Survey of Petri nets Slicing

Khan, Y. I., Konios, A. & Guelfi, N.

Original citation & hyperlink:
Khan, YI, Konios, A & Guelfi, N 2018, 'Survey of Petri nets Slicing' ACM Computing Surveys (CSUR), vol. 51, no. 5, 109.
https://dx.doi.org/10.1145/3241736

DOI 0360-0300
ISSN 1557-7341
ESSN 1557-7341

Publisher: ACM

1 A Survey of Petri nets Slicing YASIR IMTIAZ KHAN, Institute for Future Transport and Cities, Coventry University, United Kingdom ALEXANDROS KONIOS, Institute for Future Transport and Cities, Coventry University, United Kingdom NICOLAS GUELFI, Laboratory of Advanced Software Systems, University of Luxembourg, Luxembourg Petri nets slicing is a technique that aims to improve the verification of systems modeled in Petri nets. Petri nets slicing was first developed to facilitate debugging but then used for the alleviation of the state space explosion problem for the model checking of Petri nets. In this article, different slicing techniques are studied along with their algorithms introducing: i) a classification of Petri nets slicing algorithms based on their construction methodology and objective (such as improving state space analysis or testing), ii) a qualitative and quantitative discussion and comparison of major differences such as accuracy and efficiency, iii) a syntactic unification of slicing algorithms that improve state space analysis for easy and clear understanding, and iv) applications of slicing for multiple perspectives. Furthermore, some recent improvements to slicing algorithms are presented, which can certainly reduce the slice size even for strongly connected nets. A noteworthy use of this survey is for the selection and improvement of slicing techniques for optimizing the verification of state event models. CCS Concepts: 1 INTRODUCTION Petri nets have been extensively used to model and analyze concurrent and distributed system since their birth. Among several dedicated analysis techniques for Petri nets, model checking and testing are more widely and commonly used. A typical challenge in model checking is the limitations imposed by the state space explosion problem, which signifies that as systems get moderately complex, the complete enumeration of their states demands a growing amount of resources. Therefore, in some cases model checking is impractical in terms of time and memory consumption [2, 11, 12, 34]. Similarly, testing suffers from problems such as large input amount of test data, test case selection, etc [4, 5]. As a result, an intense field of research is targeting to optimize these verification techniques, either by reducing the state space or by improving the test input data. A technique called Petri net slicing falls into the first category. Petri net slicing (PN Authors' addresses: Y. I. Khan, A. Konios and N.Guelfi, School of Computing, Electronics and Mathematics, Faculty of Engineering, Environment and Computing, Coventry University, 3 Gulson Road, Coventry, Warwickshire, CV1 2JH, United Kingdom and University of Luxembourg, Luxembourg. Authors' addresses: Yasir Imtiaz Khan, Institute for Future Transport and Cities, Coventry University, Coventry, United Kingdom; Alexandros Konios, Institute for Future Transport and Cities, Coventry University, Coventry, United Kingdom; Nicolas Guelfi, Laboratory of Advanced Software Systems, University of Luxembourg, Luxembourg, Luxembourg. is a syntactic technique, which is used to reduce a Petri net model (PN model) based on a given criterion. The given criterion refers to the point of interest, e.g. a PN place or code line number, for which the PN model is analysed. The sliced part constitutes only that part of a PN model that may affect the given criterion. Existing PN slicing techniques that can be found in the present literature are being reviewed in this article [9, 17, 22, 24–26, 28–30, 35]. The classification of Petri nets slicing algorithms is proposed based on their construction methodology and objective (meaning whether they are designed to improve state space analysis or testing). Further to that, a discussion about qualitative and quantitative contributions of each slicing construction and a comparison describing the major differences between them are given. A particular convention is adopted to study the proposed slicing techniques i.e., at first, the objective of slicing algorithm is given and then the different steps required to generate the slice are presented with the help of a process-flow diagram. Moreover, by taking a simple example of a Petri net model and a property, the algorithm is explained and evaluated in terms of state space reduction. A syntactic unification of slicing algorithms either designed to improve state space analysis or testing is also presented for easy and clear understanding. Additionally, a discussion about the application of slicing in general and with respect to other state space reduction techniques is given. Finally, some recent improvements to existing slicing algorithms are presented, which can certainly be helpful for the reduction of slice size, even for strongly connected nets. The remaining part of the paper is structured as follows: in Section 2, an overview and background of slicing is discussed in the context of programming and Petri nets. A classification of Petri nets slicing is presented in Section 3. Section 4 consists of formal definitions necessary for the understanding of studied slicing algorithms. In Sections 5 and 6, a review of existing Petri nets slicing techniques is presented to provide details about the underlying theory and techniques for each slicing construction. A comparative analysis is given for all the studied algorithms in Section 7. In Section 8, the application of slicing is described in general and with respect to other state space reduction techniques. Finally, in Section 9, conclusions are drawn with respect to the review of the algorithms and the future work related to the Petri nets slicing is presented. 2 OVERVIEW AND BACKGROUND The term slicing was coined by Mark Weiser for the first time

in the context of program debugging [36]. According to Weiser's proposal, a program slice (PS) is a reduced, executable program that can be obtained from a program P based on the variables of interest and line number by removing statements such that PS replicates part of the behavior of program. To explain the basic idea of program slicing, according to Wieser [36], an example program shown in the Fig. 1(a) is considered. This program requests a positive integer number n as input and computes the sum and the product of the first n positive integer numbers. The slicing criterion that is examined is a line number and a set of variables, e.g. C = (line10, {product }). Figure 1(b) shows the sliced program that is obtained by tracing backwards possible influences on the variables. For instance, in line 7, product is multiplied by i, and in line 8, i is incremented too, so all the instructions that impact the value of i need to be kept. As a result, all the computations that do not contribute to the final value of the product have been sliced away1 . Petri nets slicing is a technique used to syntactically reduce a Petri net model in such a way that the reduced Petri net model contains only those parts that may influence the property the Petri net model is analyzed for. In general, slicing starts by identifying which places or transitions in the Petri net model are directly concerned by a property. These places constitute the slicing 1 Interested readers can find more details about program slicing in [1, 32, 37]). A Survey of Petri nets Slicing 1:3 Fig. 1. (a) An example program and (b) a sliced program w.r.t. a given criterion criterion. The slicing construction takes all the transitions that deposit or consume tokens to or from the criterion places, plus all those places that are pre-condition for those transitions. This step is repeated for later places until reaching a fixed point (see Alg. 7). A simple example of a Petri net model is provided, as shown in Fig. 2(a) representing the semantics of the operation of an insurance claim system. This behavioral model contains labelled places and transitions. In the insurance claim system, claims are received and approved, where a legal expert assesses the case and a settlement is offered to the customer. The customer may accept or reject that offer. Money is paid to the customer if he agrees upon the settlement offer, otherwise it is proceeded legally or the offer is revised. For example, if the property to be verified is 'every accepted claim is settled'. Formally this property can be specified in temporal logic as $\varphi = AG(ac \Rightarrow AFcs)$ implying that the place ac (resp. cs) is not empty. The slice to be built is based on the slicing criteria Q = {ac,cs} , where ac and cs are places extracted from the temporal description of the property. The resultant sliced net can be observed in Fig. 2(b), which is smaller than the original net. 3 TYPES OF SLICING Roughly, the PN slicing can be distinguished into two major classes (as shown in the Fig. 3): • Static Slicing and • Dynamic Slicing 3.1 Static Slicing: A slice is said to be static if the initial markings of the places are not considered for generating the slice. In this type of slicing, only set of places are considered as a slicing criterion. The static slicing starts from the given criterion place(s) and includes all the pre and post sets of transitions together with their incoming places. It may exist a sequence of transitions in the resultant slice that is not fireable because some of the pre places of these transitions are not initially marked and eventually cannot acquire any marking. Static slicing algorithms are useful in improving the state space analysis. Figure 2 shows an example of static slicing, where the slice is generated for the criterion places ac and cs without considering initial markings. 3.2 Dynamic Slicing: ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:4 Y.I. Khan, A. Konios and N. Guelfi. record accept ac reject emergency measure offer accept cs pay revise close end assess by expert legal proceedings record accept ac reject offer accept cs pay revise assess by expert legal proceedings Fig. 2. (a) An example Petri net model and (b) the sliced Petri net model w.r.t. the given criterion Static Slicing Dynamic Slicing Forward Slicing Backward Slicing Fig. 3. Classification of PN slicing constructions A slice is said to be dynamic if the initial markings of places are considered for generating the slice. The general idea is to use available information of the initial markings to generate smaller slice. For a given slicing criterion that consists of the initial markings and a set of places for a PN model, the main interest is to extract a subnet with those places and transitions of PN model that can contribute to the marking change of criterion places for any execution that starts from the initial marking. Dynamic slicing can be useful, e.g., in debugging. For example, consider if the user is analysing a particular trace of a marked PN model (using a simulation tool) such that an erroneous state is reached. In this case, it is interesting to extract a set of places and transitions (more formally, a subnet) that may erroneously contribute tokens to the places of interest (termed as criterion places) such that the user can more easily locate the bug. Dynamic slicing will produce a reduced net consisting of all the paths that contribute tokens to the criterion places for which test cases can be generated. There are two ways to compute static and dynamic slices, forward and backward slicing. Forward slicing starts from the initially marked places and by forward traversing a PN model until the criterion places, a slice is generated. Whereas, backward slicing starts from the criterion places and then by backward traversing all the incoming transitions together with their input places, a slice is obtained. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:5 An extension to the dynamic slicing can also be used to further reduce the slice size. This extension is called 'condition

slicing' and the rationale behind is to include a subset of behaviours in the sliced PN model instead of all the behaviours. The Slicing criterion consists of places and sequence of transitions. The resultant slice obtained by the condition slicing is smaller as compared to that produced by the dynamic slicing. The reason of getting a smaller slice is the inclusion of a particular sequence of transitions around the criterion places. The 'condition slicing' is very useful when analysing a particular behavior, but limits the scope of verification due to the exclusion of some sequences of transitions. 4 FORMAL DEFINITIONS In this section, basic definitions needed to understand the slicing algorithms considered by this study are provided. Most of the slicing algorithms are either designed for low-level Petri nets or Algebraic Petri nets (an instance of high-level Petri nets). A Petri net is a directed bipartite graph consisting of two essential elements, the places and transitions. Informally, Petri nets places hold resources (also known as tokens) and transitions are linked to places by input and output arcs, which can be weighted. Usually, a Petri net has a graphical concrete syntax consisting of circles for places, boxes for transitions and arrows to connect the two. The semantics of a Petri net express the non-deterministic firing of transitions in the net. Firing a transition means consuming tokens from the set of places linked to the input arcs of the transition and producing tokens into the set of places linked to the output arcs of the transition. Various advancements of Petri nets have been created, among others are the Algebraic Petri nets, where the level of abstraction of Petri nets is raised by using complex structured data [31]. Algebraic Petri Nets have two aspects, the control aspect, which is handled by a Petri Net and the data aspect, which is handled by one or many algebraic abstract data types (AADTs). Definition 4.1. (Petri net) A Petri Net PN =< P,T,w,m0 > is a tuple where: ◦ P and T are finite and disjoint sets, called places and transitions respectively. ◦ w : (P ×T) ∪ (T × P) → N is a function that assigns weights to arcs. ◦ a marking function m0 : P → N. Definition 4.2. (Pre(resp.Post) set places(resp.transitions) of PN) Let PN =< P,T,w,m0 > be a Petri net, with p ∈ P being a place, then the preset and postset of p, denoted by •p and p•, are defined as follows: •p = {t ∈ T |w(t,p) > 0}. p• = {t ∈ T |w(p,t) > 0}. Analogously •t and t• are defined. The notation •P and P• is also used representing the pre(resp.post) set of transitions of all the places in set P. Analogously, •T and T • are denoted. Definition 4.3. (Reading(resp.Non-reading) transitions of PN) Let t ∈ T be a transition in PN, t is called a reading-transition iff its firing does not change the marking of any place p ∈ (•t ∪ t•) , i.e., iff ∀p ∈ (•t ∪ t•),w(p,t) = w(t,p). Conversely, t is called a non-reading transition iff w(p,t) , w(t,p). Definition 4.4. (Algebraic Petri net) A marked Algebraic Petri Net APN =< SPEC, P,T, F, asд,cond, λ,m0 > consist of ◦ an algebraic specification SPEC = (Σ,E)2 . 2For further reading on the algebraic specifications used in the formal definition of APNs refer to [20, 22, 31]. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:6 Y.I. Khan, A. Konios and N. Guelfi. ◦ P and T are finite and disjoint sets, called places and transitions respectively. ◦ F ⊆ (P ×T) ∪ (T × P), the elements of which are called arcs. ◦ a sort assignment asд : P → S. ◦ a function,cond : T → Pf in(Σ−equation), assigning to each transition a finite set of equational conditions. ◦ an arc inscription function λ assigning to every (p,t) or (t,p) in F a finite multiset over TO P,asд(p) , ◦ an initial marking m0 assigning a finite multiset over TO P,asд(p) to every place p. Definition 4.5. (Reading(resp.Non-reading) transitions of APN) Let t ∈ T be a transition in an unfolded APN. t is called a reading-transition iff its firing does not change the marking of any place p ∈ (• t ∪ t •), i.e., iff ∀p ∈ (• t ∪ t •), λ(p,t) = λ(t,p). Conversely, t is called a non-reading transition iff λ(p,t) , λ(t,p). 5 STATIC SLICING ALGORITHMS There are two types of slicing algorithms according to the classification presented in Section 2, which are either used i) to improve the state space analysis or ii) to improve testing. Furthermore, the proposed algorithms vary with respect to the class of Petri nets for which they are designed, meaning that some are only applicable to low-level Petri nets and some other only to high-level Petri nets. Bearing this classification in mind, a discussion about the slicing algorithms designed to improve model checking for different classes of Petri nets (as shown in Fig. 4) are presented in the next section. Later, a similar discussion is followed about the slicing algorithms proposed to improve testing. Fig. 4. Static Slicing algorithms designed to improve model checking As has already been noted, a particular convention is followed to study the slicing algorithms in this survey. At first, the objective of slicing algorithm is given (whether it is designed to improve model checking or testing). Secondly, with the help of a process-flow diagram, different steps required to generate sliced net are reviewed. Finally, the proposed algorithm is explained and evaluated by taking a simple example of a Petri net model by examining a specified property. 5.1 CTL*−X and Safety Slicing Algorithms 5.1.1 CTL*−X : Astrid Rakow presented the first slicing algorithm to improve the model checking of Petri nets [30]. The basic idea of CTL*−X slicing algorithm is to reduce a Petri net model in such a way that the reduced model contains only that part of the model that is sufficient to verify a given ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:7 property. Figure 5 shows the process-flow diagram that highlights the central idea of the CTL*−X algorithm, where the criterion places are extracted from the temporal description of properties. Then, these criterion places

are used to slice a Petri net model and finally the state space is generated to verify the examined properties. Astrid Rakow introduced the notion of reading and non-reading transitions to generate smaller sliced model (see definition 4.3). Informally, reading transitions are those transitions that do not change the markings of a place, while non-reading transitions are the transitions that change the markings of a place (as shown in Fig. 6). The idea is to include only non-reading transitions in the sliced Petri net model and exclude possible reading transitions. Excluding the reading transitions and including the non-reading transitions can certainly reduce the size of a slice and it is proved that there is no impact on the verification of results. SLICING PETRI NET MODEL PETRI NET MODEL PROPERTY MODEL CHECKING SLICED PETRI NET MODEL PROEPERTY FULFILLED NO NOTIFICATION YES EXTRACTING CRITERION PLACES Fig. 5. Process-flow diagram: Astrid Rakow's Slicing Approach P 1 1 P 1 2 Reading Transition Non Reading Transition Fig. 6. Reading and Non-reading transitions of a PN It is important to note that there are two restrictions with respect to the verification by a sliced Petri net model. The first one is on the formulas and the other one on the set of admissible firing sequences in terms of fairness assumptions. A Petri net model has a more intrinsic behaviour to slicing, as intentionally not all the behaviours are captured. Fairness assumptions help to restrict the non-captured behaviour, so that verification of formulas without next-time operator becomes possible. The CTL*−X algorithm takes as input a Petri net (PN) and the criterion places (Crit), which are extracted from the temporal description of properties. The algorithm iteratively builds the sliced net by taking all the incoming and outgoing transitions together with their input places. The CTL*−X algorithm starts with the GenerateSlice function, which takes as input a Petri net model and a set of criterion places. T ′ and P ′ are the sets of transitions and places in the sliced Petri net respectively. All the non-reading transitions (together with their input places) starting from the criterion places are added in the sliced net until all the places which contribute tokens to ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:8 Y.I. Khan, A. Konios and N. Guelfi. the criterion places. The resultant sliced Petri net model contains only those places and transitions that contribute tokens to the criterion places. Then, the obtained sliced Petri net model is used to verify the given properties. ALGORITHM 1: CTL*−X Slicing GenerateSlice(⟨P,T, f , λ,m0⟩,Crit){ T ′ ← ∅; /* representing set of transitions in the slice */ Pdone ← ∅; /* representing set of place that are already included in the slice. */ P ′ ← Crit ; /* representing set of places in the slice */ while (∃p ∈ (P ′ \ Pdone)) do while (∃t ∈ (•p ∪ p•) \ T ′) : λ(p,t) , λ(t,p)) do P ′ ← P ′ ∪ •t; T ′ ← T ′ ∪ {t }; end Pdone ← Pdone ∪ {p}; end return ⟨P ′ ,T ′ , f|P′ ,T′ , λ|P′ ,T′ ,m0|P ′; } 5.1.2 Safety Slicing: Astrid Rakow presented another slicing algorithm to improve model checking of Petri nets. The Safety slicing algorithm focuses on the preservation of stutter-invariant linear time safety properties. In contrast to CTL*−X , the Safety slicing algorithm iteratively takes into consideration only the transitions that increase the token count in the sliced net places. The reason why the Safety slicing algorithm can produce a reduced sliced model for safety properties is due to the fact that the satisfiability of safety properties can already be determined by inspecting finite prefixes of traces of the transition system of a Petri net model. Remark that the Safety slicing algorithm does not preserve liveness properties. ALGORITHM 2: Safety Slicing GenerateSlice(⟨P,T, f , λ,m0⟩,Crit){ T ′ ← {t ∈ T | ∃p ∈ Crit : λ(p,t) , λ(t,p)}; /* representing set of transition in the slice by considering non-reading transitions */ P ′ ← •T ∪Crit ; /* representing set of places in the slice starting from criterion places */ Pdone ← Crit; /* representing set of place that are already included in the slice. */ while (∃p ∈ (P ′ \ Pdone)) do while (∃t ∈ (•p \ T ′) : λ(p,t) < λ(t,p)) do P ′ ← P ′ ∪ •t; T ′ ← T ′ ∪ {t }; end Pdone ← Pdone ∪ {p}; end return ⟨P ′ ,T ′ , f|P′ ,T′ , λ|P′ ,T′ ,m0|P ′; } The Safety slicing algorithm follows the same construction methodologies as CTL*−X . The main difference between these constructions is the selective inclusion of transitions. In the Safety slicing algorithm, only those transitions (together with their incoming places) that are producing more tokens to the places than they are consuming are added in the sliced Petri net model. Let's take a simple example of a Petri net model and a property to show the application of CTL*−X and Safety slicing algorithms respectively. Considering the example of the Petri net model ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:9 shown in Fig. 7(A), the property to be verified expresses that (ϕ1 = P3 , ∅ is never empty). Using this property, the evaluation of both slicing algorithms can be performed. As discussed above, the Safety slicing algorithm works only for the safety properties whereas the scope of CTL*−X is more generic and it can be used for both liveness and safety properties. Figure 7 shows the resultant sliced Petri net models by applying Basic, CTL*−X and Safety slicing algorithms respectively. The basic slicing algorithm is similar to (Alg. 7) with a slight difference of adding all the pre and post places of transitions included in the slice. Petri Net After applying Basic Slicing After applying CTL*-X Slicing After applying Safety Slicing P1 1 1 1 1 P21 1 P3 P4 1 1 1 1 t1 t2 t3 t6 t5 t4 P5 P6 1 1 P1 1 1 1 1 P21 1 P3 1 1 t1 t2 t3 t5 t4 P5 P6 1 1 P1 1 1 P21 1 P3 1 1 t1 t2 t5 t4 P6 1 P1 1 1 P21 1 P3 1 t1 t5 t4 Fig. 7. An example Petri net model and its sliced Petri net models by applying

A.Rakow's proposed algorithms Let's now compare the number of states required to verify the given property (ϕ1) without slicing and after applying different slicing algorithms. The first column of Table 3 shows the property ϕ1 and the second one presents the total number of states required to verify the property without slicing. Properties Total States Basic Slicing CTL*−X Safety Slicing ϕ1 60 24 12 9 Table 1. Comparison of slicing algorithms proposed by Astrid Rakow. Similarly the third, fourth and fifth columns show the number of states that are reduced by applying the Basic, CTL*−X and Safety slicing algorithms respectively. The results clearly indicate that the Safety slicing algorithm is more aggressive in terms of reducing the number of states required to model check the given property but it can only be used for safety properties. 5.2 APNSlicing and Abstract slicing algorithms 5.2.1 APNSlicing: The first slicing algorithm introduced to improve the model checking of Algebraic Petri nets (a variant of high- level Petri nets) was presented by Khan et al [22]. The construction methodology of the slicing algorithms proposed by Khan et al is fairly similar to the ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:10 Y.I. Khan, A. Konios and N. Guelfi. construction methodology proposed by Astrid Rakow. One limitation of the slicing algorithms proposed by Astrid Rakow is that they cannot be applied directly to Algebraic Petri nets or any other variant of high-level Petri nets. However, the basic slicing algorithm can be directly applied to Algebraic Petri nets, but the sliced net would not be optimal. As has already been discussed, Astrid Rakow introduced the notion of reading and non-reading transitions to generate reduced sliced net. In Algebraic Petri nets, reading transitions cannot be determined straightforwardly (see Fig.8). [1] t1 P x 1 y Syntactically and semantically reading transition Syntactically non-reading but semantically reading transition [1] t1 P x 1 x x=y Fig. 8. Reading transition of APN The process-flow diagram shown in the Fig. 9 gives an overview of the proposed approach by Khan et al. The first step is to unfold the Algebraic Petri net model to know the ground substitutions of the variables such that reading and non-reading transitions could be identified3 . They used a particular unfolding technique developed by SMV group i.e., a partial unfolding [8]. However, any unfolding technique can be used to identify reading transitions. UNFOLDING APN-MODEL APN-MODEL PROPERTY SLICING UNFOLDED APN-MODEL MODEL CHECKING PROPERTY FULFILLED? NOTIFICATION COUNTER EXAMPLE NO YES REFINING APN-MODEL EXTRACTING CONCERNED PLACE(S) Fig. 9. Process-flow diagram: Khan et al Slicing Construction 3Note that details of unfolding are not discussed as it is out of the scope of this article. Interested readers can find more details about unfolding in [8]. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:11 ALGORITHM 3: APN Slicing APNSlice($\langle$SPEC, P,T, f , as$_д$,cond, λ,m0$\rangle$,Crit){ $T' = \{t \in T \mid \exists p \in Crit : t \in (\bullet p \cup p\bullet) : \lambda(p,t) , \lambda(t,p)\}$; /* representing set of transitions in the slice */ $P' = Crit \cup \{\bullet T'\}$; /* representing set of places in the slice */ Pdone = $\emptyset$; /* representing set of place that are already included in the slice. */ while (($\exists p \in (P' \setminus Pdone)$) do while ($\exists t \in (\bullet p \cup p\bullet) \setminus T'$) : $\lambda(p,t)$, $\lambda(t,p)$) do $P' = P' \cup \bullet t$; $T' = T' \cup \{t\}$; end Pdone = Pdone $\cup \{p\}$; end return $\langle$SPEC, P' ,T' , f|P' ,T' , as$_д$|P' ,cond|T' , λ|P' ,T' ,m0|P' $\rangle$; } The slicing algorithm starts by taking an unfolded Algebraic Petri net model and a set of the criterion places. Initially T' (representing transitions set of the slice) contains the set of all pre and post transitions of the given criterion place, which correspond to non-reading transitions only. P' (representing the places set of the slice) contains all the preset places of the transitions in T'. The algorithm then iteratively adds other preset transitions together with their preset places in T' and P'. 5.2.2 Abstract Slicing: Khan et al, in [21], proposed another slicing algorithm to improve model checking of Algebraic Petri nets. The Abstract Slicing algorithm can also be applied to low-level Petri nets with slight modifications. They extend the previous slicing proposals of Rakow and Khan et al by introducing a new notion, the neutral transitions. The Abstract Slicing algorithm preserves properties expressed in CT L$*$ −X formulas. [1] t1 P x 1 y Syntactically and semantically reading transition Syntactically non-reading but semantically reading transition [1] t1 P x 1 x x=y Syntactically and semantically neutral transition Syntactically non-neutral but semantically neutral transition [1] t1 x P1 x x=y [] P2 [1] t1 x P1 y [] P2 Fig. 10. Neutral and Reading transitions in APNs Informally, a neutral transition consumes and produces the same token from its incoming place to an outgoing place. The cardinality of the incoming arcs of a neutral transition is strictly equal to one and the cardinality of the outgoing arcs from an incoming place of a neutral transition is equal to one as well. Another restriction is that the cardinality of the outgoing arcs from the incoming place of a neutral transition is strictly equal to one and the reason is to preserve all possible behaviours of the net (shown in Fig. 10). Some behaviours could be lost when incoming and outgoing places are merged if more outgoing arcs are allowed from the incoming place of a neutral transition. The idea is to use reading transitions and neutral transitions to generate a smaller sliced net, as shown in Fig. 11. READING TRANSITIONS NEUTRAL TRANSITIONS ABSTRACT SLICING Fig. 11. Construction methodology of Abstract slicing algorithm ALGORITHM 4: Abstract slicing algorithm AbsSlicing($\langle$SPEC, P,T, F, as$_д$,cond, λ,m0$\rangle$,Crit){ $T' \leftarrow \{t \in T / \exists p \in Crit \wedge t \in (\bullet p \cup p \bullet) \wedge \lambda(p,t)$,

λ(t,p)}; /* representing set of transitions in the slice */ P ′ ← Crit ∪ {•T ′ } ; /* representing set of places in the slice */ Pdone ← ∅ ; /* representing set of places that are already included in the slice */ while ((∃p ∈ (P ′ \ Pdone)) do while (∃t ∈ ((•p ∪ p •) \ T ′) ∧ λ(p,t) , λ(t,p)) do P ′ ← P ′ ∪ {• t }; T ′ ← T ′ ∪ {t }; end Pdone ← Pdone ∪ {p}; end while (∃t∃p∃p ′ /t ∈ T ′ ∧ p ∈ • t ∧ p ′ ∈ t • ∧ |• t| = 1 ∧ |t • | = 1 ∧ |p • | = 1 ∧p < Crit ∧ p ′ < Crit ∧ λ(p,t) = λ(t,p ′)) do m(p ′) ← m(p ′) ∪ m(p); while (∃t ′ ∈ • p/t ′ ∈ T ′) do λ(p ′• ,p) ← λ(p ′• ,p ′) ∪ λ(t ′ ,p); end T ′ ← T ′ \ {t ∈ T ′ /t ∈ p • ∧ t ∈ • p ′ }; P ′ ← P ′ \ {p}; end return ⟨SPEC, P ′ ,T ′ , F |P′ ,T′ , asд|P′ ,cond|T′ , λ|P′ ,T′ ,m0|P′ ⟩; } The Abstract slicing algorithm starts with an unfolded APN and a slicing criterion Crit ⊆ P containing criterion place(s). In this algorithm, initially T ′ (representing transitions set of the slice) contains a set of all the pre and post transitions of the given criterion places, which correspond to the non-reading transitions only. P ′ (representing the places set of the slice) contains all the preset places of the transitions in T ′ . The algorithm then iteratively adds other preset transitions together with their preset places in ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:13 T ′ and P ′ . Finally, the neutral transitions are identified and their pre and post places are merged to one place together with their markings. Algebraic Petri net [1,2] t1 t2 t3 y x x x x A D B x x=y [1,2] t0 x x C [1,2] B t21 t23 t22 D t33 C 3 1 2 3 1 2 3 1 2 3 t32 t31 2 1 t11,1 t12,2 t13,3 1 1 2 2 3 3 Unfolded Algebraic Petri net t21 t23 t2 [1,2] 2 A 1 2 3 1 2 3 After applying APNslicing [1,2] B t21 t23 t22 D t33 C 3 1 2 3 1 2 3 1 2 3 t32 t31 2 1 t21 t23 t22 [1,2] A 1 2 3 1 2 3 [1,2 1,2] AB t21 t23 t22 D t33 C 3 1 2 3 1 2 3 1 2 3 t32 t31 2 1 After applying Abstract slicing E [1] E [1] x 1 2 3 Fig. 12. An APN and its sliced unfolded APN by applying APNslicing algorithm Considering the example of the Algebraic Petri net model shown in Fig. 12, let's take the example property ϕ2 = AF(C , ∅) (informally, it means that 'eventually place C will not be empty') and apply APNSlicing and Abstract slicing algorithms respectively. Afterwards, the comparison of the reduction in terms of state space is presented. The first column of Table 2 shows the property ϕ2 which is to be model checked. The second column presents the total number of states required to verify the property without slicing. Similarly, the third and fourth column show the number of states that are reduced by applying APNSlicing and Abstract slicing algorithms respectively. The results clearly indicate that the Abstract slicing algorithm is more aggressive in terms of reducing states to model check the given property. Properties Total States APNSlicing Abstract Slicing ϕ2 162 81 36 Table 2. Comparison slicing algorithms proposed by Khan et al. 5.3 Lee et al Slicing Algorithm Lee et al proposed first slicing algorithm for performing compositional verification. The idea is to develop partitioning criteria and using slicing algorithm Petri nets model is divided into meaningful and manageable modules. Then, compositional analysis technique is applied for verifying boundedness and liveness properties. The main objective proposed of slicing algorithm is to find concurrent units and to divide a huge Petri nets model into slices. The resultant slices preserve the behaviours of original model by using concurrent units. The process-flow diagram shown in the Fig.13 gives an overview of the proposed approach by Lee et al. The first step is to slice given Petri net model into concurrent units and then reachability graphs are generated for each unit. Finally, through compositional reachability analysis, properties are verified. Fig. 13. Construction methodology of Lee et al slicing algorithm ALGORITHM 5: Lee et al Slicing SliceSet ← ∅ ;/* representing set of computed slices */ SOI; /* representing set of invariants */ SOI ← FindMinimalInvariants(N); while (Place(SOI) ⊆ Place(SliceSet) ∨ Place(PN) == Place(SliceSet)) do SmallInvariant ← f indSmallestInvariant(SOI); SliceSet ← SliceSet ∪ {SmallInvariant }; SOI ← SOI \ {SmallInvariant }; end while (Place(SliceSet) , Place(PN)) do U ncoveredPlaceSet ← Place(PN) \ Place(SliceSet); (∀p ∈ U ncoveredPlaceSet) slice ← FindMinimallyConnected(SliceSet,p); siice ← slice ∪ {p}; end Lee et al algorithm starts with an empty set representing resultant slices. For the given Petri net model, minimal invariants are computed using FindMinimalInvariants(N) function. Small invariants i.e., minimal number of elements are added into the slice set until all the places are covered or there is no invariant which includes a new place in the given Petri net model. Fig. 14(b) and 14(c) show the resultant slices by applying the slicing algorithm to the model Fig. 14(a). These slices represent different behaviours of the net and there is no uncovered place. It is important to note that lee et al slicing algorithm does not directly reduce the state space, therefore, state space alleviation is not included. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:15 Fig. 14. Petri net model and its slices 6 DYNAMIC SLICING ALGORITHMS In this section, a study of the basic algorithms for dynamic PN slicing is presented consulting the existing literature [9, 19, 26, 35] (as shown in Fig. 15). The dynamic slicing algorithms give more reduced sliced Petri nets because only the dependencies that occur in a specific execution of the examined Petri net model are taken into account. Fig. 15. Dynamic Slicing algorithms ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:16 Y.I. Khan, A. Konios and N. Guelfi. 6.1 Chang et al Slicing Chang et al presented the first slicing algorithm for Petri nets in the context of testing [9]. The presented algorithm slices out all the

concurrency set of a Petri net model. The concurrency set is defined as a set of paths in different processes that could be executed concurrently. Based on the information about which parts of the system would be executed, test input data can be generated. The algorithm takes as input the set of concurrent Petri net models, which is ProcessPN[1], ..., ProcessPN[N], and produces an output of the concurrency set, which is S[1], S[2], . . . , S[I]. The important sets required to extract the concurrency set are defined as follows: The algorithm first finds a base path that covers at least one communication transition (denoted by CS) and adds it into the concurrency set (denoted by S[I]). To select a path which covers all the marked transitions each process is scanned. The path may generate new communication transitions that have relations with the previous process (i.e., which has been scanned) or the succeeding process that has not been scanned yet. If this path does not involve any new communication transitions having relations with the previous processes or these transitions are already in the concurrency set, then this path is added into the concurrency set and the transitions having relations with the succeeding processes are marked. Otherwise, if this path involves new communication transitions having relations with a previous process, say x, a new path needs to be found to cover both the marked and temporarily marked transitions. If there is such a path, then replace the one already in the concurrency set with this new path and mark again the transitions in other process. Otherwise, erase temporary marks and try to find a new path other than the old one that was already in the concurrency set. Afterwards, restart the scanning process from x till all the processes have been scanned and a concurrency set has been found. The procedure is repeated until all the communication transitions are included in the certain concurrency set. The procedure named ProcedureScanninд(ProcessPN[1], . . . , ProcessPN[N] : in;CS, Mar,TM, S[I] : in &out) is central to the slicing construction. By executing this procedure once, a concurrency set can be obtained. The formal description of the procedure is skipped and the interested reader is referred to [9] for the formal description of the algorithm. t 11 t 12 t 13 t 21 t 23 t 22 P11 P12 P13 P21 P22 P23 P24 t 11 t 12 t 13 t 21 t 23 t 22 P11 P12 P13 P21 P22 P23 P24 Fig. 16. Example Petri net model and its concurrency set by applying Chang algorithm ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:17 ALGORITHM 6: Chang Slicing ChangSlicing (ProcessPN[1], . . . , ProcessPN[N],CS : in; S[1], S[2], . . . , S[I] : out); CS ← {t|t ∈ T,t is a communication transition}; Mar ← {t|t ∈ T,t has a mark}; TM ← {t|t ∈ T,t has a temporary mark}; W S ← {t|t ∈ CS,t is in current process beinд scanned}; for (j ← 1 to N) do if there exist more than one path in ProcessPN[j] then chanдable[j] ← true; else chanдable[j] ← f alse end I ← 0; terminate ← f alse; end end while (CS , ∅ and terminate = f alse) do I < −I + 1; S[I] ← ∅ ; Mar ← ∅;TM ← ∅; W S ← ∅;W S ← W S ∪ {t }; Procedure_f indpath(ProcessPN[1],W S : in; PA : out) ; S[I] ← S[1] ∪ PA; M ← M ∪ {t|t ∈ ProcessPN[x], x = 1 and t has relation with PA}; ProcedureScanninд(ProcessPN[1], . . . , ProcessPN[N] : in;CS, Mar,TM, S[I] : in & out); CS ← CS − CS ∩ S[I]; end endslicinд; A simple Petri net model (shown in Fig. 16) is taken to apply to it the slicing algorithm proposed by Chang et al (refer to Alg. 6). The path shown with the red doted line is put into the concurrency set. 6.2 Backward, Forward and Trace Slicing 6.2.1 Backward + Forward: Llorense et al presented a slicing algorithm to improve testing for Petri nets in [26]. Llorense et al utilized the initial markings for the first time to generate a smaller sliced net. The basic idea is to generate backward and forward slices respectively for the criterion places and then a resultant sliced net is obtained by their intersection (as shown in the Fig. 17). It is important to note that in the forward slicing algorithm, initial markings are taken into consideration such that only those paths that contribute tokens to the criterion places are added to the sliced net. Whereas, in the backward slicing, initial markings are not utilized. For the clarity of concept, their slicing proposal is divided into three steps. In the first step, the basic algorithm given below computes a backward slice by taking as input a Petri net model and a set of criterion places. Starting from the set of initially marked places, the algorithm proceeds further by checking the enabled transitions. Then, the post set of places is included in the slice. The algorithm computes the paths that may be followed by the tokens of the initial marking. In the third step, both forward and backward slices are intersected to get the resultant slice. By bearing a slight overhead, more reduced slices can be obtained. Consider the Petri net model shown in Fig. 18 to generate a slice for the criterion place B. At first, both the forward and backward slices are computed and the resultant sliced net is obtained. Consequently, test input data can be. BACKWARD SLICING PETRI NET + INITIAL MARKING CRITERION PLACE(S) COMBINE SLICES FORWARD SLICING RESULTANT SLICED PETRI NET Fig. 17. Llorens et al Slicing Construction generated for the sliced net, which is less than the test input data generated for the whole Petri net model. ALGORITHM 7: Bakward Slicing GenerateBackwardSlice(⟨P,T, f , λ,m0⟩,Crit){ T ′ ← ∅; /* representing set of transitions in the slice */ P ′ ← Crit ; /* representing set of places in the slice */ while (•P ′ , T ′) do T ′ ← T ′ ∪ •P ′ ; P ′ ← P ′ ∪ •T ′ ; end return ⟨P ′ ,T ′ , f|P′ ,T′ , λ|P′ ,T′ ,m0|P ′; } Similarly for the Forward Slicing, where the algorithm starts from the criterion place and iteratively includes all the incoming transitions together

with their input places until reaching a fixed point. In the second step, a forward slicing is computed by the following algorithm. ALGORITHM 8: Forward Slicing GenerateForwardSlice(⟨P,T, f , λ,m0⟩){ T ′ ← {t ∈ T | m0[t⟩}; /* representing set of transitions in the slice */ P ′ ← {p ∈ P | m0(p) > 0} ∪T ′•; /* representing set of places in the slice */ Tdo ← {t ∈ T \ T ′ | •t ⊆ P ′ }; /* representing set of transitions that are enabled */ while (Tdo , ∅) do T ′ ← T ′ ∪Tdo ; P ′ ← P ′ ∪T •do ; Tdo ← {t ∈ T \ T ′ | •t ⊆ P ′ } end return ⟨P ′ ,T ′ , f|P′ ,T′ , λ|P′ ,T′ ,m0|P ′; } 6.2.2 Trace Slicing: This is the second slicing algorithm introduced by Llorens et al. The rationale behind it is that by fixing the firing sequences, a smaller slice net can be generated. The algorithm is defined by an auxiliary function and takes as input the initial markings together with the firing sequence denoted by σ and the set of places Q of the slicing criterion. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:19 Petri net Model Resultant Forward Slice Resultant Backward Slice Backward Slice Forward Slice \ D A B E C F t2 t1 t3 t4 t5 A B E C F t1 t3 t4 t5 D A B t2 t1 A B t1 Fig. 18. Petri net model and its sliced net by applying Llorens slicing algorithms For a particular marking, a firing sequence σ and a set of places Q, the function slice just moves backwards if there is no place in Q and increases its tokens by the considered firing sequence if the places are not initially marked. Otherwise, the fired transition ti increases the number of tokens of some place in Q and in this case, the function slice already returns this transition ti and, moreover, it moves backwards by also adding the places in •ti to the previous set Q. When the initial marking is reached, the function slice returns the accumulated set of places. Unfortunately, both techniques are not evaluated to case studies which is a big question mark on the usefulness and practicality. ALGORITHM 9: Trace Slicing slice(m0, σ,Q) = □□□□□ □□□ □ Q if i = 0, slice(m0, σ,Q) if ∀p ∈ Q.m0−1(p) ≥ m0(p) ,i > 0 {ti } ∪ slice(m0, σ,Q ∪ •ti) if ∃p ∈ Q.m0−1(p) < m0(p) ,i > 0 6.3 Wangyang Slicing Wangyang et al presented another slicing algorithm to improve testing [35]. Later on, they proposed some improvements in [38]. The basic idea of the proposed algorithm in [35] is similar to the algorithm proposed by Lloren et al [26]. At first, for both algorithms, a backward slice (see Alg. 7) is computed for a given criterion place(s). Secondly, in case of Llorens et al, a forward slice is computed for the complete Petri net model, whereas in case of Wangyang et al, a forward slice is computed for the resultant Petri net model obtained from static backward slice. Let's suppose that there are n number of places in a Petri net model. Now, after applying the backward slicing algorithm, let's assume that there are n/2 number of places. The algorithm of Llorens et al will compute the forward slice for n number of places whereas Wangyang et al algorithm will compute the forward slice only for n/2. The algorithm starts by taking a backward sliced Petri net model ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:20 Y.I. Khan, A. Konios and N. Guelfi. and produces a local reachability graph LRG for the Petri net model. LRG is a directed graph, where its node set is the set of places and the mark of an arc is a transition. From the initially marked places a root node is constructed and then the enabled transitions are added together with their places. If the old node can contribute tokens to new ones, then the LRG(PN ′) can be obtained by tracking the backward static slice forward and the parts associated with slicing criterion under the initial marking m0. Finally, the backward slice can be obtained being coupled with the initial marking and corresponding flow relation. Consider the example of the Petri net model shown in Fig. 19, where its resultant sliced net for the criterion place B consists of transition t1 and its incoming place. ALGORITHM 10: Wangyang Slicing MP ← {p ∈ P ′ | m0(p) > 0}, /* be the root node, and mark with ''New'' */ while MP , ∅ do Choose an arbitrary New node as MP ′ ; if MP ′• ← ∅ then mark MP ′ with Terminate node; Return to while loop; end make that every place p ∈ MP ′ has a token; if there does not exist t ∈ T ′ and is enabled under this situation then mark B ′ with Terminate node; Return to while loop; end else if there is no transition set then Tl ⊆ T ′ and is enabled under this situation for t ∈ Tl do Compute a new set of places MP ″ = MP ′ \ •t ∪ t•; if MP ″ exists in LRG(PN ′) then create a directed edge from MP ′ to MP ″, mark the edge with t; else if MP ″ does not exist in LRG(PN ′) then create a new node MP ″ end and create a directed edge from MP ′ to MP ″, mark edge with t; end Mark MP ″ with "New"; end end Remove mark "New" of MP ′ ; end Repeat 6.4 Concerned Slicing Khan et al presented a slicing algorithm to improve the testing for Algebraic Petri nets (APNs) [19]. Its objective is to generate a sliced net with those places and transitions of the APN model that can contribute to the marking change of a given criterion place in any execution starting from the initial marking. In the introduced concerned slicing algorithm, the available information about the initial marking is utilized and it is directly applied to APNs instead of their unfolding. Starting from the criterion place, the algorithm iteratively includes all the incoming transitions together with their input places until reaching a fixed point. Then, starting from the set of initially marked places, the algorithm proceeds further by checking the enabled transitions. Then, the post ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:21 Petri net Model Resultant Slice D A B E C F t2 t1 t3 t4 t5 A B t1 Fig. 19. Petri net model and its sliced net by applying Wangyang slicing algorithms set of these places are included in the slice. The

algorithm computes the paths that may be followed by the tokens of the initial marking. ALGORITHM 11: Concerned slicing algorithm ConcernedSlicing(⟨SPEC, P,T, F, asд,cond, λ,m0⟩,Crit){ T ′ ← ∅; /* representing set of transitions in the slice */ P ′ ← Crit ; /* representing set of places in the slice starting from criterion places */ while (•P , T ′) do P ′ ← P ′ ∪ • T ′ ; T ′ ← T ′ ∪ • P ′ ; end T ″ ← {t ∈ T ′ /m0[t⟩}; P ″ ← {p ∈ P ′ /m0(p) > 0} ∪T ″• ; Tdo ← {t ∈ T ′ \ T ″/ • t ⊆ P ″}; while (Tdo , ∅) do P ″ ← P ″ ∪T • do ; T ″ ← T ″ ∪Tdo ; Tdo ← {t ∈ T ′ \ T ″/ • t ⊆ P ″}; end return ⟨SPEC, P ″ ,T ″ , F |P″ ,T″ , asд|P″ ,cond|T″ , λ|P″ ,T″ ,m0|P″ ⟩; } Considering the example of the Algebraic Petri net model shown in Fig. 20(A), for the criterion place D, the resultant sliced APN model is presented in Fig. 20(B). The test input data can be generated for the sliced APN model to observe which tokens are coming to the criterion place. 7 COMPARISON OF STATIC SLICING ALGORITHMS AND GENERAL OBSERVATIONS This section presents a comparison of the static slicing algorithms discussed earlier and some useful general observations with respect to the type, use and performance of both the static and dynamic slicing algorithms. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:22 Y.I. Khan, A. Konios and N. Guelfi. [] A C [] [] x [] [1,2] t1 [1] [1,2] t3 t2 t5 t4 x x x y x y y y y z z z B C E D F G APN-Model [] A x [] [1,2] t1 [1,2] t3 t2 x x x y y z B E D Sliced APN-Model Fig. 20. The sliced APN model (by applying concerned slicing) 7.1 Comparison of Static Slicing Algorithms It is worth noting that the comparison presented in this section considers only the static slicing algorithms as the dynamic ones do not contribute to the alleviation of the generated state space of Petri nets, but mostly to the testing process. Furthermore, although Lee et al. slicing algorithm is a static slicing algorithm, it is not included in the comparison as it does not reduce the state space of the examined nets4 . Consequently, the static slicing algorithms that will be compared against the state space of the unsliced Petri net models are the APN, CTL*−X , Abstract and Safety algorithms. To compare these algorithms, two real-world case studies are used as examples, the WSU-CASAS (Washington State University - Center for Advanced Studies in Adaptive Systems) smart home project [14, 16] and the insurance claim system [33]. These case studies are actually used to investigate the 'performance' on the slicing of strongly and non-strongly connected Petri nets respectively. Thus, it is examined how the application of the different slicing algorithms on these two types of nets affects the total number of states, edges, places and transitions required for the model checking and construction of the analysed model. Specifically, the efficiency of those algorithms is compared with respect to the reduction of the state space against which the examined system properties will be verified in each case. The properties used are actually liveness and safety properties expressed in Computation Tree Logic (CTL) [10]. 7.1.1 WSU-CASAS: A Strongly Connected Net Case Study. For the slicing of the strongly connected Petri nets, the example that is used to compare the static slicing algorithms refers to the behavioural model of one of the apartments that were used for the WSU CASAS smart home project [13]. This smart apartment is equipped with different kind of sensors like motion (M), item (I), stove (A001), hot/cold water (A002), door (D) and refrigerator door (R) sensors, which are all captured in the Petri net model shown in Fig. 21. The smart apartment of the WSU CASAS project intends to identify the normal daily activities performed by its resident(s). Some of these daily activities include the meal preparation, personal hygiene, go to toilet, watch TV, sleep, etc. The identification of all these activities requires the analysis of the data collected from the system's sensors in order to detect behavioural patterns that are based on different sequences of activated sensors each time that the environment interacts with 4 It is worth mentioning that this algorithm is mostly used in compositional verification. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:23 the user(s). To analyse the behaviour of the system with respect to its specification (i.e. through model checking), the sensors' dependencies have been considered in the Petri net model of the smart apartment, illustrating in this way their location in it and the sequence of activated sensors that can be followed while a user's activity or action is detected by them. Also, it is worth mentioning that the modelling process considers that the resident(s) can have the initiative and perform any of the daily activities without being intercepted by the system. Now, having described the operation of the smart apartment, the comparison of the static slicing algorithms takes place by examining two system properties that are linked to the activities or sub-activities that can be detected and supported by the smart environment. The system properties that are verified consist of a liveness and safety property respectively. The liveness property that is used for the model checking and slicing of the CASAS smart apartment model is that "a user can eventually use one of the items in the cupboards while he/she is in the kitchen". The temporal logic representation of which can be expressed as follows: φ1 = EF((M016 ∨ M017 ∨ M018) ⇒ (D007 ∨ D014 ∨ D015 ∨ D016)) This property is actually used to examine whether the sensors are eventually activated or not when somebody is in a specific room. Further, the safety property that is used for the comparison of the algorithms examines whether two activities can be performed by the same user at the same time. Expressly, a property that could indicate this is that "a user cannot use the tap and the stove at the same time". The temporal logic proposition of which is

presented below. φ2 = AG(¬((A002_hot_tap ∨ A002_cold_tap) ∧ A001_Stove)) Once, the temporal logic expressions of two properties have been formed, the slicing process can be conducted using the places mentioned as input criteria to the static slicing algorithms considered. It is noted that for the verification process of the CASAS model, a different slice is created for each specified property. This results from the fact that the criterion places used as input to the static slicing algorithms differ in each case, as it can be noticed from their temporal logic propositions. The obtained slices, produced by each algorithm, are then used in the Charlie model checker [15] to generate the state space required for the checking of these properties. 7.1.2 Insurance Claim System: A Non-strongly Connected Net Case Study. For the non-strongly connected Petri nets, the insurance claim system is considered as the example case for the comparison of the slicing algorithms. The functioning of this system has already been explained in Section 2 describing how the customers' claims are initially received and assessed and then are either approved or rejected. The model of the insurance claim system is shown in Fig. 2. As with the example of the strongly-connected net, the two system properties considered for the model checking and slicing process are a liveness and safety property respectively. The liveness property used is the one presented in Section 2 examining whether "every accepted claim is settled". The temporal logic proposition of which is recalled below: φ1 = AG(ac ⇒ AFcs) Also, the safety property that is required for the comparison of the static slicing algorithms is shown below examining whether there is an upper threshold regarding the settled claims. φ2 = AG(|ac| ≤ 50) Explicitly, φ2 checks if the number of the settled claims is always less or equal to fifty, as this value is set as the upper threshold in this case. Now, the formed temporal logic expressions of the two properties are used for the slicing process, where the criteria places of the static slicing algorithms considered are ac, cs and ac for the φ1 (liveness) and φ2 (safety) property respectively. It is noted that, as in the first example, for the model Fig. 21. Petri net model for CASAS smart apartment ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:25 checking of the insurance claim system, a different slice is created for each specified property as the criterion places used as input to the static slicing algorithms are slightly different in the two temporal logic propositions. Again, the obtained slices, from the application of each static algorithm, are also used in the model checker to generate the state space required for the checking of the insurance claim system properties. 7.1.3 Evaluation of Static Slicing Algorithms. Having produced all the slices and having generated their state space using the Charlie model checker 5 [15], the evaluation of the algorithms is conducted based on criteria related to the state space reduction and the structural representation of the slices, i.e. the total number of states generated, the number of edges that link those states, the number of place and transitions in the produced sliced nets. The evaluation of the efficiency of the compared static slicing algorithms starts with the comparison of the total number of states and edges generated for the produced slice of each algorithm for the property φ1 of both case studies. The number of states and edges required for the verification of this property is shown in Fig. 22. Observing the charts of this figure, it is concluded that the APN and CTL*−X algorithms do not reduce the number of states and edges of the reachability graph of both the CASAS and insurance claim model. This results from the fact that the total number of states and edges of the reachability graph for the slices of these algorithms is equal to the total number of states and edges required for the verification of the 'unsliced' models. For instance, the total number of states and edges for all these models of the CASAS example are 45 and 108 respectively. This implies that there is no alleviation of the state space when the APN and CTL*−X algorithms are applied to strongly connected nets. Similarly for the case of the insurance claim system, where the total number of states and edges produced is 29 and 59 for the entire model and the slices of the APN and CTL*−X algorithms respectively. (a) (b) Fig. 22. Comparison of different static slicing algorithms w.r.t the reduction of (a) states and (b) edges for the φ1 liveness properties. On the contrary, it should be noted that the only static slicing algorithm that can effectively reduce the number of states and edges for both the strongly and non-strongly connected nets is the Abstract algorithm. At this point, it is reminded that the Safety slicing algorithm is not considered in the examination of the liveness properties as this slicing algorithm is applied only to the checking of safety properties. Actually, for the strongly (resp. non-strongly) connected net, the Abstract algorithm reduced the states by approximately 16% (resp. 45%) and the edges by 7% (resp. 46%). This means that compared to the total number of states and edges required for the model checking 5Charlie is a java-based tool with a GUI environment that supports the formal analysis and model checking of different Petri net classes using temporal logic. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. 1:26 Y.I. Khan, A. Konios and N. Guelfi. of the entire CASAS (resp. insurance claim) model, the reachability graph of the sliced model is reduced by 7 states and edges (resp. 13 states and 27 edges). (a) (b) Fig. 23. Comparison of different static slicing algorithms w.r.t the reduction of (a) places and (b) transitions for the φ1 liveness properties. Regarding the size of the produced slices for the liveness (i.e. φ1) properties of the two models, the charts of Fig. 23 show that

the number of places and transitions of the APN and CTL*−X slices is not reduced compared to the number of places and transition of the ‘unsliced’ models of the CASAS and insurance claim systems. Contrariwise, in the case of the strongly (resp. non-strongly) connected net, the Abstract algorithm reduces the number of places by 16% (resp. 25%) and the number of transitions by 7% (resp. 22%) for the slice to be model checked. This means that the places (resp. transitions) of the Abstract slice for the CASAS case are decreased by 7 (resp. 7) compared to the places (resp. transitions) of the entire model. Similarly, the set of places (resp. transitions) of the Abstract slice for the insurance claim system contains 4 places (resp. 4 transitions) less. (a) (b) Fig. 24. Comparison of different static slicing algorithms w.r.t the reduction of (a) states and (b) edges for the φ2 safety properties. Next, the comparison of the slicing algorithms is carried out with respect to the safety properties (i.e. φ2 properties) of the two types of nets described above. As is shown in Fig. 24, the APN and CTL*−X algorithms do not contribute to the reduction of the total number of states and edges as it is equal to the number of states and edges required for the model checking of the ‘unsliced’ CASAS and insurance claim models. For the CASAS (resp. insurance claim) system, the number of states and edges generation for the APN and CTL*−X algorithms is 45 and 106 (resp. 29 states and 59 edges). Moreover, for the case of the strongly connected model, the Safety algorithm seems not to alleviate the state space as it consists of the same number of states and edges as this of the ‘unsliced’ ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:27 CASAS model. On the contrary, the Abstract algorithm is the only one that reduces the number of states and edges through its slicing as its state space comprises 30 states and 91 edges. This could be interpreted as 33% and 14% fewer states and edges respectively. Now, regarding the insurance claim system, it is observed from the charts (a) and (b) of Fig. 24 that the Abstract and Safety algorithms can reduce the number of states and edges for the non-strongly connected net drastically. For instance, the slice created by the Abstract algorithm produces a state space that is roughly 46% smaller than that of the entire model. This derives from the fact that the number of states is reduced by 13 (or 45%) and the number of edges is fewer by 27 (or 46%). It is worth noting that the Safety algorithm performs even better as it reduces the states and edges of the state space of the insurance claim system by 65% and 83% respectively. In actual numbers, this means that the number of states drops from 29 to 10 and the numbers of edges from 59 to 10. (a) (b) Fig. 25. Comparison of different static slicing algorithms w.r.t the reduction of (a) places and (b) transitions for the φ2 safety properties. The slicing algorithms are now compared with respect to the size of the produced slices for the safety (i.e. φ2) properties of the two models. According to the charts of Fig. 25, the number of places and transitions of the APN, CTL*−X and Safety slices is not reduced compared to that of the ‘unsliced’ model of the CASAS example. On the contrary, in the case of the strongly (resp. non-strongly) connected net, the Abstract algorithm reduces the number of places by 33% (resp. 25%) and the number of transitions by 14% (resp. 22%) for the slice to be model checked. This means that the places (resp. transitions) of the Abstract slice for the CASAS model are reduced by 15 (resp. 4) compared to the places (resp. transitions) of the entire model. Similarly, the set of places (resp. transitions) of the Abstract slice for the insurance claim system comprises 4 places (resp. 4 transitions) less. Once again, it is observed that the Safety algorithm performs better than any other slicing algorithm when it is applied to the non-strongly connected net of the insurance claim system. It actually reduces the net place and transitions by 43% and 44% respectively, which implies that the number of places is deducted from 16 to 9 and that of the transitions from 18 to 10. Having applied the static slicing algorithms (i.e. APN, CTL*−X , Abstract and Safety Slicing) to the models of the two example cases, the comparison outcomes show that the Safety slicing algorithm performs better than any other algorithm when it comes to the model checking of the safety properties of non-strongly connected nets, but it does not effectively reduce the state space or the size of the examined model. Finally, it can be concluded that the ‘average slicing efficiency’ of the Abstract algorithm is better than that of the others as it can reduce the state space and the size of the examined model for both the strongly and non-strongly connected Petri nets. 7.2 General Observations for Slicing Algorithms In this section, some useful conclusions about the slicing algorithms are presented, which have derived from the comparison conducted and some general observations with respect to both the dynamic and static slicing algorithms described in this work. It should be mentioned that one major difference between static and dynamic slicing algorithms is their slicing criterion. The static slicing algorithms extract slicing criteria from the temporal description of the properties. These slicing criteria consist of a set of places/transitions, and a slice is generated around them. In Fig. 26, the different static slicing algorithms are highlighted with respect to their generated slice size. It can be observed that safety slicing algorithm may generate the smallest slice as compared to other algorithms (due to the fact that safety properties can be determined by inspecting finite prefixes of traces of the transition system of a Petri net model.), but the scope of safety slicing algorithms is limited to safety properties only. The scope of abstract slicing is broader in

terms of preserved properties as compared to safety slicing. The algorithm adapts the notion of reading transitions to exclude transitions that do not change the markings of concerned places. Also, the inclusion of neutral transitions makes the resultant slice size small by combining places through transitions producing and consuming same amount of tokens. Perhaps, it can be combined with safety slicing to generate smaller slices. The APN slicing and CTL*−X are based on the similar construction methodology, where the objective is to identify and exclude reading transitions. Remark that all the existing slicing algorithms do not allow properties specified with the next time operator. On the other hand, the dynamic slicing algorithms can take directly places or transitions as slicing criteria. Additionally, the following conclusions can be drawn from the evaluation results of the static slicing algorithms: i) the reduction in terms of sliced net can vary with respect to the net structure and markings of the places. The slicing refers to the part of a net that concerns to the examined property, while the remaining part may have more places and transitions that increase the overall number of states. If the slicing removes parts of the net that expose highly concurrent behaviour, the savings may be huge and if the slicing removes dead parts of the net, in which transitions are never enabled then there is no effect on the state space. ii) The choice of the criterion place can have an important influence on the reduction effects, as the basic idea of slicing is to start from the criterion place and iteratively include all the transitions contributing tokens on them together with their incoming places. The fewer transitions are attached to the criterion place, the more reduction is possible. iii) For certain strongly connected nets, slicing may not produce a reduced number of states. For all the strongly connected nets that contain reading transitions, the slicing can produce noteworthy reductions. iv) It has been empirically proved that in general slicing produces best results for work-flow nets. Finally, let's summarise the basic features of the static and dynamic algorithms as these have derived from theoretical and empirical observations and their application. This information is presented in Table 3, where the first column shows the different slicing algorithms under observation. For each algorithm, the table lists i) in which context the slicing algorithm is presented i.e., to improve the testing process or to improve the state space of model checking process, ii) the reduction effect describing i.e., either that the PN model can be reduced or there is no effect of slicing on the model, iii) the properties that are preserved by the slicing construction. As some of the algorithms are designed in the context of testing and their objective is to find a particular trace for the analysis jointly referring to those properties as particular, iv) the slicing type that refers to the construction methodology i.e., either it is static or dynamic (see section 2 for slicing types) and is following backward or forward propagation (or both), v) the time complexity for each construction and vi) whether the algorithm has been implemented or not. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:29 Fig. 26. Slicing algorithms for APNs and PNs w.r.t slice size Algorithm Context Reduction Preserved properties Type slicing T.Comp Impl Lee et al slicing MC × Boundedness and liveness SBS O(N) 3 × Rakow CTL*−X slicing MC √ CTL*−X SBS O(N) Own Rakow Safety slicing MC √ Safety SBS O(N) Own Khan APNSlicing MC √ LTL−X SBS O(N) S LAPn tool Khan Abstract slicing MC √ CTL*−X SBS O(N) S LAPn tool Chang et al slicing Tes × Particular DBS O((n/N))N × Llorens et al APNevo slicing MC/Tes √ Particular DBS/FBS O(T) × Llorense et al trace slicing Tes √ Particular DBS/DFS O(T) × Wangyang et al slicing MC/Tes √ Particular DBS O(T) × Khan Concerned slicing Tes √ Particular DBS/DFS O(T) × Table 3. Overview of PN slicing algorithms features MC = Model checking; Tes = Testing; SBS = Static Backward Slicing; FBS = Forward Backward Slicing; DFS = Dynamic Forward Slicing; DBS = Dynamic Backward Slicing, T.Comp = Time Complexity; Impl = Implementation 8 APPLICATION OF SLICING AND POSSIBLE IMPROVEMENTS Slicing is playing a valuable role in the domain of verification and validation (especially in programs verification) for more than two decades [1, 3, 6, 7, 23]. The major portion of research work on slicing is dedicated to the improvement of slicing algorithms in such a way that verification effort can be minimized. Let's now discuss some applications of slicing in general and some possible improvements to slicing algorithms, which can be adapted. Note that the discussion is restricted to state event model's slicing.. 8.1 Slicing a pre-processing step to model checking: Model checking has been proved to be very useful technique to verify concurrent and distributed systems [12]. The main problem with the practicality of model checking is that it suffers from the state space explosion problem. There are many active research groups working on the alleviation of the state space explosion problem. Slicing, on the other hand, not only can be utilized to produce small input test data, but it could also be used as a pre-processing step to model checking process. In general, to verify a property over a model, the complete state space is generated which can be improved by using slicing. The idea is to generate partial state space by removing the irrelevant parts of the model, meaning keeping only that part of the model that is concerned by the property. It is believed that adding slicing to existing or new model checkers could significantly improve their performance. 8.2 Slicing with other methods for alleviating state space:

Four different methods can be broadly used to alleviate the state space, such as the symbolic representations, the on the fly model checking, the compositional reasoning and the reduction methods. The symbolic methods avoid the state space explosion problem by not explicitly representing the states of the model. McMillan, in his PhD thesis, proposed to use symbolic representation for the state transition graph [27]. He showed that by using symbolic representations such as binary decision diagrams much larger systems can be verified. Later by using original CTL model checking algorithms and their refinements, it was showed that models with more than 10120 states could be verified. Similarly, other methods use different strategies to cope with the state space explosion problem. The results achieved using a symbolic method or others are encouraging but still a long way to go to make model checking practical. If these methods were combined with the slicing, then better results could have been produced. For example slicing with compositional reasoning. The compositional method verifies each component of a system in the isolation and allows global properties to be inferred about the entire system. This method is not only better suited for improving verification, it can also be used to reason about the property satisfaction when an evolution happens to any component of the system. Thus combining compositional reasoning with slicing could provide an effective solution to the verification and re-verification. 8.3 Alleviating repeated model checking: In general, the behavioural models of a system expressed in event based modeling formalism are subject to evolution, where an initial version goes through a series of evolutions generally aimed at improving its capabilities. Considering model checking as a verification technique all the proofs are redone after every evolution which is very expensive in terms of time and memory. To avoid the repeated model checking, a slicing can serve a base step to reason about the preservation of previous properties. Khan et al proposed a slicing based solution in the context of context of Algebraic Petri nets [18]. The central idea is to classify properties and evolutions following their approach, at first, slices are built for the evolved and non-evolved models with respect to the property by the slicing algorithm. For example, if the evolution is taking place outside the sliced of evolved model, then it is obvious that the property is preserved and there is no need to repeat the model checking. Another scenario could be when the evolution is taking place inside the sliced evolved model, then by looking at the evolution and property, it could be determined if the properties are still preserved (as shown in Fig. 27). It is argued that even when the preservation of a property fails to be determined, then model checking can be performed on the sliced model, which is also an improvement to the repeated model checking process. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:31 Model ModelSL Model'SL re-verif Model' NO YES (re-verify on sliced model) build slice build slice evolution Fig. 27. An overview of proposed approach for alleviating state space explosion 8.4 Improving slicing algorithms: This article studied several slicing algorithms, which are designed to improve model checking for different Petri nets classes. These algorithms still have limitations, for example many of them fail to reduce the number of states for strongly connected nets. The reason is the backward dependability and the inclusion of all the transitions together with their incoming places. As discussed in section 5, among others, the abstract slicing algorithm can produce good results even for strongly connected nets. The central idea is to look for those transitions that produce and consume the same number of tokens and merge these places together by summing up their markings together with the reading transitions. + + Parallel Reading Neutral Fig. 28. Parallel transitions an improvement to abstract slicing algorithm Interestingly, this idea can be improved further by looking for parallel transitions having common incoming and outgoing places and eliminating any one of them, as shown in Fig. 28. The slicing algorithm using reading, neutral, parallel transitions can produce significant results even for strongly connected nets. Another drawback of the existing slicing algorithms is their static slicing criterion, which restricts the possible improvement one may have. Almost every slicing algorithm starts from the set of places as criterion and generates the slice around them. One possible direction could be to change the slicing criteria, for example, by building the slices around the transitions instead of the places. It is argued that if timed Petri nets were to be model checked, a criterion based on the transitions along with their time intervals can significantly improve the state space.. 9 CONCLUSION AND FUTURE WORK This article presented a survey of the PN slicing constructions that can be found in the present literature. This work fills the gap for the people who are interested in PN slicing and need more information about the up to date researches. The presented syntactic unification of PN slicing algorithms will facilitate the user to have a clear and easy understanding. By comparing existing PN slicing constructions, it was highlighted that most of them are limited to low-level Petri nets focusing more on reducing the state space explosion problem for the model checking of Petri nets. Very few slicing techniques are described in the context of testing. It was also identified that some possible future directions in this domain could be the description of more refined constructions in the context of testing and more slicing constructions for high-level Petri nets. Furthermore, slicing can serve as a base step towards the reasoning of preserved properties by adding

slicing to existing model checkers as a pre-processing step to see its practical usability. REFERENCES
[1] Hiralal Agrawal and Joseph R Horgan. 1990. Dynamic program slicing. In ACM SIGPLAN Notices, Vol. 25. ACM, 246–256. [2] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. Principles of model checking. MIT press. [3] Jon Beck and David Eichmann. 1993. Program and interface slicing for reverse engineering. In Proceedings of the 15th international conference on Software Engineering. IEEE Computer Society Press, 509–518. [4] Boris Beizer. 2003. Software testing techniques. Dreamtech Press. [5] Antonia Bertolino. 2007. Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering. IEEE Computer Society, 85–103. [6] David Binkley. 1998. The application of program slicing to regression testing. Information and software technology 40, 11 (1998), 583–594. [7] David W Binkley and Keith Brian Gallagher. 1996. Program slicing. Advances in Computers 43 (1996), 1–50. [8] Didier Buchs, Steve Hostettler, Alexis Marechal, Alban Linard, and Matteo Risoldi. 2010. Alpina: A symbolic model checker. Springer Berlin Heidelberg (2010), 287–296. [9] Juei Chang and Debra J. Richardson. 1994. Static and Dynamic Specification Slicing. In In Proceedings of the Fourth Irvine Software Symposium. [10] E. Clarke, O. Grumberg, and D. Long. 1994. Verification tools for finite-state concurrent systems. In A Decade of Concurrency Reflections and Perspectives, J. W. de Bakker, W. P. de Roever, and G. Rozenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 124–175. [11] Edmund M Clarke, Orna Grumberg, and David E Long. 1994. Model checking and abstraction. ACM transactions on Programming Languages and Systems (TOPLAS) 16, 5 (1994), 1512–1542. [12] Edmund M Clarke, Orna Grumberg, and Doron Peled. 1999. Model checking. MIT press. [13] Diane Cook and Maureen Schmitter-Edgecombe. 2009. Assessing the quality of activities in a smart environment. Methods of Information in Medicine 48, 5 (2009), 480–485. [14] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan. 2013. CASAS: A Smart Home in a Box. Computer 46, 7 (July 2013), 62–69. https://doi.org/10.1109/MC.2012.328 [15] M Heiner, M Schwarick, and JT Wegener. 2015. Charlie – An Extensible Petri Net Analysis Tool. In Application and Theory of Petri Nets and Concurrency: 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings, R Devillers and A Valmari (Eds.). Springer International Publishing, Cham, 200–211. https: //doi.org/10.1007/978-3-319-19488-2_10 [16] Yang Hu, Dominique Tilke, Taylor Adams, Aaron S. Crandall, Diane J. Cook, and Maureen Schmitter-Edgecombe. 2016. Smart home in a box: usability study for a large scale self-installation of smart home technologies. Journal of Reliable Intelligent Environments 2, 2 (01 Jul 2016), 93–106. [17] Yasir Imtiaz Khan. 2013. Optimizing Verification of Structurally Evolving Algebraic Petri nets. In Software Engineering for Resilient Systems, V. Kharchenko A. Gorbenko, A. Romanovsky (Ed.). Lecture Notes in Computer Science, Vol. 8166. Springer Berlin Heidelberg. [18] Yasir Imtiaz Khan. 2013. Optimizing verification of structurally evolving algebraic petri nets. In Software Engineering for Resilient Systems. Springer, 64–78. [19] Yasir Imtiaz Khan. 2015. Property based model checking of structurally evolving Algebraic Petri nets. Ph.D. Dissertation. University of Luxembourg. [20] Yasir Imtiaz Khan and Nicolas Guelfi. 2013. Survey of petri nets slicing. Technical Report. lassy. ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018. A Survey of Petri nets Slicing 1:33 [21] Yasir Imtiaz Khan and Nicolas Guelfi. 2014. Slicing high-level petri nets. In International Workshop on Petri Nets and Software Engineering (PNSE'14). 20. [22] Yasir Imtiaz Khan and Matteo Risoldi. 2013. Optimizing Algebraic Petri Net Model Checking by Slicing. International Workshop on Modeling and Business Environments (ModBE'13, associated with Petri Nets'13) (2013). [23] Bogdan Korel and Janusz Laski. 1990. Dynamic slicing of computer programs. Journal of Systems and Software 13, 3 (1990), 187–195. [24] W. J. Lee, H. N. Kim, S. D. Cha, and Y. R. Kwon. 2000. A slicing-based approach to enhance Petri net reachability analysis. Journal of Research Practices and Information Technology 32 (2000), 131–143. [25] Marisa Llorens, Javier Oliver, Josep Silva, and Salvador Tamarit. 2017. An Integrated Environment for Petri Net Slicing. In International Conference on Applications and Theory of Petri Nets and Concurrency. Springer, 112–124. [26] M. Llorens, J. Oliver, J. Silva, S. Tamarit, and G. Vidal. 2008. Dynamic Slicing Techniques for Petri Nets. Electron. Notes Theor. Comput. Sci. 223 (Dec. 2008), 153–165. https://doi.org/10.1016/j.entcs.2008.12.037 [27] Kenneth L McMillan. 1993. Symbolic model checking. Springer. [28] Astrid Rakow. 2008. Slicing Petri nets with an application to workflow verification. In Proceedings of the 34th conference on Current trends in theory and practice of computer science (SOFSEM'08). Springer-Verlag, Berlin, Heidelberg, 436–447. http://dl.acm.org/citation.cfm?id=1785934.1785974 [29] Astrid Rakow. 2011. Slicing and Reduction Techniques for Model Checking Petri Nets. Ph.D. Dissertation. University of Oldenburg. [30] Astrid Rakow. 2012. Safety Slicing Petri Nets. In Application and Theory of Petri Nets, Serge Haddad and Lucia Pomello (Eds.). Lecture Notes in Computer Science, Vol. 7347. Springer Berlin Heidelberg, 268–287. https://doi.org/10.1007/ 978-3-642-31131-4_15 [31] Wolfgang Reisig. 1991. Petri nets and algebraic specifications. Theor. Comput. Sci. 80, 1 (1991), 1–34. [32] F. Tip. 1995. A Survey of

Program Slicing Techniques. JOURNAL OF PROGRAMMING LANGUAGES 3 (1995), 121–189. [33] Wil Van Der Aalst and Kees Max Van Hee. 2004. Workflow management: models, methods, and systems. MIT press. [34] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. 2003. Model checking programs. Automated Software Engineering 10, 2 (2003), 203–232. [35] Yu Wangyang, Yan Chungang, Ding Zhijun, and Fang Xianwen. 2013. Extended and improved slicing technologies for Petri nets. High Technology Letters 19, 1 (2013). [36] Mark Weiser. 1981. Program slicing. In Proceedings of the 5th international conference on Software engineering (ICSE '81). IEEE Press, Piscataway, NJ, USA, 439–449. [37] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. 2005. A brief survey of program slicing. SIGSOFT Softw. Eng. Notes 30, 2 (March 2005), 1–36. https://doi.org/10.1145/1050849.1050865 [38] Wangyang Yu, Zhijun Ding, and Xianwen Fang. 2015. Dynamic slicing of petri nets based on structural dependency graph and its application in system analysis. Asian Journal of Control 17, 4 (2015), 1403–1414. Received August 2017 ACM Computing Surveys, Vol. 1, No. 1, Article 1. Publication date: January 2018.