# An Integrated Environment for Petri Net Slicing*

Marisa Llorens, Javier Oliver, Josep Silva, and Salvador Tamarit

Universitat Politècnica de València, Camí de Vera S/N, E-46022 València, Spain
{mllorens,fjoliver,jsilva,stamarit}@dsic.upv.es

**Abstract.** Petri net slicing is a technique to automatically isolate the part of a marked Petri net that influences or is influenced by a given set of places. There exist different algorithms for Petri net slicing with different objectives. Nevertheless, they have never been evaluated or compared from a practical point of view. In fact, because there does not exist a public implementation of some of them, their performance and scalability have remained unknown. In this paper we present three tools for the analysis and transformation of Petri nets. The three tools are complementary, and they allow us to extract from a Petri net a set of slices that preserve a given set of properties (e.g., boundedness, reversibility, etc.). For this, they include the first public, free, and open-source implementation of the most important algorithms for Petri net slicing, including a new algorithm that reduces the size of the slices. Our implementation of the algorithms allowed us to compare all of them and to measure and report for the first time about their individual performance.

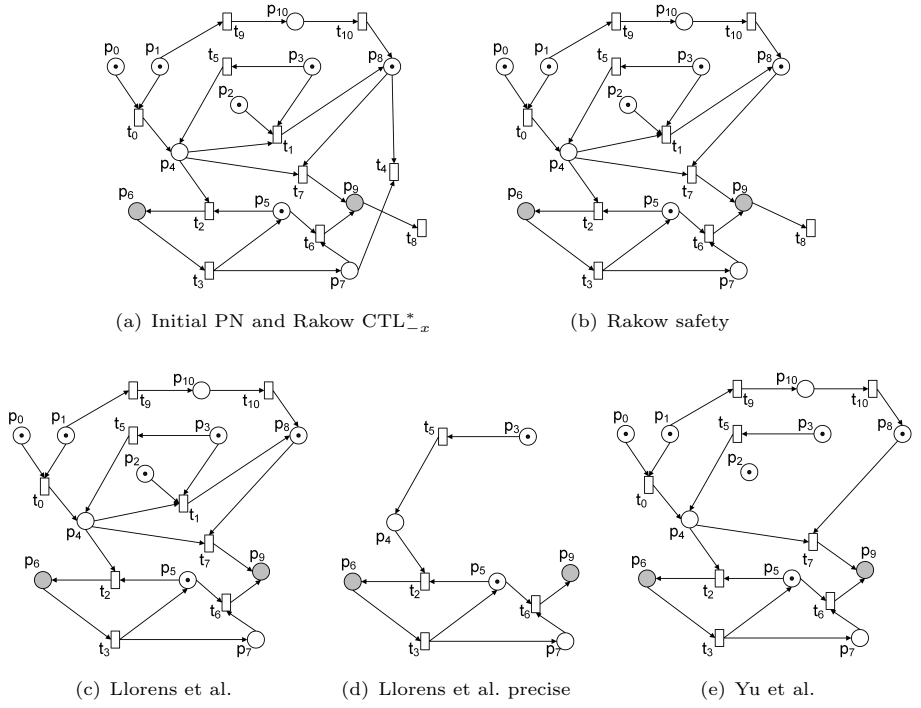
## 1 Introduction and Objectives

*Petri net slicing* [2] is a technique to extract the part of a Petri net that is relevant with respect to a so-called *slicing criterion*. In general, the slicing criterion is a set of places of a marked Petri net, and the slice produced is a new Petri net that contains the subset of the original Petri net that can contribute tokens to the slicing criterion. The main motivation for Petri net slicing is lessening the state space explosion problem for model checking [2–4, 6, 7]. As a reduction technique, it is supposed to preserve some properties, so the sliced net (which ideally has a smaller state space) can be model checked instead of the original net. Additionally, Petri net slicing is useful to enhance reachability analyses [4, 6, 3], for Petri net comprehension [5], and for debugging [5, 7], among others.

*Example 1.* Consider the Petri net of Figure 1(a), where the slicing criterion is the set of places in grey color (places $\{p_6, p_9\}$). The Petri nets shown in Figures

* This work has been partially supported by MINECO/AEI/FEDER (EU) under grants TIN2013-44742-C4-1-R and TIN2016-76843-C4-1-R, and by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic). Salvador Tamarit was partially supported by the *Conselleria de Educación, Investigación, Cultura y Deporte de la Generalitat Valenciana* under the grant APOSTD/2016/036.

1(a), 1(b), 1(c), 1(d), and 1(e) are slices computed with different algorithms. This example is a contribution by itself because it presents a Petri net whose slices computed with the five algorithms described are pairwise different. In order to find this Petri net and its slicing criterion we used our implementation.




(a) Initial PN and Rakow CTL$^*_{-x}$ (b) Rakow safety

(c) Llorens et al. (d) Llorens et al. precise (e) Yu et al.

**Fig. 1.** Five different slices (a), (b), (c), (d), and (e) of the same Petri net (a).

Petri net slicing was first defined in [1]. Since then, several other techniques have appeared with different interpretations about what a Petri net slice is, and how it should be computed [5–7]. A recent survey on Petri net slicing [2] compares the algorithms proposed so far from a theoretical perspective and it reveals that they are complementary. Unfortunately, some of these algorithms have been only proposed theoretically, and no public implementation of any of them exists. Therefore, they have not been empirically evaluated or compared, and thus, their performance and precision is unknown in practice.

In this paper we present three complementary tools for the analysis and transformation of Petri nets. The first tool, `pn_slicer`, includes the first public, free, and open-source implementation of the most important algorithms for Petri net slicing, including a new algorithm that reduces the size of the slices. This tool is not only a standard slicer, but it is also able to extract from a Petri net a

set of slices that preserve a given set of properties (e.g., liveness, boundedness, etc.), using LoLA[1] and APT[2]. The second tool, `pn_prop`, allows us to compare a Petri net and its slice by analyzing what properties have been preserved or not after the slicing transformation. The third tool, `pn_tools`, implements many useful features such as Petri net animation, Petri net format conversion, and an algorithm for Petri net slicing based on a firing sequence.

Besides the three tools presented, we provide the first empirical evaluation and comparison of the main Petri net slicing algorithms. We also study what properties are kept by each algorithm. We use as benchmarks the whole *Model Checking Contest @ Petri Nets 2017*[3].

## 2 Petri Net Slicing Algorithms

There exist two important dimensions that must be considered when selecting a Petri net slicing algorithm:

• **static/dynamic**: A static slicing algorithm does not consider any initial marking for the Petri net being sliced. Contrarily, a dynamic slicing algorithm does consider a concrete initial marking. Hence, in general, dynamic slicing algorithms can produce smaller slices than their static counterparts.

• **backward/forward**: A backward slicing algorithm computes the part of the Petri net that can contribute tokens to the slicing criterion. A forward slicing algorithm computes the part of the Petri net to which the slicing criterion can contribute tokens.

In the rest of this section, we review and summarize the most important Petri net slicing techniques that can be found in the literature [5–7].

**Rakow - CTL$^*_{-x}$ Slicing** [6]: static backward slicing approach preserving any property expressed in CTL$^*_{-x}$, i.e., CTL$^*$ without next-time operator. The slicing criterion is a set of places (which may, for instance, be the places that a CTL$^*_{-x}$ property $\varphi$ refers to). This algorithm starts in the criterion places and, iteratively, builds the sliced net by taking all the incoming and outgoing non-reading transitions (those that change the marking of a place) together with their input places. Rakow's CTL$^*_{-x}$ algorithm computes all the paths $P$ of the Petri net that could change (increase or decrease) the token count of any place of the slicing criterion, and also those paths that can enable or disable the transitions in $P$.

**Rakow - Safety Slicing** [6]: static backward slicing approach that preserves stutter-invariant linear-time safety properties. The slicing criterion is also a set of places $Q$. This algorithm takes all non-reading transitions connected to $Q$ and all their input places and, iteratively, takes only transitions that increase the token count on places in the sliced net and their input places. Rakow's safety algorithm computes all the paths of the Petri net that contribute tokens to the

[1] LoLA - Low Level Petri Net Analyzer: `http://home.gna.org/service-tech/lola/`

[2] APT - Analysis of Petri nets and labelled transition systems: `https://github.com/CvO-theory/apt`

[3] Model Checking Contest @ Petri Nets 2017: `http://mcc.lip6.fr/models.php`

slicing criterion, in such a way that the places in the slice contain at least as many tokens as the original net, and the same token count on the slicing criterion.
**Llorens et al.** [5]: It was the first dynamic slicing approach, and combines both backward and forward slicing. The slicing criterion is a pair $\langle M_0, Q\rangle$, being $M_0$ an initial marking and $Q$ a set of places. The algorithm iteratively computes a backward slice by taking all the incoming transitions together with their input places. Then, the algorithm computes a forward slice by collecting first all places that are marked in $M_0$ and all transitions initially enabled in $M_0$ and, iteratively, it moves forwards adding their outgoing places and the transitions whose input places are in the set of collected places. The final slice is obtained as the intersection of the backward and forward slices. Llorens et al.'s algorithm computes all the paths of the Petri net that could increase the token count of any place of the slicing criterion from the initial marking.
**Llorens et al. - precise**: The precision of the algorithm by Llorens et al. can be improved if we compute the smallest path that could increase the token count of the slicing criterion (instead of all paths). Also, if the forward slice is only computed for the resultant Petri net obtained from the backward slice. We have implemented an improved version of this algorithm. It is integrated into the Petri net slicer.
**Yu et al.** [7]: dynamic backward slicing approach based on the Structural Dependency Graph (SDG). It uses two algorithms: Algorithm 1 takes a set of places $Q$ as the slicing criterion, and it constructs the $SDG(N)$ with a backward traversal process from $Q$. Algorithm 2 takes $SDG(N)$, and the slicing criterion $\langle M_0, Q\rangle$, and it extracts the dynamic slice, a subnet that can dynamically affect the slicing criterion from the initial marking $M_0$. If the initial marking $M_0$ cannot affect the slicing criterion, the dynamic slice is set to null and this means that there does not exist a dynamic slice that can transport tokens to the places of interest. Yu et al.'s algorithm computes one single path of the Petri net that could increase the token count of at least one place of the slicing criterion from the initial marking.

*Example 2.* Consider the Petri net of Fig. 1(a), where the user wants to produce a slice w.r.t. the slicing criterion $\{p_6, p_9\}$. With Rakow's $\text{CTL}^*_{-x}$ algorithm, the slice obtained is the original net (Fig. 1(a)). Figure 1(b) shows the safety slice obtained in Rakow's safety algorithm. The slice obtained by Llorens et al. is shown in Fig. 1(c). Figure 1(d) shows the slice obtained with the improved version of Llorens et al.'s algorithm. Finally, Fig. 1(e) shows the slice obtained with the algorithm by Yu et al.

## 3 A Universal Tool for Petri Net Slicing

In this section, we present PN-Suite, a system prepared to implement, combine, compare, and evaluate Petri net slicing algorithms. Roughly, this system can be seen as a workbench that implements the currently most important algorithms for Petri net slicing, and it is prepared to easily integrate more algorithms into

it. This system provides a new functionality that is particularly useful for the analysis and optimization of Petri nets: it combines all the slicing algorithms with the analysis of properties in such a way that one can reduce the size of a Petri net producing a slice that preserves some desired properties.

PN-Suite implements interfaces to communicate with other systems such as LoLA and APT. This means that it takes advantage of LoLA and APT analyses to report about the properties kept or lost by the slices produced. Hence, they are only invoked, through their public interfaces, when a property must be validated.

In the rest of this section we describe the main features and functionality of PN-Suite, and its architecture.

### 3.1 Installation and Usage

PN-Suite is free and open-source, and it is publicly available[4]. There are two prerequisites to use this tool, and both are free: Graphviz[5] and the Erlang/OTP framework[6]. The (free) system LoLA is optional: it enables the use of LoLA expressions as properties to preserve when performing slicing. Listing 1.1 shows the few steps needed to have PN-Suite installed in a Unix system.

**Listing 1.1.** Installation of the tool

```
$ git clone https://github.com/tamarit/pn_suite.git
$ cd pn_suite/
$ make
$ sudo make install
```

The first step clones the GitHub's repository to the local system. Then, `make` is used to compile source files and, finally, three executables (`pn_slicer`, `pn_prop`, and `pn_tools`) are generated and installed by `make install`.

The independence of these three tools is an important design decision. Separating their functionality increases their cohesion because each tool is specific for a concrete task. This allows other tools to use a single command that produces exactly what they need, or to combine them, if they need a more complex result. In particular, the tools `pn_slicer` and `pn_prop` are independent because the former is in charge of producing slices, and the later studies the preservation of properties of the slices produced.

**pn_slicer.** This tool allows us to extract slices using one of the algorithms, or to extract all the slices (using all algorithms) that preserve a given set of properties.

**Listing 1.2.** `pn_slicer` command format

```
$ pn_slicer PNML_FILE SLICING_CRITERION [PROPERTY_LIST | ALGORITHM] [-json]
```

where `SLICING_CRITERION` is a quoted list of places separated with commas. `PROPERTY_LIST` is optional. It accepts both APT properties and LoLA expressions. Valid APT properties can be found at the GitHub's repository of our tool.

[4] `https://github.com/tamarit/pn_suite`
[5] Graphviz - Graph Visualization Software: `http://www.graphviz.org/`
[6] `http://www.erlang.org/`

LoLA expressions are preceded by `lola:`, e.g., `lola:EF DEADLOCK`. `ALGORITHM` is also optional. If not specified, all algorithms will be used. To choose a slicing algorithm we have to write `alg:ALGORITHM` (no quotes required). The names of the algorithms are: `rakow_ctl`, `rakow_safety`, `llorens`, `llorens_prec`, and `yu`.

For instance, all the slices in Figure 1 can be computed with the following command (observe that they all preserve the property `conflict_free` and the same deadlock freedom, i.e., `lola:EF DEADLOCK`).

**Listing 1.3.** `pn_slicer` command usage

```
$ pn_slicer pn_example.xml "P6,P9" "conflict_free,lola:EF DEADLOCK"
Petri net named pn_example successfully read.
Slicing criterion: [P6, P9]
1.- Llorens et al's slicer precise -> Reduction: 54.55 %
2.- Llorens et al's slicer -> Reduction: 9.09 %
3.- Rakow's slicer CTL -> Reduction: 0.00 %
4.- Yu et al's slicer -> Reduction: 13.64 %
5.- Rakow's slicer safety -> Reduction: 4.55 %
```

Each slice is stored in a file named `output/<PNML_NAME>_<OUTPUT_NUMBER>.pnml`, where `<OUTPUT_NUMBER>` indicates the algorithm used (1.-Llorens, 3.-Rakow, 4.-Yu, etc.). For example, Yu's slice generated in Listing 1.3 can be found at `output/example_4.pnml`. A PDF file is also generated. If flag `-json` is used, a JSON output is generated with exact details about locations and other data.

**pn_prop.** This tool allows us to study the preservation of properties of a slice.

**Listing 1.4.** `pn_prop` command format

```
$ pn_prop PNML_FILE PNML_FILE [PROPERTY_LIST]
```

Given two Petri nets (often a Petri net and its slice), it shows a list of properties that hold in both Petri nets, and a list of properties that only hold in the original Petri net. It is also possible to specify some specific properties, and only them will be analyzed. For the analysis of properties, `pn_prop` conveniently communicates with either LoLA, or APT, or both. With the information provided by these tools, it decides whether the required properties are preserved. For instance, Listing 1.5 shows that both properties, `simply_live` and `EF DEADLOCK`, are preserved between the two Petri nets given as arguments.

**Listing 1.5.** `pn_prop` command usage examples

```
$ pn_prop pn_example.xml output/example_1.pnml
Preserved properties:
weakly_connected, output_nonbranching, strongly_connected, free_choice, pure,
plain, strongly_live, k-marking, s_net, nonpure_only_simple_side_conditions,
simply_live, t_net, weakly_live, restricted_free_choice, homogeneous,
isolated_elements, bounded, reversible, conflict_free
Changed properties:
backwards_persistent, num_places, bicf, persistent, bcf, num_tokens,
asymmetric_choice, num_arcs, num_transitions, num_labels, safe, k-bounded

$ pn_prop pn_example.xml output/example_1.pnml "simply_live,lola:EF DEADLOCK"
true
```

**pn_tools.** This tool is interactive and it can be used to animate a Petri net, slice it, or convert it to several formats.

**Listing 1.6.** `pn_tools` command usage

```
$ pn_tools pn_example.xml
Petri net example successfully read.

These are the available options:
1 .- Run the Petri Net
2 .- Export the Petri Net
3 .- Slicing Llorens et al.
4 .- Slicing Llorens et al. (precise)
5 .- Slicing Llorens et al. (for a given transition seq.)
6 .- Slicing Yu et al.
7 .- Slicing Rakow CTL
8 .- Slicing Rakow Safety
What do you want to do?
[1/2/3/4/5/6/7/8]:
```

*Animation*: The Petri net can be animated either manually or randomly. If manual animation is chosen, the system iteratively shows to the user the enabled transitions, and they can select the transitions that must be fired. As a visual support, the Petri net that is being animated can be found at `output/<PN_NAME>_run.pdf`. In this PDF, the enabled transitions are highlighted in red, so the user can see them clearly. In the random animation we can specify the number `n` of random steps to be performed, and the Petri net fires `n` random transitions, or until no more transitions can be fired (Listing 1.7).

*Slicing*: The user can select from a menu a slicing algorithm. Then, according to the chosen algorithm, the user can specify a slicing criterion. The Petri net is then automatically sliced and a new Petri net (the slice) is produced. It is important to highlight that this tool implements another slicing algorithm (option 5 in Listing 1.6) besides those of `pn_slicer`. This algorithm allows us to extract a slice from a Petri net considering a specific firing sequence (instead of all possible firing sequences as all the other algorithms do).

*Output*: The output of PN-Suite can be produced in many different formats, including standard PNML[7] (compatible with PIPE5[8]), LoLA, APT, DOT and more than 50 other formats provided by Graphviz (Listing 1.8).

[7] The Petri Net Markup Language reference site: `http://www.pnml.org/`

[8] The Platform Independent Petri Net Editor version 5: `http://sarahtattersall.github.io/PIPE/`

**Listing 1.7.** The Petri net can be animated either randomly or manually

```
[1/2/3/4/5/6/7/8]:  1
Available modes:
0.- Manually
n.- n random steps (at most)
How do you want to run the PN? $ 10
Selected transition: T5
Selected transition: T9
Selected transition: T10
Selected transition: T7
Selected transition: T8
There is not any fireable transition.
Execution:
T5,T9,T10,T7,T8
```

**Listing 1.8.** Output formats

```
[1/2/3/4/5/6/7/8]:  2
1 .- pdf
2 .- dot
3 .- PNML (compatible with PIPE)
4 .- LoLA
5 .- APT
6 .- Other formats
What format do you need?
[1/2/3/4/5/6]:  1
```

Even though we strongly encourage all users to install PN-Suite, we have implemented an online web interface[9]. This interface allows for testing many features of PN-Suite without the need to install it. It includes the five Petri net slicing algorithms that can be studied and compared with any PNML Petri net or with a collection of Petri nets available in the interface. For security reasons, the runtime has been limited to 2 seconds. Figure 2 shows a screenshot of the web interface showing the slice of the Petri net in Example 1(a).

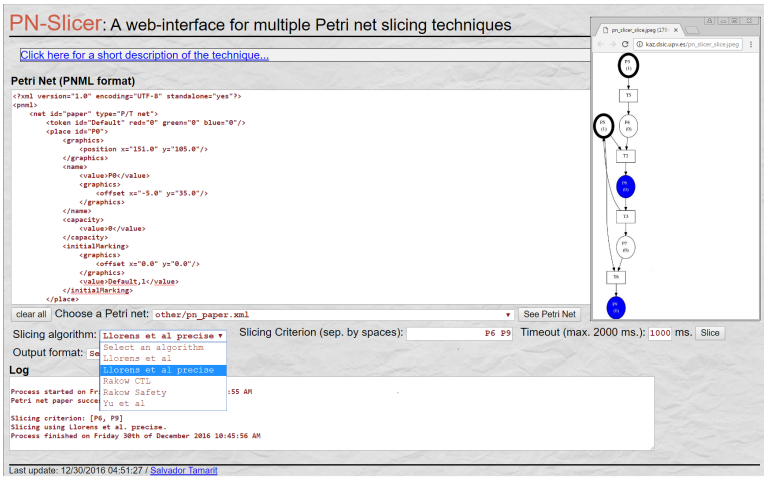



**Fig. 2.** Screenshot of PN-Suite web-interface with the slice of Figure 1(d).

### 3.2 Architecture

The internal architecture of our system is depicted in Figure 3. In the figure, the white rectangles are files or text (e.g., the original Petri net, its slice, the

[9] `http://kaz.dsic.upv.es/pn_slicer`

slicing criterion, or reports about properties preserved after the transformation), whereas dark rectangles represent modules of the system.

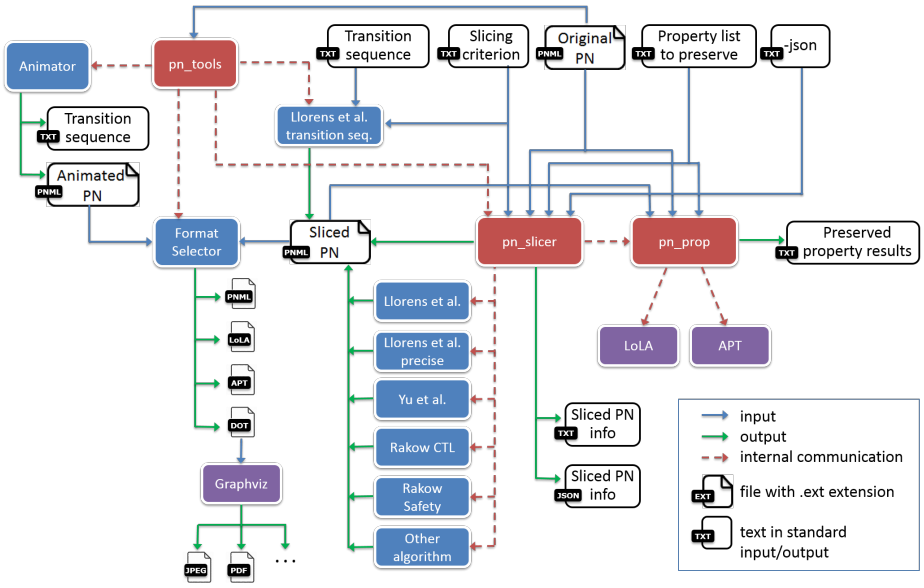



**Fig. 3.** Architecture of PN-Suite.

It is important to highlight that, currently, PN-Suite uses five different slicing algorithms, but it is prepared to easily integrate more algorithms. For this, it contains interfaces that allow the system to invoke the slicing algorithm with given inputs, and receive the answer with a specified format. The *Other algorithm* dark rectangle in Figure 3 represents this scenario. Hence, the system can be extended by other researchers that want to include their slicing algorithms, and it can be used to compare future slicing algorithms against the current empirical evaluation (see Section 4).

## 4 Empirical Evaluation

We conducted several experiments to empirically evaluate and compare all the Petri net slicing algorithms. For the evaluation we used the suite *Model Checking Contest @ Petri Nets 2017*. For each one of the 43 Petri nets used, we produced 20 random slicing criteria with a size between 1 and 5 places. Hence, in total we produced 860 slicing criteria. Each slicing criteria with its associated Petri net was then used to produce a slice with each of the five slicing algorithms implemented. The Petri nets of the benchmarks can be found, classified by year, in folder `examples/mcc_models` at: `https://github.com/tamarit/pn_suite`.

For the replicability and validation of the experiments, all data, including the slicing criteria is available, classified by year, in folder `data`.

In order to evaluate the performance of our tool, all benchmarks were executed in the same hardware configuration: Intel® Core™ i7-3520M (2 cores, 4MB Cache, 2.90GHz) with 8GB RAM. During the execution of the benchmarks, all processes of the system except PN-Suite were stopped to avoid interference of external programs.

We produced the set of measures shown in Table 1. This table compares the five slicing algorithms, one in each column: Llorens et al. (`L`), Llorens et al. precise (`LP`), Rakow CTL$^*_{-x}$ slicing (`RC`), Rakow Safety slicing (`RS`), and Yu et al. (`Y`). The rows have been divided into two sections.

1. *Property preservation.* The upper part of the table studies the preservation of properties—a document in our repository[10] contains a short description of these properties—after the slicing process carried out with each algorithm. Of course, given a Petri net where a specific property (e.g., *strong liveness*) is preserved, there can coexist slices of this Petri net where the property is preserved and where the property is not preserved. Therefore, we are interested in statistical information. Specifically, we want to know, for each property and algorithm, the ratio of slices produced by the algorithm that do preserve the property. This information is useful to select one slicing algorithm if one is interested in preserving some specific properties in the slice. Note that this information is complementary to theoretical results that prove whether an algorithm preserves or not certain property, i.e., a boolean value. With our study, we are not providing a boolean value, instead a trusting ratio. For instance, although according to the theory the property *backwards persistent* would not be preserved by any algorithm, here we show that by using Rakow CTL$^*_{-x}$ slicing we can get a slice where this property is preserved in 99% of the cases. Moreover, the results obtained can lead new theoretical proofs, since new property preservations can be discovered and then proved. To design this experiment, we sliced all the benchmarks with each algorithm, and evaluated the percentage of times that each property holds in the slice. In each row, the best value is in bold.

2. *Performance and efficiency.* In the bottom part of the table we first measure the percentage of times that the slice is equal to the original Petri net, considering the number of places, tokens, transitions, and labels. Row `Size` shows the proportion of the size of the slice with respect to the original net. For this, it considers places and transitions. Finally, row `Time` shows the average time needed by each of the algorithms to compute a slice.

In the table we can see that the algorithm whose slices do preserve more properties is Rakow's CTL$^*_{-x}$ algorithm. It presents the best values in all cases except in two. This algorithm preserves all properties with a probability higher than 0.9 except for one case: *traps*. This is an expected result, because slices usually destroy traps when they remove transitions. However, 58% of the traps were preserved with this algorithm.

[10] `https://github.com/tamarit/pn_suite/blob/master/doc/glossary.pdf`

| Property preservation | | L | LP | RC | RS | Y |
|---|---|---|---|---|---|---|
| Behavioural | *strongly-live* | **1.00** | 0.80 | **1.00** | **1.00** | 0.95 |
| | *weakly-live* | **1.00** | 0.91 | **1.00** | **1.00** | 0.92 |
| | *simply-live* | **0.91** | **0.91** | **0.91** | **0.91** | 0.78 |
| | *deadlock free* | **1.00** | 0.82 | **1.00** | **1.00** | **1.00** |
| | *persistent* | 0.89 | 0.61 | **1.00** | 0.97 | 0.57 |
| | *backwards persistent* | 0.98 | 0.47 | **0.99** | 0.98 | 0.77 |
| | *bounded* | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| | *k-bounded* | **1.00** | 0.85 | **1.00** | **1.00** | 0.88 |
| | *safe* | **1.00** | 0.88 | **1.00** | **1.00** | **1.00** |
| | *k-marking* | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| | *reversible* | **1.00** | 0.80 | **1.00** | **1.00** | 0.95 |
| | *binary conflict free* | 0.89 | 0.61 | **1.00** | 0.97 | 0.57 |
| | *behaviourally conflict free* | 0.89 | 0.61 | **1.00** | 0.97 | 0.57 |
| Structural | *strongly-connected* | **1.00** | 0.81 | **1.00** | **1.00** | 0.80 |
| | *weakly-connected* | 0.99 | 0.88 | **1.00** | **1.00** | 0.71 |
| | *isolated elements* | 0.99 | 0.98 | **1.00** | **1.00** | 0.77 |
| | *free-choice* | 0.90 | 0.67 | **0.94** | 0.92 | 0.83 |
| | *restricted-free-choice* | 0.97 | 0.97 | **0.98** | **0.98** | 0.87 |
| | *asymmetric-choice* | 0.89 | 0.75 | **1.00** | 0.92 | 0.86 |
| | *siphons* | 0.91 | 0.51 | **0.94** | 0.90 | 0.63 |
| | *traps* | 0.52 | 0.12 | **0.58** | **0.58** | 0.24 |
| | *pure* | **1.00** | 0.95 | 0.94 | 0.94 | **1.00** |
| | *nonpure-only-simple-side-conditions* | **1.00** | 0.95 | 0.94 | 0.94 | **1.00** |
| | *conflict free* | 0.89 | 0.61 | **1.00** | 0.97 | 0.57 |
| | *output-nonbranching* | 0.89 | 0.61 | **1.00** | 0.97 | 0.57 |
| | *s-net* | 0.94 | 0.94 | **0.95** | **0.95** | 0.94 |
| | *t-net* | 0.90 | 0.61 | **1.00** | 0.97 | 0.69 |
| Performance and efficiency | | L | LP | RC | RS | Y |
| | *num_places* | 0.52 | **0.12** | 0.50 | 0.50 | 0.24 |
| | *num_tokens* | 0.91 | **0.62** | 0.94 | 0.90 | 0.91 |
| | *num_arcs* | 0.52 | **0.12** | 0.50 | 0.50 | 0.23 |
| | *num_transitions* | 0.52 | **0.12** | 0.53 | 0.50 | 0.23 |
| | *num_labels* | 0.52 | **0.12** | 0.53 | 0.50 | 0.23 |
| | Size (% reduction of the slice) | 0.78 | 0.53 | 0.84 | 0.80 | **0.50** |
| | Time (runtime in milliseconds) | 10.77 | 774.95 | 7.11 | **7.03** | 14.39 |

**Table 1.** Benchmark results showing statistical information about the preservation of properties by the different slices.

Another interesting information is that those cells where the value is not 1.00 do ensure that the corresponding algorithm does not always preserve the property (we found a counterexample). The opposite is not true: those cells with a value of 1.00 do not necessarily ensure that the property will be preserved in all cases by the algorithm.

In the second part of the table we see that computing slices is an efficient task (less than 1 second in all cases). Almost all algorithms have similar time values (between 7 and 14 ms.) except for the improved version of Llorens et al.'s algorithm, with a runtime of two orders of magnitude more. This algorithm is much more exhaustive than the others (i.e., it computes all possible paths and takes the smaller one). For this reason, its computational cost is significantly higher, but the size of its slices is usually smaller.

Finally, the size of the slices is a very interesting result. It ranges over 50% and 84% of the size of the original Petri net. We provide five rows *num_X* to give an idea of the slice's size considering places, transitions, etc.

## 5 Conclusions

We have presented PN-Suite, a suite that includes three independent and complementary tools to analyse Petri nets. Specifically, our implementation includes animation and analysis tools, and the most powerful Petri net slicer in the current state of the art, which integrates the implementation of the main slicing algorithms. The implementation of all the algorithms is necessary and useful, because they have different purposes, and they retain different properties in their slices.

The implementation of the main slicing algorithms have produced another important result: their first empirical evaluation and comparison. Until our implementation, some of the algorithms were only theoretical results, and no public implementation existed. Thus, their individual performance and scalability remained unknown. Their relative efficiency was also unknown. The evaluation of the algorithms have produced the first measures of their average runtime, the relative size of the slices, and the properties that they preserve.

There are two other side results of our work: in the theoretical side, we have proposed (and implemented) a new algorithm that improves the dynamic slicing algorithm by Llorens et al. [5]. In the practical side, we have implemented a web interface to test the suite and compare the algorithms without the need to install the system. All our research and implementation is public, free and open-source. This includes our experiments and empirical evaluation.

## References


1. Chang, C., Wang, H.: A Slicing Algorithm of Concurrency Modeling Based on Petri Nets. In: Proc. of the Int'l Conf. on Parallel Processing. pp. 789–792. ICPP'86, IEEE Computer Society Press (1986)
2. Khan, Y., Guelfi, N.: Survey of Petri Nets Slicing. Tech. rep., University of Luxembourg, Faculty of Science, Technology and Communication (FSTC), Computer Science and Communications Research Unit (CSC), Luxembourg (2013), `http://hdl.handle.net/10993/13606`
3. Khan, Y., Risoldi, M.: Optimizing Algebraic Petri Net Model Checking by Slicing. In: Int'l Workshop on Modeling and Business Environments. pp. 275–294. ModBE 2013, associated with Petri Nets 2013 (2013)
4. Lee, W., Cha, S., Kwon, Y., Kim, H.: A Slicing-based Approach to Enhance Petri Net Reachability Analysis. Journal of Research and Practice in Information Technology 32(2), 131–143 (2000)
5. Llorens, M., Oliver, J., Silva, J., Tamarit, S., Vidal, G.: Dynamic Slicing Techniques for Petri Nets. Electronic Notes in Theoretical Computer Science 223, 153–165 (2008)
6. Rakow, A.: Safety Slicing Petri Nets. In: Int'l Conf. on Application and Theory of Petri Nets and Concurrency. pp. 268–287. Petri Nets'12, Springer LNCS 7347 (2012)
7. Yu, W., Ding, Z., Fang, X.: Dynamic Slicing of Petri Nets Based on Structural Dependency Graph and its Application in System Analysis. Asian Journal of Control 17(4), 1403–1414 (2015), `http://dx.doi.org/10.1002/asjc.1031`