Property Based Verification of Evolving Petri Nets

Yasir Imtiaz Khan and Ehab Al-shaer

Department of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, USA
Email: ykhan2,alshaer@uncc.edu

***Abstract*—Software evolution is inevitable in the field of information and communication technology systems. Existing software systems continue to evolve to progressively reach important qualities such as completeness and correctness. Iterative refinements and incremental developments are considered to be well suitable for the development of evolving systems among other approaches. The problem with iterative refinements and incremental development is the lack of support of an adequate verification process. In general, all the proofs are redone after every evolution, which is very expensive in terms of cost and time. In this work, we propose a slicing based solution to add an adequate verification process to iterative refinements and incremental development technique. Our proposal has two objectives, the first is to perform verification only on those parts that may influence the property satisfaction by the analyzed model. The second is to classify the evolutions and properties to identify which evolutions require re-verification. We argue that for the class of evolutions that requires re-verification, instead of verifying the whole system only a part that is concerned by the property would be sufficient. We use Petri nets as a modeling formalism and model checking as a verification approach to show the viability of the proposed approach.**

Keywords–Software evolution; Re-verification; Model checking; Iterative refinements; Slicing.

I. Introduction

Software systems are playing an important role in our daily life. Companies are spending millions of dollars and are dependent on them. The software development process does not stop when a system is delivered, but continues throughout the lifetime of software. In general, existing software systems continue to evolve due to various reasons such as the emergence of new requirements, performance may need to be improved, business environment is changing [1] . According to the survey report conducted by Erlikh [2], 90% of software costs are software evolution costs and about 75% of all software professionals are involved in some form of evolution activity. These facts point out the importance of software evolution and demand tools and techniques for its better management.

Iterative refinements and incremental developments is a commonly used technique for handling complex systems in hardware and software engineering and is considered well suitable for software development and managing its evolution. The idea involves creating a new specification or implementation by modifying an existing one [3]. In general, the modeler provides a first model that satisfies a set of initial requirements. Then, the model can undergo several iterations or refinements until all the requirements are satisfied. In most cases, it is desirable for the developer to be able to assess the quality of model as it evolves.

The problem with the iterative and incremental development is that there is no guarantee that after each iteration or evolution of the model, it will still satisfy the previously satisfied properties.

Considering Petri nets as a modeling formalism and model checking as a verification technique all the proofs are redone which is very expensive in terms of cost and time. In this work, we propose a solution to improve the verification and re-verification of evolving systems by re-using, adapting and refining state of the art techniques. Our proposal pursues two main goals, the first is to perform verification only on those parts that may affect the property a model is analyzed for and the second is to classify evolutions, to identify which evolutions require re-verification. We argue that for a class of evolutions that require re-verification, instead of verifying the whole system only a part that is concerned with the property would be sufficient.

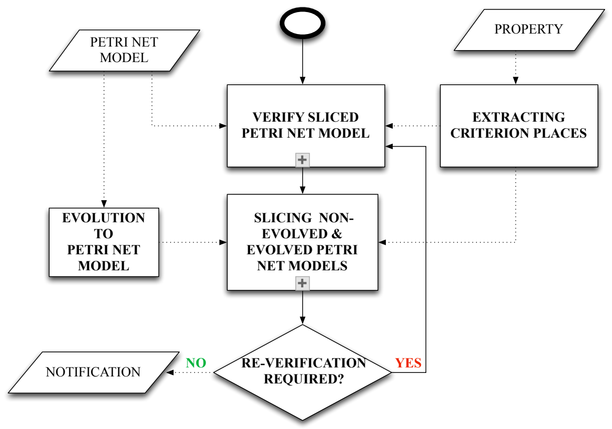
PETRI NET MODEL
PROPERTY
VERIFY SLICED PETRI NET MODEL
EXTRACTING CRITERION PLACES
EVOLUTION TO PETRI NET MODEL
SLICING NON-EVOLVED & EVOLVED PETRI NET MODELS
NOTIFICATION
NO
RE-VERIFICATION REQUIRED?
YES

Figure 1. Process Flowchart property based verification of evolving Petri nets

Figure 1, gives an overview using Process Flowchart of the proposed approach, i.e., a slicing based verification of evolving Petri nets. At first, verification is performed on the sliced Petri net model by taking a property into an account. Secondly, we build slices for evolved and non-evolved Petri nets models. By comparing the resultant sliced models (i.e., Petri net and its evolved model), it is determined if the evolution has an impact on the property satisfaction and if it requires re-verification. In the worst case, if an evolution has an impact on the property satisfaction only the resultant sliced evolved Petri net model would be used for the verification. The process can be iterated as per Petri net evolution.

The rest of the paper is structured as follows: in Section II, we give a informal and formal definition of Petri nets. In Section III, we give formal and informal description of the slicing algorithm and all the steps of slicing based verification of Petri nets. In Section IV, a slicing based solution is given for re-verification of evolving Petri nets. Details about the underlying theory and techniques are given for each activity of the process. In Section V, we discuss related work and a comparison with the existing approaches. In Section VI, we draw the conclusions and discuss future work concerning to the proposed work.

II. Informal and Formal Definition of Petri nets

Petri nets are a very well known formalism to model and analyze concurrent and distributed systems indroduced by C.A. Petri in his Ph.D. Dissertation [4].

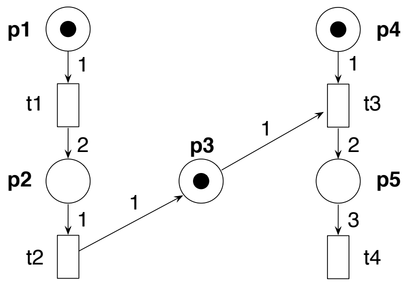

Figure 2. Example of a Petri net model

A Petri net is a directed bipartite graph, whose two essential elements are 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. Formally, we can define :

Definition 1. Petri net: A Petri Net is: $PN = \langle P, T, w, m_0 \rangle$ consist of

∘ P and T are finite and disjoint sets, called places and transitions, resp.,

∘ a function $w : (P \times T) \cup (T \times P) \to \mathbb{N}$, assigns weights to the arcs,

∘ a marking function $m_0 : P \to \mathbb{N}$.

The semantics of a Petri net expresses the *non-deterministic* firing of transitions in the net. Firing a transition means consuming tokens from a set of places linked to the input arcs of a transition and producing tokens into a set of places linked to the output arcs of a transition. A transition can be fired only if its incoming places have a token quantity greater or equal to the weight attached to the arc. As shown in Figure 2, transitions *t1 and t2* are enabled from the initial marking and *non-deterministically* any one of them can fire. Let us consider that if *t1* fires, the result of transition firing will remove a token from place *p1* and adds a token to place *p2*.

Definition 2. (Pre(resp.Post) set places(resp.transitions) of PN): Let $pn =< P, T, f, w, m_0 >$ be a Petri net, $p \in P$ a place then, preset and postset of p, noted $\bullet p$ and $p\bullet$, are defined as follows:

$\bullet p = \{t \in T / w(t, p) > 0\}$.

$p\bullet = \{t \in T / w(p, t) > 0\}$.

Analogously $\bullet t$ and $t\bullet$ are defined. We also note $\bullet P$ and $P\bullet$ representing pre(resp.post) set of transtions of all the places in set P. $\bullet T$ and $T\bullet$ are defined Analogously.

III. Abstract Slicing

Petri net slicing is a syntactic technique, which is used to reduce a PN model based on a given *criteria*. A *criteria* is a property for which the PN model is analyzed for. A sliced part is equal to only that part of a PN model that may affect the *criteria*. Considering a property over PN model, we are interested to define a syntactically smaller PN model that could be equivalent with respect to the satisfaction of the property of interest. To do so the slicing technique starts by identifying the places directly concerned by the property. Those places constitute the *slicing criterion*. The algorithm then, keeps all the transitions that create or consume tokens from the criterion places, plus all the pre-set places for those transitions. This step is iteratively repeated for the latter places, until reaching a fixed point. (It is important to note that the proposed slicing algorithms preserve certain specific properties as we intentionally do not capture all the behaviors to generate a smaller sliced net). Many algorithms are proposed for slicing Petri nets and their main objective is to generate reduced sliced net [5]–[11]. The first slicing algorithm to generate reduced sliced net was proposed by Astrid Rakow by introducing a notion of *reading and non-reading transitions*. Later, this idea was adapted by Khan et al in the context of Algebraic Petri nets (i.e., an advancement of Petri nets) [6], [10].

Informally, *reading transitions* do not change the marking of a place, meaning they consume and produce the same token. On the other hand, *non-reading transitions* change the markings of a place (see Figure 3), meaning they consume and produce different tokens. A reduced sliced net can be generated by discarding the *reading transitions* (as reading transitions do not impact the behavior of Petri net) and to include only *non-reading transitions*. Formally, we can define reading and non-reading transitions:

Definition 6. (Reading(resp.Non-reading) transitions of Petri nets): Let $t \in T$ be a transition in a PN. We call t a reading-transition iff its firing does not change the marking of any place $p \in ({}^{\bullet}t \cup t^{\bullet})$, i.e., iff $\forall p \in ({}^{\bullet}t \cup t^{\bullet}), w(p, t) = w(t, p)$. Conversely, we call t a non-reading transition iff $w(p, t) \neq w(t, p)$.

We extend the slicing proposal of Rakow and Khan et al by introducing a new notion of *neutral transitions*. The abstract slicing algorithm preserves properties expressed in CTL^{*}_{-X} formulas, we refer the interested reader to [12] for the detailed proofs. Informally, a *neutral transition* consumes and produces the same token from its incoming place to an outgoing place. The cardinality of incoming (resp.) outgoing arcs of a neutral transition is strictly equal to one and the cardinality of outgoing arcs from an incoming place of a neutral transition is equal to one as well. Another restriction is that the cardinality of outgoing arcs from the incoming place of a neutral transition is strictly equal to one and the reason is that we want to preserve all possible behaviors of the net. We may loose some behaviors when we merge incoming and outgoing places if we allow more outgoing arcs from the incoming place of a neutral transition. The idea is to use *reading transitions and neutral transitions* to generate smaller sliced net.

Definition 7. (Neutral transitions of Petri nets): Let $t \in T$ be a transition in a PN. We call t a neutral-transition iff it consumes token from a place $p \in^{\bullet} t$ and produce the same token to $p' \in t^{\bullet}$, i.e., $t \in T \wedge \exists p \exists p' / p \in^{\bullet} t \wedge p' \in t^{\bullet} \wedge |p^{\bullet}| = 1 \wedge |^{\bullet}t| = 1 \wedge |t^{\bullet}| = 1 \wedge w(t,p) = w(t,p')$.

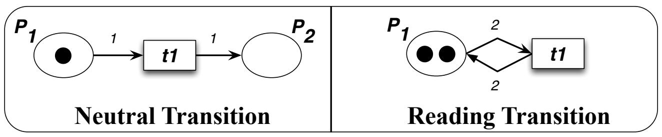

Figure 3. Neutral and Reading transitions of PN

1) Abstract Slicing Algorithm:: The abstract slicing algorithm starts with a Petri net model and a slicing criterion $Q \subseteq P$ containing place(s). We build a slice for an Petri net based on Q by applying the following algorithm:

Algorithm 1: Abstract slicing algorithm

```
AbsSlicing($\langle P, T, f, w, m_0 \rangle, Q$){
$T' \leftarrow \{t \in T / \exists p \in Q \wedge t \in ({}^{\bullet}p \cup p^{\bullet}) \wedge w(p,t) \neq w(t,p)\}$;
$P' \leftarrow Q \cup \{{}^{\bullet}T'\}$ ;
$P_{done} \leftarrow \emptyset$ ;
while  $((\exists p \in (P' \setminus P_{done}))$ do
    while $(\exists t \in (({}^{\bullet}p \cup p^{\bullet}) \setminus T') \wedge w(p,t) \neq w(t,p))$ do
        $P' \leftarrow P' \cup \{{}^{\bullet}t\}$;
        $T' \leftarrow T' \cup \{t\}$;
    end
    $P_{done} \leftarrow P_{done} \cup \{p\}$;
end
while $(\exists t \exists p \exists p' / t \in T' \wedge p \in^{\bullet} t \wedge p' \in t^{\bullet} \wedge |^{\bullet}t| =$
$1 \wedge |t^{\bullet}| = 1 \wedge |p^{\bullet}| = 1$
$\wedge p \notin Q \wedge p' \notin Q \wedge w(p,t) = w(t,p'))$ do
    $m(p') \leftarrow m(p') \cup m(p)$;
    $w(t,p') \leftarrow w(t,p') \cup w(t,p)$;
    while $(\exists t' \in^{\bullet} t / t' \in T'$ ) do
        $w(p',t) \leftarrow w(p',t) \cup w(p,t')$;
        $T' \leftarrow T' \setminus \{t \in T' / t \in p^{\bullet} \wedge t \in^{\bullet} p'\}$;
        $P' \leftarrow P' \setminus \{p\}$;
    end
end
return $\langle P', T', f_{|_{P',T'}}, w_{|_{P',T'}}, m_{0_{|_{P'}}} \rangle$;
}
```

In the Abstract slicing algorithm, initially T' (representing transitions set of the slice) contains a set of all the *pre and post* transitions of the given criterion places. Only the *non-reading transitions* are added to T'. 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 the T' and P'. Then, the *neutral transitions* are identified and their *pre and post* places are merged to one place together with their markings.

Considering an example Petri net model shown in figure 4, let us now apply our proposed algorithm on two example properties (i.e., one from the class of *safety* properties and one from *liveness* properties). Informally, we can define the properties:

ϕ_1 : *"The cardinality of tokens inside place P3 is always less than 5".*

ϕ_2 : *"Eventually place P3 is not empty".*

Formally, we can specify both properties in the *CTL* as:

$\phi_1 = \mathbf{AG}(|m(P3)| < 5)$.

$\phi_2 = \mathbf{AF}(|m(P3)| = 1)$.

For both properties, the slicing criterion $Q = \{P3\}$, since *P3* is the only place concerned by the properties. The resultant sliced Petri net can be observed in figure4, which is smaller than the original Petri net.

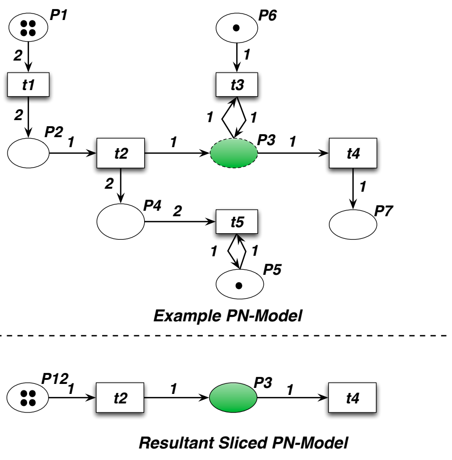

Figure 4. Petri net model and resultant sliced model after applying Abstract slicing algorithm

Let us compare the number of states required to verify the given properties without slicing and after applying abstract slicing. The total number of states required without slicing is ***985***, whereas with the sliced model number of states is ***15***.

IV. Classifications of Evolutions

The behavioral model of a system expressed in terms of Petri nets is subject to evolve, where an initial version goes through a series of evolutions generally aimed at improving its capabilities. Informally, Petri nets can evolve with respect to the structural changes such as: *add/remove places, transitions, arcs, tokens and terms over the arcs*. By notation, different Petri nets will be noted with superscripts such as $pn' = \langle P', T', f', w', m'_0 \rangle$. As there is no guarantee that after every evolution of a Petri net model, it still satisfies the previously satisfied properties. A naive solution is to repeat model checking after every evolution, which is very expensive in terms of time and space.

We propose a slicing based solution to improve the repeated model checking. Since it has already been proved that a sliced net is sufficient to verify properties. (Note: We refer the interested reader to [10], [12], [13] for the detailed proofs of all the theorems used in this paper and for slicing algorithms). According to our proposed approach, at first, slices

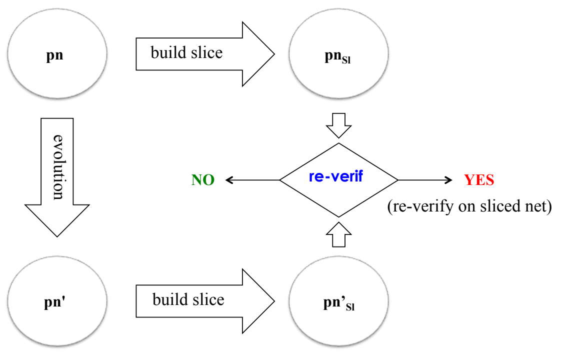

Figure 5. Overview

are generated for evolved and non-evolved Petri nets models with respect to the property by the abstract slicing algorithm as shown in Figure 5. Then, by comparing both sliced nets it is decided whether re-verification is required or not. If the answer is no then, re-verification is not required, whereas if the answer is yes, then, re-verification is performed on the sliced net. The good thing is that in both cases re-verification cost is improved. To decide for which evolutions re-verification is not required, we divide the evolutions into two major classes (by comparing both sliced Petri nets models as shown in the Figure 6), i.e., the evolutions that are taking place outside the slice, the evolutions that are taking place inside the slice. Furthermore, we divide the evolutions that are taking place inside the slice into two classes, i.e., the evolutions that disturb and those that do not disturb the previously satisfied properties.

A. Evolutions taking place outside the Slice:

The aim of slicing is to syntactically reduce a model in such a way that of the best reduced model contains only those parts that may influence the property the model is analyzed for. And if something is happening outside those parts of the system, then, it is guaranteed that previously satisfied properties are still true. We can generalize the notion, for all the evolutions that are taking place outside the slice do not influence the property satisfaction. Consequently, re-verification can be completely avoided for these evolutions. We formally specify how to avoid the verification if the evolutions are taking place outside the slice.

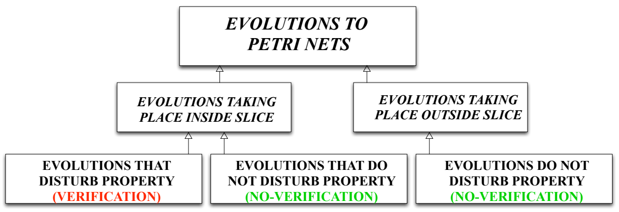
EVOLUTIONS TO PETRI NETS
EVOLUTIONS TAKING PLACE INSIDE SLICE
EVOLUTIONS TAKING PLACE OUTSIDE SLICE
EVOLUTIONS THAT DISTURB PROPERTY (VERIFICATION)
EVOLUTIONS THAT DO NOT DISTURB PROPERTY (NO-VERIFICATION)
EVOLUTIONS DO NOT DISTURB PROPERTY (NO-VERIFICATION)

Figure 6. Classification of evolutions to Petri nets

Theorem 1: Let $pn_{sl} = \langle P, T, f, w, m_0 \rangle$ be a sliced Petri net model and $pn'_{sl} = \langle P', T', w', m'_0 \rangle$ be an evolved sliced Petri net model w.r.t the property ϕ. $pn_{sl} \models \phi \Leftrightarrow pn'_{sl} \models \phi$ if and only if

$$pn_{sl} = pn'_{sl}$$

Informally, this theorem states that if an evolution is taking place outside the slice then, the evolved Petri net model preserves the previously satisfied properties. According to the conditions imposed by the theorem, both the sliced net and evolved sliced net are same and if the Petri net model satisfy a given property then, this property will also be true in its evolved model. Conversely, if the Petri net model does not satisfy a given property then, this property will be false in its evolved model.

Let us recall the Petri net model and example property given in the section III. The example property is following $\mathbf{AG}(|m(P3)|) < 5)$. Figure 7, shows some possible examples of the evolutions to Petri nets model that are taking place outside the slice. All the places, transitions and arcs that constitute a slice with respect to the property are shown with the blue doted lines (remark that we follow the same convention for all examples). In the example evolution, weight attached to the arc between transition *t2* and place *P4* is changed and shown with the red color. For all such kind of evolutions that are taking place outside the slice, we do not require verification because they do not disturb any behavior that may impact the satisfaction of the property.

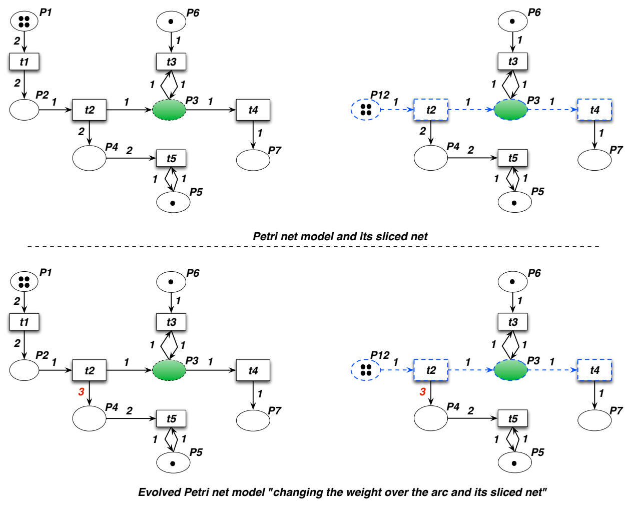
Petri net model and its sliced net
Evolved Petri net model "changing the weight over the arc and its sliced net"

Figure 7. Evolutions to Petri net model taking place outside the slice

B. Evolutions taking place inside the slice:

For all the evolutions that are taking place inside the slice, we divide them into two classes, i.e., evolutions that require verification and the evolutions that do not require re-verification. Identifying such class of evolutions is extremely hard due to non-determinism of the possible evolutions. Specifically, in Petri nets small structural changes can impact the behavior of the model. It is also hard to determine whether a property would be disturbed after an evolution or it is still satisfied by the model.

To identify evolutions that are taking place inside the slice and do not require re-verification, we propose to use the temporal specification of properties to reason about the satisfaction of properties with respect to the specific evolutions. For an example, for all the safety properties specified by the temporal formula $\mathbf{AG}(\varphi)$ *or* $\exists\mathbf{G}(\varphi)$, if φ an atomic formula, using the ordering operators $\leq$ *or* $<$ between the *places* and their *cardinality* or tokens inside *places*, then, all the evolutions that decrease the tokens from places do not require re-verification because they do not impact the behavior required for the property satisfaction.

Theorem 2: Let $pn_{sl} = \langle P, T, f, w, m_0\rangle$ be a sliced Petri net and $pn'_{sl} = \langle P', T', w', m'_0\rangle$ be an evolved sliced Petri net model (in which tokens are decreased from places) w.r.t the property ϕ. For all the safety properties specified by temporal formulas, i.e., $\mathbf{AG}(\phi)$ *or* $\exists\mathbf{G}(\phi)$, and ϕ a formula using $\leq$ *or* $<$ ordering operator between the places and their cardinality or tokens inside places. $pn_{sl} \models \phi \Rightarrow pn'_{sl} \models \phi$ if and only if

$$\forall p \in (P \cap P')/m_0(p) \geq m'_0(p) \wedge T = T' \wedge f = f' \wedge w = w'$$

Let us recall the Petri net model and example property given in the Section III. The example property is following $\mathbf{AG}(|m(P3)| < 5)$, we can avoid the re-verification for several evolutions even if they are taking place inside the slice. Some possible examples of the evolutions are shown in Figure 8. In the first example, tokens are decreased from a place and in the second example, tokens are decreased from an arc, but the property is still satisfied.

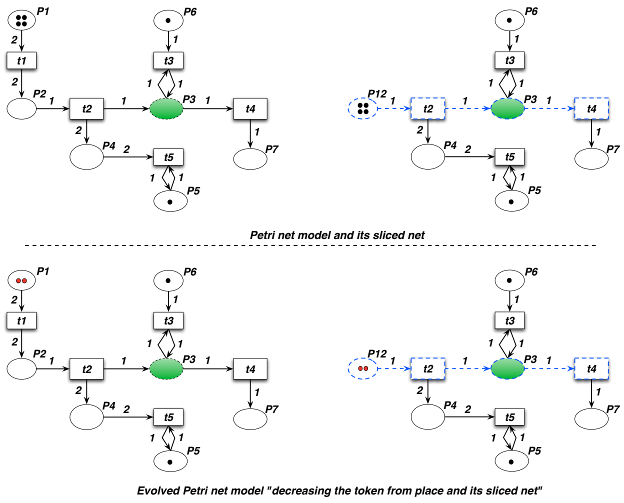
P1
P6
P2
P3
P4
P5
P7
P12
t1
t2
t3
t4
t5
Petri net model and its sliced net
Evolved Petri net model "decreasing the token from place and its sliced net"

Figure 8. Evolutions to Petri net model taking place inside the slice

For all the liveness properties specified by a temporal formula $\exists\,\mathbf{F}(\varphi)$, and if φ a formula using the ordering operators ($\geq$ *or* $>$) the *places* and their *cardinality* or tokens inside *places* and their *values*, then, for all the evolutions that increase the token count, it is not required to verify them as they do not impact the behavior required for the property satisfaction.

Theorem 3: Let $pn_{sl} = \langle P, T, f, w, m_0\rangle$ be a sliced Petri net and $pn'_{sl} = \langle P', T', w', m'_0\rangle$ be an evolved sliced Petri net model (in which tokens are increased from places) w.r.t the property ϕ. For all the liveness properties specified by a temporal formula $\exists\mathbf{F}(\phi)$, and ϕ is using the ordering operators $\geq$ *or* $>$ between the *places* and their *cardinality* or tokens inside *places* and their *values*. $pn_{sl} \models \phi \Rightarrow pn'_{sl} \models \phi$ if and only if

$$\forall p \in (P \cap P')/m_0(p) \leq m'_0(p) \wedge T = T' \wedge f = f' \wedge w = w'$$

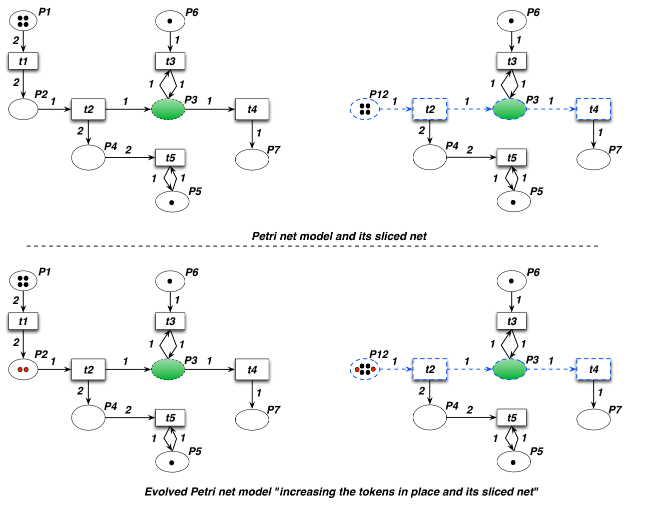
P1
P6
P2
P3
P4
P5
P7
P12
t1
t2
t3
t4
t5
Petri net model and its sliced net
Evolved Petri net model "increasing the tokens in place and its sliced net"

Figure 9. Evolutions to Petri net model taking place inside the slice

Let us consider again Petri net model given in the Section III , if we are interested to verify the example property such as: $\exists\mathbf{F}(|P3| > 3)$, verification can be avoided completely for several evolutions even if they are taking place inside the slice. Some possible examples of the evolutions are shown in Figure d9. In the first and second examples, tokens are increased but the property is still satisfied.

We identified above that for several specific evolutions and properties verification could be completely avoided, and for the rest of evolutions we can perform verification only on the part that concerns the property by following Section **??**. Even in this case we significantly improve the verification of evolution.

V. Related Work

Slicing is a technique used to reduce a model syntactically. The reduced model contains only those parts that may affect the property the model is analyzed for. Slicing Petri nets is gaining much attention in the recent years [5]–[11], [13]. Mark Weiser [14] introduced the slicing term, and presented slicing as a formalization of an abstraction technique that experienced programmers (unconsciously) use during debugging to minimize the program. The first algorithm about Petri net slicing was presented by Chang et al [5]. They proposed an algorithm on Petri nets testing that slices out all sets of paths, called concurrency sets, such that all paths within the same set should be executed concurrently. Astrid Rakow developed two slicing algorithms for Petri nets, i.e., CTL^*_{-X} slicing and *Safety slicing* in [10]. We introduced the Algebraic Petri net slicing for the first time [6], [12]. We adapt the notion of *reading and non-reading transitions* defined by Rakow [10] in the context of low-level Petri nets and applied to Algebraic Petri nets [6]. We extend the previous proposal by introducing a new notion of *neutral transitions* and applied to Algebraic Petri nets [12]. In this work, we designed abstract slicing algorithm in the context of low-level Petri nets and used to reason about the re-verification. To the best of your knowledge this is the first proposal to use slicing to improve the re-verification of Petri nets models.

Most of the work regarding the improvement of the re-verification of evolving Petri nets is oriented towards the preservation of properties. Padberg and several other authors published extensively on the invariant preservation of APNs by building a full categorical framework for APNs, i.e., rule-based refinements [15]–[17]. Padberg consider the notion of a rule-based modification of Algebraic high level nets preserving the safety properties. The theory of a rule-based modification is an instance of the high-level replacement system. Rules describe which part of a net are to be deleted and which new parts are to be added. It preserves the safety properties by extending the rule-based modification of Algebraic Petri nets in contrast to transition preserving morphisms in [15]. These morphisms are called the place preserving morphisms by allowing transferring of specific temporal logic formulas expressing net properties from the source to the target net. Lucio presented a preliminary study on the invariant preservation of behavioral models expressed in Algebraic Petri nets in the context of an iterative modeling process [16]. They proposed to extend the property preserving morphisms in a way that it becomes possible to strengthen the guards without loosing previous behaviours.

In contrast to the property preservation, the scope of our work is broader. At first, we try to find out which evolutions require re-verification independent of the temporal representations of properties. Secondly, we focus on the specific properties and evolutions to improve the re-verification. We do not restrict the type of evolutions and properties to give more flexibility to a user. It is important to note that our proposed technique can further refine the previous proposals about the property preservation. The proposal is to preserve the morphisms restricted to the sliced part of the net.

VI. Conclusion and Future Work

In this work, we developed an approach to improve the verification and re-verification of systems modeled in Petri nets. At first, a Petri net model is syntactically reduced based on the given temporal property. The reduced model which we call a sliced model constitutes only that part of a model that may affect the property satisfaction. The sliced model preserves CTL^*_{-X} properties. Secondly, we classify evolutions and properties to determine whether re-verification is required or not. We do not restrict the types of evolutions and the properties to give more flexibility to the user. Our results show that slicing is helpful to alleviate the state space explosion problem of Petri nets model checking and the re-verification of evolving Petri nets.

The future work has two objectives; first is to implement the proposed approach. A tool named $SLAP_N$ (a tool for slicing Algebraic Petri nets) is under development [18]. It is important to note that the $SLAP_N$ tool is a generic tool over the Petri net classes such as Petri net, Algebraic Petri nets. It provides a graphical interface to draw a Petri net or Algebraic Petri net model together with the temporal description of properties. It contains the implementation of different slicing algorithms and a user can select any of them to generate a sliced Petri net model. The future work consists of implementation of the classification of evolutions and properties to automate the proposed approach. The second objective of future work is concerned to enhance the theory of preservation of properties. The aim is to develop a property preserving domain specific language for the evolving Petri nets based on the slicing and the classification of evolutions and properties proposed in this work.

References

[1] I. Sommerville, *Software Engineering: (Update) (8th Edition) (International Computer Science Series)*. Addison Wesley, June 2006.

[2] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, May 2000.

[3] C. Larman and V. Basili, "Iterative and incremental developments. a brief history," *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[4] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Universität Hamburg, 1962.

[5] J. Chang and D. J. Richardson, "Static and dynamic specification slicing," in *In Proceedings of the Fourth Irvine Software Symposium*, 1994.

[6] Y. I. Khan and M. Risoldi, "Optimizing algebraic petri net model checking by slicing," *International Workshop on Modeling and Business Environments (ModBE'13, associated with Petri Nets'13)*, 2013.

[7] M. Llorens, J. Oliver, J. Silva, S. Tamarit, and G. Vidal, "Dynamic slicing techniques for petri nets," *Electron. Notes Theor. Comput. Sci.*, vol. 223, pp. 153–165, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2008.12.037

[8] A. Rakow, "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*, ser. SOFSEM'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 436–447. [Online]. Available: http://dl.acm.org/citation.cfm?id=1785934.1785974

[9] ——, "Slicing and reduction techniques for model checking petri nets," Ph.D. dissertation, University of Oldenburg, 2011.

[10] ——, "Safety slicing petri nets," in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds., vol. 7347. Springer Berlin Heidelberg, 2012, pp. 268–287. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31131-4_15

[11] W. J. Lee, H. N. Kim, S. D. Cha, and Y. R. Kwon, "A slicing-based approach to enhance petri net reachability analysis," *Journal of Research Practices and Information Technology*, vol. 32, pp. 131–143, 2000.

[12] Y. I. Khan and N. Guelfi, "Slicing high-level petri nets," *International Workshop on Petri Nets and Software Engineering (PNSE'14) associated with Petri Nets'14)*, vol. 2, no. 3, pp. 201–220, 2014. [Online]. Available: http://ceur-ws.org/Vol-1160/

[13] Y. I. Khan, "Property based model checking of structurally evolving algebraic petri nets," Ph.D. dissertation, University of Luxembourg, 2015.

[14] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering*, ser. ICSE '81. Piscataway, NJ, USA: IEEE Press, 1981, pp. 439–449.

[15] J. Padberg, M. Gajewsky, and C. Ermel, "Rule-based refinement of high-level nets preserving safety properties," in *Fundamental approaches to Software Engineering*. Springer Verlag, 1998, pp. 22 123–8.

[16] M. A. Q. Z. Levi Lucio, Eugene Syriani and H. Vangheluwe, "Invariant preservation in iterative modeling," *Proceedings of the ME 2012 workshop*, 2012.

[17] S. P. Er, "Invariant property preserving extensions of elementary petri nets," Technische Universitat Berlin, Tech. Rep., 1997.

[18] Y. I. Khan and N. Guelfi, "Slapn: A tool for slicing algebraic petri nets," *International Workshop on Petri Nets and Software Engineering (PNSE'14) associated with Petri Nets'14)*, vol. 2, no. 3, pp. 343–345, 2014.