Properties preserved by Petri net slicing algorithms

This document summarizes the results obtained from a collection of experiments performed to compare different Petri net slicing algorithms. The algorithms considered are the following:

CTL$^*_{-x}$ Slicing (RC): This approach is backwards and static. It produces slices that preserve a CTL$^*_{-x}$ property (that is CTL* but without next-time operator). In this approach, the slicing criterion is defined by a list of places (for example, those places referred to in a CTL$^*_{-x}$ property). The slice is computed from the places in the slicing criterion by (iteratively) collecting all the transitions that change the marking of a place, i.e. the incoming and outgoing non-reading transitions together with their related input places. The corresponding CTL$^*_{-x}$ algorithm identifies all those paths that can change (decrease or increase) tokens in the slicing criterion (any place), and it also identifies the paths that could disable or enable a transition in those paths.

Safety Slicing (RS): This approach is backwards and static. It extracts a subnet that preserves stutter-invariant linear-time safety properties. The slicing criteria in this approach is composed of sets of places Q. The corresponding algorithm collects all non-reading transitions connected to Q and all their input places. Then, iteratively, it collects (only) those transitions (and their input places) that could increase the tokens in the sliced net. This algorithm identifies all the paths that can increase the tokens in the slicing criterion ensuring that in the resulting slice: (i) all the places contain the same or more tokens as in the original net; and (ii) the places in the slicing criterion keep the same number of tokens.

Maximal slicing by Llorens et al. (LM): This algorithm was the first approach for dynamic slicing. It is backwards, but it uses a forwards and a backwards traversal of the Petri net. The slicing criterion in this approach is a pair $\langle M_0, Q\rangle$, where M_0 is the initial marking and Q is a set of places. This algorithm computes two slices: (i) a backward slice is computed collecting all incoming transitions plus their relating input places. (ii) a forward slice is computed as follows: first, it takes all places marked in M_0 and all transitions initially enabled in M_0. Then, the outgoing places and the transitions whose input places are in the slice are iteratively included in the slice. Finally, the algorithm computes the intersection of the forward and backward slices to produce the final slice. This algorithm identifies all the paths that, from the initial marking, can contribute tokens to any place in the slicing criterion.

Minimal Slicing by Llorens et al. (Lm): The idea of this algorithm is that the slice is formed from the places and transitions in the path with the smallest number of transitions that contribute tokens to the slicing criterion. This means that the slice contains the minimum firing sequence that contribute tokens to the slicing criterion. As in the algorithm by Llorens et al., all paths are computed, but only one is included in the slice. Moreover, the backward slice is computed first, and then, the forward slice is calculated from the backward slice (instead of the original Petri net).

Slicing by Yu et al. (Yu): This approach is backwards and dynamic. The underlying data structure used to produce slices is the Structural Dependency Graph (SDG). Two algorithms are used: (i) The first algorithm defines the slicing criterion as a set of places Q, and it builds a $SDG(N)$ by traversing the SDG backwards from Q. (ii) The second algorithm starts from $SDG(N)$. It defines the slicing criterion as $\langle M_0, Q \rangle$. The dynamic slice is the subnet of N that can dynamically influence the slicing criterion from the initial marking M_0. It could be possible that the initial marking M_0 cannot affect the slicing criterion. In such a case, the slice is empty. This means that there does not exist any subnet that can introduce tokens to the slicing criterion. This algorithm identifies one single path (there could be others) that, from the initial marking, can augment the number of tokens in at least one place of the slicing criterion.

Table 1 indicates for each algorithm what properties are preserved or not. The rows have been divided into two sections.

1. *Property preservation.* The upper part of the table studies the preservation of properties—a description of these properties can be found here:

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

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.

Another interesting information is that those cells where the value is not 100% 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 100% 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 3 and 16 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 the minimal slice). For this reason, its computational cost is significantly higher, but the size of its slices is usually very small.

Finally, the size of the slices is a very interesting result. It ranges over 9% and 90% 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.

Property preservation		LM	Lm	RC	RS	Yu
Behavioural	*strongly-live*	**100%**	80%	**100%**	**100%**	95%
	weakly-live	**100%**	91%	**100%**	**100%**	92%
	simply-live	**91%**	**91%**	**91%**	**91%**	78%
	deadlock free	**100%**	82%	**100%**	**100%**	**100%**
	persistent	89%	61%	**100%**	97%	57%
	backwards persistent	98%	47%	**99%**	98%	77%
	bounded	**100%**	**100%**	**100%**	**100%**	**100%**
	k-bounded	**100%**	85%	**100%**	**100%**	88%
	safe	**100%**	88%	**100%**	**100%**	**100%**
	k-marking	**100%**	**100%**	**100%**	**100%**	**100%**
	reversible	**100%**	80%	**100%**	**100%**	95%
	binary conflict free	89%	61%	**100%**	97%	57%
	behaviourally conflict free	89%	61%	**100%**	97%	57%
Structural	*strongly-connected*	**100%**	81%	**100%**	**100%**	80%
	weakly-connected	99%	88%	**100%**	**100%**	71%
	isolated elements	99%	98%	**100%**	**100%**	77%
	free-choice	90%	67%	**94%**	92%	83%
	restricted-free-choice	97%	97%	**98%**	**98%**	87%
	asymmetric-choice	89%	75%	**100%**	92%	86%
	siphons	91%	51%	**94%**	90%	63%
	traps	52%	12%	**58%**	**58%**	24%
	pure	**100%**	95%	94%	94%	**100%**
	nonpure-only-simple-side-conditions	**100%**	95%	94%	94%	**100%**
	conflict free	89%	61%	**100%**	97%	57%
	output-nonbranching	89%	61%	**100%**	97%	57%
	s-net	94%	94%	**95%**	**95%**	94%
	t-net	90%	61%	**100%**	97%	69%

Performance and efficiency	LM	Lm	RC	RS	Y
num_places	91.53%	**14.19%**	90.14%	88.68%	65.14%
num_tokens	96.55%	**47.38%**	96.49%	95.31%	96.74%
num_arcs	91.88%	**5.49%**	89.52%	85.78%	41.56%
num_transitions	91.99%	**6.18%**	90.23%	86.07%	42.78%
Size (w.r.t. the original net)	91.89%	**9.52%**	89.93%	86.52%	51.79%
Time (runtime in milliseconds)	16.32	131.95	**3.33**	3.64	139.03

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