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) from the backward slice, 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 final dynamic slice is composed of all places and transitions in the computed forward slice for which there exists a path to some place in the slicing criterion. 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 minimum firing sequence that contribute tokens to the slicing criterion. The algorithm performs the following phases: (i) a backward slice that iteratively collects all the incoming transitions together with their input places from the slicing criterion (same as in the Maximal slicing); (ii) discard useless branches in the

backward slice (those that do not contain any token nor any enabled transition are discarded); and (iii) compute the minimal forward slice by iteratively constructing the reachability tree, expanding those branches that can form a minimum increasing firing sequence. Therefore, only a part of the reachability tree is build. The final output is the net formed from the places and transitions needed to fire the minimum transition sequence that contributes tokens to the slicing criterion.

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.

The experiments were performed using a collection of Petri nets from the benchmark suite *Model Checking Contest @ Petri Nets 2017*. We randomly selected 1–5 places to define a slicing criterion. This was done 20 times for each benchmark. Then, from each triple (slicing algorithm, Petri net, slicing criterion) we extracted the corresponding slice. All data is available at: `https://github.com/tamarit/pn_suite/data/2022_experiments/`.

Table 1 presents the performance and efficiency of the algorithms. 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. Row `Time` shows the average time needed by each of the algorithms to compute a slice.

We can see that the computation of slices is a relatively efficient process (< 1 s. in all benchmarks). Comparatively, practically all algorithms showed similar runtimes ([5,13] ms.) except for Llorens et al.'s minimal slicing and Yu et al.'s algorithms, which showed a runtime of two and one order of magnitude more, respectively.

Tables 2 and 3 indicate for each algorithm what properties are preserved or not. Table 2 shows the preservation of behavioural properties and Table 3 shows the preservation of structural properties. A description of these properties can be found here:

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

Performance and efficiency	LM	Lm	RC	RS	Y
num_places	91.60%	**16.80%**	92.97%	91.94%	67.64%
num_tokens	98.20%	**46.40%**	99.57%	98.90%	98.70%
num_arcs	92.19%	**6.34%**	93.66%	88.68%	47.71%
num_transitions	91.82%	**7.06%**	93.91%	88.34%	49.40%
Size (w.r.t. the original net)	91.80%	**11.39%**	93.15%	88.60%	56.65%
Time (runtime in milliseconds)	13.00	749.87	5.69	**5.41**	60.04

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

We distinguish in the table four different situations, using columns `Orig` and `Slice`:

(a) The property holds in the original Petri net and in the slice (yes | yes).

(b) The property holds in the original Petri net and not in the slice (yes | no).

(c) The property does not hold in the original Petri net but it holds in the slice (no | yes).

(d) The property does not hold in the original Petri net nor in the slice (no | no).

For each property and algorithm, the percentage indicates the number of times over the total that the property was preserved or not.

An 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.

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. Moreover, the results obtained can lead new theoretical proofs, since new property preservations can be discovered and then proved.

In Tables 4 and 5, we provide a counterexample for each property that is not preserved (`Orig` = yes and `Slice` = no in Tables 2 and 3) with algorithms `LM` and `Lm`.

Property preservation	Orig	Slice	LM	Lm	RC	RS	Yu
strongly-live	yes	yes	**40.0%**	0.0%	**40.0%**	**40.0%**	30.24%
	yes	no	0.0%	**40.0%**	0.0%	0.0%	9.76%
	no	yes	**9.51%**	0.0%	0.0%	1.46%	4.88%
	no	no	50.49%	**60.0%**	**60.0%**	58.54%	55.12%
weakly-live	yes	yes	**50.49%**	0.0%	**50.49%**	**50.49%**	35.85%
	yes	no	0.0%	**50.49%**	0.0%	0.0%	14.63%
	no	yes	**9.51%**	0.0%	0.0%	1.46%	4.88%
	no	no	40.0%	**49.51%**	**49.51%**	48.05%	44.63%
simply-live	yes	yes	**80.93%**	**80.93%**	**80.93%**	**80.93%**	**80.93%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	6.05%	**19.07%**	1.16%	1.16%	10.47%
	no	no	13.02%	0.0%	**17.91%**	**17.91%**	8.6%
persistent	yes	yes	**9.27%**	**9.27%**	**9.27%**	**9.27%**	**9.27%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.71%	**90.49%**	0.49%	0.49%	23.17%
	no	no	89.02%	0.24%	**90.24%**	**90.24%**	67.56%
backwards-persistent	yes	yes	10.73%	**10.98%**	**10.98%**	10.73%	**10.98%**
	yes	no	**0.24%**	0.0%	0.0%	**0.24%**	0.0%
	no	yes	0.49%	**89.02%**	0.49%	0.49%	32.44%
	no	no	88.54%	0.0%	88.54%	88.54%	56.59%
bounded	yes	yes	**95.35%**	**95.35%**	**95.35%**	**95.35%**	**95.35%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	0.0%	**4.65%**	0.0%	0.0%	**4.65%**
	no	no	**4.65%**	0.0%	**4.65%**	**4.65%**	0.0%
k-bounded	yes		**100.00%**	78.78%	**100.00%**	**100.00%**	98.78%
safe	yes	yes	**63.26%**	**63.26%**	**63.26%**	**63.26%**	**63.26%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	0.0%	**19.07%**	0.0%	0.0%	4.65%
	no	no	**36.74%**	17.67%	**36.74%**	**36.74%**	32.09%
k-marking	yes		**100.00%**	97.67%	**100.00%**	**100.00%**	**100.00%**
reversible	yes	yes	**48.78%**	0.0%	**48.78%**	**48.78%**	30.24%
	yes	no	0.0%	**48.78%**	0.0%	0.0%	18.54%
	no	yes	**4.88%**	0.0%	0.0%	1.46%	**4.88%**
	no	no	46.34%	**51.22%**	**51.22%**	49.76%	46.34%
binary conflict free	yes	yes	**9.27%**	**9.27%**	**9.27%**	**9.27%**	**9.27%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.71%	**90.49%**	0.49%	0.49%	21.71%
	no	no	89.02%	0.24%	**90.24%**	**90.24%**	69.02%
behavioural conflict free	yes	yes	**9.27%**	**9.27%**	**9.27%**	**9.27%**	**9.27%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.71%	**90.49%**	0.49%	0.49%	21.71%
	no	no	89.02%	0.24%	**90.24%**	**90.24%**	69.02%

Table 2: Benchmark results showing statistical information about the preservation of behavioural properties by the slicing algorithms.

Property preservation	Orig	Slice	LM	Lm	RC	RS	Yu
strongly-connected	yes	yes	**56.74%**	0.0%	**56.74%**	**56.74%**	16.28%
	yes	no	0.0%	**56.74%**	0.0%	0.0%	40.47%
	no	yes	**11.86%**	0.0%	2.79%	4.19%	4.65%
	no	no	31.4%	**43.26%**	40.47%	39.07%	38.6%
weakly-connected	yes	yes	**94.65%**	**94.65%**	**94.65%**	**94.65%**	55.12%
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**39.53%**
	no	yes	**5.35%**	**5.35%**	1.86%	1.86%	0.0%
	no	no	0.0%	0.0%	3.49%	3.49%	**5.35%**
isolated elements	yes	yes	0.0%	0.0%	3.49%	3.49%	**5.35%**
	yes	no	**5.35%**	**5.35%**	1.86%	1.86%	0.0%
	no	yes	0.0%	0.0%	0.0%	0.0%	**35.35%**
	no	no	**94.65%**	**94.65%**	**94.65%**	**94.65%**	59.3%
free-choice	yes	yes	**13.49%**	**13.49%**	**13.49%**	**13.49%**	**13.49%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.63%	**83.49%**	3.02%	3.26%	16.51%
	no	no	84.88%	3.02%	83.49%	83.26%	70.0%
restricted-free-choice	yes	yes	**2.33%**	**2.33%**	**2.33%**	**2.33%**	**2.33%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	0.93%	**30.7%**	1.16%	1.16%	7.67%
	no	no	**96.74%**	66.98%	96.51%	96.51%	90.0%
asymmetric-choice	yes	yes	**31.16%**	**31.16%**	**31.16%**	**31.16%**	**31.16%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	2.09%	**68.84%**	1.86%	7.91%	24.88%
	no	no	66.74%	0.0%	66.98%	60.93%	43.95%
siphons	yes		88.37%	19.53%	**92.09%**	86.28%	40.47%
traps	yes		61.86%	0.00%	**78.60%**	65.58%	20.70%
pure	yes	yes	**48.14%**	**48.14%**	**48.14%**	**48.14%**	**48.14%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	0.0%	**33.95%**	3.02%	3.02%	9.53%
	no	no	**51.86%**	17.91%	48.84%	48.84%	42.33%
nonpure-only-simple-side-conditions	yes	yes	**51.86%**	17.91%	48.84%	48.84%	42.33%
	yes	no	0.0%	**33.95%**	3.02%	3.02%	9.53%
	no	yes	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	no	**48.14%**	**48.14%**	**48.14%**	**48.14%**	**48.14%**
homogeneous	yes	yes	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
plain	yes	yes	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
conflict-free	yes	yes	**8.84%**	**8.84%**	**8.84%**	**8.84%**	**8.84%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.63%	**88.14%**	0.47%	0.47%	22.56%
	no	no	89.53%	3.02%	**90.7%**	**90.7%**	68.6%
output-nonbranching	yes	yes	**8.84%**	**8.84%**	**8.84%**	**8.84%**	**8.84%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.63%	**88.14%**	0.47%	0.47%	21.16%
	no	no	89.53%	3.02%	**90.7%**	**90.7%**	70.0%
t-net	yes	yes	**8.84%**	**8.84%**	**8.84%**	**8.84%**	**8.84%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	1.63%	**81.86%**	0.47%	0.47%	20.0%
	no	no	89.53%	9.3%	**90.7%**	**90.7%**	71.16%
s-net	yes	yes	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	yes	no	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	no	yes	3.02%	**30.47%**	3.49%	3.49%	8.6%
	no	no	**96.98%**	69.53%	96.51%	96.51%	91.4%

Table 3: Benchmark results showing statistical information about the preservation of structural properties by the slicing algorithms.

Property preservation		LM	Counterexample	Slicing criterion
Behavioural	*strongly-live*	0%		
	weakly-live	0%		
	simply-live	0%		
	persistent	0%		
	backwards persistent	0.24%	QCertifProtocol_02-unfold.pnml	AstopAbort,n4_0
	bounded	0%		
	k-bounded	0%		
	safe	0%		
	k-marking	0%		
	reversible	0%		
	binary conflict free	0%		
	behaviourally conflict free	0%		
Structural	*strongly-connected*	0%		
	weakly-connected	0%		
	isolated elements	5.35%	railroad-005-pt.pnml	pl_P7_2,pl_P7_6,pl_P4_1
	free-choice	0%		
	restricted-free-choice	0%		
	asymmetric-choice	0%		
	siphons	11.63%	railroad-005-pt.pnml	pl_P7_2,pl_P7_6,pl_P4_1
	traps	38.14%	raft_02.pnml	p13,p10
	pure	0%		
	nonpure-only-simple-side-cond.	0%		
	homogeneous	0%		
	plain	0%		
	conflict free	0%		
	output-nonbranching	0%		
	s-net	0%		
	t-net	0%		

Table 4: Counterexamples to show the non-preservation of properties by the LM algorithm.

Property preservation		Lm	Counterexample	Slicing criterion
Behavioural	*strongly-live*	40.0%	erk-000001.pnml	RKIPP
	weakly-live	50.49%	erk-000001.pnml	RKIPP
	simply-live	0%		
	persistent	0%		
	backwards persistent	0%		
	bounded	0%		
	k-bounded	21.22%	CircularTrain-012.pnml	Section_8,F6,Section_2
	safe	0%		
	k-marking	2.33%	FMS-2.pnml	P12,P2wM2,P1s,P2s,P1M1
	reversible	48.78%	erk-000001.pnml	RKIPP
	binary conflict free	0%		
	behaviourally conflict free	0%		
Structural	*strongly-connected*	56.74%	erk-000001.pnml	RKIPP
	weakly-connected	0%		
	isolated elements	5.35%	railroad-005-pt.pnml	pl_P7_2,pl_P7_6,pl_P4_1
	free-choice	0%		
	restricted-free-choice	0%		
	asymmetric-choice	0%		
	siphons	80.47%	erk-000001.pnml	RKIPP
	traps	100.0%	erk-000001.pnml	RKIPP
	pure	0%		
	nonpure-only-simple-side-conditions	33.95%	afcs_01_a.pnml	p17,p4,p11,p1
	homogeneous	0%		
	plain	0%		
	conflict free	0%		
	output-nonbranching	0%		
	s-net	0%		
	t-net	0%		

Table 5: Counterexamples to show the non-preservation of properties by the Lm algorithm.