 10
- 2.3 Contributions and Structure of the Report . . . 11

3 State-of-the-art 13
- 3.1 Binary Decision Diagrams . . . 13
- 3.2 SAT and SMT . . . 14
- 3.3 Model Checking Contest (MCC) . . . 15
- 3.4 Use of Systems of Linear Equations and the Polyhedral Approach . . . 15

4 Reachability Equivalence and Net-Abstractions 17
- 4.1 Satisfiability Conditions on the Systems of Equation . . . 18
- 4.2 E-Abstraction Equivalence . . . 18
- 4.3 Basic Properties of Polyhedral Abstraction . . . 19
- 4.4 Reductions Rules . . . 20
- 4.5 Composition Laws . . . 20
  - 4.5.1 Compatibility of Equations . . . 28
  - 4.5.2 Composability (COMP) . . . 28
  - 4.5.3 Transitivity (TRANS) . . . 29
  - 4.5.4 Relabeling (RENAME) . . . 30

4.6 Deriving E-Equivalences using Reductions . . . 30
4.7 An Example of Totally Reducible Nets . . . 31

5 Implementation **35**
5.1 Enumerative Markings . . . 35
5.2 Bounded Model Checking (BMC) . . . 37
5.3 Property Directed Reachability (PDR) . . . 41

6 Concurrent Places Problem **49**
6.1 Nested-Unit Petri Net (NUPN) . . . 49
6.2 Concurrent Places Problem . . . 50
6.3 Concurrency Matrix Construction . . . 51
6.4 Change of Basis using Reduction Equations . . . 54
6.4.1 Algorithm . . . 55
6.4.2 Example . . . 55

7 Experimental Results **57**

8 Conclusion and Perspectives **63**

A Appendix **65**
A.1 Proof of Correctness for a Reduction rule . . . 65
A.2 SMPT Usage . . . 68
A.3 AirplaneLD Concurrency Matrix . . . 69

Bibliography **71**

— 1 —
Introduction

To overcome the increasing complexity of critical systems, it is necessary to improve the methods and tools used in system engineering. Complex systems nowadays, such as satellites, nuclear power plants, or airplanes, rely on control software comprised of several million lines of code. A bug on one of these systems can be a human and financial disaster. We can cite some classical examples of "computer disasters". In June 1996, the crash caused by an integer overflow of the European Ariane 5-missile cost more than 500 million US$. The European Space Agency did not detect the error due to a lack of proper integration test with realistic acceleration value as it would occur. Several other historical examples, but also a string of success stories [Garavel, 2012], have raised interest in the use of formal methods to help avoid similar problems in the future.

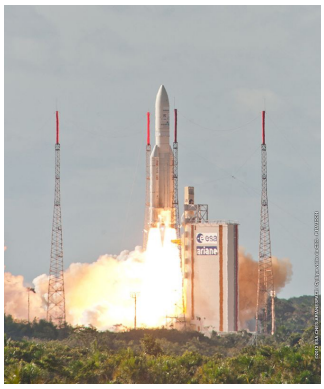


Figure 1.1: Ariane 5

Figure 1.2: Ariane 5 crash, 1996

The appeal of using *formal methods*, and of formal verification techniques in particular, is to provide rigorous mathematical techniques to verify some properties (correct behavior) on software or hardware systems. Indeed, many complex systems cannot be fully tested, their state-space is often too big to enumerate all the scenarios that can occur, and their complexity far-exceeds what any individual engineer can manage on their own.

Verification tools can be divided into three main categories:

- **Automated theorem proving**: formal proofs of the program, usually based on the use of a specific logical system and with the support of a proof assistant,
- **Model checking**: verification of the *behavioral properties* of a system by exploring its possible states, where properties may be expressed using a temporal logic,
- **Static analysis** techniques, such as abstract interpretation for instance, that relies on an analysis of program code, without execution, generally by computing an over-approximation of the program behavior.

During my internship at LAAS, I have studied and developed new methods and algorithms for the model checking of systems modeled as Petri nets. I have also implemented some of these new ideas into a prototype tool called SMPT.

1.1 Model Checking Overview

Model Checking is a formal method for checking whether a model of a system meets a given specification [Baier and Katoen, 2008, Clarke et al., 1999]. A (property) **specification** describes the properties of interest, in other words, what the system should do and the characteristics it should have. A **model** defines the idealized behavior of the system and how it interacts with the external world. This technique is used at the different stages of systems development (design, architecture, etc.) and is based, roughly, on exploration over all the states that the system can take. In my work, I focus on *reachability properties* (sometimes also called *safety* properties), meaning properties on the states that the system can reach.

Model Checking is composed of three main elements to perform verification:

- **A property specification language**, that is a mathematical formalism to describe the properties that the designed system must verify. Different temporal logic can be used, such as LTL (Linear Temporal Logic) or CTL (Computation Tree Logic).
- **A behavioral specification language**, that is a formalism to describe the system and its behavior. A model-checker can work with different formalisms such as automata, transition system, Petri net, and many others.
- **A verification technique**, that is a method to prove that the system satisfies the given properties or return a counter-example if it is not the case. Besides "traditional" enumerative techniques, two main approaches can be found: the first one based on the use of decision diagrams (such as Binary Decision Diagrams, BDD, for example); and a second one based on SAT solvers.

Nowadays, we have sophisticated academic and industrial tools [Kordon et al., 2019] that can be used to model-check systems. During my work, I have mostly focused on the TINA model-checking toolbox [LAAS-CNRS, 2020], developed by the Vertics team at LAAS.

1.2 Petri Nets

Petri nets, also called *Place/Transition (P/T) nets*, are a mathematical model of concurrent systems defined by Carl Adam Petri. The idea is to describe the state of a system using *places*, containing tokens. A change of state of the system is represented by *transitions*. Places are connected to transitions by *arcs*. If a condition on the number of tokens in the *inputs places* is met, the transition can *fire*, in this case some tokens are removed from the *input places*, and some are added to the *output places*. Basically, places are a representation of the states, conditions, and resources of a system, while transitions symbolize actions. A complete formalization of Petri nets can be found in [Murata, 1989] and [Diaz, 2009].

Syntax

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). A state m of a net, also called a *marking*, is a mapping $m : P \to \mathbb{N}$ which assigns a number of *tokens*, $m(p)$, to each place p in P. A marked net (N, m_0) is a pair composed from a net and an initial marking m_0. In the following, we will often consider that each transition is associated with a label (a symbol taken from an alphabet Σ). In this case, we assume that a net is associated with a labeling function $l : T \to \Sigma \cup \{\tau\}$, where τ is a special symbol for the silent action name. Every net has a default labeling function l_N such that $\Sigma = T$ and $l_N(t) = t$ for every transition $t \in T$.

Useful Notations

The *pre-set* of a transition $t \in T$ is denoted ${}^\bullet t = \{p \in P \mid \mathbf{pre}(t, p) > 0\}$, respectively the *post-set* of a transition t is denoted $t^\bullet = \{p \in P \mid \mathbf{post}(t, p) > 0\}$. We call $\mathbf{pre}(t, p)$ and $\mathbf{post}(t, p)$ the weight of the arc between p and t. A Petri net is called *ordinary* when the (non-zero) weights on all arcs are equal to 1. These notations can be extended to the *pre-set* and *post-set* of a place p, with ${}^\bullet p = \{t \in T \mid \mathbf{post}(t, p) > 0\}$ and $p^\bullet = \{t \in T \mid \mathbf{pre}(t, p) > 0\}$.

Given a set of constants A, we define the set of *finite sequences* on A to be the free monoid $A^\star$, where ε stands for the "empty sequence". We will use $s \cdot s'$ for the concatenation operation between sequences, that we should often write $s\,s'$.

In the remainder of this report, we will use the two notations $A \to B$ and B^A interchangeably, for the set of functions from A to B.

Reachability Graph

A transition $t \in T$ is *enabled* at marking $m \in \mathbb{N}^P$ when $m(p) \geq \mathbf{pre}(t, p)$ for all places p in P. (We can also simply write $m \geq \mathbf{pre}(t)$, where $\geq$ stands for the component-wise

comparison of markings.) A marking $m' \in \mathbb{N}^P$ is reachable from a marking $m \in \mathbb{N}^P$ by firing transition t, denoted $m \xrightarrow{t} m'$, if: (1) transition t is enabled at m; and (2) $m' = m - \mathbf{pre}(t) + \mathbf{post}(t)$. By extension, we say that a *firing sequence* $\sigma = t_1 \dots t_n \in T^*$ can be fired from m, denoted $m \overset{\sigma}{\Rightarrow} m'$, if there exists markings $m_0, \dots, m_n$ such that $m = m_0$, $m' = m_n$ and $m_i \xrightarrow{t_{i+1}} m_{i+1}$ for all $i < n$.

We denote $R(N,m)$ the set of markings reachable from m in N. A marking m is k-bounded when each place has at most k tokens; property $\bigwedge_{p \in P} m(p) \leq k$ is true. Likewise, a marked Petri net (N,m_0) is bounded when there is k such that all reachable markings are k-bounded. A net is *safe* when it is 1-bounded. In our work, we consider *generalized* Petri nets (in which net arcs may have weights larger than 1) and we do not restrict ourselves to bounded nets.

The *reachability graph* is the rooted, directed graph, denoted $G(N,m_0)$, such that, the set of vertices is $R_N(m_0)$, the root is m_0, and we have an edge from m to m', labeled by t, if and only if $m \xrightarrow{t} m'$.

Observable Sequence

We can extend the notion of labels to sequences of transitions in a straightforward way. Given a relabeling function, l, we can extend it into a function from $T^\star \to \Sigma^\star$ such that $l(\varepsilon) = \varepsilon$, $l(\tau) = \varepsilon$ and $l(\sigma t) = l(\sigma)\, l(t)$. Given a sequence of labels σ in $\Sigma^\star$, we write $(N,m) \overset{\sigma}{\Rightarrow} (N,m')$ when there is a firing sequence ρ in $T^\star$ such that $(N,m) \overset{\rho}{\Rightarrow} (N,m')$ and $\sigma = l(\rho)$. We say in this case that σ is an *observable sequence* of the marked net (N,m).

We define a final operation on sequences that will be useful. Given an observable sequence σ, we denote $\sigma_{\lfloor \Sigma}$ the *projection* of the sequence on Σ, that is the sequence obtained by erasing all labels that are not in Σ, such that $\varepsilon_{\lfloor \Sigma} = \varepsilon$, $(\sigma a)_{\lfloor \Sigma} = \sigma$ if $a \notin \Sigma$ and $(\sigma a)_{\lfloor \Sigma} = (\sigma)_{\lfloor \Sigma} a$ otherwise.

Given two observable sequences σ_1 and σ_2, we say that σ_1 and σ_2 are *compatible* over the alphabet Σ, denoted $\sigma_1 \approx_\Sigma \sigma_2$ when they have equal projections on Σ, that is: $\sigma_{1\,\lfloor \Sigma} = \sigma_{2\,\lfloor \Sigma}$.

Graphical Syntax

We use the standard graphical notation for nets, where places are depicted as circles and transitions as squares. In the example of Fig. 1.3 (left), transition t_0 is fireable, because place p_0 has a token. The same net, right, depicts the evolution after firing transition t_0. We give another, more complex example of net in Fig. 1.4, that is taken from [Stahl, 2011].

(a) t_0 fireable (b) t_0 fired

Figure 1.3: Basic example of the behavior of a Petri net

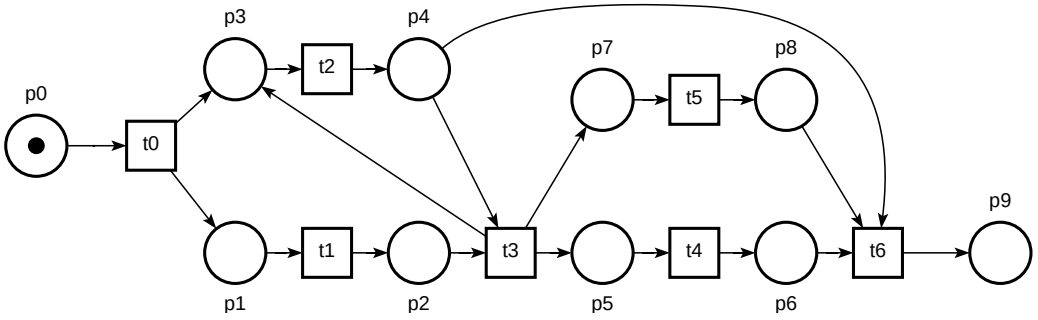


Figure 1.4: An example of Petri net

Extensions

Different extensions of standard Petri nets have been defined in order to increase the expressiveness of the model. Two extensions are treated in our study [Busi, 1998]:

- **Inhibitor arc**: an inhibitor arc with weight k between a place, p, and a transition, t, can be used to express an inhibition constraint between p and t. In this case, transition t is enabled in a marking m only when $m(p) < k$,

- **Read arc**: a read arc with weight k between a place, p, and a transition, t (also called a *test arc*), is equivalent to the combination of an incoming and an outgoing arc with weight k (such that $\mathbf{pre}(t,p) = k$ and $\mathbf{post}(t,p) = k$); that is, it is enabled when the marking of p is at least k and the marking of p does not change when t fires. Read arcs do not add any expressive power, but it can be useful to identify them when we reason about transitions as equations between integer vectors.

1.3 SAT/SMT Solving

Our approach is focused on SMT (*Satisfiability Modulo Theory*) solving to prove properties on the reachable markings of a net. One of the main references about the algorithms implemented in the state-of-the-art SAT and SMT solvers is [Kroening and Strichman, 2008].

SAT Problem

A SAT solver is a software that answers questions about the *satisfiability* of Boolean formulas.

A Boolean formula is composed of variables, $x, y, \ldots$ that can take one of two possible values: either *true* (denoted $\top$) or *false* (denoted $\bot$). Some basic operations can be performed between variables: conjunction (and) denoted as $\wedge$, disjunction (or) denoted as $\vee$, and negation (not) denoted as $\neg$ (or sometimes simply $\bar{x}$ when it is applied to a variable).

A formula ϕ can be evaluated to *true* or *false* when we fix the values of its variables. If a certain assignment of variables s evaluates the formula ϕ to *true*, we say that ϕ is *satisfiable* (*SAT*) and s is a *model* (denoted as $s \models \phi$). If there is no assignment such that ϕ is *SAT*, we say that ϕ is *UNSAT*. On the opposite, if all evaluations satisfy ϕ, we say that ϕ is *valid* (denoted as $\models \phi$).

SMT Extension

It is not possible to use Boolean formulas to reason about nets that are unbounded. In general, places can contain more than one token and, in the case of non-ordinary nets, we must encode conditions using inequalities between integers. That is why we will be using SMT solvers, like **z3** [Bjørner, 2020], in our approach.

We use the standard language for SMT solvers named *SMT-LIB 2*. A full documentation can be found in [Barrett et al., 2017]. The main advantage of using the SMT-LIB format, instead of using the API of the solver, is to be independent of a single solver. Hence it should be possible to adapt our results to work with other solvers supporting the SMT-LIB format.

Markings Seen as Systems of Equations

We can define many properties on the markings of a net N using Boolean combinations of linear constraints with integer variables (what is called the QF-LIA theory in SMT-LIB). Assume that we have a marked net (N, m_0) with set of places $P = \{p_1, \ldots, p_n\}$. We can associate a marking m over P to the formula $\underline{m}(x_1, \ldots, x_n)$, below. Formula $\underline{m}$ is obviously a conjunction of literals, what is called a *cube* in [Bradley, 2011].

$$\underline{m}(x_1, \ldots, x_n) \triangleq (x_1 = m(p_1)) \wedge \cdots \wedge (x_k = m(p_k)) \tag{1.1}$$

In the remainder, we use the notation $\phi(\vec{x})$ for the declaration of a formula ϕ with variables in $\vec{x}$, instead of the more cumbersome notation $\phi(x_1,\ldots,x_n)$. We also simply use $\phi(\vec{v})$, instead of $\phi\{x_1 \leftarrow v_1\}\ldots\{x_n \leftarrow v_n\}$, for the substitution of $\vec{x}$ with $\vec{v}$ in ϕ. We should often use place names as variables (or parameters) and use $\vec{p}$ for the vector $(p_1,\ldots,p_n)$. We also simply use $\underline{m}$ instead of $\underline{m}(\vec{p})$.

We say that a marking m *satisfies* a property ϕ, denoted $m \models \phi$, when formula $\phi \wedge \underline{m}(\vec{p})$ is satisfiable. In this case ϕ may use variables that are not necessarily in P. We can use this approach to reframe many properties on Petri nets. For instance the notion of safe markings, described previously: a marking m is safe when $m \models \mathrm{SAFE}_1(\vec{p})$, where $\mathrm{SAFE}_k(\vec{x}) \triangleq \bigwedge_{i\in 1..n}(x_i \leq k)$.

Likewise, the property that transition t is enabled corresponds to formula $\mathrm{ENBL}_t(\vec{x}) \triangleq \bigwedge_{i\in 1..n}(x_i \geq \mathbf{pre}(t_i,p))$, in the sense that t is enabled at m when $m \models \mathrm{ENBL}_t(\vec{p})$. Another example is the definition of *deadlocks*, which are characterized by formula $\mathrm{DEAD}(\vec{x}) \triangleq \bigwedge_{t\in T} \neg\mathrm{ENBL}_t(\vec{x})$. We give other examples in Chapter 2 and Chapter 5, when we encode the transition relation of a Petri net using formulas.

— 2 —
Motivations

A significant focus in model checking research is finding algorithmic solutions to avoid the "state explosion problem", that is finding ways to analyse models that are out of reach from current methods. The goal of this study is to explore a new technique that we call *polyhedral model checking*, in reference to the polyhedral, or polytope model, used in program optimization and static analysis [Besson et al., 1999, Feautrier, 1996].

The Vertics team is working on a new approach for the symbolic model checking of Petri nets that relies on the combined use of reductions and "state equations", see for example [Berthomieu et al., 2019]. By symbolic, we mean a method capable of computing the state-space of a system without enumerating all its states exhaustively. This method is based on representing some *reduction constraints*, expressed as a set of linear equations on the places of a net. When possible, instead of analysing a net N, we analyse a reduced net, N', and generate a system of equations E linking markings in the two nets.

2.1 Net Reductions

We consider nets reductions of the form (N_1, E, N_2), where N_1 is the initial net, N_2 the reduced net, and E a system of linear equation. This relation is formally defined in Chapter 4 of this report, where we define a new notion of equivalence denoted $N_1 \rhd_E N_2$, that we named a *polyhedral abstraction*.

The main idea is to decrease the size of the net N_1 by preserving some properties, such as the reachability graph. With our approach, we know how to reconstruct a partition of the states reachable in N_1 from the states reachable in N_2.

The concept of *reductions between nets* has been introduced in [Berthelot, 1987] and is known as *structural reduction*. It is still a topic of interest in recent works such as structural reductions implemented in ITSTools [Thierry-Mieg, 2020b]. The approach in [Berthomieu et al., 2018, Berthomieu et al., 2019], considered in this study, permits to reconstruct the state-space of N_1 as explained before. More powerful reductions can be applied when we are interested by less general properties, such as the detection of deadlocks for instance. Several tools use reductions for checking reachability properties.

TAPAAL [Bønneland et al., 2019], for instance, is an explicit-state model checker that combines reductions and partial-order reduction techniques and can apply them on Petri nets with weighted arcs and inhibitor arcs.

We consider two main kinds of reductions (and so equations) detailed in [Berthomieu et al., 2018, Berthomieu et al., 2019, Berthelot, 1987]: removal of redundant transitions and places and agglomeration of places.

In some cases, a Petri net can be fully reducible, i.e, the set of places P_2 and the set of transitions T_2 of the reduced net N_2 are empty ($P_2 = T_2 = \emptyset$). An empty-net has only one marking and no transitions.

We can schematize this approach by the Figure 2.1, in which the top state-space (corresponding to N_1) is obtained as the tesselation of a finite set of polyhedra (the pre-image of states from N_2 by the system of equations E).

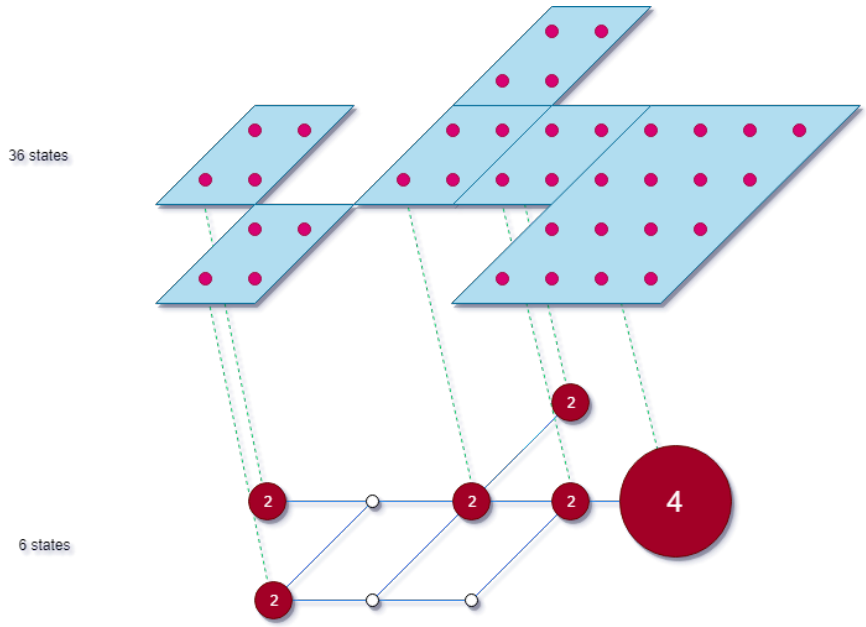


Figure 2.1: Polyhedral reduction representation

The implementation of this technique has shown its effectiveness in practice. In particular, it has been used to compute the reachable states of many use cases that where considered as beyond reach so far.

2.2 Properties Verification

We use our concept of *reduction equations* to automatically check properties on the initial system, N_1, from an analysis of the reduced net, N_2. In this context, a possible approach is to transform the proof into the verification of an invariant on the set of reachable states of N_2. We use a SMT solver to check, at the same time, the conditions on the invariant and on the reduction equations E.

In this work, we focus on a subset of the properties defined for the *Model Checking Contest*, see for instance [Amparore et al., 2019, Hillah and Kordon, 2017, LIP6, 2020], that are *safety* properties on the reachable markings of a marked net (N, m_0). Examples of the property that we want to check include: checking if some transition t is enabled (commonly known as *quasi-liveness*); checking if there is a deadlock; checking whether some invariant between place markings is true; ... We consider the proof of both formulas, $\mathrm{AG}\,F$ (true only when every marking $m \in R(N, m_0)$ is a model of F), and $\mathrm{EF}\,F$ (true if there is some m such that $m \models F$).

Place reachability corresponds to the satisfiability of property $\mathrm{REACH}(p)$:

$$\mathrm{COVER}(p,k) \equiv m(p) \geq k \tag{2.1}$$
$$\mathrm{REACH}(p) \equiv m(p) \geq 1 \tag{2.2}$$

We can define a notion of *quasi-liveness*, or "event reachability", that is similar to state reachability but for transitions. We say that a transition t is live if there is a reachable marking m such that $m \xrightarrow{t}$. We can extend this property to a set of transitions $\{t_1, \ldots, t_n\}$. Again, checking the liveness of a transition, say t, can be reduce to checking the satisfiability of a conjunction over the set of reachable markings, that is:

$$\mathrm{LIVE}^1(t) \equiv \bigwedge_{p \in {}^\bullet t} \mathrm{COVER}(p, \mathbf{pre}(t,p)) \tag{2.3}$$

We are also interested in the verification of *deadlocks*. A deadlock is a state of the system in which the system cannot fire any transition, and so is blocked. The deadlock problem is a common problem in concurrent computing, operating system, and communication system.

$$\mathrm{DEAD} \equiv \bigwedge_{t \in T} \neg \mathrm{LIVE}(t) \tag{2.4}$$

We describe in Chapter 5 two main methods, called BMC and PDR, for checking the properties that we just defined. During my work, I have also studied a problem related to the decomposition of Petri nets into a hierarchical structure of safe sub-nets, called NUPNs [Garavel, 2019a]. Computing a NUPN decomposition requires to compute a "concurrency relation" between places in the net, where two places p_1 and p_2 are said concurrent if property $\mathrm{REACH}(p_1) \wedge \mathrm{REACH}(p_2)$ is satisfiable. One of the contributions of my work is to propose a new method for finding pairs of concurrent places, see Chapter 6.

2.3 Contributions and Structure of the Report

I have made a number of contributions during my internship. All the methods defined or improved during my work have been implemented on a prototype model-checker called

[1] Property LIVE is often referred to as quasi-liveness in the literature.

SMPT that is based on a SMT approach. This tool includes two popular approaches for model-checking transition systems that I have updated in order to handle generalized nets (markings are not necessarily 1-safe and edges can have weights larger than 1) and also reductions (see Chapter 5). On a more theoretical-side, we defined a "principled" method for using reductions rules used and defined a method to prove the correctness of the approach based on a new notion of *E-abstraction equivalence* (see Chapter 4). Another contribution of this project was to apply our approach on the *Concurrent Places Problem* (see Chapter 6).

The report is organized as follows:

Chapter 3 gives a brief State-of-the-Art on symbolic model-checking methods by presenting two main approaches.

Chapter 4 contains the theoretical part of my work and includes the formalization of net reductions and the notion of net equations. It also introduces a new equivalence relation that is used in our proofs.

Chapter 5 is a technical description of the two main methods implemented in our model-checker: BMC and PDR. algorithms, as well as their extension to reductions. We also describe a simpler approach, called "enumerative", that can be surprisingly efficient when the net can be almost totally reduced.

Chapter 6 describe how we can apply our work to solve the *Concurrency Places Problem*. It introduces a new method, which takes advantage of net reductions, in order to compute the concurrency matrix of very "large nets".

Chapter 7 contains experimental results obtained with SMPT and that we used to validate our approach.

Chapter 8 is the conclusion of the report. It also presents some perspectives for future works.

— 3 —
State-of-the-art

Several verification techniques have been developed in order to overcome, at least partially, the *state explosion problem*. Some of these techniques can be grouped into the category called *symbolic model checking*. Instead of enumerating the whole state-space exhaustively, state by state, we encode and manipulate "groups of states" using a symbolic representation. Many different kinds of symbolic representations can be used: logical formulas, binary decision diagrams (BDD) or other related data structures, etc.

3.1 Binary Decision Diagrams

A *Binary Decision Diagram* or *BDD* is a data structure that permits to represent a Boolean function, and therefore sets of Boolean vectors, in a compact way. Assuming a state s is a vector of Boolean value of size n ($s \in \mathbb{B}^n$), the idea is to represent a set of states by a Boolean formula $f : \mathbb{B}^n \to \mathbb{B}$, such that $f(s)$ is true if and only if s is in the set. In the following, we use 0 for false and 1 for the truth value.

A BDD with variables $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph with two terminal nodes, labeled with 0 and 1, where all non-terminal node is labeled with a variable in X. Each such node has exactly two outgoing nodes, labeled with 0 and 1 respectively. These edges correspond to an assignment for the variable labeling the node. We display a graphical version of a BDD in Fig. 3.1, where dashed lines denote 0-edges and solid lines denote 1-edges. Many approaches work with a constrained version of BDD, called *Reduced Ordered BDD* (ROBDD), where the order of the variables along a path in the graph is fixed and such that isomorphic nodes (same variables and same outgoing nodes) are merged. Given an order on the variables of X, the ROBDD associated with a set is unique. A nice property of ROBDD is that it is possible to check equivalence between Boolean sets/functions in linear time (in the size of the input ROBDDs) and satisfiability in constant time.

Figure 3.1 is a BDD encoding of the Boolean formula $a \wedge (\neg b \vee c)$. Figure 3.2 is the ROBDD for the same formula and the variable ordering ($a < b < c$).

The first model-checker to use BDDs was **SMV** [Burch et al., 1992, Bérard et al., 2001].

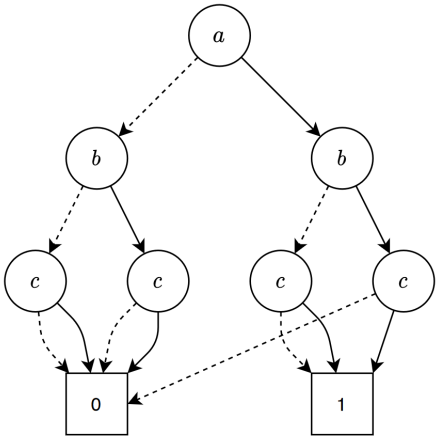


Figure 3.1: BDD

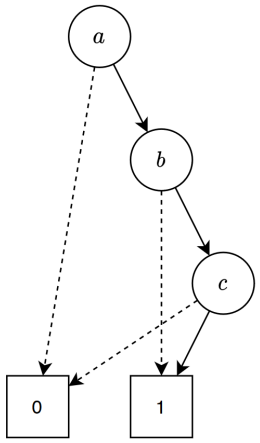


Figure 3.2: ROBDD

This is, for instance, the model-checker used for the verification of the **IEEE Future+ cache coherence protocol** [Garavel, 2012], which is one of the breakthroughs in the area of formal verification. Many state-of-the-art model-checkers today use BDD-like structures (called decision diagrams) to encode set of states and state transitions. For example, there are extensions of decision diagrams that can encode sets of values taken from a finite set (for example a bounded interval of integers).

3.2 SAT and SMT

Another approach for symbolic model checking is to encode set of states (and also transitions relations) into *logical formulas*. These logic can include more complex theories than the Booleans, such as *linear arithmetic* for example. Given a logical specification of a property, verification can be handled using SAT or SMT solvers.

An efficient method to find counter-examples is *Bounded Model Checking* or *BMC* [Biere et al., 1999]. The idea is to unroll transitions until we reach a state that does not satisfy the property. This method is good at answering questions of the form: "is there a marking falsifying the property reachable in k transitions from the initial marking?". The problem with this approach is that is is not always possible to know a bound for the value of k. Therefore, when the property is always true (we say it is an invariant), the BMC methods do not terminate.

One of the extensions of *BMC* is *k-induction* [de Moura et al., 2003]. It is a generalization of induction to k transitions. In addition to finding counter-examples, this method is able to prove invariants. Another more efficient technique, that is also complete, is based on the use of interpolation. See for example the *Interpolation-based Model Checking* (*IMC*) of [McMillan, 2003].

One of the most recent SAT/SMT based method is called *Property Directed Reachability* (PDR). This approach is also known as *IC3*, which stands for *Incremental Construction*

of Inductive Clauses for Indubitable Correctness [Bradley, 2011]. This method is based on a combination of induction and over-approximations. The interest of this method is that the analysis is split into many "simple" problems.

We provide a partial implementation of PDR that is correct and complete when the property is monotonic (see Chapter 5), even in the case of nets that are not bounded. This temporary solution[1] can be understood as a restriction to the case of "coverability properties". This seems to be the current state of the art with Petri nets; see for example [Esparza et al., 2014] or the extension of PDR to "well-structured transition systems" [Kloos et al., 2013]. We can also mention the works on inductive procedure for infinite state and/or parametrized systems, such as the verification methods used in Cubicle [Conchon et al., 2012], or in [Cimatti et al., 2016, Gurfinkel et al., 2016].

In this report, I describe my results when extending both the BMC and PDR methods to the verification of generalized Petri nets (therefore not only on 1-safe nets) and taking into account reductions.

3.3 Model Checking Contest (MCC)

The *Model Checking Contest* [Amparore et al., 2019] is an annual competition where tool developers can compare the efficiency of their model-checkers, and therefore (partially) the effectiveness of the algorithms and implementation techniques that they use. The competition is composed of several categories, such as *StateSpace* or *Reachability*.

The new approach developed by the Vertics team, which combines the use of reductions and "state equations", is one of the reasons behind the victory of TINA [LAAS-CNRS, 2020] in the StateSpace during the last edition of the contest [Amparore et al., 2019].

The tool used in this competition, called tedd, uses a combination of Set Decision Diagrams and reduction equations [Berthomieu et al., 2019]. We should not use decision diagram in this report, since my goal was to try an approach mixing SMT solvers (see below) and reductions. Nonetheless, for future works, I plan to study a combination of SMT solvers with decision diagrams.

3.4 Use of Systems of Linear Equations and the Polyhedral Approach

My work relies on several results about systems of linear equations with solutions over the positive integers. The set of solutions of these system are convex sets of $\mathbb{N}^n$, also called polyhedra. I also work with the union (disjunction) of such systems. These systems are commonly used in the domains of *linear programming* and *convex optimization*. They also occur frequently when studying Petri nets, which are equivalent in

[1] We have several ideas for improving our current implementation of PDR.

many ways to "Vector Addition Systems with States" (VASS), for instance when studying invariants. For instance, a large part of my approach relies on SMT solvers that support procedures for satisfiability modulo Linear Integer Arithmetic.

While I use results and tools that deal with linear system, I did not work on this domain during my internship and I did not try to adapt the existing tools and techniques to my particular use case. This could be part of my future work.

There are many domains of theoretical computer science where convex sets are important, including for formal verification. For instance, they are used as abstract domains in Abstract Interpretation [Cousot and Halbwachs, 1978]. See for example the work on the APRON numerical abstract domain library. They also play a central role in the automatic parallelization of programs [Feautrier and Lengauer, 2011]. Closer to my work, Difference Bound Matrices (DBM), are a special class of convex structures that have been used for the model checking of a time extension of Petri nets. Besides Time Petri nets, DBM are heavily used for the analysis of Timed Automata. I am certain that there are many interesting ideas, coming from these different domains, that could be used in my project.

— 4 —

Reachability Equivalence and Net-Abstractions

We define a new notion, called *E-abstraction equivalence*, that is used to state a correspondence between the set of reachable markings of two Petri nets "modulo" some linear system of equations, E. Basically, we have that (N_1, m_1) is E-equivalent to (N_2, m_2) when, for every sequence $m_2 \stackrel{\sigma_2}{\Longrightarrow} m_2'$ in N_2, there must exist a sequence $m_1 \stackrel{\sigma_1}{\Longrightarrow} m_1'$ in N_1 such that $E \wedge \underline{m_1'} \wedge \underline{m_2'}$ is satisfiable (and reciprocally). Therefore, knowing E, we can compute the reachable markings of N_1 from those of N_2, and vice versa. We also ask for the observable sequences, σ_1 and σ_2 in this case, to be equal. As a result, we will prove that our equivalence is also a congruence.

We can illustrate these notions using the two nets M_1, M_2 in Fig. 1.4, we have that $m_1' \triangleq p_0{*}3\ p_5{*}2\ p_6{*}3$ is reachable in M_1 and $E_M \wedge \underline{m_1'}$ entails $(a_2 = 3) \wedge (a_3 = 3)$, which means that marking $m_2' \triangleq a_2{*}3\ a_3{*}3\ p_5{*}2\ p_6{*}3$ is reachable in M_2. Conversely, we have several markings in M_1 that corresponds to the constraint $E_M \wedge \underline{m_2'} \equiv (p_2 = p_1 + p_4) \wedge (3 = p_o + p_1 + p_3) \wedge (3 = p_0 + p_1 + p_4) \wedge (p_5 = 2) \wedge (p_6 = 3)$. All these markings are reachable in M_1 using the same observable sequence **bcc**. More generally, each marking m_2' of N_2 can be associated to a convex set of markings of N_1, defined as the set of positive integer solutions of $E \wedge \underline{m_2'}$. Moreover, these sets form a partition of $R(N_1, m_1)$. This motivates our choice of calling this relation a *polyhedral abstraction*.

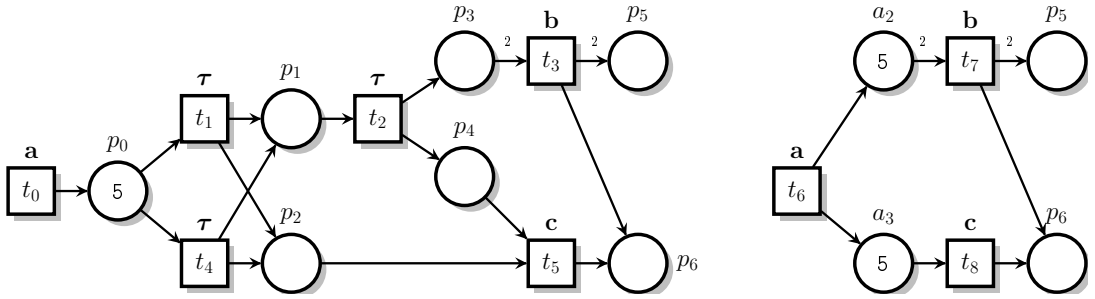


Figure 4.1: An example of Petri net, M_1 (left), and one of its polyhedral abstraction, M_2 (right), with $E_M \triangleq (p_2 = p_1 + p_4) \wedge (a_2 = p_o + p_1 + p_3) \wedge (a_3 = p_0 + p_1 + p_4)$.

While our approach does not dictate a particular method for finding pairs of equivalent nets, we propose an automatic approach based on the use of *net reductions*. When the net N_1 can be reduced, we will obtain a resulting net (N_2) and a condition (E) such that N_2 is a polyhedral abstraction of N_1. In this case, E will always be expressed as a conjunction of equality constraints between linear combination of integer variables (the marking of places). This is why we should often use the term *reduction equations* when referring to E. Our goal is to transform any reachability problem on the net N_1 into a reachability problem on the (reduced) net N_2, which should hopefully be simpler to check.

4.1 Satisfiability Conditions on the Systems of Equation

Before defining our equivalence more formally, we need to introduce some constraints on the condition, E, used to correlate the markings of two different nets. We say that a pair of markings (m_1, m_2) are *compatible* (over respective sets of places P_1 and P_2) when they have equal marking on their shared places, meaning $m_1(p) = m_2(p)$ for all p in $P_1 \cap P_2$. This is a necessary and sufficient condition for formula $\underline{m_1} \wedge \underline{m_2}$ to be satisfiable. When this is the case, we denote $m_1 \uplus m_2$ the unique marking in $(P_1 \cup P_2)$ such that $(m_1 \uplus m_2)(p) = m_1(p)$ if $p \in P_1$ and $(m_1 \uplus m_2)(p) = m_2(p)$ otherwise. Hence, with our conventions, $\underline{m_1 \uplus m_2} \Leftrightarrow \underline{m_1} \wedge \underline{m_2}$.

In the following we ask that condition E be *solvable for* P_1, P_2, meaning that for all markings m_1 over P_1 there must exists at least one marking m_2 over P_2, compatible with m_1, such that $m_1 \uplus m_2 \models E$ (and reciprocally). While this property is not essential for most of our results, it simplifies our presentation and it will always be true for the reduction equations generated with our method. On the other hand, we do not prohibit to use variables in E that are not in $P_1 \cup P_2$. Actually, such situation will often occur in practice, when we start to chain several reductions.

4.2 E-Abstraction Equivalence

We start by defining a notion of E-abstraction. An E-abstraction equivalence is an abstraction in both directions.

Definition 4.1 (E-abstraction equivalence)**.** *Assume N_1, N_2 are two Petri nets with respective sets of places P_1, P_2 and labeled functions l_1, l_2, over the same alphabet Σ. We say that the marked net (N_2, m_2) is a E-abstraction of (N_1, m_1), denoted $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$, if and only if:*

(A1) *system E is solvable for P_1, P_2 and the initial markings are compatible with E, meaning $m_1 \uplus m_2 \models E$.*

(A2) *for all observation sequences* $\sigma \in \Sigma^\star$ *such that* $(N_1,m_1) \stackrel{\sigma}{\Rightarrow} (N_1,m_1')$ *then for all marking* m_2' *over* P_2 *such that* $m_1' \uplus m_2' \models E$ *we have* $(N_2,m_2) \stackrel{\sigma}{\Rightarrow} (N_2,m_2')$.

We say that (N_1,m_1) *is* E*-equivalent to* (N_2,m_2)*, denoted* $(N_1,m_1) \rhd_E (N_2,m_2)$*, when we have both* $(N_1,m_1) \sqsupseteq_E (N_2,m_2)$ *and* $(N_2,m_2) \sqsupseteq_E (N_1,m_1)$.

Notice that condition (A2) is defined only for sequences starting from the initial marking of N_1. Hence the relation is usually not true on every pair of matching markings; it is not a simulation. By definition, relation $\rhd_E$ is symmetric. We deliberately use a "comparison symbol" for our equivalence, $\rhd$, in order to stress the fact that N_2 should be a reduced version of N_1. In particular, we expect that $|P_2| \leq |P_1|$.

4.3 Basic Properties of Polyhedral Abstraction

We prove that we can use E-equivalence to check the reachable markings of N_1 simply by looking at the reachable markings of N_2. We give a first property that is useful in the context of bounded model checking, when we try to find a counter-example to a property by looking at firing sequences with increasing length. Our second and third properties are useful for checking invariants, and is at the basis of our implementation of the PDR method for Petri nets.

Lemma 4.1 (Bounded Model Checking)**.** *Assume* $(N_1,m_1) \rhd_E (N_2,m_2)$*. Then for all* m_1' *in* $R(N_1,m_1)$ *there is* m_2' *in* $R(N_2,m_2)$ *such that* $m_1' \uplus m_2' \models E$.

Proof. Since m_1' is reachable, there must be an observable sequence σ in N_1 such that $(N_1,m_1) \stackrel{\sigma}{\Rightarrow} (N_1,m_1')$. By condition (A1), system E is solvable. Hence there must be some marking m_2' over P_2, compatible with m_1', such that $m_1' \uplus m_2' \models E$. By condition (A2), we have that $(N_2,m_2) \stackrel{\sigma}{\Rightarrow} (N_2,m_2')$. Therefore we have m_2' reachable in N_2 such that $m_1' \uplus m_2' \models E$. □

Lemma 4.1 can be used to find a counter-example m_1', to some property F in N_1, just by looking at the reachable markings of N_2. Indeed, it is enough to find a marking m_2' reachable in N_2 such that $m_2' \models E \wedge \neg F$. This is the result we use in our implementation of the BMC method (see Lemma 5.2). Our second property can be used to prove that every reachable markings of N_2 can be traced back to at least one marking of N_1 using the reduction equations. (While this mapping is surjective, it is not a function, since a state in N_1 could be associated with multiple states in N_2.)

Lemma 4.2 (Invariance Checking)**.** *Assume* $(N_1,m_1) \rhd_E (N_2,m_2)$*. Then for all pairs of markings* m_1', m_2' *over* N_1, N_2 *such that* $m_1' \uplus m_2' \models E$ *and* $m_2' \in R(N_2,m_2)$ *it is the case that* $m_1' \in R(N_1,m_1)$.

Proof. Take m'_1, m'_2 a pair of markings in N_1, N_2 such that $m'_1 \uplus m'_2 \models E$ and $m'_2 \in R(N_2, m_2)$. Hence there is an observable sequence σ such that $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m'_2)$. By condition (A2), since $m'_1 \uplus m'_2 \models E$, we have that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m'_1)$. Hence $m'_1 \in R(N_1, m_1)$. □

The last result (see Th. 4.1) ensures that we can easily extract an invariant on N_1 (a property valid for every marking in $R(N_1, m_1)$) from an invariant on N_2. This is the property that ensure the soundness of our model checking technique (see Chapter 5).

Theorem 4.1 (Invariant Conservation)**.** *Assume* $(N_1, m_1) \rhd_E (N_2, m_2)$. *If* $\tilde{E}(\vec{p_1}, \vec{p_2}) \wedge F(\vec{p_1})$ *is an invariant on* N_2 *then* $F(\vec{p_1})$ *is an invariant on* N_1.

Proof. Assume property $\tilde{E}(\vec{p_1}, \vec{p_2}) \wedge F(\vec{p_1})$ is an invariant on N_2 (where F is a formula with variables in P_1). Take a marking m'_1 in N_1. By definition of E-equivalence, we have at least one marking m'_2 reachable in N_2 such that $m'_1 \uplus m'_2 \models E$. We also have $m'_2(\vec{p_2}) \models \tilde{E}(\vec{p_1}, \vec{p_2}) \wedge F(\vec{p_1})$ (from the invariant) and therefore $\underline{m'_1} \wedge \underline{m'_2} \wedge E \wedge F$ is satisfiable. This implies that $m'_1(\vec{p_1}) \models F(\vec{p_1})$, which means that F is an invariant on N_1 when $E \wedge F$ is an invariant on N_2. □

4.4 Reductions Rules

We introduce different kinds of net reductions in Figures 4.1 to 4.8. For each rule, we define the matching net (N_1, m_1), the corresponding reduced net (N_2, m_2), and the resulting equation. In each case, the rule define a triplet (N_1, E, N_2) such that $(N_1, m_1) \rhd_E (N_2, m_2)$. We prove the soundness of some of these rules in Appendix A.1. In each case, we draw the nets with their initial marking, which are expressed using integer parameters. We also add a condition that should be true initially.

4.5 Composition Laws

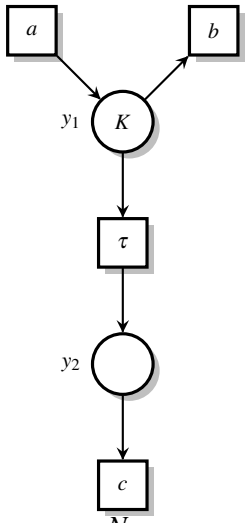
Next we prove that polyhedral abstractions are closed by synchronous composition, relabeling, and chaining. Before defining these operations, we start by describing sufficient conditions in order to safely compose equivalence relations. From theses results we want to state that the E-equivalence is a congruence (see Th. 4.2).

Theorem 4.2 (E-equivalence is a congruence)**.** *Assume we have two compatible equivalence statements* $(N_1, m_1) \rhd_E (N_2, m_2)$ *and* $(N_2, m_2) \rhd_{E'} (N_3, m_3)$, *and that* M *is compatible with respect to these equivalences, then:*

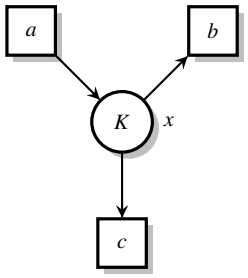
- $(N_1, m_1) \| (M, m) \rhd_E (N_2, m_2) \| (M, m)$ *(see Th. 4.3).*
- $(N_1, m_1) \rhd_{E,E'} (N_3, m_3)$ *(see Th. 4.4).*
- $(N_1[a/b], m_1) \rhd_E (N_2[a/b], m_2)$ *and* $(N_1[a/\tau], m_1) \rhd_E (N_2[a/\tau], m_2)$ *(see Th. 4.5).*

Concatenate (CONCAT)

This rule is similar to one of the original reductions proposed in [Berthelot, 1987].



N_1



N_2

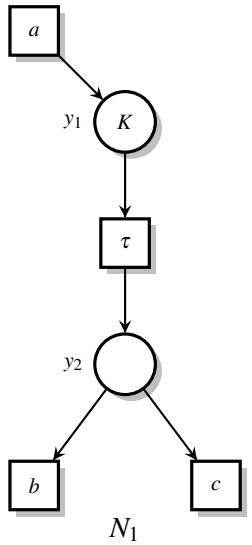
Condition: $\emptyset$

Equation: $x = y_1 + y_2$

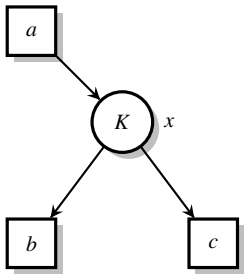
Figure 4.2: Rule Concatenate (CONCAT)

Concatenate (CONCAT')

We can define a scheme of rules similar to (CONCAT) by adding or removing labeled transitions with inputs from y_1 and y_2 and outputs to y_1 only. We give an example below. The only constraints are that place y_2 should be initially empty and that no extra transition can add a token to y_2.



N_1



N_2

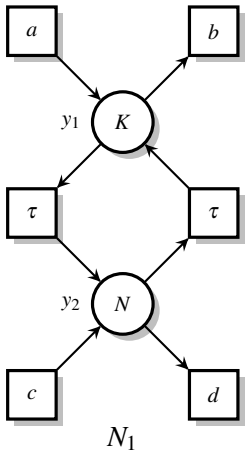
Condition: $\emptyset$

Equation: $x = y_1 + y_2$

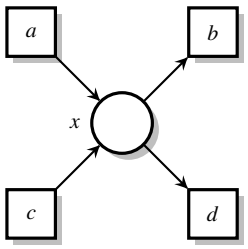
Figure 4.3: Rule Concatenate (CONCAT')

Agglomeration of Places (AGG)

Agglomeration is used to simplify a “cluster of places” between which tokens can move freely.



N_1



N_2

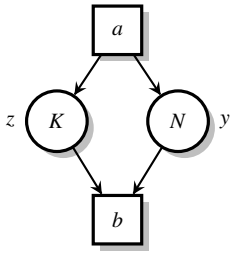
Condition: $\emptyset$

Equation: $x = y_1 + y_2$

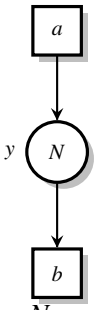
Figure 4.4: Rule Agglomeration of Places (AGG)

Redundant Places (RED1)

We provide several rules for the elimination of redundant places.



N_1



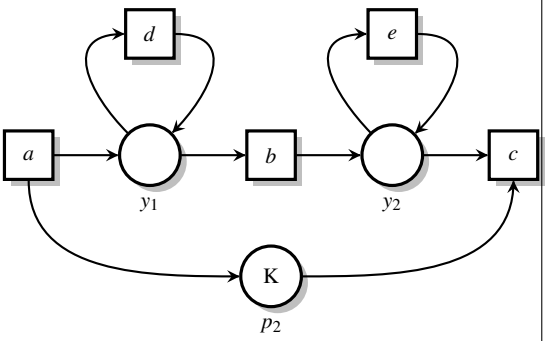
N_2

Condition: $K > N$

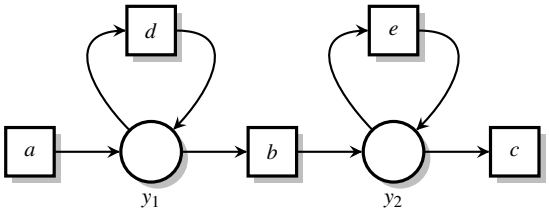
Equation: $z = y + K - N$

Figure 4.5: Rule Redundant Places (RED1)

Redundant Places (RED2)



N_1



N_2

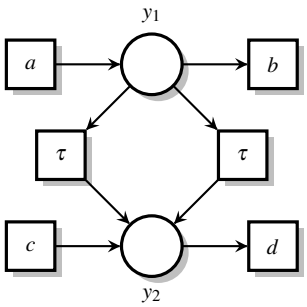
Condition: $\emptyset$

Equation: $z = y_1 + y_2 + K$

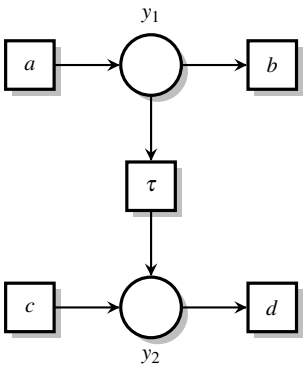
Figure 4.6: Rule Redundant Places (RED2)

Redundant Transitions (REDT)

This is the first example of a rule that does not decrease the number of places but that can be used to simplify transitions. Such rules are interesting because, when applied in collaboration with others, they can create new opportunities to apply reductions. We give an example of such mechanism in the example of Sect. 4.7.



N_1



N_2

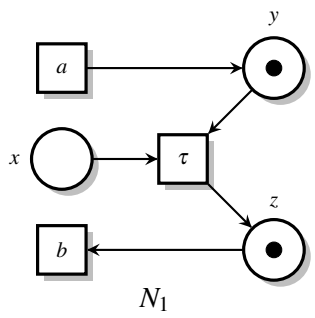
Condition: $\emptyset$

Equation: $\emptyset$

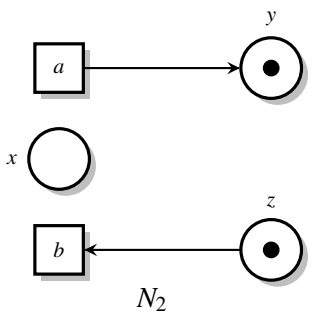
Figure 4.7: Rule Redundant Transitions (REDT)

Dead Transition Removal (DEADT)

Since place x is initially empty, and no transition can increase its marking, the τ transition can never be fired and can therefore be removed.



N_1



N_2

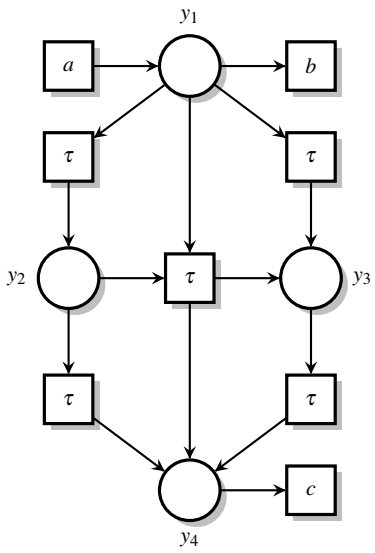
Condition: $\emptyset$

Equation: $\emptyset$

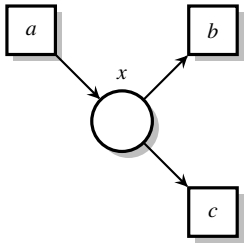
Figure 4.8: Rule Dead Transition Removal (DEADT)

Magic Concatenate (MAGIC)

This last rule is an example of reduction that cannot be obtained using the system defined in [Berthomieu et al., 2019]. This is one of our motivation for developing a "reduction system" that can be easily extended.



N_1



N_2

Condition: $\emptyset$

Equation: $x = y_1 + y_2 + y_3 + y_4$

Figure 4.9: Rule Magic Concatenate (MAGIC)

These composition laws are useful to build larger equivalences from simpler axioms. We show two examples of reductions in the next section and how they occur in the example of Fig. 1.4.

4.5.1 Compatibility of Equations

The goal here is to avoid inconsistencies that could emerge if we inadvertently reuse the same variable in different reduction equations.

The *fresh variables* in an equivalence statement EQ : $(N_1,m_1) \rhd_E (N_2,m_2)$ are the variables occurring in E but not in $P_1 \cup P_2$. (These variables can be safely "alpha-converted" in E without changing any of our results.) We say that a net N_3 is *compatible* with respect to EQ when $(P_1 \cup P_2) \cap P_3 = \emptyset$ and there are no fresh variables of EQ that are also places in P_3. Likewise we say that the equivalence statement EQ$'$: $(N_2,m_2) \rhd_{E'} (N_3,m_3)$ is *compatible* with EQ when $P_1 \cap P_3 \subseteq P_2$ and the fresh variables of EQ and EQ$'$ are disjoint.

4.5.2 Composability (COMP)

In this section we rely on the classical synchronous product operation between labeled Petri nets. Let $N_1 = (P_1,T_1,\mathbf{pre}_1,\mathbf{post}_1)$ and $N_2 = (P_2,T_2,\mathbf{pre}_2,\mathbf{post}_2)$ be two labeled Petri nets, with labeling functions l_1 and l_2 on the respective alphabets Σ_1 and Σ_2. We can assume, without loss of generality, that the sets P_1 and P_2 are disjoint. We introduce a new symbol, $\circ$, used to build (structured) names for transitions that are not synchronized. The *synchronous product* between N_1 and N_2, denoted as $N_1 \| N_2$, is the net $(P_1 \cup P_2, T, \mathbf{pre}, \mathbf{post})$ where T is the smallest set containing: (1) transition $(t,\circ)$ if $l_1(t) \notin \Sigma_1$; transition $(\circ,t)$ if $l_2(t) \notin \Sigma_2$; (3) and transition (t_1,t_2) if $l_1(t_1) = l_2(t_2)$. The flow functions of $N_1 \| N_2$ are such that $\mathbf{pre}((t_1,t_2),p) = \mathbf{pre}_1(t_1,p)$ if $p \in P_1$ and $t_1 \neq \circ$, or $\mathbf{pre}_2(t_2,p)$ if $p \in P_2$ and $t_2 \neq \circ$ (and 0 in all the other cases). Similarly for **post**. Since the places in N_1 and N_2 are disjoint, we can always see a marking m in $N_1 \| N_2$ as the disjoint union of two markings m_1, m_2 from N_1, N_2. In this case we simply write $m = m_1 \| m_2$. More generally, we extend this product operation to marked nets and write $(N_1,m_1) \| (N_2,m_2)$ for the marked net $(N_1 \| N_2, m_1 \| m_2)$.

Lemma 4.3 (Projection and product of sequences)**.** *If there is a firing sequence σ in the synchronous product $N_1 \| N_2$, say $(N_1 \| N_2, m_1 \| m_2) \stackrel{\sigma}{\Rightarrow} (N_1 \| N_2, m_1' \| m_2')$, then its projections are also firing sequences: $(N_i,m_i) \stackrel{\sigma \cdot i}{\Longrightarrow} (N_i,m_i')$ for all $i \in 1..2$ and $l_1(\sigma \cdot 1) \approx_\Sigma l_2(\sigma \cdot 2)$. Conversely, if $(N_i,m_i) \stackrel{\sigma_i}{\Rightarrow} (N_i,m_i')$ for all $i \in 1..2$ and $\sigma \in (\sigma_1 \parallel_\Sigma \sigma_2)$ then $(N_1 \| N_2, m_1 \| m_2) \stackrel{\sigma}{\Rightarrow} (N_1 \| N_2, m_1' \| m_2')$.*

We can now prove that E-abstraction equivalence is stable by synchronous composition.

Theorem 4.3. *Assume we have $(N_1,m_1) \rhd_E (N_2,m_2)$ and that M is compatible with respect to this equivalence, then $(N_1,m_1) \| (M,m) \rhd_E (N_2,m_2) \| (M,m)$.*

Proof. It is enough to prove the result on E-abstraction, since it will directly entail the result for equivalence.

By hypothesis system E is solvable for N_1, N_2. Hence, since M is compatible, no place in P_M can occur in one of the equation of E. Therefore E is also solvable for the pair of nets $(N_1 \| M)$ and $(N_2 \| M)$. Likewise, the initial markings $(m_1 \| m)$ and $(m_2 \| m)$ are compatible together and $(m_1 \| m) \uplus (m_2 \| m) \models E$ (the constraints in m have no effect on the equations of E). Therefore condition (A1) is valid for the marked nets $(N_1, m_1) \| (M, m)$ and $(N_2, m_2) \| (M, m)$.

We are left with proving condition (A2). Note that, since N_1 and N_2 are equivalent, they must have the same alphabet: $\Sigma_1 = \Sigma_2$ and therefore $\Sigma_1 \cap \Sigma_M = \Sigma_2 \cap \Sigma_M = \Sigma$.

Assume we have a firing sequence σ in $N_1 \| M$. By our projection property (Lemma 4.3) it must be the case that $(N_1 \| M, m_1 \| m) \stackrel{\sigma}{\Rightarrow} (N_1 \| M, m_1' \| m')$ with $(N_1, m_1) \stackrel{\sigma \cdot 1}{\Longrightarrow} (N_1, m_1')$. We also have that $(M, m) \stackrel{\sigma \cdot 2}{\Longrightarrow} (M, m')$ with $l_1(\sigma \cdot 1) \approx_\Sigma l_M(\sigma \cdot 2)$.

By condition (A2) on the abstraction between N_1 and N_2, it must be the case that $(N_2, m_2) \stackrel{\sigma_2}{\Longrightarrow} (N_2, m_2')$, for some firing sequence σ_2 of N_2, for all markings over N_2 such that $m_1' \uplus m_2' \models E$. Moreover we have $l_1(\sigma \cdot 1) = l_2(\sigma_2)(\star)$ and therefore $l_2(\sigma_2) \approx_\Sigma l_M(\sigma \cdot 2)$. Hence, using the second direction in Lemma 4.3, $\sigma_2 \|_\Sigma (\sigma \cdot 2)$ is well-defined and we can choose σ' in it such that $(N_2 \| N_3, m_2 \| m_3) \stackrel{\sigma'}{\Longrightarrow} (N_2 \| N_3, m_2' \| m_3')$. Like in the proof of condition (A1), we obtain that $(m_1' \| m_3') \uplus (m_2' \| m_3') \models E$ from the fact that $m_1' \uplus m_2' \models E$, and E is solvable, and N_3 is compatible.

We are left to prove that σ and σ' have the same observable sequences. This is a consequence of the fact that $l_1(\sigma \cdot 1) = l_2(\sigma_2)$ (property $\star$ above); and the fact that, by construction of σ', we have $\sigma' \cdot 1 = \sigma_2$ and $\sigma' \cdot 2 = \sigma \cdot 2$. □

4.5.3 Transitivity (TRANS)

In this section we prove that we can chain equivalences together in order to derive more general reduction rules.

Theorem 4.4. *Assume we have two compatible equivalence statements* $(N_1, m_1) \rhd_E (N_2, m_2)$ *and* $(N_2, m_2) \rhd_{E'} (N_3, m_3)$*, then* $(N_1, m_1) \rhd_{E, E'} (N_3, m_3)$.

Proof. It is enough to prove the result on E-abstraction, since it will directly entail the result for equivalence.

For condition (A1), we use the fact system E, E' is solvable for N_1, N_3. This is a consequence of the compatibility assumption, since no fresh variable in E can clash with a fresh variable in E'. For similar reason, we have that $m_1 \uplus m_2 \models E$ and $m_2 \uplus m_3 \models E'$ entails $m_1 \uplus m_3 \models E, E'$. Indeed we even have the stronger property that $\underline{m_1} \wedge \underline{m_2} \wedge \underline{m_3} \wedge E \wedge E'$ is satisfiable.

For condition (A2), we assume that $(N_1, m_1) \stackrel{\sigma}{\Rightarrow} (N_1, m_1')$. Hence, using the fact that $(N_1, m_1) \rhd_E (N_2, m_2)$, we have $(N_2, m_2) \stackrel{\sigma}{\Rightarrow} (N_2, m_2')$ for every marking m_2' over N_2 such that $m_1' \uplus m_2' \models E$. Using a similar property from $(N_2, m_2) \rhd_{E'} (N_3, m_3)$, we have $(N_3, m_3) \stackrel{\sigma}{\Rightarrow} (N_3, m_3')$ for every marking m_3' over N_3 such that $m_2' \uplus m_3' \models E$. The result follows from the observation that, since E and E' are both solvable and the nets are compatible, for all marking m_1'' over N_1, if a marking m_3'' over N_3 satisfies $m_1' \uplus m_3'' \models E, E'$ then there must be a marking m_2'' over N_2 such that both $m_1'' \uplus m_2'' \models E$ and $m_2'' \uplus m_3'' \models E'$. □

4.5.4 Relabeling (RENAME)

Another standard operation on labeled Petri net is *relabeling*, denoted as $N[a/b]$, that apply a substitution to the labeling function of a net. Assume l is the labeling function over the alphabet Σ. We denote $l[a/b]$ the labeling function on $(\Sigma \setminus \{a\}) \cup \{b\}$ such that $l[a/b](t) = b$ when $l(t) = a$ and $l[a/b](t) = l(t)$ otherwise. Then $N[a/b]$ is the same as net N but equipped with labeling function $l[a/b]$. Relabeling has no effect on the marking of a net. The relabeling law is true even in the case where b is the silent action τ. In this case we say that we *hide* action a from the net.

Theorem 4.5. *If* $(N_1, m_1) \rhd_E (N_2, m_2)$ *then* $(N_1[a/b], m_1) \rhd_E (N_2[a/b], m_2)$.

Proof. Assume $(N_1, m_1) \rhd_E (N_2, m_2)$. Condition (A1) does not depend on the labels and therefore it is also true between $N_1[a/b], E$ and $N_2[a/b]$. For condition (A2), we simply use the fact that for any firing sequences σ_1 and σ_2, $l_1(\sigma_1) = l_2(\sigma_2)$ implies $l_1[a/b](\sigma_1) = l_2[a/b](\sigma_2)$. □

The relabeling law is true even in the case where b is the silent action τ. In this case we say that we have *erased* action a from the nets.

4.6 Deriving E-Equivalences using Reductions

We can compute net reductions by reusing a tool, called Reduce, developed by the Vertics team on combining reductions with decision diagrams [Berthomieu et al., 2019]. The tool takes a marked Petri net as input and returns a reduced net and a sequence of linear equations. For example, given the net M_1 of Fig. 4.1, Reduce returns net M_2 and equations $p_2 = p_1 + p_4, a_1 = p_0 + p_1, a_2 = a_o + p_3, a_3 = a_0 + p_4$, whose conjunctions are equivalent to formula E_M from Fig. 4.1.

The tool works by applying successive reduction laws, in a compositional way, and it is possible to prove that each reduction step, from a net (M_i, m_i) to (M_{i+1}, m_{i+1}) with equations E_i, is such that $(M_i, m_i) \rhd_{E_i} (M_{i+1}, m_{i+1})$. Therefore, by Th. 4.2, the reductions computed by Reduce always translate into valid polyhedral abstractions.

We can look at our running example to explain the inner working of Reduce. It is always safe to remove a *redundant transition*, e.g. a transition with the same **pre** and **post** than another one. Indeed, this preserves the reachable markings. This is the case with the pair t_1,t_4. After removing t_4, for instance, it is apparent that place p_2 is redundant (see the first equivalence in Fig. 4.10). Our tool can find such occurrences by solving an integer linear programming problem [Silva et al., 1998]. After the removal of p_2, we are left with a residual net similar to the one in the second equivalence of Fig. 4.10. In this case, we can simplify places p_0 and p_1, since the tokens in these places can only "move to" p_3 and p_4 in a deterministic way. This is one of the original net reduction rules found in [Berthelot, 1987]. Similar situations, where we can aggregate several places, can be found by searching patterns in the net. Our tool can also identify other opportunities for reductions, like specific structural or behavioural restrictions such that the set of reachable markings is exactly defined by the solutions of the state equation [Hujsa et al., 2020].

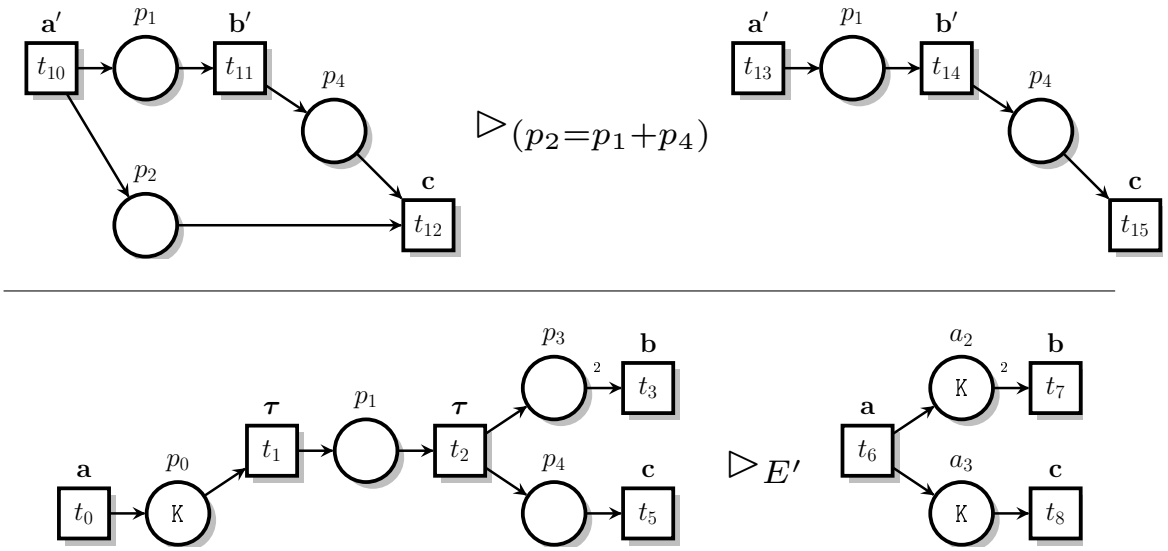


Figure 4.10: An example of removal of redundant place (above), and of agglomeration rule (below), with $E' \triangleq (a_1 = p_0 + p_1) \wedge (a_2 = a_1 + p_3) \wedge (a_3 = a_1 + p_4)$.

In conclusion, we can use Reduce to compute polyhedral abstractions automatically. In the other direction, we can use our notion of equivalence to prove the correctness of new reduction patterns that could be added in the tool. While it is not always possible to reduce the complexity of a net using this approach, we observed in our experiments (Chapter 7) that, on a benchmark suite that includes almost 1 000 instances of nets, about half of them can be reduced by a factor of more than 30%.

4.7 An Example of Totally Reducible Nets

We show how we could apply our different rules on the Petri net given in Fig. 1.4, that was taken from [Stahl, 2011]. This net can be *fully reduced*, meaning that we can apply

reductions until we reached the "singleton net" (a Petri net with only one place and no transitions). This net has only one marking (its initial one).

When we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and the initial markings are obvious from the context we simply write $N_1 \rhd_E N_2$.

When looking at the initial net, Fig. 4.11, we can see several occurrences of the "pattern" in rule (CONCAT) (sub-nets that are isomorphic to N_1 in Fig. 4.1). We have emphasized a particular example in blue that involves places p_1 and p_2[1].

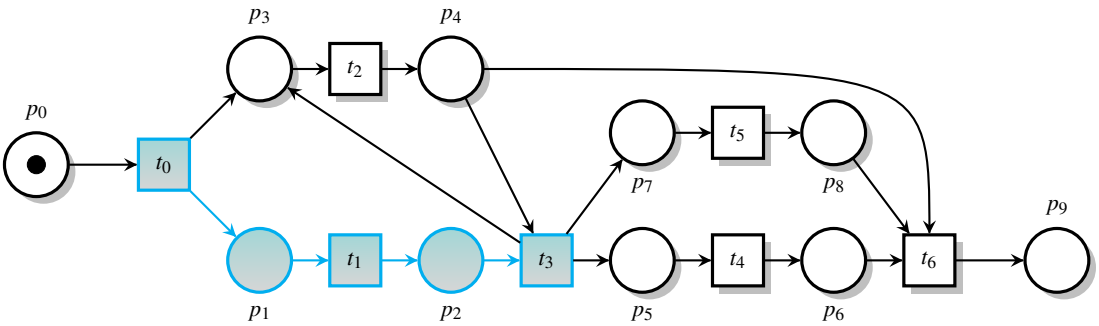


Figure 4.11: Initial net, S_1, with a pattern for rule (CONCAT) emphasized in blue

We can always use a default labeling function where each transition names are used as labels: we have $\Sigma = T$ and $l(t) = t$. With this default, by successive application of rule (CONCAT), (RENAME) (to erase the label on transition t_1) and (COMP) we can derive a reduced net S_2, see Fig. 4.12, such that $S_1 \rhd_E S_2$ where E is the single equation $a_1 = p_1 + p_2$.

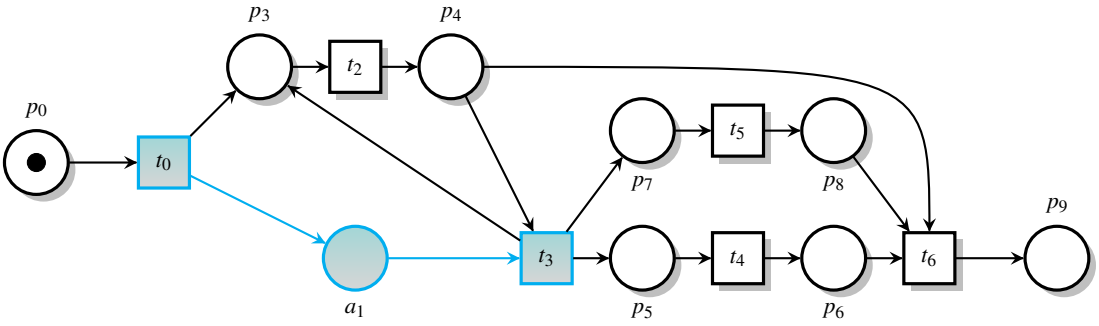


Figure 4.12: Net S_2, with the result of applying rule (CONCAT) emphasized in blue

A similar process can be applied on the pairs of places $(\{p_3, p_4\}, \{t_2\})$, $(\{p_5, p_6\}, \{t_4\})$ and $(\{p_7, p_8\}, \{t_5\})$, which are also a variation of the (CONCAT) rule. Hence, using the transitivity law, we can derive a sequence of reduced net S_3, S_4, S_5 such that:

$$S_1 \rhd_{a_1 = p_1 + p_2} S_2 \rhd_{a_2 = p_3 + p_4} S_3 \rhd_{a_3 = p_5 + p_6} S_4 \rhd_{a_4 = p_7 + p_8} S_5$$

[1] In this example, we use a variation of rule (CONCAT) where the transition with label b in N_1 and N_2 is removed.

We display net S_5 in Fig. 4.13. From law (TRANS), we have that $S_1 \rhd_{E_5} S_5$, where E_5 is the (solvable) system of equations:

$$E_5 = \begin{cases} a_1 &= p_1 + p_2, \\ a_2 &= p_3 + p_4, \\ a_3 &= p_5 + p_6, \\ a_4 &= p_7 + p_8 \end{cases} \tag{4.1}$$

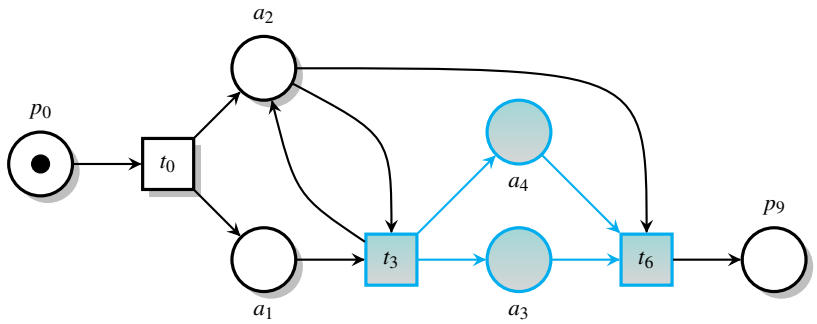


Figure 4.13: Net S_5, with a pattern for rule (RED1) emphasized in blue

After these first steps of reduction, we obtain a net with an opportunity to use rule (RED1); on the sub-net of S_5 with places a_3, a_4. (Again, we have emphasized this sub-net in blue in Fig. 4.13.) By the (COMP) and (TRANS) law, we can therefore infer that $S_1 \rhd_{E_6} S_6$, where E_6 is the system $E_5, a_4 = a_3$.

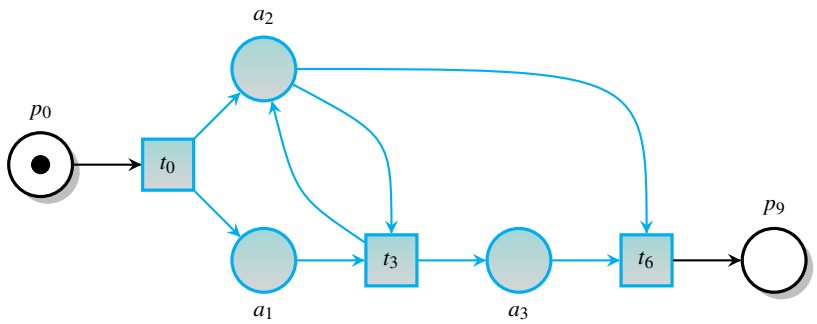


Figure 4.14: Net S_6, with a pattern for rule (RED3) emphasized in blue

We now gained the ability to use rule (RED3) on the sub-net with places a_1, a_2, a_3 to remove a_2, resulting in net S_7 and the system E_7 that is equal to $E_6, a_2 = a_1 + a_3$.

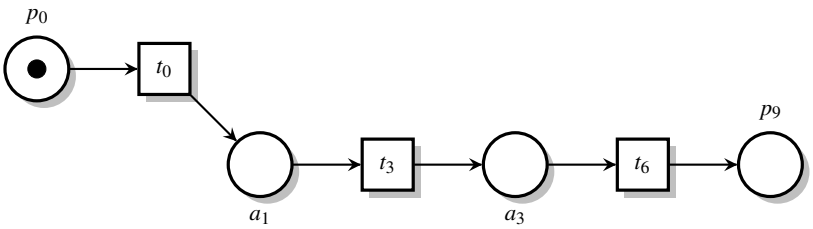


Figure 4.15: Net S_7

At this stage, we can apply rule (CONCAT) several times to remove places p_0, a_1, a_3, p_9 until we obtain a net, see S_8 below, containing only one place and no transitions. Each application of rule (CONCAT) introduces a fresh variable name, say a_5, a_6, a_7[2]. All in all, we infer that $S_1 \rhd_{E_8} S_8$, where E_8 is the system $E_7, a_5 = p_0 + a_1, a_6 = a_5 + a_3, a_7 = a_6 + p_9$; giving the final system E such that:

$$E = \begin{cases} a_1 &= p_1 + p_2, \\ a_2 &= p_3 + p_4, \\ a_3 &= p_5 + p_6, \\ a_4 &= p_7 + p_8, \\ a_4 &= a_3, \\ a_2 &= a_1 + a_3, \\ a_5 &= p_0 + a_1, \\ a_6 &= a_5 + a_3, \\ a_7 &= a_6 + p_9 \end{cases} \tag{4.2}$$

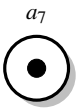


Figure 4.16: Net S_8

Because the resulting net has no transitions left, its only reachable marking is the initial marking: this is the only marking over $\{a_7\}$ such that $a_7 = 1$. As a consequence of Lemma 4.1, every marking m that is reachable in S_1 must be such that $\underline{m} \models E, a_7 = 1$. Conversely, from Lemma 4.2, every marking that satisfies this set of constraints is reachable in $S - 1$. As a consequence, we can represent the set of reachable markings in S_1 as the set of (positive, integer) solutions to the linear system $E, a_7 = 1$.

[2]We can apply these rules in a different order. If we always obtain the same result in this example, our reduction system is not confluent: a different order in the application of rules may not ultimately lead to the same result.

— 5 —
Implementation

We developed a prototype model-checker, called **SMPT** (for **Satisfiability Modulo Petri Net**) which is based on a SMT approach and that takes advantage of net reductions. The SMPT tool is open source, under the GPLv3 licence, and is freely available on GitHub github.com/nicolasAmat/SMPT/ (see the explanations in Chapter 7). The current tool offers three different analysis methods that have been developed for generalized Petri nets. No optimizations have been implemented, yet, in the case where the Petri net is safe.

The main components of the tool correspond to the implementation of the BMC and PDR methods, that we describe in Sect. 5.2 and 5.3. Before detailing these methods, we give a brief overview of the "enumerative method", which corresponds to the simplest possible approach and gives a good opportunity to understand the advantages of using reductions.

5.1 Enumerative Markings

The simplest method implemented in our model-checker is what we call the *enumerative method*. Assume we want to find states reachable in the net (N, m_0) that satisfies formula F, that usually models a set of "feared events". The idea is to build a formula, ρ, that represents the reachable states as a union (a disjunction) of all the markings in $R_N(m_0)$ and test whether $F \wedge \rho$ is satisfiable. Otherwise, we have proven that $\neg F$ is an invariant over the states of N.

Given a marking m over the set $\{p_1, \ldots, p_n\}$, we denote $\underline{m}$ the formula that is "satisfiable only for m". This is a simple, conjunctive formula in Presburger arithmetic.

$$\underline{m} \triangleq (p_1 = m(p_1)) \wedge \cdots \wedge (p_n = m(p_n)) \tag{5.1}$$

$$\rho(N, m_0) \triangleq \bigvee \{\underline{m} \mid m \in R_N(m_0)\} \tag{5.2}$$

$$\phi_R(N, m_0) \triangleq F(\vec{p}) \wedge \rho(N, m_0) \tag{5.3}$$

Formula ϕ_R has as many variables that places in N. Moreover, it has a size proportional to the number of markings in $R_N(m_0)$. The formula only uses Boolean operators and equality between variables and integers (no quantifiers). This category of constraints can be solved using procedures for satisfiability modulo Quantifier-free Linear Integer Arithmetic (QF-LIA), which are part of most SMT solvers.

If the SMT solver returns *UNSAT* on formula ϕ_R then property $\neg F$ is a safety invariant; otherwise, if the solver returns *SAT*, we can extract a model corresponding to a counter-example for property $\neg F$.

This method can only be applied on bounded Petri nets and, in practice, only on nets with a "reasonable" number of reachable markings (less than a few thousands). The SMPT tool provides an option, `-auto-enumerative`, that uses the tool TINA to enumerate all the reachable markings of a net and to build an equivalent of formula ϕ_R above. We can also use option `-enumerative PATH_MARKINGS` when we have already computed the state space in `.aut` format, the description format used for Labeled Transition Systems in the Aldebaran tool, part of the CADP toolbox.

Enumerative markings with reductions

We can use a more efficient approach when net N has reductions, since we can decrease both the number of variables and the size of the formula.

We denote $\tilde{E}(\vec{x}_i, \vec{y}_i)$ the formula obtained from E where place names in N_1 are replaced with variables in $\vec{x}_i$ and place names in N_2 are replaced with variables in $\vec{y}_i$. When we have the same place in both nets, say x_j and y_l stand for the same place in $P_1 \cap P_2$, this operation add the constraint $(x_j = y_l)$ to $\tilde{E}$.

Assume that the net N_1 can be reduced to N_2, that is $(N_1, m_1) \rhd_E (N_2, m_2)$. Then it is enough to check the satisfiability of formula $\phi_R^r(N_2, m_2) \triangleq F(\vec{p}_1) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2) \wedge \rho(N_2, m_2)$.

The correctness of this method is a direct corollary of the basic properties of E-abstraction (Section 4.3). Note that formula ϕ_R^r includes both F and E. Therefore it may have more variables than ϕ_R, since it can also include variables that are in E but not in N_1 or N_2. On the other hand, it may be a lot smaller.

Theorem 5.1. *Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. There is a marking m'_1 over N_1 such that $m'_1 \models \phi_R(N_1, m_1)$ if and only if there is m'_2 over N_2 such that $m'_2 \models \phi_R^r(N_2, m_2)$.*

Proof. For the first direction of this equivalence, we assume that $(N_1, m_1) \rhd_E (N_2, m_2)$ and $m'_1 \models \phi_R(N_1, m_1)$. Hence formula $\underline{m'_1}(\vec{p}_1) \wedge F(\vec{p}_1)$ is satisfiable. By Lemma 4.1, there must be m'_2 reachable in N_2 such that $m'_1 \uplus m'_2 \models E$. Which means that $\underline{m'_1}(\vec{p}_1) \wedge \underline{m'_2}(\vec{p}_2) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2)$ is satisfiable. Therefore $\underline{m'_1}(\vec{p}_1) \wedge \underline{m'_2}(\vec{p}_2) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2) \wedge F(\vec{p}_1)$ is also satisfiable, which entails $m'_2 \models \phi_R^r(N_2, m_2)$. The other direction is similar but relies on Lemma 4.2. □

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. If the SMT solver returns *UNSAT* on formula $\phi_R^r(N_2, m_2)$ then we know that formula $\phi_R(N_1, m_1)$ is also *UNSAT*, and therefore that $\neg F$ is an invariant over the reachable markings of N_1. Same thing if the solver returns *SAT*. In this case, we can extract a model m_2' corresponding to a counter-example for property $\tilde{E}(\vec{p}_1, \vec{p}_2) \wedge F(\vec{p}_1)$ in N_2. By definition of E-abstraction equivalence, we know that every marking m_1' that is a model of $m_2'(\vec{p}_2) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2)$ is reachable in N_1. Every such solution is a counter-example for $\overline{F(\vec{p}_1)}$ in N_1. This method can be extremely efficient when the net has a lot of reductions. For instance when the net is fully (or almost fully) reducible.

5.2 Bounded Model Checking (BMC)

The *Bounded Model Checking* analysis method, or *BMC* for short, is an iterative method exploring the state-space of finite-state systems by unrolling their transitions [Biere et al., 1999]. The method was originally based on an encoding of transition systems into (a family of) propositional logic formulas and the use of SAT solvers to check these formulas for satisfiability [Clarke et al., 2001]. More recently, this approach was extended to more expressive models, and richer theories, using SMT solvers [Armando et al., 2006]. We can also find recent works focused on High-Level Petri Nets [Liu and He, 2015].

In BMC, we try to find a reachable marking m that is a model for a given formula F, that models a set of "feared events". The algorithm starts by computing a formula, say ϕ_0, representing the initial marking and checking whether $\phi_0 \wedge F$ is satisfiable (meaning F is initially true). If the formula is *UNSAT*, we compute a formula ϕ_1 representing all the markings reachable in one step, or less, from the initial marking and check $\phi_1 \wedge F$. This way, we compute a sequence of formulas $(\phi_i)_{i \in \mathbb{N}}$ until either $\phi_i \wedge F$ is *SAT* (in which case a counter-example is found) or we have $\phi_{i+1} \Rightarrow \phi_i$ (in which case we reach a fixpoint and no counter-example exists).

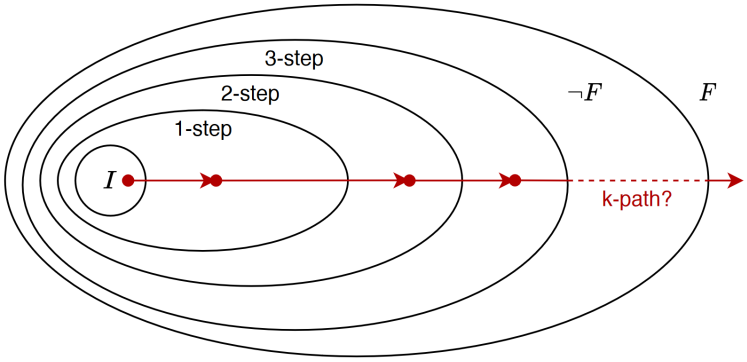


Figure 5.1: BMC method representation

The BMC method is not complete since it is not possible, in general, to bound the number of iterations needed to give an answer. Also, when the net is unbounded, we

may very well have an infinite sequence of formulas $\phi_0 \subsetneq \phi_1 \subsetneq \ldots$ However, in practice, this method can be very efficient to find a counter-example when it exists.

The crux of the method is to compute formulas ϕ_i that represents the set of markings reachable using firing sequences of length at most i. We show how we can build such formulas incrementally. We assume that we have a marked net (N, m_0) with places $P = \{p_1, \ldots, p_n\}$ and transitions $T = \{t_1, \ldots, t_k\}$. In the remainder of this section, we build formulas that express constraints between markings m and m' such that $m \to m'$ in N. Hence we define formulas with $2n$ variables. We use the notation $\psi(\vec{x}, \vec{x'})$ as a shorthand for $\psi(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$.

Given a transition t in N we define the formula $\text{ENBLD}_t(\vec{x})$ that is true when t is enabled at m. We also define formula Δ_t that describes the evolution of a marking after transition t fires.

$$\text{ENBLD}_t(\vec{x}) \triangleq \bigwedge \{(x_i \geq k) \mid k = \mathbf{pre}(t, p_i) > 0\} \tag{5.4}$$

$$\Delta_t(\vec{x}, \vec{x'}) \triangleq \bigwedge \{(x'_i = x_i + \delta_i) \mid \delta_i = \mathbf{post}(t, p_i) - \mathbf{pre}(t, p_i), 1 \leq i \leq n\} \tag{5.5}$$

In a similar way than with the previous method, we use $\underline{m}(\vec{x})$ to denote a formula with variables in $\vec{x}$ that is only "satisfiable at marking m". (With our notations, formula $\underline{m}$ is equivalent to $\underline{m}(\vec{p})$.)

$$\underline{m}(\vec{x}) \triangleq \bigwedge_{i \in 1..n} (x_i = m(p_i)) \tag{5.6}$$

Next, we define the helper formula $t(\vec{x}, \vec{x'})$ such that $(\underline{m}(\vec{x}) \wedge t(\vec{x}, \vec{x'}) \wedge \underline{m'}(\vec{x'}))$ entails that $m \xrightarrow{t} m'$ when t is enabled at m or that $m = m'$ otherwise. We will use the simpler notation $t(m, m')$ as a shorthand for formula $\underline{m}(\vec{x}) \wedge t(\vec{x}, \vec{x'}) \wedge \underline{m'}(\vec{x'})$.

$$\text{EQ}(\vec{x}, \vec{x'}) \triangleq \bigwedge_{i \in 1..n} x_i = x'_i \tag{5.7}$$

$$t(\vec{x}, \vec{x'}) \triangleq \big(\text{ENBLD}_t(\vec{x}) \Rightarrow \Delta_t(\vec{x}, \vec{x'})\big) \wedge \big(\neg\text{ENBLD}_t(\vec{x}) \Rightarrow \text{EQ}(\vec{x}, \vec{x'})\big)$$

Note that formula t can be more clearly defined using an `if - then - else` operator, *ite*, since $t(\vec{x}, \vec{x'}) \triangleq ite(\text{ENBLD}_t(\vec{x}), \Delta_t(\vec{x}, \vec{x'}), \text{EQ}(\vec{x}, \vec{x'}))$.

With all these notations, it is easy to define formula $T(\vec{x}, \vec{x'})$ as the disjunction of all the $t(\vec{x}, \vec{x'})$ for t a transition in T. Hence formula $\underline{m}(\vec{x}) \wedge T(\vec{x}, \vec{x'}) \wedge \underline{m'}(\vec{x'})$ entails that m' is at most "one step of reduction" from m. We should use a simplified version of this formula where we remove sub-terms of the form $(\neg\text{ENBLD}_j(\vec{x}) \Rightarrow \text{EQ}(\vec{x}, \vec{x'}))$ which are entailed by the top-level term $\text{EQ}(\vec{x}, \vec{x'})$, see below.

$$T(\vec{x}, \vec{x'}) \triangleq \text{EQ}(\vec{x}, \vec{x'}) \vee \bigvee_{t \in T} \big(\text{ENBLD}_t(\vec{x}) \wedge \Delta_t(\vec{x}, \vec{x'})\big) \tag{5.8}$$

Formula ϕ_i is the result of connecting i successive occurrences of formulas of the form $T(\vec{x}_j, \vec{x}_{j+1})$. We define the formulas inductively, with a base case (ϕ_0) which states that only m_0 is reachable initially. To define the ϕ_i's, we assume that we have a collection of (pairwise disjoint) sequences of variables, $(\vec{x}_i)_{i \in \mathbb{N}}$.

$$\phi_0(N, m_0) \triangleq \underline{m_0}(\vec{x}_0) \qquad \phi_{i+1}(N, m_0) \triangleq \phi_i(N, m_0) \wedge T(\vec{x}_i, \vec{x}_{i+1})$$

We say that marking m is a model of formula $\phi_i(N, m_0)$, denoted $m \models \phi_i(N, m_0)$, when formula $\underline{m}(\vec{x}_i) \wedge \phi_i(N, m_0)$ is satisfiable. In this case, m is exactly the mapping from P to $\mathbb{N}$ defined by the valuation of variables in $\vec{x}_i$. We can prove that the BMC formula provide a way to find counter-examples to F in the reachable markings of (N, m_0).

We can prove that this family of BMC formulas provide a way to check invariants on the reachable markings of (N, m_0).

Theorem 5.2 (BMC reachability)**.** *Assume $F(\vec{p})$ is a formula with variables in P, the set of places in N. Formula $F(\vec{x}_i) \wedge \phi_i(N, m_0)$ is satisfiable if and only if there is a firing sequence σ such that $m_0 \stackrel{\sigma}{\Rightarrow} m$ in N and $m \models F(\vec{p})$.*

Proof (sketch). The proof is by induction on the value of i and use the fact that $T(m, m')$ entails $m \Rightarrow m'$. It is also possible to show that $F(\vec{x}_i) \wedge \phi_i(N, m_0)$ satisfiable implies that σ is of length at most i. □

Therefore we can find a counter-example to F by checking the satisfiability of formulas of the form $\phi_i(N, m_0) \wedge F(\vec{x}_i)$, where $F(\vec{x})$ is obtained from F by substituting variables $\vec{p}$ with $\vec{x}$.

We display in Fig. 5.2 the general architecture of the BMC method. This method is more efficient when combined with an SMT solver supporting `push / pop` operations on the assertion stack. Indeed, to check the satisfiability of a formula at a new iteration, we do not need to reload a new formula entirely.

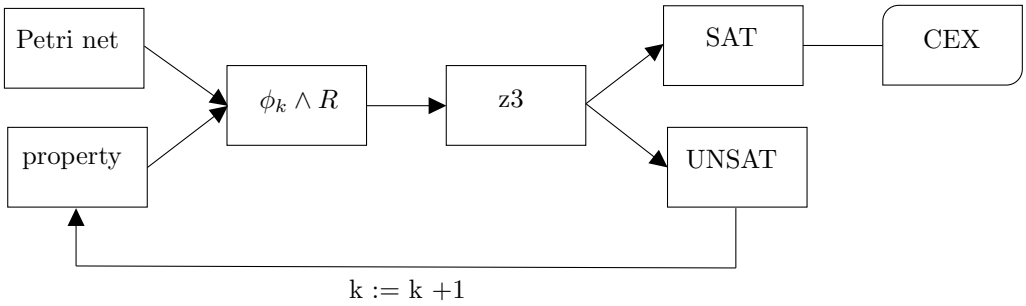


Figure 5.2: BMC Algorithm

Extension to inhibitor arcs

The transition formula (5.8) must be modified in the case of inhibitor arcs. If a place p is linked to transition t by an inhibitor arc, we should replace the term $p_i \geq k$ with $p_i < k$ in the formula ENBLD_t. At the opposite, no modifications are necessary for read arcs, since they are equivalent to the combination of two regular arcs (one "incoming" and one "outgoing").

Extension of BMC with reductions

The approach we describe here is well-known. It is also quite simplified. Actual model-checkers that rely on BMC apply several optimizations techniques, such as compositional reasoning; acceleration methods; or the use of invariants on the underlying model to add extra constraints. We do not consider such optimizations here, on purpose, since our motivation is to study the impact of polyhedral abstractions. We believe that our use of reductions is orthogonal and do not overlap with many of these optimizations, in the sense that we do not preclude them, and that the performance gain we observe with reductions could not be obtained with these optimizations.

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. We denote T_1, T_2 the equivalent of formula T, above, for the nets N_1, N_2 respectively. In the following, we use $\vec{x}$, $\vec{y}$ for sequence of variables ranging over (the places of) N_1 and N_2 respectively. We should use $\phi(N_1, m_1)$ for the family of formulas build using operator T_1 and variables $\vec{x}_0, \vec{x}_1, \ldots$ and similarly for $\phi(N_2, m_2)$, where we use T_2 and variables of the form $\vec{y}$.

As for the enumerative method, we denote $\tilde{E}(\vec{x}_i, \vec{y}_i)$ the formula obtained from E where place names in N_1 are replaced with variables in $\vec{x}_i$ and place names in N_2 are replaced with variables in $\vec{y}_i$. When we have the same place in both nets, say x_j and y_l stand for the same place in $P_1 \cap P_2$, this operation add the constraint $(x_j = y_l)$ to $\tilde{E}$.

Lemma 5.1. *Assume m_1, m_2 are markings over N_1, N_2 respectively. With the notations used in this section, we have $m_1 \uplus m_2 \models E$ if and only if formula $\underline{m_1}(\vec{p}_1) \wedge \underline{m_2}(\vec{p}_2) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2)$ is satisfiable.*

The following property states that, to find a model of $F(\vec{p}_1)$ in the reachable markings of N_1, it is enough to find a model for $F(\vec{p}_1) \wedge \tilde{E}(\vec{p}_1, \vec{p}_2)$ in N_2.

Theorem 5.3 (BMC with reductions)**.** *Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$ and that $F(\vec{p}_1)$ is a formula with variables in P_1, the set of places in N_1. Formula $F(\vec{x}_i) \wedge \tilde{E}(\vec{x}_i, \vec{y}_i) \wedge \phi_i(N_2, m_2)$ is satisfiable if and only if there is a firing sequence σ such that $m_1 \stackrel{\sigma}{\Rightarrow} m_1'$ in N_1 and $m_1' \models F(\vec{p}_1)$.*

Proof. We can observe that, since $\phi_i(N_2, m_2) \wedge \tilde{E}(\vec{x}_i, \vec{y}_i) \wedge \underline{m_1}'(\vec{x}_i)$ is of the form $\phi_i(N_2, m_2) \wedge \psi$, we have that $m_2' \models \phi_i(N_2, m_2) \wedge \tilde{E}(\vec{x}_i, \vec{y}_i) \wedge \underline{m_1}'(\vec{x}_i)$ implies $m_2' \models \phi(N_2, m_2)$.

We prove this property by induction on i. For the base case, we need to compare formulas $\phi_0(N_1, m_1)$ and $\phi_0(N_2, m_2)$. The result follows from Lemma 5.1 and the fact that, by definition of E-abstraction, we have $m_1 \uplus m_2 \models E$.

For the induction case, we prove each direction of the equivalence separately. Assume we have $m_1' \models \phi_i(N_1, m_1)$. By Th. 5.2 it follows that m_1' is reachable in (N_1, m_1) by some firing sequence of length i, say σ_1. Hence, by property of E-abstractions, we know that there is (at least) a firing sequence σ_2 and a marking m_2' such that (1) $m_2 \overset{\sigma_2}{\Longrightarrow} m_2'$ and (2) $m_1' \uplus m_2' \models E$. By property (2) and Lemma 5.1 we have that $m_2' \models \tilde{E}(\vec{x}, \vec{y}) \wedge \underline{m_1}'(\vec{x})$. By property (2) and Th. 5.2, it must be the case that $m_2' \models \phi_j(N_2, m_2)$, where j is the length of σ_2. When we have "real reductions" (meaning we only eliminate places and transitions, never the opposite), we also know that σ_2 is smaller than σ_1. Since the models of ϕ_j are also models of ϕ_i when $j \leq i$, we finally obtain $m_2' \models \phi_i(N_2, m_2) \wedge \tilde{E}(\vec{x}_i, \vec{y}_i) \wedge \underline{m_1}'(\vec{x}_i)$, as needed. The induction case in the opposite direction is similar. □

Our proof actually shows that we can find a counter-example of length i in N_1 by finding a counter-example of length $j \leq i$ in N_2. This is because reductions may compact a sequence of several transitions into a single one. Take the example of rule (CONCAT). Therefore BMC benefits from reductions in two ways. First because we can reduce the size of formulas (which are proportional to the size of the net), but also because we can "accelerate" transitions in the reduced net.

5.3 Property Directed Reachability (PDR)

While BMC is the right choice when we try to find counter-examples, it usually performs poorly when we want to check an invariant property, $\mathrm{AG}\neg F$. There are techniques that are better suited to prove *inductive invariants* in a transition system; that is a property that is true initially and stays true after firing any transition.

In order to check invariants with SMPT, we have implemented a method called PDR [Bradley, 2011, Bradley, 2012], which incrementally generates clauses that are inductive "relative to stepwise approximate reachability information". PDR is a combination of induction, over-approximation, and SAT solving. For SMPT, we developed a similar method that uses SMT solving, to deal with markings and transitions, and that can take advantage of polyhedral abstractions.

Notations

We use the same notations than with BMC. The PDR method requires to define a set of *safe states*, described as the models of some property $\neg F$. It also requires a set of initial states, I. In our case $I \triangleq \underline{m_0}(\vec{x})$. The procedure is complete for finite transition systems, for instance with bounded Petri nets. We can also prove termination in the general case when property $\neg F$ is *monotonic*, meaning that $m \models \neg F$ implies that $m' \models \neg F$ for all markings m' that covers m (that is when $m' \geq m$, component-wise). An intuition is that

it is enough, in this case, to check the property on the minimal coverability set of the net, which is always finite (see e.g. [Finkel, 1991]).

The PDR method computes sets of states $I, P, Q, \ldots$. We represent these sets using formulas ranging over the set of places $X = \{p, q, \ldots\}$ and we will often use set-theoretic symbols for formulas, such as $P \subseteq Q$ instead of $P \Rightarrow Q$. By convention, we use primed variables $p', q', \ldots$ for marking m' (reached from m by firing T). and we use $I', P', \ldots$ for the formulas $I, P, \ldots$ where variables are substituted with their primed counterparts.

Definition 5.1 (Inductive formula)**.** *A formula F is said inductive, if both $I \Rightarrow F$ and $F(\vec{x}) \wedge T(\vec{x}, \vec{x}') \Rightarrow F(\vec{x}')$ hold. A formula F is inductive relative to a formula G if both $I \Rightarrow F$ and $G(\vec{x}) \wedge F(\vec{x}) \wedge T(\vec{x}, \vec{x}') \Rightarrow F(\vec{x}')$ hold.*

We have adapted the original PDR method so that it can deal with general markings (and not only markings of safe Petri nets). The original definition of PDR [Bradley, 2011] relies heavily on a syntactical restriction over states, that must be expressed as cubes.

A *witness* for a formula $\phi(\vec{x})$ is a marking m such that $\underline{m}(\vec{x}) \wedge \phi(\vec{x})$ is satisfiable; also denoted $m \models \phi$. Because of our extension to PDR, it could be the case that we have infinitely many states in $\phi(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge (\neg F)(\vec{x}')$, especially if we cannot add an explicit bound on the marking of a place. Nonetheless, we are sure to terminate if the system is finite. To overcome the problem of a potential infinite number of witnesses, we define formula $\underline{\hat{m}}$ that is valid for every state m' that has more tokens than m on all places: formula $\underline{\hat{m}}(\vec{x}) \wedge \underline{m'}(\vec{x})$ is valid when m' covers m.

$$\underline{\hat{m}}(\vec{x}) \triangleq \bigwedge \{x_i \geq m(x_i) \mid m(x_i) > 0\} \tag{5.9}$$

By virtue of the monotonicity of the flow function of Petri nets, when $\neg F$ is monotonic and m is a witness, we know that all models of $\underline{\hat{m}}$ are also witnesses (if we do not use inhibitor arcs). Another benefit of this choice is that $\underline{\hat{m}}$ is a conjunction of inequalities of the form $(x_j \geq k_i)$, which greatly simplifies the computation of the *minimal inductive clause*, defined later. When F is anti-monotonic ($\neg F$ is monotonic), we can prove the completeness of the procedure using an adaptation of Dickson's lemma, which states that we cannot find an infinite decreasing chain of witnesses (but the number of possible witness may be extremely large).

Over Approximated Reachability Sequence (OARS)

We want to build an *Over Approximated Reachability Sequence* (OARS), meaning a monotone sequence of formulas $F_0, \ldots, F_{k+1}$, with variables in $\vec{x}$, such that:

1. $(F_0 = I \subseteq F_1 \subseteq \cdots \subseteq F_{k+1} \subseteq \neg F)$,
2. for all $i \in 0..k+1$. $F_i(\vec{x}) \wedge T(\vec{x}, \vec{x}') \Rightarrow F_{i+1}(\vec{x}')$ (*consecution*).

The idea is to stop when we find a fixpoint; an index i such that $F_i = F_{i+1}$. In this case, we can return *SAFE* because there can be no sequence of transitions starting from I and reaching F. We can also stop during the iteration if we find a counter-example.

By construction, each of the F_i is a formula in CNF (conjunctive normal form); basically, it is the conjunction of a set of clauses of the form $\neg\underline{\hat{m}}$. Also, since we usually have that I is a singleton set, we already have that F_0 is in CNF.

We define $CL(F_i)$ to be the set of clauses in F_i. By design, the PDR algorithm will build an OARS where CL is always decreasing, meaning that $CL(F_{i+1}) \subseteq CL(F_i)$. This is much more restrictive than the condition $F_i \subseteq F_{i+1}$. This constraint has a positive impact on performances since we can check the equality $F_i = F_{i+1}$ by syntactically looking at the equivalence of the clauses (as set equality) rather than as semantical equality.

With these notations we have that (apart from $F_0 = I$ that may be considered as a special case) $F_i \triangleq (\neg F) \wedge \bigwedge CL(F_i)$. The invariant preserved during the computation is that each F_i describes a set of states that:

1. includes the markings m less than i steps from I, $\{m' \mid \forall m \in I. \forall \sigma.\ m \stackrel{\sigma}{\Rightarrow} m' \Rightarrow |\sigma| \leq i\} \subseteq F_i$,

2. contains only states m which are more than $k-i+1$ steps from F (which include states that cannot reach F), meaning $F_i \subseteq \{m \mid \forall m' \in F. \forall \sigma.\ s \stackrel{\sigma}{\Rightarrow} m' \Rightarrow |\sigma| \geq k-i+1\}$.

Therefore formula F_i represents an over-approximation of the states reachable in up to i steps. A modification we add in our version of PDR is that F_i is a union of $\underline{\hat{m}}$ and not of $\underline{m}$.

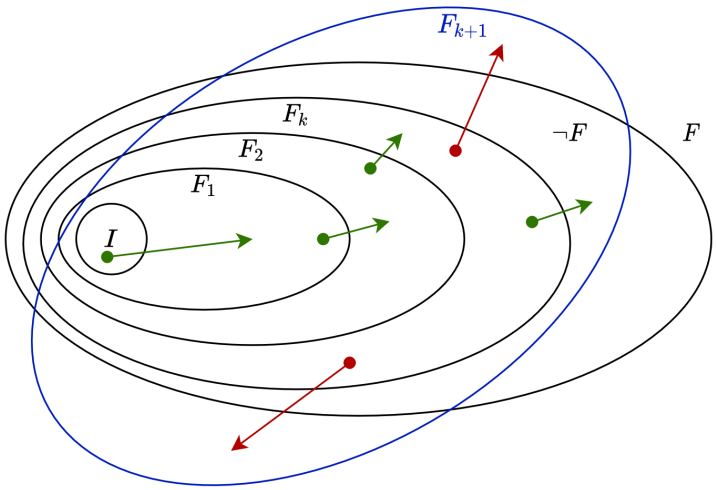


Figure 5.3: PDR method representation

Algorithm

Initialization

1. We check $I(\vec{x}) \wedge F(\vec{x})$. If it is *SAT* then we have a trivial counter-example; return *CEX*.

2. We check $I(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$. Likewise, if it is *SAT* then return *CEX*; we have a counter-example that can reach F after only one transition.

3. We check $(\neg F)(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$. If it is *UNSAT* then return *SAFE*; because then property $\neg F$ is inductive. Indeed, from step 1 we have that $I(\vec{x}) \Rightarrow (\neg F)(\vec{x})$ and from step 3 we have $(\neg F)(\vec{x}) \wedge T(\vec{x},\vec{x}') \Rightarrow (\neg F)(\vec{x}')$.

4. At this point, we have that the sequence $(F_0 = I, F_1 = P)$ is an OARS. Indeed, we have $I \subseteq \neg F$ from step 1 (and therefore $F_0 \subseteq F_1$), and we have $F_0(\vec{x}) \wedge T(\vec{x},\vec{x}') \Rightarrow F_1(\vec{x}')$ from step 2.

5. Since $\neg F$ is not inductive from step 3, we also have that $(\neg F)(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$ is *SAT*. Therefore we can enumerate solutions to $(\neg F)(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$, say $m_1, m_2, \ldots$ Each of these witness corresponds to a *dangerous marking*, a state that can reach F by firing T. We do not need to enumerate all the witnesses beforehand, but we will need to iterate over them.

6. Iterate through the witnesses and add $\neg\underline{\hat{m}_i}$ to the set $CL(F_1)$ until you have F_1 inductive. If at one point we have $I(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \underline{\hat{m}_i}(\vec{x}')$ *SAT* then return *CEX*; because we can reach F from I going through $\hat{s}_i$. We stop as soon as we have $F_1(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$ *UNSAT*, meaning $F_1 \wedge T(\vec{x},\vec{x}') \Rightarrow (\neg F)(\vec{x}')$. Actually, we do not directly add $\neg\underline{\hat{m}_i}$ but we try, instead, to add a minimal inductive cube $c_i \subseteq \neg\underline{\hat{m}_i}$, see procedure *GenerateClause* in the next section.

7. At this point, we have that the sequence (F_0, F_1) is an OARS.

DOWN, UP and MIC

The algorithm relies on cubes of the form $\neg\underline{\hat{m}}$ to extend the set of clauses in the OARS. When we update a formula F_i (by adding a new clause), we will also add the same clause to all the previous $(F_j)_{j<i}$. It is not necessary to update F_0, but it simplifies the presentation to do so.

From a cube, say c_0, we try to find a *minimal inductive clause* (MIC) for some property G (usually one of the top-level F_i) using the DOWN algorithm of Manna and Bradley [Bradley and Manna, 2007]: start with $G(\vec{x}) \wedge c_0(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \neg c_0(\vec{x}')$; if it is not inductive (*UNSAT*) take a valuation s and consider $c_1 \triangleq c_0 \wedge \neg\underline{\hat{m}}$; repeat until you find an inductive clause. This is exactly what is done in step 5 of initialization.

The final clause, c_m, may not be minimal. This is why we may want to take the result from DOWN and successively remove literals form it while we stay inductive. We just

need to consider a random literal in c_m; remove it, and check if the result is still inductive for G. This optimization is called MIC in [Bradley and Manna, 2007]. Another possibility is to use the UNSAT core of z3 that permits the extraction of an unsatisfiable core, i.e., a subset of clauses that are mutually unsatisfiable. We implemented both methods.

This leads to a general procedure $GenerateClause(\hat{m}, i)$ to be applied to F_i whenever we have a witness.

Algorithm 1: *GenerateClause*

Input: $\hat{m}, i$

Find a "minimal inductive cube" $c \Rightarrow \neg\underline{\hat{m}}$ that is inductive relative to F_i;

(Minimal here means that no strict sub-clause of c is inductive relative to F_i.)

for $1 \leq j \leq i$ **do**
  $CL(F_j) := CL(F_j) \cup \{c\}$;

Iterations

The algorithm alternates between two different kinds of iterations: a *main iteration*, where we add a new element in the OARS (we increase k), and a *minor iteration*, where we generate new *minimal inductive clauses* in the F_i. At the end of the initialization phase, we are in a situation where $k = 1$, and we start a minor iteration.

1. Main iteration

Assume that the sequence $(F_0, \ldots, F_k)$ is an OARS such that $F_k(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$ is *UNSAT*. We extend the sequence with a new element $F_{k+1} = \neg F$. We also assume that $CL(F_{k+1}) = \emptyset$.

Before continuing with the minor iteration, we check every formula F_i, $i \geq 1$, and look for a clause $c \in CL(F_i)$ such that: (1) $c \notin CL(F_{i+1})$ and (2) $F_i(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge \neg c(\vec{x}')$ is *UNSAT*. When this is the case, we propagate the clause forward, i.e., we add c to $CL(F_{i+1})$. This phase is equivalent to function *PropagateClauses* in [Bradley, 2011]. (In this phase we grow the formulas F_i from left to right.)

We continue until no such clause can be propagated. We can stop and return *SAFE* during this process if we find an index i such that $CL(F_i) = CL(F_{i+1})$.

2. Minor iteration

When we enter the minor iteration, we have an OARS with $F_{k+1} = \neg F$. By construction we know that $F_k(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$ is *UNSAT* (represented in Figure 5.4) and that $F_{k+1} \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$ is *SAT* (see step 4 of the initialization phase).

Here we proceed with the *Strengthen* phase of [Bradley, 2011] where we can add new clauses to the F_i and propagate them from right to left. We look at set of witnesses $\hat{m}$

from $F_{k+1} \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$, and generalize them inductively with respect to F_{k-1}. This is done using a helper function, *Push*, defined below. There are three cases to consider:

1. If $F_k(\vec{x}) \wedge \neg\underline{\hat{m}}(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \underline{\hat{m}}(\vec{x}')$ is *SAT* then $GenerateClause(\hat{m}, k-1)$ and $Push(\{(\hat{m},k)\}, k+1)$; we have a state in $F_k \cap \neg\hat{m}$ that leads to $\hat{m}$, this could lead to a *CEX* at distance k from I.

2. Otherwise, if $F_{k+1}(\vec{x}) \wedge \neg\underline{\hat{m}}(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \underline{\hat{m}}(\vec{x}')$ is SAT then $GenerateClause(\hat{m},k)$ and $Push(\{(\hat{m},k+1)\}, k+1)$; we have another state in F_{k+1} that leads to F, we choose to push $\hat{m}$ instead.

3. Otherwise $GenerateClause(\hat{m}, k+1)$; it is not possible to reach $\hat{m}$ from F, but we have an occasion to strengthen our invariants.

We continue until $F_{k+1}(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$ is *UNSAT*, if we have not found a *CEX* before. Then we continue with a main iteration step.

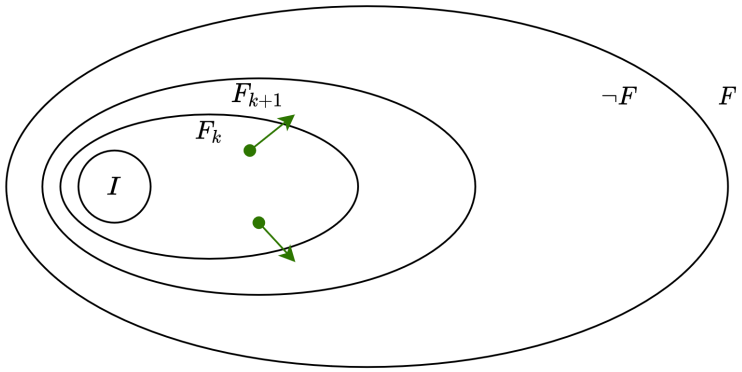


Figure 5.4: $F_k(\vec{x}) \wedge T(\vec{x},\vec{x}') \Rightarrow (\neg F)(\vec{x}')$ equivalent to $F_k(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge F(\vec{x}')$ *UNSAT*

3. Push and Generalize functions

We define two recursively defined functions *Push* and *Generalize*.

The goal of function *Push* is to apply an inductive generalization of a dangerous set of states $\hat{m}$ to its F_i state predecessors. The goal of *Generalize* is to strengthen the invariants in F by adding cubes generated during the *Push* procedure. *Generalize* tries to find the smallest index in $1 \ldots k$ that can lead to the dangerous set of states $\hat{m}$. Actually, a step in the minor iteration is almost like a call to $Generalize(\hat{m}, k-1, k+1)$.

Remarks: Since we always execute *GenerateClause* before calling *Push*, it is always the case that (a minimal inductive clause for) $\neg\underline{\hat{m}}$ is added to F before pushing it. Also, function *Push* works with a set of pairs (t, j) corresponding to possible paths leading to F, where t is interpreted as a state in F_j. Part of the complexity here is to avoid the problematic case of loops in the transition system.

Algorithm 2: *Push*

Input: set, k

Output: int

$(\hat{m}, n)$:= choose from set minimizing n, we skip the pairs were n is bigger than k;

if $SAT(F_n(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \underline{\hat{m}}(\vec{x}'))$ **then**

  $\hat{t}$:= set of witnesses that is not in the current set;

  m := $Generalize(\hat{t}$,n,k);

  $set := set \cup \{(\hat{t}, m+1)\}$;

else

  $m := Generalize(\hat{m}, n, k)$;

  $set := set \setminus \{(\hat{m}, n)\} \cup \{(\hat{m}, m+1)\}$;

Algorithm 3: *Generalize*

Input: $\hat{m}, i, k$

Output: int

if $i < 0$ *and* $SAT(F_0(\vec{x}) \wedge T(\vec{x},\vec{x}') \wedge \neg\underline{\hat{m}}(\vec{x}) \wedge \underline{\hat{m}}(\vec{s}'))$ **then**

  return CEX;

for $i \in i+1 \ldots k$ **do**

  if $SAT(F_i(\vec{x}) \wedge \neg\underline{\hat{m}}(x) \wedge T(\vec{x},\vec{x}') \wedge \underline{\hat{m}}(\vec{x}'))$ **then**

    $GenerateClause(\hat{m}, i-1)$;

    return $i-1$;

$GenerateClause(\hat{m}, k, k)$;

return k;

Extension of PDR with reductions

As it was the case with the enumerative and the BMC methods, we can check the property on the reduced net by adding a conjunction of constraints stating that states are linked in E and that F should be true. In the case of PDR, we should also add a formula expressing the constraint on $\neg F$.

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. We denote T_1, T_2 the equivalent of formula T, from equation (5.8), for the nets N_1, N_2 respectively. In the following, we use $\vec{x}$, $\vec{y}$ for sequence of variables ranging over (the places of) N_1 and N_2 respectively. We use the same formula $\tilde{E}(\vec{x},\vec{y})$ than the one defined for the BMC method (see for instance Lemma 5.1).

To check an invariant on N_1, it is enough to check a slightly modified sequence of

formulas $F_0^r, \ldots, F_i^r$ on N_2, where F_i^r is obtained from the (OARS) formulas $F_0, \ldots, F_i$ as follows:

$$F_0^r(\vec{x},\vec{y}) \triangleq I(\vec{x}) \wedge \bigwedge CL(F_0)(\vec{y}) \wedge \tilde{E}(\vec{x},\vec{y}) \tag{5.10}$$

$$F_i^r(\vec{x},\vec{y}) \triangleq (\neg F)(\vec{x}) \wedge \bigwedge CL(F_i)(\vec{y}) \wedge \tilde{E}(\vec{x},\vec{y}), \text{ for all } i \geq 1 \tag{5.11}$$

The soundness of this approach can be handled by using Th. 4.1. Assume $(N_1, m_1) \rhd_E (N_2, m_2)$, since $\tilde{E}(\vec{p}_1, \vec{p}_2) \wedge (\neg F)(\vec{p}_1)$ is an invariant on N_2 we have that $(\neg F)(\vec{p}_1)$ is an invariant on N_1.

— 6 —
Concurrent Places Problem

In this chapter, we study a possible application of our approach to a problem recently proposed by [Garavel, 2019b, Bouvier et al., 2020] and related to the computation of Nested Units in Petri Nets [Garavel, 2019a]. We show how to use our approach to compute pairs of *concurrent places* in a net. This part of my work is still a work in progress, but it shows a nice application of reductions for the BMC and PDR methods. This is why we decided to include it in this report.

6.1 Nested-Unit Petri Net (NUPN)

Nested-Unit Petri nets are an extension of Petri nets in which *units* of independent places are made explicit. A unit is a group of places that cannot have more that one token in any reachable marking [Garavel, 2019a]. (We only focus on *unit-safe* NUPNs.) NUPN are used during the *Model Checking Contest* and are used by more than a dozen verification tools.

In the following, we consider Petri nets that are *safe* (all place capacities are equal to one) and *ordinary* (such that all arc weights are equal to one).

A *marked Nested-Unit Petri net* is a tuple $(N, m_0, U, u_0, \sqsubseteq, \mathbf{unit})$ where:

- (N, m_0) is a marked net with places and transitions P, T,
- U is the set of units ($U \cap T = U \cap P = \emptyset$),
- $u_0 \in U$ is the root unit,
- $\sqsubseteq$ is a binary relation over U, for which u_0 is the greatest element of this relation and $u_1 \sqsubseteq u_2$ expresses that unit u_1 is transitively nested in or equal to unit u_2,
- $\mathbf{unit} : P \to U$ is a function s.t. $(\forall u \in U \setminus \{u_0\})(\exists p \in P)\ \mathbf{unit}(p) = u$, in other words, $\mathbf{unit}(p)$ expresses that unit u directly contains place p.

NUPNs provide an interesting set of invariants that can be used for model checking or for compositional reasoning on the reachable markings of a Petri net. Computing units in a safe Petri net requires to answer several problems: finding dead places; finding dead transitions; and finding concurrent places.

My work is focused on solving the *Concurrent Places problem*.

6.2 Concurrent Places Problem

Given a marked net (N, m_0) with set of places P, two places p and p' in P are concurrent when there exists a reachable marking m such that both p and p' are marked in m. The concurrency relation characterizes those net parts that can be simultaneously active. It is mentioned in many publications, under various names, such as "concurrency relation" for instance [Van Glabbeek et al., 2012, Garavel, 2019b].

Definition 6.1 (Concurrent places)**.** *Assume (N, m_0) is a marked Petri net. We say that two places p_1 and p_2 of N are concurrent, denoted as $p_1 \| p_2$, if and only if there exists a reachable marking m in $R_N(m_0)$ such that $m(p_1) > 0$ and $m(p_2) > 0$. By extension, we say that $p \| p$ when p is not dead.*

Deciding when places are concurrent is useful for the decomposition of safe nets into NUPN [Bouvier et al., 2020]. The use of net reductions allow us to solve this problem on some very big nets, that were not feasible before.

To simplify our presentation, we define the dual relation to concurrency, called *independence*.

Definition 6.2 (Independent places)**.** *Assume (N, m_0) is a marked Petri net. We say that two places p_1 and p_2 of N are independent, denoted as $p_1 \# p_2$, if and only if for all markings m in $R_N(m_0)$ it is the case that $m(p_1) = 0$ or $m(p_2) = 0$.*

The *Concurrent Places problem* consists of enumerating all pairs of places $(p_1, p_2) \in P \times P$ such that $p_1 \| p_2$. By definition, this relation depends on the choice of the initial marking of the net.

Given a total order on the set of places, $P = \{p_1, \ldots, p_n\}$, the concurrency relation can be represented using a $n \times n$ matrix C where $C[i, j]$ is 1 if $p_i \| p_j$ and 0 if $p_i \# p_j$. Since the concurrency relation is symmetric, we only need a half-matrix, that we call the *concurrency matrix* of (N, m_0). To keep up with the notations used in CADP, we can also use symbol ? when the relation is undecided. We say that the matrix is *incomplete*. Due to the large dimension of the matrix, the output will often be displayed compressed using the RLE method (run-length encoding). We give an example of such matrices in Annex A.3.

The CADP tool integrates an option (`-concurrent-places`) to build the concurrency matrix from a net [INRIA, 2020]. However, with large nets (more than a few hundreds

places), the tool often cannot compute the full relation. We implemented a similar option in the SMPT model-checker by taking advantage of nets reductions. The idea is to compute the concurrency matrix on the reduced net and transpose the matrix to the initial net. Our approach can be applied to nets that are unsafe.

6.3 Concurrency Matrix Construction

We propose an algorithm to compute the *concurrency matrix* from a marked net (N, m_0). This algorithm relies on two main component: a *stepper*, that can compute the "successors" of a marking, and a model-checker, for finding potential pairs of concurrent places, or the absence thereof. To simplify the presentation, we assume that we have no dead places. This case can be handled separately or by a slight modification of our algorithm.

To check if two places are concurrent, we need to find a *witness*: a reachable marking m where the two places are marked. In the following, we should often refer to the concurrency relation, $\|$, as an undirected graph (P,R) where the vertices are places and there is an edge $(p,q) \in R$ when $p\|q$. While the BMC method is well-suited for finding witnesses, proving that two places are independent corresponds to proving an invariant. In this case, the PDR method is most suited. Hence BMC is for finding the 1 in the concurrency matrix and PDR are for the 0.

Definition 6.3 (c-stable)**.** *A c-stable set is a clique in the (graph-equivalent of) concurrency relation: a subset S of P is c-stable when $p\|q$ for all p,q in S.*

By definition the empty set, $\emptyset$, is c-stable. Likewise, the singleton $\{p\}$ is c-stable when p is not dead. More generally, any reachable marking m corresponds to a c-stable set, in the sense that the set $\{p \mid m(p) > 0\}$, of places marked in m is obviously a c-stable. Of course this is not necessarily a "maximal clique".

In the following, we use m both for a marking and its corresponding c-stable set of places. By extension, we should also use symbol m to range over c-stable sets.

Definition 6.4. *If $\mathcal{S}$ is a collection of c-stable sets, we say that $\mathcal{S} \models p\|q$ when there is m in $\mathcal{S}$ such that $\{p,q\} \subset m$. We say that a collection of c-stable sets entails another, denoted $\mathcal{S}_1 \subseteq \mathcal{S}_2$, when $\mathcal{S}_1 \models p\|q$ implies $\mathcal{S}_2 \models p\|q$.*

Our algorithm computes an under-approximation of the concurrency relation as a union of cliques (c-stable sets), starting with the initial marking. The main procedure, see Algorithm 4, computes a set of "non-redundant" witnesses $\mathcal{S}$. We say that $\mathcal{S}$ is non-redundant when, for all element m in $\mathcal{S}$, we have $\{m\} \not\subseteq \mathcal{S} \setminus \{m\}$. In the algorithm, we keep a stack of "tentative markings", $\mathcal{N}$, that we try to add to $\mathcal{S}$. A marking from $\mathcal{N}$ can be added to $\mathcal{S}$ if $\{m\} \not\subseteq \mathcal{S}$. Conversely, when we add a marking m to $\mathcal{S}$, we make sure that no c-stable set in $\mathcal{S}$ is entailed by m. This is the purpose of function add.

We have two methods for including new markings. First, we can find new candidates by firing transitions from markings in $\mathcal{N}$. This can be computed, inexpensively, using a

Algorithm 4: Computing concurrency relation $\mathcal{S}$

Input: Petri net (N, m_0)
Output: Concurrency relation $\mathcal{S}$
$\mathcal{S} \longleftarrow \{\}$;
$\mathcal{N} \longleftarrow \{m_0\}$;
while $\mathcal{N} \neq \emptyset$ **do**
  choose m in $\mathcal{N}$;
  $\mathcal{N} \longleftarrow \mathcal{N} \setminus \{m\}$;
  $\mathcal{S} \longleftarrow$ add($\mathcal{S}$, m);
  foreach $\{m' \mid m \to m'\}$ **do**
    if $\{m'\} \not\subseteq \mathcal{S} \cup \mathcal{N}$ **then**
      $\mathcal{N} \longleftarrow$ add($\mathcal{N}$, m');
  if $\mathcal{N} = \emptyset$ **then**
    $\mathcal{N} \longleftarrow$ check($\mathcal{S}$);
return $\mathcal{S}$;

stepper. This corresponds to the **foreach** clause in the algorithm. If no new candidates are found this way, we can use SMPT try to find a witness that is not entailed by $\mathcal{S}$ (using the BMC method) or to prove that all the concurrent places are covered by $\mathcal{S}$ (using PDR). Because the BMC may not terminate, we can run these two methods in parallel. In practice, we may exit from the algorithm at any time, for instance when computation takes too long. In this case we have an incomplete matrix, where all elements of the concurrency matrix are uncertain (?) of the independent elements.

Algorithm 5: Function add

Input: a concurrency relation $\mathcal{S}$ and a c-stable set m
Output: an updated concurrency relation $\mathcal{S}'$
$\mathcal{S}' \longleftarrow \{m\}$;
foreach $m' \in \mathcal{S}$ **do**
  if $m' \not\subseteq m$ **then**
    $\mathcal{S}' \longleftarrow \mathcal{S}' \cup \{m'\}$;
return $\mathcal{S}'$;

Call to Function add($\mathcal{S}$, m) is a simple "list comprehension", where we filter out the c-stable sets in $\mathcal{S}$ that are redundant with m. For function check($\mathcal{S}$), it is enough to define the property that we want to test over the set of reachable states. This formula is the conjunction of two constraints. We are looking for a marking m such that: (1) at least two places are marked; and (2) that is not "covered" by one of the set in $\mathcal{S}$.

For property (1), it is enough to have at least two valid clauses in the sequence $(m(p_1) > 0, \ldots, m(p_n) > 0)$. This can be directly expressed in SMT-LIB using oper-

ator $\geq_k (\phi_1, \ldots, \phi_n)$, which implements an "(at-least) k-out-of-n-out" test.

For property (2), we need to check that $m \not\subseteq m'$ for all m' in $\mathcal{S}$, where the test $m \not\subseteq m'$ can be encoded as the disjunction $\bigvee\{m(p) > 0 \mid m'(p) = 0\}$ (at least one place is marked in m but not in m'). We follow the notations used in Chapter 5 and denote this formula $\Omega_{\mathcal{S}}(\vec{x})$. Hence we have a possible candidate m to be added to $\mathcal{N}$, in Algorithm 4, if we find m reachable in (N, m_0) such that $\underline{m}(\vec{x}) \wedge \Omega_{\mathcal{S}}(\vec{x})$ is true.

$$\Omega_{\mathcal{S}}(\vec{x}) \quad = \quad \geq_2 (x_1 > 0, \ldots, x_n > 0) \wedge \bigwedge_{m' \in \mathcal{S}} \Big(\bigvee_{i \in 1..n} (x_i > 0 \wedge m'(p) = 0) \Big)$$

Algorithm 6: Function check

Input: a concurrency relation $\mathcal{S}$
Output: either a non-redundant witness $\{m\}$, or $\emptyset$ if none exist
parallel
  begin
    if *PDR proves* $\neg(\underline{m}(\vec{x}) \wedge \Omega_{\mathcal{S}}(\vec{x}))$ *is an invariant* **then**
      return $\emptyset$;
    else
      return *counter-example* $\{m\}$*;*
  begin
    if *BMC finds a counter-example m to* $\neg(\underline{m}(\vec{x}) \wedge \Omega_{\mathcal{S}}(\vec{x}))$ **then**
      return $\{m\}$;

A special case of formula Ω is when $\mathcal{S}$ contains only one set of the form $\{p_i, p_j\}$, with $i \neq j$. We simply denote this formula $\Omega_{i,j}$. This formula can be used to check if p_i, p_j are independent: we have $p_i \# p_j$ in (N, m_0) if and only if $\underline{m}(\vec{x}) \wedge \neg\Omega_{i,j}(\vec{x})$ is an invariant over the reachable markings of (N, m_0). We can use the PDR method defined in Chapter 5 to test whether the invariant holds. At the opposite, we can use BMC to try and find counter-examples. Each counter-example provides a "certificate" that $p_i \| p_j$. In this case, it may be more efficient to start from the markings in $\mathcal{S}$ (which are all reachable) rather than from m_0.

We can give some remarks about the complexity of our algorithm. The most complex part in the computation are in the call to function check. This is why we try to limit them as much as possible. Formula $\Omega_{\mathcal{S}}$ has n variables but its size is proportional to the size of $\mathcal{S}$, which may be quite big. Indeed we may have up-to $\frac{n(n-1)}{2}$) non-redundant markings, one for each possible pair of concurrent places. (This is also an upper-bound on the number of iterations in Algorithm 4). We could optimize our algorithm by limiting the size of $\mathcal{S}$. One possible solution will be to collect "maximal cliques" and not only c-stable sets that corresponds to reachable markings. This can easily be implemented. Apart from the complexity of the calls to functions add and check, the only other non-trivial computation is in the test $\{m'\} \not\subseteq \mathcal{S} \cup \mathcal{N}$ (in the conditional inside the **foreach**

loop). Our current implementation just look at all the concurrent pairs entailed by m and check whether they are also in $\mathcal{S}$. This test could be optimized with the help of a better data structure. Another solution would be to implement it using a SAT solver.

6.4 Change of Basis using Reduction Equations

To follow-up with the goal of my work, we consider the possible use of net reductions for computing concurrent places. We propose a method that can infer the concurrency matrix of a Petri net from the concurrency relation of a reduced version. We call this operation a *change of basis*.

Assume we have $(N_1, m_1) \rhd_E (N_2, m_2)$. (We consider the case where N_2 has less places than N_1.) To simplify our presentation, we assume that places prefixed by p, q, r, s are from the initial net N_1 (with set of places P_1), and that variables prefixed by a, b are additional variables, which are potentially places in N_2 (with set of places P_2).

We can deduce some of the concurrent places in N_1 by looking at the equations in E. For instance, if E entails both $a = p + q$ and $a \leq 1$ then it must be the case that p, q are independent (otherwise we could reach a marking where $p + q \geq 2$, contradicting our invariant). In the following, we give a sequence of properties that makes this remark more formal.

In the following, we use the notation $E \models F$ when $f_v(F) \subseteq f_v(E)$ and, for all possible solutions of system F, system E is satisfiable. We give a name to each of our properties: (RED) when it corresponds to equations generated by the reduction of redundant places and (AGG) when it comes from an "agglomeration" or a "concat" rule. We do not provide the proofs of these rules in this report.

(RED1) If $E \models p \geq k$, where k is a constant and $k \geq 1$, then we have $p \| s$ in (N_1, m_1) for all places $s \in P_1$ that are not dead.

(RED2) If $E \models p = q$ and $p \| r$ then $q \| r$ in (N_1, m_1). This is also true when p and r are the same places (remember that $p \| p$ means that p is not dead), meaning that p not dead implies q not dead..

(RED3) If $E \models p = q + r$ and $q \| q$ (place q is not dead) then $p \| p$.

(RED4) If $E \models p = q + r$ and $q \| s$ then $p \| s$ in (N_1, m_1).

(AGG1) If $E \models a = p_1 + \cdots + p_n$ and $a \| q$ in (N_2, m_2) (with q also a place in N_1) then we have $p_i \| q$ in (N_1, m_1) for all $i \in 1..n$.

(AGG2) If $E \models a = p_1 + \cdots + p_n, b = q_1 + \cdots + q_m$ and $a \| b$ in (N_2, m_2) then we have $p_i \| q_j$ in (N_1, m_1) for all $i, j \in 1..n \times 1..m$. Note that, by definition of the agglomeration rules used in our reductions, places p_i and q_j are necessary different from each others.

(AGG3) If $E \models a = k, a = p_1 + \cdots + p_n$ with $k \geq 2$ then the set $\{p_1, \ldots, p_n\}$ is a c-stable.

(AGG4) If $E \models a = k, a = p_1 + \cdots + p_n$ with $k \geq 1$ and s is a place in N_1 different from the p_i and s is not dead then we have $s \| p_1$ for all $i \in 1..n$.

(AGG5) If $E \models a = p_1 + \cdots + p_n, b = q_1 + \cdots + q_m, a = b$ and a is not dead in (N_2, m_2) then we have $p_i \| q_j$ in (N_1, m_1) for all $i, j \in 1..n \times 1..m$. We can make the same remark than for rule (AGG2).

6.4.1 Algorithm

Assume we have the concurrency matrix C of the reduced net N_2, which can be obtained by using 6.3. The idea of our method is to iterate over the system of equations to learn concurrent places, until all concurrent places are found. This *change of basis* can be seen as a fixed point on the reduction equations.

A second way is to build the relation between places as a *directed acyclic graph* (DAG), and to obtain the whole concurrency relation by exploring recursively this graph. This algorithm is not in the scope of this report since it is much more complicated than the first one, and the "spirit" of our *change of basis* is more understandable using a fixed point over the reduction equations.

6.4.2 Example

To illustrate our *change of basis* algorithm, we consider the case of model Kanban. This is one of the model used in our benchmarks of Chapter 7, which is fully-reducible. We provide the set of reduction equations obtained by running the Reduce tool in 6.1. By rewriting the equations we obtain the following system of equations:

$$\begin{cases} (1) & \texttt{5 = a11 = P1 + Pback1 + Pout1 + Pm1} \\ (2) & \texttt{5 = a10 = P2 + Pback2 + Pout2 + Pm2} \\ (3) & \texttt{5 = a9 = P4 + Pback4 + Pout4 + Pm4} \\ (4) & \texttt{a6 = a4 = Pback2 + Pout2 + Pm2} \\ (5) & \texttt{5 = a10 = a4 + P2} \end{cases}$$

To show up our *change of basis* we proceed to the first iteration of the algorithm, that is sufficient to obtain the full concurrency relation in that example.

From (1) by using the (AGG3) rule we obtain that {P1, Pback1, Pout1, Pm1} is a c-stable set. Similarly for (2) and (3), we know that {P2, Pback2, Pout2, Pm2} and {P4, Pback4, Pout4, Pm4} are also c-stable sets.

From (4) we have $a6 = a4$, and from (5) we can infer that a4 $= 5$. Therefore we can infer the c-stable set {Pback3, Pout3, Pm3} by using again the (AGG3) rule.

During the second iteration over the reduction equations, since a11 $= 5 \geq 1$, places in {P1, Pback1, Pout1, Pm1} are concurrent to all the non dead places in net Kanban. Similarly with a10, a9 and a6 by using the (AGG4) rule.

```
# R |- P3 = P2
# A |- a1 = Pout1 + Pm1
# A |- a2 = Pback1 + a1
# A |- a3 = Pout2 + Pm2
# A |- a4 = Pback2 + a3
# A |- a5 = Pout3 + Pm3
# A |- a6 = Pback3 + a5
# A |- a7 = Pout4 + Pm4
# A |- a8 = Pback4 + a7
# A |- a9 = a8 + P4
# R |- a9 = 5
# R |- a6 = a4
# A |- a10 = a4 + P2
# R |- a10 = 5
# A |- a11 = a2 + P1
# R |- a11 = 5
```

Figure 6.1: Output of tool Reduce on the Kanban instance for $N = 5$

Since the agglomerations `a6`, `a9`, `a10` and `a11` cover all the places in the initial net, we have that all places are pairwise concurrent.

In our example, we consider an "instance" of net Kanban where the initial marking of places P1, P2, P3 and P4 is 5: this is the instance of Kanban for parameter $N = 5$. All instances with $N \geq 2$ are not safe (our approach is still valid in this case) and, for all such instances, the concurrency relation is total.

— 7 —
Experimental Results

In this chapter, we report on some experimental results obtained with SMPT (for Satisfiability Modulo P/T Nets), our prototype implementation of a SMT-based model-checker with reduction equations, that implements the ideas developed in Chapter 5. The tool is open source, under the GPLv3 licence, and is freely available on GitHub (https://github.com/nicolasAmat/SMPT/). We have used the extensive database of models and formulas provided by the Model Checking Contest (MCC) [Amparore et al., 2019, Hillah and Kordon, 2017] to experiment with our approach.

SMPT serves as a front-end to generic SMT solvers, such as z3 [de Moura and Bjørner, 2008, Bjørner, 2020]. The tool can output sets of constraints using the SMT-LIB format [Barrett et al., 2017] and pipe them to a z3 process through the standard input. We have implemented our tool with the goal to be as interoperable as possible, but we have not conducted experiments with other solvers yet. SMPT takes as inputs Petri nets defined using the `.net` format of the TINA toolbox. For formulas, we accept properties defined with the XML syntax used in the MCC competition. The tool does not compute net reductions directly but relies on the tool Reduce, that we described at the end of Chapter 4.

Our benchmark suite is built from a collection of 102 models used in the MCC competition. Most of the models are parametrized, and therefore there can be several different *instances* for the same model. There are about 1 000 different instances of Petri nets whose size vary widely, from 9 to 50 000 places, and from 7 to 200 000 transitions. Most nets are ordinary, but a significant number of them use weighted arcs. Overall, the collection provides a large number of examples with various structural and behavioral characteristics, covering a large variety of use cases.

Since our approach relies on the use of net reductions, it is natural to wonder if reductions occur in practice. To answer this question, we computed the reduction ratio (r), obtained using Reduce, as a quotient between the number of places before (p_{init}) and after (p_{red}) reduction: $r = (p_{\text{init}} - p_{\text{red}})/p_{\text{init}}$. We display the results for the whole collection of instances in Fig. 7.1, sorted in descending order. A ratio of 100% ($r = 1$) means that the net is *fully reduced*; the resulting net has only one (empty) marking. We see that there is a surprisingly high number of models that are totally reducible with our

approach (about 20% of the total number), with approximately half of the instances that can be reduced by a ratio of 30% or more.

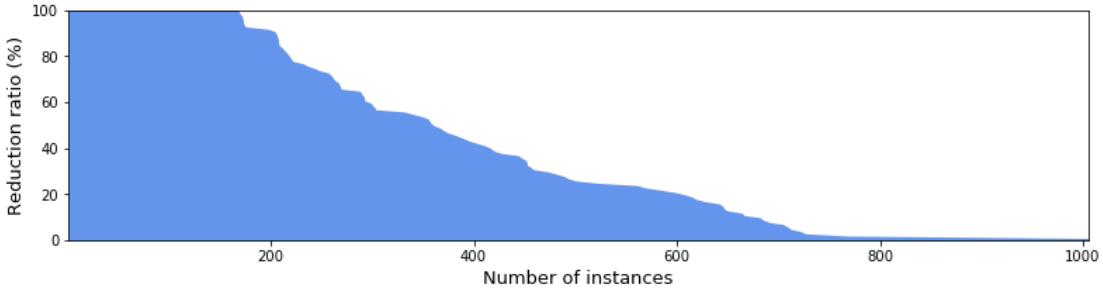


Figure 7.1: Distribution of reduction ratios over the instances in the MCC

For each edition of the MCC, a collection of about 30 random reachability properties are generated for each instance. We evaluated the performance of SMPT using the formulas of the MCC2020, on a selection of 234 Petri nets taken from instances with a reduction ratio greater than 30%. (To avoid any bias introduced by models with a large number of instances, we selected at most 5 instances from each model.) We ran SMPT on each of those instances, for all the formulas, using both the BMC and PDR methods in parallel. In each case, we check the formulas with and without the help of reductions, using a timeout of 120 seconds for each property. That adds up to a total of 7655 *test cases* (and therefore about 15000 queries for the solver) which required the equivalent of 400 hours of CPU time. We report our results on the table below. Out of the 7655 test cases, we were able to compute 4933 results using reductions and only 1922 without reductions. For each formula, we compared our results with the ones provided by an *oracle* [Thierry-Mieg, 2020a], which gives the correct answer (as computed by a majority of tools, using different techniques, during the MCC competition). Our reliability on the benchmark is at 100% (all answers are correct).

Reduction Ratio (r)		# Test Cases	Results (BMC/PDR)			
			With reductions		Without	
All	$r \in [0.3, 1]$	7655	4933	(4697/236)	1922	(1761/161)
Good	$r \in [0.3, 0.7]$	3384	1485	(1392/93)	697	(631/66)
High	$r \in [0.6, 1[$	2143	1092	(956/136)	344	(303/41)
Full	$r = 1$	2558	2557	(n.a./n.a.)	990	(n.a./n.a.)

We give the number of computed results for three different categories of test cases: *High* contains only the fully reducible instances (the best possible case with our approach); while *Good*/*High* correspond to instances with a moderate/high level of reduction. Attention was paid to the fact that these samples are of approximately the same size. We also have a general category, *All*, for the complete set of benchmarks. We observe that

we are able to compute almost 2.6 times more results when we use reductions than without. This gain is greater on the *High* (3.2) than on the *Good* (2.2) instances. But the fact that it is quite high even for *Good* instances indicates that our approach can benefit from all the reductions we can find in a model (and that our results are not skewed by the large number of fully reducible instances).

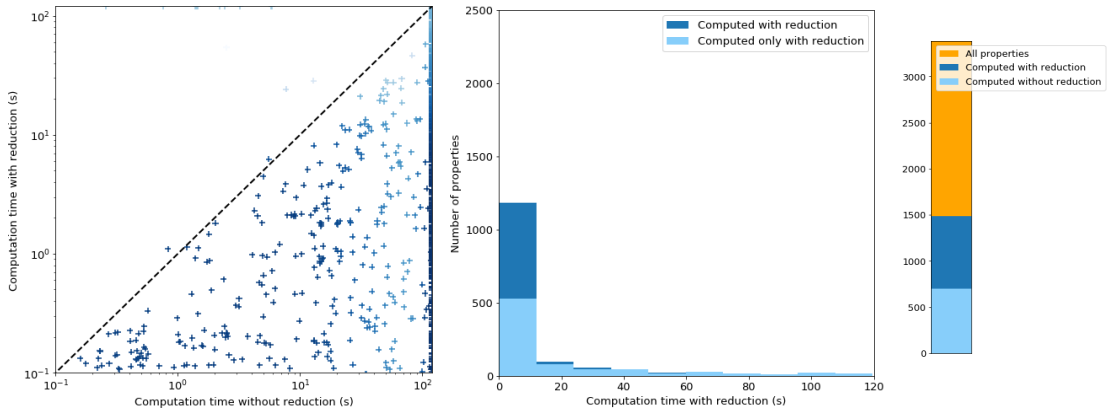
We also report the number of cases solved using BMC/PDR. This value is not relevant for fully reducible nets since, in this case, we can check the result directly on the initial marking of the reduced net (with a single call to the solver). We observe that the contribution of PDR is poor. This can be explained by several factors. Most notably, we restricted our implementation of PDR to monotonic formulas (which represents 30% of all properties). Among these, PDR is useful only when we have an invariant that is true (meaning BMC will certainly not terminate). On the other hand, PDR is able to give answer on the most complex cases. Indeed, it is much more difficult to prove an invariant than to find a counter-example (and we have other means to try and find counter-examples, like simulation for instance). This is why we intend to improve the performances and the "expressiveness" of our PDR implementation.

We also compare the computation time, with or without reductions, for each test case. These results do not take into account the time spent for reducing each instance. This time is negligible when compared to each test, usually in the order of 1*s*, and we only reduce the net once before checking the 30 properties. Figure 7.2 gives three visualizations (**Left**) for the computation time "with" (*y*-axis) and "without" reductions (*x*-axis), for the *Good*, *High* and *Full* categories of instances. To avoid overplotting, we removed all the "trivial" properties (570 in total), that can be computed with and without reduction in less than 10*ms*, and we used a logarithmic scale. These "trivial" properties could be computed really quickly for various reasons: some instances have a very small sate-space (see `ResAllocation-R002C002` instance), some properties have a counter-example on the initial marking (see `Peterson-PT-3-ReachabilityCardinality-02` property), or some models have a reachable state-space by firing small transition sequences (see `NighborGrid` instances). We observe that almost all the data points are bellow the diagonal (meaning reductions accelerate the computations), while most of the rest are on the right border (computation without reductions timeout). On the more than 7 000 test cases we have, there are only 11 cases where we timeout with reductions but compute a result without. These exceptions can be explained by border cases where the order in which transitions are processed has a sizeable impact.

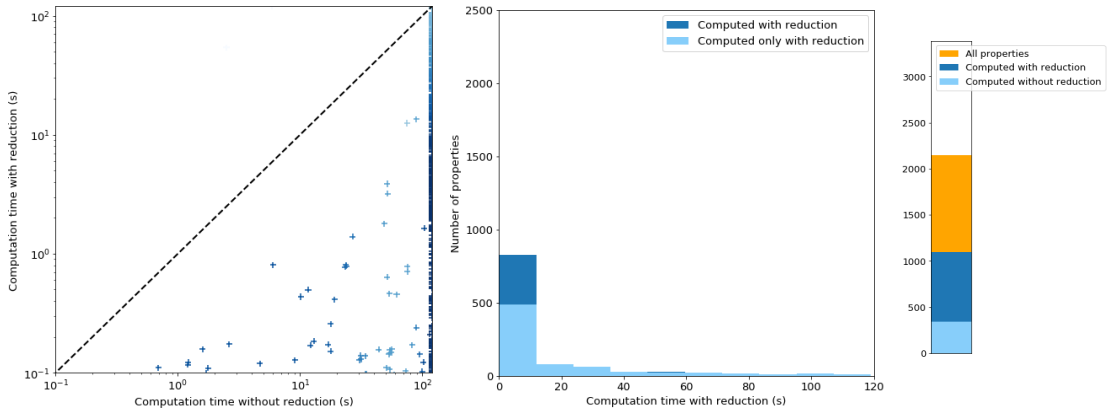
Another interesting point is the ratio of properties only computed with reduction. Figure 7.2 provides the number of computed properties "with" and "only with" reduction over computation time (**Center**) for the three categories of instances. From this, we can extract two main information. First, we observe that reductions allow us to compute twice as many properties using reductions for the three categories, even if the performances are quite better for the *Good* category. Furthermore, genrally properties computed in more than 10*ms* are properties unsolvable so far without reductions.

On Figure 7.2 (**Right**) we give the number of computed properties "with" and "without"

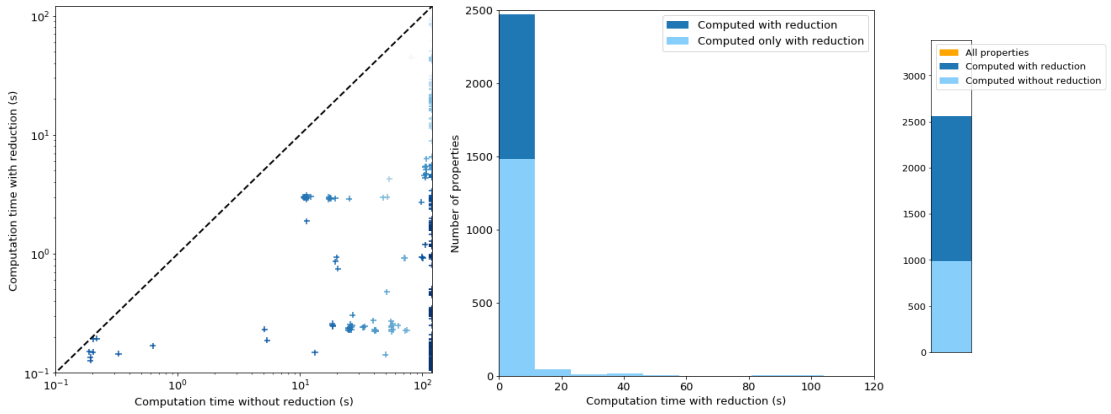
reductions over the total number of test cases. It provides an overview of the benefits of using our *polyhedral approach*. With reductions we are able to compute more than twice as many properties, on a huge amount of instances, using BMC and PDR methods. In the special case of *full reducible* nets, we can perform almost all the properties (only one cannot be computed in less than 120s).



(a) $r \in [0.3, 7]$



(b) $r \in [0.6, 1[$



(c) $r = 1$

Figure 7.2: **Left**: Comparing computation time, "with" (y-axis) and "without" (y-axis) reduction; **Center**: Number of computed properties "with" (dark blue) and "only with" (clear blue) reduction over computation time; **Right**: Number of computed properties "with" (dark blue) and "without" (clear blue) reduction compared to all properties examinated (orange).

— 8 —
Conclusion and Perspectives

This research project opened up new horizons for symbolic model checking. The results at the Model Checking Contest of the tedd tool [Amparore et al., 2019], which combines decision diagrams and reduction equations [Berthomieu et al., 2018, Berthomieu et al., 2019], were promising for the Vertics team. The SMPT model-checker [Amat, 2020] that we developed during this project, combining reduction equations and SMT-based verification methods, shows that such new methods for reachability verification are promising too.

The development of our model-checker led us to explore new algorithms. We adapted the well known BMC and PDR algorithms to handle generalized Petri nets and take advantage of our reductions. These algorithms are intended to be improved, such as PDR, for which we want to extend it to non-montonic properties. But we want to go even further by working on the state equations, adding invariants, or developing new methods combining SMT-based methods and Binary Decision Diagrams. A perfect testing place for this work would be the *Reachability* category of the Model Checking Contest.

The key element of this study is reductions. We explore some theoretical aspects to prove the correctness of these reduction rules. We formalized a new equivalence relation, called *E-abstraction equivalence*, which introduce a new notion of equivalence between Petri nets modulo a set of reduction equations. This work could lead to the definition of more "aggressive" reduction rules, preserving only some specific properties (such as deadlock). We are also interested in implementing a "reduction rule" prover, with the aim of proving automatically new reduction rules.

Another contribution of this project was to apply my tool on the *Concurrent Places Problem*, which is useful for the decomposition into sub-nets, called NUPNs, introduced by the Convecs team at INRIA [Garavel, 2019a]. We proposed a new approach, which reconstructs the set of concurrent places of the initial net from the ones in the reduced net.

To conclude, I believe that this new "polyhedral model checking" approach, that mixes symbolic model checking and convex analysis, could lead to some very interesting new developments. For instance, this approach seems to be interesting for model counting

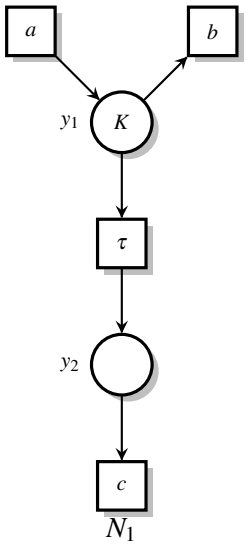
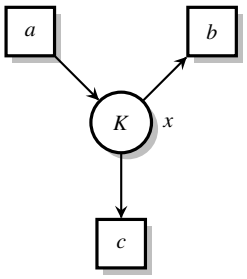
[Berthomieu et al., 2019]. This is one problem, among others, that I want to explore in the future.

— A —
Appendix

A.1 Proof of Correctness for a Reduction rule

We give an example of correctness proof for rule (CONCAT), below, meaning that we prove that $(N_1, m_1) \rhd_E (N_2, m_2)$ for the nets defined in the rule below, with E a system containing the single equation $x = y_1 + y_2$, and with the constraints on the initial marking displayed in the nets. Namely, we assume that $m_1(y_1) = m_2(x) = K \geq 0$ and that $m_1(y_2) = 0$. To ease the presentation, we should use τ, a, b, c as the name of the transitions, and not only as labels.

Note that, in this rule, nets N_1 and N_2 are not bounded, since transition a can always be fired to increase the marking of places y_1 and x.

 N_1	 N_2
Condition: $\emptyset$	**Equation**: $x = y_1 + y_2$

We start by proving condition (A1). By construction, we have $m_1 \uplus m_2 \models E$ and E is solvable for N_1, N_2. Indeed, equation $x = y_1 + y_2$ is always satisfiable when we fix either the values of variables y_1, y_2, or the value of x.

We now prove condition (A2) for the relation $(N_1, m_1) \sqsupseteq_E (N_2, m_2)$. Assume that $(N_1, m_1) \stackrel{\sigma_1}{\Longrightarrow} (N_1, m_1')$ and that $m_1' \uplus m_2' \models E$. By definition of E, when m_1' is fixed, there is a unique solution for m_2' such that $m_1' \uplus m_2' \models E$; which is $m_2'(x) = m_1'(y_1) + m_1'(y_2)$. We prove that there is a firing sequence σ_2 such that $(N_2, m_2) \stackrel{\sigma_2}{\Longrightarrow} (N_2, m_2')$ and $l_1(\sigma_1) = l_2(\sigma_2)$. We proceed by induction on the length of σ_1.

(Base Case): if $\sigma_1 = \varepsilon$ then we can choose $\sigma_2 = \varepsilon$ and $m_2' = m_2$.

(Induction Case): We have $\sigma_1 = \varsigma_1 t$ where t is one of the transition τ, a, b or c. Therefore there is a marking m_1'' over N_1 such that $(N_1, m_1) \stackrel{\varsigma_1}{\Longrightarrow} (N_1, m_1'') \stackrel{t}{\Rightarrow} (N_1, m_1')$. By induction hypothesis, there is a firing sequence ς_2 and a marking m_2'' over N_2 such that $m_2''(x) = m_1''(y_1) + m_1''(y_2)$ and $(N_2, m_2) \stackrel{\varsigma_2}{\Longrightarrow} (N_2, m_2'')$. The property follows by a case analysis on the possible choice of t.

case $t = \tau$: in this case the overall number of token is left unchanged and we can simply choose $\sigma_2 = \varsigma_2$ and $m_2' = m_2''$.

case $t = a$: transition a can always be fired, we can choose $\sigma_2 = \varsigma_2 a$ and m_2' the unique marking such that $m_2'' \stackrel{a}{\rightarrow} m_2'$.

case $t = b$: since b can be fired from m_1'' it must be the case that $m_1''(y_1) \geq 1$. Hence $m_2''(x) \geq 1$ and b can also fire from m_2'' in N_2. We can choose $\sigma_2 = \varsigma_2 b$ and m_2' the unique marking such that $m_2'' \stackrel{b}{\rightarrow} m_2'$. The proof is similar in the case where $t = c$.

We are left to prove condition (A2) for the relation $(N_2, m_2) \sqsupseteq_E (N_1, m_1)$. Assume we have $(N_2, m_2) \stackrel{\sigma_2}{\Longrightarrow} (N_2, m_2')$. We prove that there is a firing sequence σ_1 such that $(N_1, m_1) \stackrel{\sigma_1}{\Longrightarrow} (N_1, m_1')$ and $l_1(\sigma_1) = l_2(\sigma_2)$, where m_1' is the marking defined by $m_1'(y_1) = m_2'(x)$ and $m_1'(y_2) = 0$ (all the tokens are in y_1). Like in the previous case, we proceed by induction on the length of the firing sequence and by a case analysis on the last transition fired in N_2.

(Base Case): if $\sigma_2 = \varepsilon$ then we can choose $\sigma_1 = \varepsilon$ and $m_2' = m_2$.

(Induction Case): We have $\sigma_2 = \varsigma_2 t$ where t is one of the transition a, b or c. Therefore there is a marking m_2'' over N_2 such that $(N_2, m_2) \stackrel{\varsigma_2}{\Longrightarrow} (N_2, m_2'') \stackrel{t}{\Rightarrow} (N_2, m_2')$. By induction hypothesis, there is a firing sequence ς_1 and a marking m_1'' such that $m_1''(y_1) = m_2''(x)$ and $(N_1, m_1) \stackrel{\varsigma_1}{\Longrightarrow} (N_1, m_1'')$.

case $t = a$: transition a can always be fired, we can choose $\sigma_1 = \varsigma_1 a$ and m_1' the unique marking such that $m_1'' \stackrel{a}{\rightarrow} m_1'$.

case $t = b$: since b can be fired from m_2'' it must be the case that $m_2''(x) \geq 1$. Hence $m_1''(y_1) \geq 1$ and b can also fire from m_1'' in N_2. We can choose $\sigma_1 = \varsigma_1\, b$ and m_2' the unique marking such that $m_1'' \xrightarrow{b} m_1'$.

case $t = c$: since c can be fired from m_2'' it must be the case that $m_2''(x) \geq 1$. Hence $m_1''(y_1) \geq 1$ and it is possible to fire the sequence $\tau\, c$ from m_1''. Hence we have $\sigma_1 = \varsigma_1\, \tau\, c$.

Condition (A2) follows from the fact that, when marking m_2' is fixed, then all solutions m_1' to the constraint $m_1' \uplus m_2' \models E$ can be reached by firing a sequence of τ transitions from $m_1^{0'}$ such that $m_1^{0'}(y_1) = m_2'(x)$ and $m_1^{0'}(y_2) = 0$.

A.2 SMPT Usage

```
usage: smpt [-h] [--version] [-v] [--debug]
[--xml PATH_PROPERTIES | --deadlock | --quasi-liveness QUASI_LIVE_TRANSITIONS
| --reachability REACHABLE_PLACES]
[--auto-reduce | --reduced PATH_PTNET_REDUCED]
[--save-reduced-net]
[--no-bmc | --no-ic3 | --auto-enumerative | --enumerative PATH_MARKINGS]
[--timeout TIMEOUT] [--skip-non-monotonic] [--display-method]
[--display-model] [--display-time] [--display-reduction-ratio]
ptnet

SMPT: Satisfiability Modulo Petri Net

positional arguments:
ptnet                                 path to Petri Net (.net format)

optional arguments:
-h, --help                            show this help message and exit
--version                             show the version number and exit
-v, --verbose                         increase output verbosity
--debug                               print the SMT-LIB input/ouput
--xml PATH_PROPERTIES                 use XML format for properties
--deadlock                            deadlock analysis
--quasi-liveness QUASI_LIVE_TRANSITIONS liveness analysis (comma separated list of transition names)
--reachability REACHABLE_PLACES       reachibility analysis (comma separated list of place names)
--auto-reduce                         reduce automatically the Petri Net (using ‘reduce‘)
--reduced PATH_PTNET_REDUCED          path to reduced Petri Net (.net format)
--save-reduced-net                    save the reduced net
--no-bmc                              disable BMC method
--no-ic3                              disable IC3 method
--auto-enumerative                    enumerate automatically the states (using ‘tina‘)
--enumerative PATH_MARKINGS           path to the state-space (.aut format)
--timeout TIMEOUT                     a limit on execution time
--skip-non-monotonic                  skip non-monotonic properties
--display-method                      display the method returning the result
--display-model                       display a counterexample if there is one
--display-time                        display execution times
--display-reduction-ratio             display the reduction ratio
```

A.3 AirplaneLD Concurrency Matrix

stp4	1
SpeedPossibleVal_1	11
SpeedPossibleVal_2	111
SpeedPossibleVal_3	1(4)
SpeedPossibleVal_4	1(5)
SpeedPossibleVal_5	1(6)
SpeedPossibleVal_6	1(7)
SpeedPossibleVal_7	1(8)
SpeedPossibleVal_8	1(9)
SpeedPossibleVal_9	1(10)
SpeedPossibleVal_10	1(11)
Speed_Left_Wheel_1	01(11)
Speed_Left_Wheel_2	01(10)01
Speed_Left_Wheel_3	01(10)001
Speed_Left_Wheel_4	01(10)0001
Speed_Left_Wheel_5	01(10)0(4)1
Speed_Left_Wheel_6	01(10)0(5)1
Speed_Left_Wheel_7	01(10)0(6)1
Speed_Left_Wheel_8	01(10)0(7)1
Speed_Left_Wheel_9	01(10)0(8)1
Speed_Left_Wheel_10	01(10)0(9)1
stp5	1(22)
Speed_Right_Wheel_1	1(21)01
Speed_Right_Wheel_2	1(21)001
Speed_Right_Wheel_3	1(21)0001
Speed_Right_Wheel_4	1(21)0(4)1
Speed_Right_Wheel_5	1(21)0(5)1
Speed_Right_Wheel_6	1(21)0(6)1
Speed_Right_Wheel_7	1(21)0(7)1
Speed_Right_Wheel_8	1(21)0(8)1
Speed_Right_Wheel_9	1(21)0(9)1
Speed_Right_Wheel_10	1(21)0(10)1
stp3	1(33)
AltitudePossibleVal_1	1(34)
AltitudePossibleVal_2	1(35)
AltitudePossibleVal_3	1(36)
AltitudePossibleVal_4	1(37)
AltitudePossibleVal_5	1(38)
AltitudePossibleVal_6	1(39)
AltitudePossibleVal_7	1(40)
AltitudePossibleVal_8	1(41)
AltitudePossibleVal_9	1(42)
AltitudePossibleVal_10	1(43)
AltitudePossibleVal_11	1(44)
AltitudePossibleVal_12	1(45)
AltitudePossibleVal_13	1(46)
AltitudePossibleVal_14	1(47)
AltitudePossibleVal_15	1(48)
AltitudePossibleVal_16	1(49)
AltitudePossibleVal_17	1(50)
AltitudePossibleVal_18	1(51)
AltitudePossibleVal_19	1(52)
AltitudePossibleVal_20	1(53)
TheAltitude_1	1(32)01(21)
TheAltitude_2	1(32)01(20)01
TheAltitude_3	1(32)01(20)001
TheAltitude_4	1(32)01(20)0001
TheAltitude_5	1(32)01(20)0(4)1
TheAltitude_6	1(32)01(20)0(5)1
TheAltitude_7	1(32)01(20)0(6)1
TheAltitude_8	1(32)01(20)0(7)1
TheAltitude_9	1(32)01(20)0(8)1
TheAltitude_10	1(32)01(20)0(9)1
TheAltitude_11	1(32)01(20)0(10)1
TheAltitude_12	1(32)01(20)0(11)1
TheAltitude_13	1(32)01(20)0(12)1
TheAltitude_14	1(32)01(20)0(13)1
TheAltitude_15	1(32)01(20)0(14)1
TheAltitude_16	1(32)01(20)0(15)1
TheAltitude_17	1(32)01(20)0(16)1
TheAltitude_18	1(32)01(20)0(17)1
TheAltitude_19	1(32)01(20)0(18)1
TheAltitude_20	1(32)01(20)0(19)1
stp2	1(74)
WeightPossibleVal_on	1(75)
WeightPossibleVal_off	1(76)
Weight_Right_Wheel_on	1(73)0111
Weight_Right_Wheel_off	1(73)01101
stp1	1(79)
Weight_Left_Wheel_on	1(78)01
Weight_Left_Wheel_off	1(78)001
P5	01(10)0(10)1(11)01(20)0(21)110(5)1
P6	1(78)0(4)1
Plane_On_Ground_Signal_no_T	1(78)0(4)11
Plane_On_Ground_Signal_no_F	01(10)0(22)1(20)0(21)110(6)101
P4	1(32)01(20)0(21)110(9)1
P3	1(73)0110(10)1
P2	1(78)0(9)1
P1	1(81)0(7)1

Bibliography

[Amat, 2020] Amat, N. (2020). SMPT. https://github.com/nicolasAmat/SMPT/.

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

[Armando et al., 2006] Armando, A., Mantovani, J., and Platania, L. (2006). Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. In Valmari, A., editor, *Model Checking Software*, Lecture Notes in Computer Science, pages 146–162, Berlin, Heidelberg. Springer.

[Baier and Katoen, 2008] Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

[Barrett et al., 2017] Barrett, C., Fontaine, P., and Tinelli, C. (2017). The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa. Available at www.SMT-LIB.org.

[Bérard et al., 2001] Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P., and Mckenzie, P. (2001). SMV - Symbolic Model Checking. In Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P., and McKenzie, P., editors, *Systems and Software Verification: Model-Checking Techniques and Tools*, pages 131–138. Springer, Berlin, Heidelberg.

[Berthelot, 1987] Berthelot, G. (1987). Transformations and Decompositions of Nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties*, Lecture Notes in Computer Science, pages 359–376, Berlin, Heidelberg. Springer.

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

[Berthomieu et al., 2019] Berthomieu, B., Le Botlan, D., and Dal Zilio, S. (2019). Counting Petri net markings from reduction equations. *International Journal on Software Tools for Technology Transfer*. Publisher: Springer Verlag.

[Besson et al., 1999] Besson, F., Jensen, T., and Talpin, J.-P. (1999). Polyhedral analysis for synchronous languages. In *Static Analysis Symposium (SAS)*, volume 1694 of *LNCS*, pages 51–68. Springer.

[Biere et al., 1999] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). Symbolic Model Checking without BDDs. In Cleaveland, W. R., editor, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 193–207, Berlin, Heidelberg. Springer.

[Bjørner, 2020] Bjørner, N. (2020). The z3 theorem prover. https://github.com/Z3Prover/z3/.

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

[Bouvier et al., 2020] Bouvier, P., Garavel, H., and Ponce-de León, H. (2020). Automatic decomposition of petri nets into automata networks–a synthetic account. submitted.

[Bradley, 2011] Bradley, A. R. (2011). SAT-Based Model Checking without Unrolling. In Jhala, R. and Schmidt, D., editors, *Verification, Model Checking, and Abstract Interpretation*, volume 6538, pages 70–87. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[Bradley, 2012] Bradley, A. R. (2012). Understanding IC3. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Cimatti, A., and Sebastiani, R., editors, *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[Bradley and Manna, 2007] Bradley, A. R. and Manna, Z. (2007). Checking safety by inductive generalization of counterexamples to induction. In *Proceedings of the Formal Methods in Computer Aided Design*, FMCAD '07, pages 173–180, USA. IEEE Computer Society.

[Burch et al., 1992] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L.-J. (1992). Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170.

[Busi, 1998] Busi, N. (1998). Petri nets with inhibitor and read arcs: Semantics, analysis and applications to process calculi. *BULLETIN-EUROPEAN ASSOCIATION FOR THEORETICAL COMPUTER SCIENCE*, 65:237–239.

[Cimatti et al., 2016] Cimatti, A., Griggio, A., Mover, S., and Tonetta, S. (2016). Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design*, 49(3):190–218.

[Clarke et al., 2001] Clarke, E., Biere, A., Raimi, R., and Zhu, Y. (2001). Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

[Conchon et al., 2012] Conchon, S., Goel, A., Krstic, S., Mebsout, A., and Zaïdi, F. (2012). Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems. In *Computer Aided Verification (CAV)*, LNCS, pages 718–724. Springer.

[Cousot and Halbwachs, 1978] Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96.

[de Moura and Bjørner, 2008] de Moura, L. and Bjørner, N. (2008). Z3: An Efficient SMT Solver. In Ramakrishnan, C. R. and Rehof, J., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 337–340, Berlin, Heidelberg. Springer.

[de Moura et al., 2003] de Moura, L., Rueß, H., and Sorea, M. (2003). Bounded Model Checking and Induction: From Refutation to Verification. In Hunt, W. A. and Somenzi, F., editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 14–26, Berlin, Heidelberg. Springer.

[Diaz, 2009] Diaz, M. (2009). *Petri Nets: Fundamental Models, Verification and Applications*. Wiley-ISTE.

[Esparza et al., 2014] Esparza, J., Ledesma-Garza, R., Majumdar, R., Meyer, P., and Niksic, F. (2014). An SMT-Based Approach to Coverability Analysis. In *Computer Aided Verification (CAV)*, LNCS, pages 603–619.

[Feautrier, 1996] Feautrier, P. (1996). Automatic parallelization in the polytope model. In *The Data Parallel Programming Model*, volume 1132 of *LNCS*, pages 79–103. Springer.

[Feautrier and Lengauer, 2011] Feautrier, P. and Lengauer, C. (2011). Polyhedron model. *Encyclopedia of parallel computing*, 3:1581–1591.

[Finkel, 1991] Finkel, A. (1991). The minimal coverability graph for petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 210–243. Springer.

[Garavel, 2012] Garavel, H. (2012). Three decades of success stories in formal methods. In *International Conference on Formal Methods for Industrial Critical Systems (FMICS)*.

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

[Garavel, 2019b] Garavel, H. (2019b). Proposal for adding useful features to petri-net model checkers. message to the participants of the Model-Checking Contest.

[Gurfinkel et al., 2016] Gurfinkel, A., Shoham, S., and Meshman, Y. (2016). SMT-based verification of parameterized systems. In *International Symposium on Foundations of Software Engineering*, pages 338–348. ACM.

[Hillah and Kordon, 2017] Hillah, L. and Kordon, F. (2017). 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.

[Hujsa et al., 2020] Hujsa, T., Berthomieu, B., Dal Zilio, S., and Le Botlan, D. (2020). Checking marking reachability with the state equation in petri net subclasses. *arXiv preprint arXiv:2006.05600*.

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

[Kloos et al., 2013] Kloos, J., Majumdar, R., Niksic, F., and Piskac, R. (2013). Incremental, inductive coverability. In *Computer Aided Verification (CAV)*, pages 158–173. Springer.

[Kordon et al., 2019] Kordon, F., Leuschel, M., van de Pol, J., and Thierry-Mieg, Y. (2019). Software Architecture of Modern Model Checkers. In Steffen, B. and Woeginger, G., editors, *Computing and Software Science: State of the Art and Perspectives*, Lecture Notes in Computer Science, pages 393–419. Springer International Publishing, Cham.

[Kroening and Strichman, 2008] Kroening, D. and Strichman, O. (2008). *Decision Procedures: An Algorithmic Point of View*. Springer Science & Business Media.

[LAAS-CNRS, 2020] LAAS-CNRS (2020). Tina Toolbox. http://projects.laas.fr/tina//home.php.

[LIP6, 2020] LIP6 (2020). Model Checking Contest property Language (Manual). *Petri Nets*, page 11.

[Liu and He, 2015] Liu, S. H. and He, X. (2015). PIPE+Verifier - A Tool for Analyzing High Level Petri Nets. In *ICSE 2015*.

[McMillan, 2003] McMillan, K. L. (2003). Interpolation and SAT-Based Model Checking. In Hunt, W. A. and Somenzi, F., editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 1–13, Berlin, Heidelberg. Springer.

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

[Silva et al., 1998] Silva, M., Terue, E., and Colom, J. M. (1998). Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *Advanced Course on Petri Nets*, pages 309–373. Springer.

[Stahl, 2011] Stahl, C. (2011). Decomposing Petri net State Spaces. In *18th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2011), Hagen, Germany*.

[Thierry-Mieg, 2020a] Thierry-Mieg, Y. (2020a). Oracle for the MCC 2020 edition. Available at https://github.com/yanntm/pnmcc-models-2020.

[Thierry-Mieg, 2020b] Thierry-Mieg, Y. (2020b). Structural Reductions Revisited.

[Van Glabbeek et al., 2012] Van Glabbeek, R., Goltz, U., and Schicke-Uffmann, J.-W. (2012). On distributability of petri nets. In *International Conference on Foundations of Software Science and Computational Structures*, pages 331–345. Springer.