Faculty of Science, Technology and Communication
Computer Science/ Telecommunications
University of Luxembourg
6, rue Coudenhove-Kalergi
L-1359 Luxembourg

Optimizing Algebraic Petri Net Model Checking by Slicing

Yasir Imtiaz Khan

TR_LASSY_13_02
Feb 2013

Optimizing Algebraic Petri Net Model Checking by Slicing

Yasir Imtiaz Khan and Matteo Risoldi

University of Luxembourg, Laboratory of Advanced Software Systems
6, rue R. Coudenhove-Kalergi, Luxembourg
{yasir.khan,matteo.risoldi}@uni.lu

Abstract. High-level Petri nets make models more concise and readable as compared to low-level Petri nets. However, usual verification techniques such as state space analysis remain an open challenge for both because of state space explosion. The contribution of this paper is to propose an approach for property based reduction of the state space of Algebraic Petri nets (a variant of high-level Petri nets). To achieve the objective, we propose a slicing algorithm for Algebraic Petri nets (*APNSlicing*). The proposed algorithm can alleviate state space even for certain strongly connected nets and is proved not to increase the state space. We exemplify our technique through the running case study of car crash management system.

Key words: High-level Petri nets, Model checking, Slicing

1 Introduction

Petri nets (PNs) are a well-known low-level formalism for modeling concurrent and distributed systems. Various evolutions of PNs have been created, among others *High-level Petri nets* (HLPNs), that raise the level of abstraction of PNs by using complex structured data [17]. However, HLPNs can be *unfolded* (i.e., translated) into a behaviourally-equivalent low-level PN.

For the analysis of concurrent and distributed systems (including those modeled using PNs or HLPNs) model checking [1] is a common approach, consisting in verifying a property against all possible states of a system. A typical drawback of model checking is its limits with respect to the *state space explosion* problem: as systems get moderately complex, completely enumerating their states demands a growing amount of resources, which in some cases makes model checking impractical both in terms of time and memory consumption [3,5,11,19]. This is particularly true for HLPN models, as the use of complex data (with possibly large associated data domains) makes the number of states grow very quickly.

As a result, an intense field of research is targeting to find ways to optimize model checking, either by reducing the state space or by improving the performance of model checkers. A technique called *PN slicing* falls into the first category. It proposes to reduce the state space size by syntactically reducing a PN model, taking only the portion of the model that impacts the properties to

be verified. The resultant model will typically have a smaller state space, thus reducing the cost of model checking.

Slicing was defined for the first time in [21] in the context of program debugging. The proposition was aimed at using program slicing for isolating the program statements that may contain a bug, so that finding this bug becomes simpler for the programmer. The first algorithm about PN slicing presented by Chang et al. [4] slices out all sets of paths in the PN graph, called *concurrency sets*, such that all paths within the same set should be executed concurrently. Some further refined PN slicing algorithms are proposed in [13–16].

One limitation of the cited approaches is that they only apply to low-level PNs. In order to be applied to HLPNs they need to be adapted to take into account data types.

In this work, we propose a slicing algorithm that is adapted to Algebraic Petri nets (APNs, a variant of HLPNs). To the best of our knowledge, there does not exist any algorithm for slicing APNs. The proposed algorithm iteratively builds a subnet from a given APN, according to a *slicing criterion* that is derived from the property to be verified. The resulting subnet preserves LTL_{-X} properties under weak fairness assumptions.

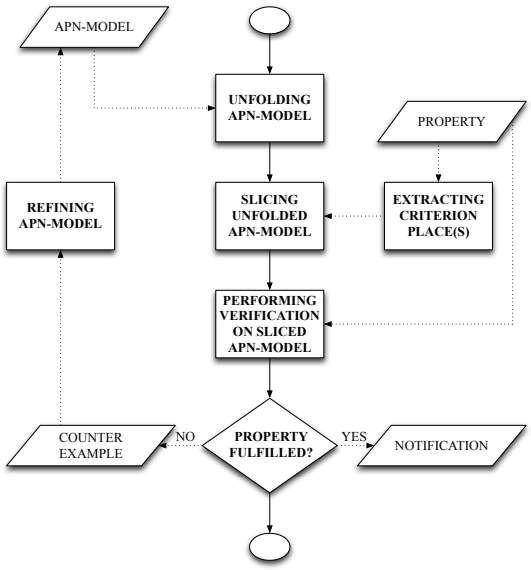

Fig. 1. Process Flowchart of *slicing* based verification of APN models

Fig.1, gives an overview of the proposed approach for slicing based verification of APNs using Process Flowchart. At first, APN-model is unfolded and then by taking properties into an account *criterion places* are extracted. Afterwards, slicing is performed for the *criterion places*. Subsequently, verification is

performed on the sliced unfolded APNs. The user may use the counterexample to refine the APN-model to correct the model to satisfy the property.

The rest of the work is structured as follows: we give basic definitions and concepts of the Algebraic Petri nets (APNs) in Section 2. Section 3, illustrates the steps of slicing based verification of APN-models shown in Fig.1. Details about the underlying theory and techniques are given for each activity of the process. In the Section 4, we discuss related work and a comparison with the existing approaches. A small case study from the domain of crisis management system (a car crash management system) is taken to exemplify the proposed slicing algorithm in Section 5. An experimental evaluation of the proposed algorithm is performed in Section 6. In the section 7, we draw conclusions and discuss future work concerning to the proposed work.

2 Basic Definitions

In this section, we give basic formal definitions of algebraic specifications used in this paper. Formal definitions, propositions, lemmas and theorems are taken as is or with slight modifications from [7, 15, 17, 18].

Definition 1. *A signature Σ = (S,OP) consists of a set S of sorts, $OP = (OP_{w,s})_{w \in S^*, s \in S}$ is a $(S^* \times S)-$sorted set of operation names of OP. For ϵ being the empty word, we call $OP_{\epsilon,s}$ the set of constant symbols.*

Definition 2. *A set X of Σ-variables is a family $X = (X_s)_{s \in S}$ of variables set, disjoint to OP.*

Definition 3. *The set of terms $T_{OP,s}(X)$ of sort s is inductively defined by:*

1. $X_s \cup OP_{\epsilon,s} \subseteq T_{OP,s}(X)$;

2. $op(t_1, \ldots, t_n) \in T_{OP,s}(X)$ for $op \in OP_{s_1,\ldots,s_n,s}, n \geq 1$ and $t_i \in T_{OP,s_i}(X)$ (for $i = 1, \ldots, n$).

The set $T_{OP,s} \equiv T_{OP,s}(\emptyset)$ contains the ground terms of sort s, $T_{OP}(X) \equiv \bigcup_{s \in S} T_{OP,s}(X)$ is the set of Σ-terms over X and $T_{OP} \equiv T_{OP}(\emptyset)$ is the set of Σ-ground terms.

Definition 4. *Let X be a finite set of Σ-variables. A substitution over X is mapping $sbt : X \to T_{OP}(X)$, whereby all $x \in X_s$ it holds $sbt(x) \in T_{OP,s}(X)$. If the image of subst is contained in T_{OP}, sbt is called ground substitution.*

Let $T \in T_{OP,s}(Y)$, X a finite subset of Y and sbt a substitution over X. Then the term sbt(T) results from T by simultaneously replacing the variables $x \in X$ by the corresponding terms sbt(x).

Definition 5. *A Σ-equation of sort s over X is a pair (l,r) of terms $l, r \in T_{OP,s}(X)$.*

Definition 6. *An algebraic specification SPEC = (Σ,E) consists of a signature Σ = (S,OP) and a set E of Σ-equations.*

Definition 7. *A Σ-algebra $A = (S_A, OP_A)$ consist of a family $S_A = (A_s)_{s \in S}$ of domains and a family $OP_A = (N_{op})_{op \in OP}$ of operations $N_{op} : A_{s_1} \times \dots A_{s_n} \to A_s$ for $op \in OP_{s_1 \dots s_n, s}$ if $op \in OP_{\epsilon,s}$, N_{op} congruent to an element of A_s.*

Definition 8. *An assignment of Σ-variables X to a Σ-algebra A is a mapping $ass : X \to A$,with $ass(x) \in A_s$iff $x \in X_s$. ass is canonically extended to $\overline{ass} : T_{OP}(X) \to A$, inductively defined by*

1. $\overline{ass}(x) \equiv ass(x)$ for $x \in X$;
2. $\overline{ass}(c) \equiv N_c$ for $c \in OP_{\epsilon,s}$;
3. $\overline{ass}(op(t_1, \dots, t_n)) \equiv N_{op}(\overline{ass}(t_1)), \dots, \overline{ass}(t_n))$ for $op(t_1, \dots, t_n) \in T_{OP}(X)$.

Definition 9. *Let SPEC-algebra is SPEC = (Σ,E) in which all equations in E are valid. Two terms t_1 and t_2 in $T_{OP}(X)$ are equivalent ($t_1 \equiv_E t_2$) iff for all assignments $ass : X \to A, \overline{ass}(t_1) = \overline{ass}(t_2)$.*

Definition 10. *Let B be a set. A multiset over B is a mapping ms_B: $B \to \mathbb{N}$. ϵ_B is the empty multiset with $ms_B(x) = 0$ for all $x \in B$. A multiset is finite iff $\{\forall b \in B \mid ms_B(b) \neq 0\}$ is finite.*

Definition 11. *Let $MS_B = \{ms_B : B \to \mathbb{N}\}$ be a set of multisets. The addition function of multisets is denoted by $+ : MS_B \times MS_B \to MS_B$. Let $ms1_B, ms2_B$ and $ms3_B \in MS_B$. $\forall b \in B, ms3_B(b) = ms1_B(b) + ms2_B(b)$.*

The subtraction function of multisets is denoted by $- : MS_B \times MS_B \to MS_B$. Let $ms1_B, ms2_B$ and $ms3_B \in MS_B$. $\forall b \in B, ms1_B(b) \geq ms2_B(b) \Rightarrow \forall b \in B, (ms1_B - ms2_B)(b) = ms1_B(b) - ms2_B(b)$.

Definition 12. *Let $MS_B = \{ms_B : B \to \mathbb{N}\}$ be a set of multisets. Let $ms1_B, ms2_B \in MS_B$. We say that $ms1_B$ is smaller than or equal to $ms2_B$ (denoted by $ms1_B \leq ms2_B$) iff*

$\forall b \in B, ms1_B(b) \leq ms2_B(b)$. Further, we say that $ms1_B \neq ms2_B$ iff
$\exists b \in B, ms1_B(b) \neq ms2_B(b)$. Otherwise, $ms1_B = ms2_B$.

Definition 13. *A marked Algebraic Petri Net $APN =< SPEC, P, T, F, asg, cond, \lambda, m_0 >$ consist of*

◦ an algebraic specification SPEC = (Σ,E),
◦ P and T are finite and disjoint sets, called places and transitions, resp.,
◦ $F \subseteq (P \times T) \cup (T \times P)$, the elements of which are called arcs,
◦ a sort assignment $asg : P \to S$,
◦ a function, $cond : T \to \mathcal{P}_{fin}(\Sigma - equation)$, 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 $T_{OP,asg(p)}$,
◦ an initial marking m_0 assigning a finite multiset over $T_{OP,asg(p)}$ to every place p.

Definition 14. *The preset of $p \in P$ is ${}^\bullet p = \{t \in T | (t,p) \in F\}$ and the postset of p is $p^\bullet = \{t \in T | (p,t) \in F\}$. The pre and post sets of $t \in T$ defined as: ${}^\bullet t = \{p \in P | (p,t) \in F\}$ and $t^\bullet = \{p \in P | (t,p) \in F\}$.*

Definition 15. *A marking m of an APN assigns to every place $p \in P$ a multiset over $T_{OP,asg(p)}$.*

Definition 16. *An occurrence mode is a ground substitution of $cond(t), m(p), \lambda(p,t)$ and $\lambda(t,p)$ where $p \in P, t \in T$. Obviously, ground substitutions are the syntactical representations of assignments.*

Definition 17. *A transition $t \in T$ of an APN is enabled in an occurrence mode at a marking m iff for all p in P with (p,t) $\in F$, $\lambda(p,t) \leq m(p)$. If a transition t is enabled in an occurrence mode at a marking m, then t may occur returning the marking m', where for all $p \in P$, $m'(p) = m(p) - \lambda(p,t) + \lambda(t,p)$. We write $m[t\rangle m'$ in this case.*

Definition 18. *A firing sequence σ of a marked APN is maximal iff either σ is of infinite length or $\nexists t \in T : m_0([\sigma t\rangle)$, where $|\sigma| \in (\mathbb{N} \cup \{\infty\})$.*

Definition 19. *Let $\sigma = t_1, t_2 \ldots$ be an infinite firing sequence of APN with $m_i[t_{i+1}\rangle m_{i+1}, \forall i, 0 \leq i$. σ permanently enables $t \in T$ iff $\exists i, 0 \leq i : \forall j, i \leq j : m_j[t\rangle$.*

3 Unfolding and Slicing APNs

One characteristic of APNs that makes them complex to model check is the use of variables on arcs. Computing variable bindings at runtime is extremely costly. Unfolding generates all possible firing sequences from the initial marking of the APN, though maintaining a partial order of events based on the causal relation induced by the net, concurrency is preserved. AlPiNA (a symbolic model checker for Algebraic Petri nets) allows the user to define partial algebraic unfolding and presumed bounds for infinite domains [2], using some aggressive strategies for reducing the size of large data domains.

The basic idea of the slicing algorithm is to start by identifying which places in the unfolded APN model are directly concerned by a property. These places constitute the *slicing criterion*. The algorithm will then take all the transitions that create or consume tokens from the criterion places, plus all the places that are pre-condition for those transitions. This step is iteratively repeated for the latter places, until reaching a fixed point.

We refine the slicing construction by distinguishing between *reading* and *non-reading transitions*. The conception of *reading and non-reading transitions* is some what similar notion introduced in [16]. The main difference is that we adapt the notion of *reading and non-reading transitions* in the context of APNs. Informally, *reading transitions* are not supposed to change the marking of a place. On the other hand *non-reading transitions* are subject to change the markings of a place. Unfolding of APNs helps us to identify syntactically *reading and non-reading transitions*. In our proposed slicing construction, we discard *reading transitions* and include only *non-reading transitions* becuase they are affecting the property satisfaction. Formally, we can define the conception of *reading and non-reading transitions* such as:

Definition 20. *Let N be an unfolded APN and $t \in T$ be a transition. 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 $\forall p \in ({}^\bullet t \cup t^\bullet), \lambda(p,t) = \lambda(t,p)$. Conversely, we call t a non-reading transition iff $\lambda(p,t) \neq \lambda(t,p)$.*

Due to partial unfolding, there could be some domains that are not unfolded. For some cases, we are still able to identify *non-reading transitions* even if domains are not unfolded. If for example, we have a case where the multiplicities or cardinalities of terms in $\lambda(p,t), \lambda(t,p)$are different then we can immediately state $\lambda(p,t) \neq \lambda(t,p)$. But for some cases, we don't have such a clear indication of the inequality between $\lambda(p,t)$ and $\lambda(t,p)$, for example, in Fig.2, we see that $\lambda(p,t) = 1 + y$ and $\lambda(t,p) = 2 + x$ (defined over naturals). Both terms has the same multiplicity and cardinality, so we need to know for which values of the variables it would be a *non-reading transition*. In general, the evaluation of terms to check their equality for all the values is undecidable. For this particular case, we would like to have a set of constraints from the user. Informally,

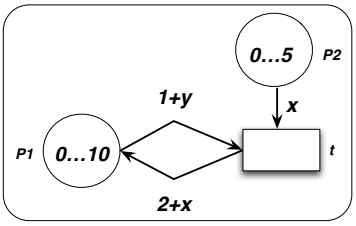

Fig. 2. An example APN model with non-unfolded terms over the arcs

a constraints set denoted by CS, is a set of propositional formulas, predicate formulas or any other logical formulas for certain specific values of variable assignements, describing the conditions under which we can evaluate terms to be equal or not. Consequently, constrains set CS will help to identify under which cases the transitions can be treated as *non-reading*.

A function $eval : T_{OP,s}(X) \times T_{OP,s}(X) \times CS \rightarrow Bool$ is used to evaluate the equivalence of terms based on the constraint set. Let us take the same terms shown over the arcs in Fig.2, $term_1 = 1 + y$, $term_2 = 2 + x$ and a constraint set $CS = \{\exists y, x \in (0, \ldots, 2) | y = x + 1\}$. It is important to note that we are not unfolding the domain but evaluating the terms for some specific values provided by user to identify *reading and non-reading transitions*. Of course, the user can provide sparse values too. Let us evaluate the terms $term_1$ and $term_2$ based on the constraints set CS provided. For all those values of x, y for which we get *eval* function result true are considered to be *reading transitions* and rest of them are *non-reading transitions*. It is also important to note that we include this step during the unfolding. The resulting unfolded APN will contain only *non-reading transitions* for the unfolded domains as shown in Fig.3.

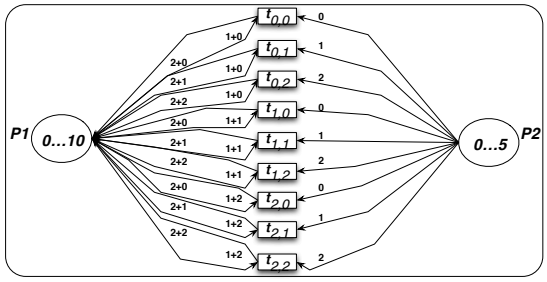

Fig. 3. Resulting unfolded APN after applying the *eval* function

The algorithm proposed in this article assumes that such an unfolding takes place before the slicing. Since this is a step that is involved in the model checking activity anyway, we do not consider this assumption to be adding to the complexity of the algorithm. In this section, we will make an extremely simple example of how the slicing algorithm works, starting from an APN, unfolding it and slicing it.

3.1 Example: Unfolding an APN

Fig. 4 shows an APN model. All places and all variables over the arcs are of sort *naturals* (defined in the algebraic specification of the model, and representing the $\mathbb{N}$ set).

Since the $\mathbb{N}$ domain is infinite (or anyway extremely large even in its finite computer implementations), it is clear that it is impractical to unfold this net by considering all possible bindings of the variables to all possible values in $\mathbb{N}$. However, given the initial marking of the APN and its structure it is easy to see that none of the terms on the arcs (and none of the tokens in the places) will ever assume any natural value above 3. For this reason, following [2], we can set a *presumed bound* of 3 for the *naturals* data type, greatly reducing the size of the data domain. By assuming this bound, the unfolding technique in [2] proceeds in three steps. First, the data domains of the variables are unfolded up to the presumed bound. Second, variable bindings are computed, and only that satisfy the transition guards are kept. Third, the computed bindings are used to instantiate a binding-specific version of the transition. The resulting unfolded APN for this APN model is shown in Fig. 5. The transitions arcs are indexed with the incoming and outgoing values of tokens. A complete explanation of the unfolding algorithm, and in particular the existence of the token *4* between transition $t2_3$ and places B, C is rather complex and out of the scope of this article. The interested reader can find details about the partial unfolding in [2].

3.2 The slicing algorithm

The slicing algorithm starts with an unfolded APN and a slicing criterion $Q \subseteq P$.

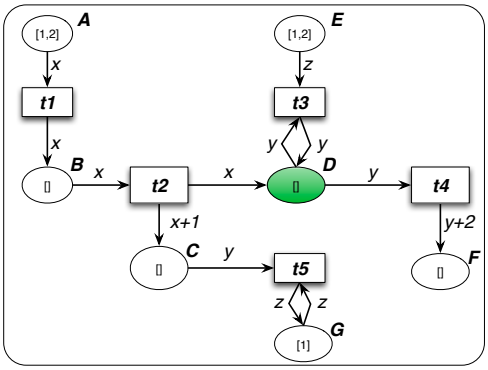

Fig. 4. An example APN model (*APNexample*)

Let $Q \subseteq P$ a non empty set called slicing criterion. We can build a slice for an *unfolded APN* based on Q, using following algorithm:

Algorithm 1: APN slicing algorithm

APNSlicing($\langle SPEC, P, T, F, asg, cond, \lambda, m_0 \rangle, Q$){
$T' = \{t \in T \mid \exists p \in Q : t \in ({}^\bullet p \cup p^\bullet) : \lambda(p,t) \neq \lambda(t,p)\}$;
$P' = Q \cup \{{}^\bullet T'\}$;
$P_{done} = \emptyset$;
while $((\exists p \in (P' \setminus P_{done}))$ **do**
  while $(\exists t \in ({}^\bullet p \cup p^\bullet) \setminus T') : \lambda(p,t) \neq \lambda(t,p))$ **do**
    $P' = P' \cup \{{}^\bullet t\}$;
    $T' = T' \cup \{t\}$;
  end
  $P_{done} = P_{done} \cup \{p\}$;
end
return $\langle SPEC, P', T', F_{|_{P',T'}}, asg_{|_{P'}}, cond_{|_{T'}}, \lambda_{|_{P',T'}}, m_{0_{|_{P'}}} \rangle$;
}

Initially, T' (representing transitions set of the slice) contains set of all *pre and post* transitions of the given criterion place. Only *non-reading* transitions are added to T' set. And P'(representing places set of the slice) contains all *preset* places of transitions in T'. The algorithm then iteratively adds other *preset* transitions together with their *preset* places in T' and P'. Remark that the *APNSlicing* algorithm has linear time complexity.

Considering the APN-Model shown in fig. 4, let us now take an example property and apply our proposed algorithm on it. Informally, we can define the property:

"The values of tokens inside place D are always smaller than 5".

Formally, we can specify the property in *LTL* as $\mathbf{G}(\forall tokens \in D | tokens < 5)$. For this property, the slicing criterion $Q = \{D\}$, as D is the only place concerned

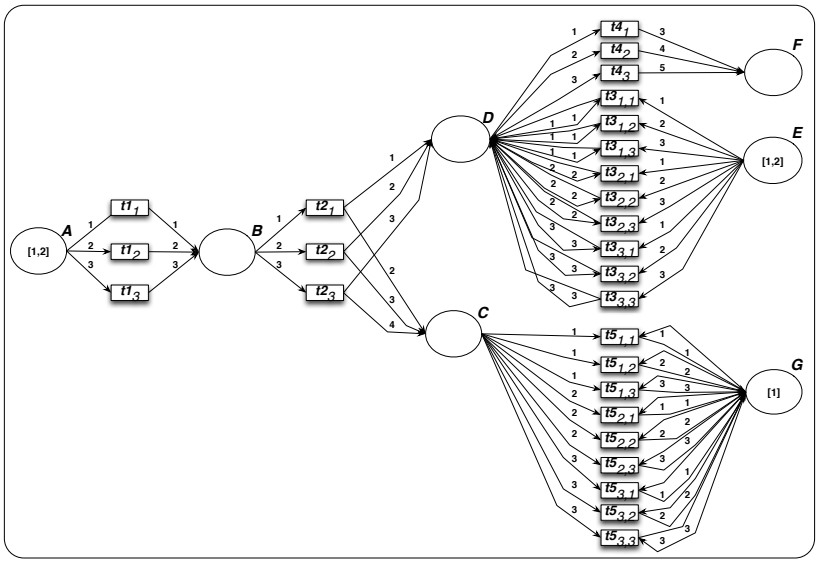

Fig. 5. The unfolded example APN model (*UnfoldedAPN*)

by the property. Therefore, the application of *APNSlicing(UnfoldedAPN, D)* returns *SlicedUnfoldedAPN* (shown in Fig. 6), which is smaller than the original *UnfoldedAPN* shown in Fig. 5).

Transitions $t3_{1,1}, t3_{1,2}, t3_{1,3}, t3_{1,3}, t3_{2,1}, t3_{2,2}, t3_{2,3}, t3_{3,1}, t3_{3,2}, t3_{3,3}, t5_{1,1}, t5_{1,2}$, $t5_{1,3}, t5_{2,1}, t5_{2,2}, t5_{2,3}, t5_{3,1}, t5_{3,2}, t5_{3,3}$, and places C, E, F, G has been sliced away. The proposed algorithm determines a slice for any given criterion $Q \subseteq P$ and always terminates. It is important to note that the reduction of net size depends on the structure of the net and on the size and position of the slicing criterion within the net.

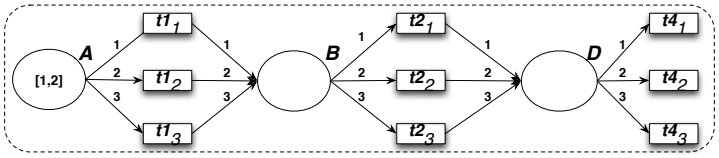

Fig. 6. Sliced and Unfolded example APN model (*SlicedUnfoldedAPN*)

3.3 Proof of the preservation of properties

To allow the verification by slice, we have to make restrictions on the formulas and on admissible firing sequences in terms of fairness assumptions. The original Algebraic Petri net has more behaviors than the sliced APN, as we intentionally do not capture all the behaviors.

Definition 21. *Let A be the set of atomic propositions. Let $\varphi, \varphi_1, \varphi_2$ be LTL formulas. The function scope associates with an LTL formula φ the set of atomic propositions used in φ i.e. scope : $LTL \rightarrow \mathcal{P}A$.*

$scope(a) = \{a\}$ for $a \in A$;
$scope(\otimes\varphi) = scope(\varphi)$ with $\otimes \in \{\neg, X\}$;
$scope(\varphi_1 \otimes \varphi_2) = scope(\varphi_1) \cup scope(\varphi_2)$ with $\otimes \in \{\wedge, U\}$.

Definition 22. *Let N be a marked APN. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let $\sigma = t_1 t_2 t_3 \ldots$be a firing sequence of N and m_i the markings with $m_i[t_{i+1}\rangle m_{i+1}, \forall i, 0 \leq i < |\sigma|$. σ is slice-fair w.r.t N' iff either σ is finite and $m_{|\sigma|}$ does not enable any transition $t \in T'$;*

or σ is infinite and if it permanently enables some $t \in T'$, it then fires infinitely often some transition of T' (which may or may not be the same as t).

Slice-fairness is a very weak fairness notion. Weak fairness determines that every transition $t \in T$ of a system, if permanently enabled, has to be fired infinitely often, slice-fairness concerns only the transitions of the slice, not of the entire system net and if a transition $t \in T$ of the slice is permanently enabled, some transitions of the slice are required to fire infinitely often but not necessarily t.

Definition 23. *Let N be a marked APN and φ an LTL formula. $N \models \varphi$ slice-fairly iff every slice-fair (not necessarily maximal) firing sequence of $\sigma \models \varphi$.*

Definition 24. *Let N and N' be two marked Algebraic Petri nets with $T' \subseteq T$ and $P' \subseteq P$. We define the function: $slice_{(N,N')} \in [(T^* \cup T^w) \rightarrow (T'^* \cup T'^w)] \cup [\mathbb{N}^{|P|} \rightarrow \mathbb{N}^{|P'|}]$ such that a finite or infinite sequence of transitions σ is mapped onto the transition sequence σ' with σ' being derived from σ by omitting every transition $t \in T \setminus T'$. A marking m of N is projected onto the marking m' of N' with $m' = m \mid_{p'}$.*

The function *slice* is used to project markings and firing sequences of a net N onto the markings and firing sequences of its slices.

Proposition 1. *Let N be a marked APN. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ be a weakly fair firing sequence of N. σ is slice fair with respect to N'.*

Proof. Let us assume, σ is not slice-fair. In case σ is finite this means that $m_{|\sigma|}[t\rangle$ for a transition $t \in T'$. In case σ is infinite, there is permanently enabled transition $t \in T'$ but all transitions of T' are fired finitely often including t. So both cases contradict the assumption that σ is weakly fair.

Lemma 1. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. The coefficients c_{ij} of the incidence matrix equal to zero for all places $p_i \in P'$ and transitions $t_j \in T \setminus T'$.*

Proof. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. A transition $t \in T$ is also an element of $T' \subseteq T$, if it is a *non-reading* transition of a place $p \in P'$. Thus a transition $t \in T \setminus T'$ either is not connected to a place $p \in P'$ or it is a *reading* transition.

Lemma 2. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let m be a marking of N and m' be a marking of N' with $m' = m \mid_{p'}$. $m[t\rangle \Leftrightarrow m'[t\rangle, \forall t \in T'$.*

Proof. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Since a transition $t \in T'$ has the same preset places in N and N' by the slicing algorithm $APNSlicing$, $m' = m \mid_{p'}$ implies $m[t\rangle \iff m'[t\rangle$.

Every firing sequence σ of N projected onto the transitions of T' is also a firing sequence of slice net N'. The resulting markings m and m' assign the same number of tokens to places $p' \subseteq P$.

Proposition 2. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ be a firing sequence of N and let m be a marking of N. $m_0[\sigma\rangle m \Rightarrow m_0 \mid_{p'} [slice(\sigma)\rangle m \mid_{p'}$.*

Proof. We prove this Proposition by induction over the length l of σ. Let N be a marked APN, σ be a firing sequence of N.

$l = 0$: In this case slice(σ) equals ϵ. Thus the initial marking of N and N' is generated by firing ϵ. By defintion 24 and the slicing algorithm $APNSlicing$, $m'_0 = m_0 \mid_{p'}$

$l \to l + 1$: Let σ be a firing sequence of length l and m_l be a marking of N with $m_0[\sigma\rangle m_l$. Let t_{l+1} be a transition in T and m_{l+1} a marking of N such that $m_l[t_{l+1}\rangle m_{l+1}$. By induction hypothesis, $m'_0[slice(\sigma)\rangle m'_k$ with $m_l \mid_{p'} = m'_k$. If t_{l+1} is an element of T', it follows by Lemma 2, that m'_k enables t_{l+1}, since m_l enables t_{l+1}. The resulting marking m'_{k+1} is determined by $m'_{k+1}(P'_i) = m'_k(P'_i) + c_{i\,l+1}, \forall p_i \in P'$ and m_{l+1} is determined by $m_{l+1}(i) = m_l(i) + c_{i\,l+1}, \forall p_i \in P'$.

Since $m_{l|_{P'}} = m'_k$, it thus follows that $m_{l_{l+1}|_{P'}} = m'_{k+1}$. If t_{l+1} is an element of $t \in T \setminus T'$, then it must be a reading transition for all $p \in P$; $slice(\sigma) = slice(\sigma t_{l+1})$ and thus $m'_0[slice(\sigma t_{l+1})\rangle m'_k$ a transition $t \in T \setminus T'$ can not change the marking of on any place $p \in P'$. By Lemma 1 and the resultant markings, $m_{l_{l+1}|_{P'}} = m'_l \mid_{P'}$. □

A firing sequence σ' of the slice net N' is also a firing sequence of N. The resulting markings of σ' on N and N', respectively assigns the same markings to places $p \in P'$.

Proposition 3. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ' be a firing sequence of N' and let m' be a marking of N'.*

$m'_0[\sigma'\rangle m' \Rightarrow \exists m \in \mathbb{N}^{|P|} : m' = m_{l|_{P'}} \wedge m_0[\sigma'\rangle m$.

Proof. We prove this Proposition by induction over the length l of σ'.

$l = 0$: The empty firing sequence generates the marking m_0 on N and the marking m'_0, which is defined as $m_{0|_{P'}}$, on N', by definition 24.

$l \rightarrow l + 1$: Let $\sigma' = t_1 \dots t_{l+1}$ be firing sequence of N' with length $l + 1$. Let m'_l and m'_{l+1} be markings of N' such that $m'_0[t_1 \dots t_l\rangle m'_l[t_{l+1}\rangle m'_{l+1}$. Let m_l be the marking of N with $m_0[t_1 \dots t_l\rangle m_l$ and $m_{l|_{P'}} = m'_l$, which exists according to the induction hypothesis. Lemma 2, m_l enables t_{l+1}. The marking m_{l+1} satisfies $m_{l+1}(P_i) = m_l(P_i) + c_{i\,l+1}, \forall p_i \in P'$ and m'_{l+1} satisfies $m'_{l+1}(P_i) = m'_l(P_i) + c_{i\,l+1}, \forall s_i \in P'$. With $m_{l|_{P'}} = m'_l$, it follows that $(m_{l+1} \mid_{P'})$ is equal to m'_{l+1}. □

Proposition 4. *Let N be a marked APN and let ϕ be an LTL_x formula such that $scope(\phi) \subseteq P$. Let N' be its sliced net for a slicing criterion $Q \subseteq P$ where $Q = scope(\phi)$. Let σ be a firing sequence of N. Let us denote the sequence of markings by $\mathcal{M}(\sigma)$. Then, $\mathcal{M}(\sigma) \models \phi \Leftrightarrow \mathcal{M}(slice(\sigma)) \models \phi$.*

Proof. We prove this Proposition by induction on the structure of ϕ. Let $\sigma = t_1 t_2 \dots$ and $slice(\sigma)$ be $\sigma' = t'_1 t'_2 \dots$. Let $\mathcal{M}(\sigma) = m_0 m_1 \dots$ and $\mathcal{M}(\sigma') = m'_0 m'_1 \dots$.

$\phi = true$: In this case nothing needs to be shown. $\phi = \neg\psi, \phi = \psi_1 \wedge \psi_2$: Since the satisfiability of ϕ depends on the initial marking of $scope(\phi)$ only and $scope(\phi) \subseteq P' \subseteq P$, both directions hold.

$\phi = \psi_1 U \psi_2$: We assume that $\mathcal{M}(\sigma') \models \psi_1 U \psi_2$. We can divide up σ' such that $\sigma' = \sigma'_1 \sigma'_2$ with $m'_{|\sigma'_1|} m'_{|\sigma'_1|+1} \dots \models \psi_2$ and $\forall i, 0 \leq i < |\sigma'_1| : m'_i m'_{i+1} \dots \models \psi_1$. There are transition sequences σ_1 and σ_2 such that $\sigma = \sigma_1 \sigma_2, slice(\sigma_1) = \sigma'_1, slice(\sigma_2) = \sigma'_2$ and σ_1 does not end with a transition $t \in T \setminus T'$.

By proposition 2, it follows that $m'_{|\sigma'_1|} = (m_{|\sigma_1|} \mid_{P'})$. Since $m'_{|\sigma'_1|} m'_{|\sigma'_1|+1} \dots \models \psi_2, m_{|\sigma_1|} m_{|\sigma_1|+1} \dots \models \psi_2$ by induction hypothesis. Let ϱ be a prefix of σ_1 such that $|\varrho| < |\sigma_1|$. Let ϱ' be $slice(\varrho)$. The firing sequence ϱ truncates at least one transition $t \in T'$, consequently $|\varrho'| < |\sigma'_1|$. Since $m'_{|\varrho'_1|} m'_{|\varrho'_1|+1} \dots \models \psi_1, m_{|\varrho|} m_{|\varrho|+1} \dots \models \psi_1$ by the induction hypothesis. Analogously, it can be shown that $\mathcal{M}(\sigma) \models \psi_1 U \psi_2$ implies $\mathcal{M}(\sigma') \models \psi_1 U \psi_2$. □

Proposition 5. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ' be a maximal firing sequence of N'. σ' is a slice-fair firing sequence of N.*

Proof. Let $\sigma' = t_1 t_2 \dots$. Let m'_i be the marking of N', such that $m'_i[t_{i+1}\rangle m'_{i+1}, \forall i, 0 \leq i < |\sigma'|$. By Proposition 3 σ' is a firing sequence of N. Let m_i be the marking of N, such that $m_i[t_{i+1}\rangle m_{i+1}, \forall i, 0 \leq i < |\sigma'|$. In case σ' is finite, $m'_{|\sigma'|}$ does not enable any transitions $t' \in T'$.

By Lemma 2, $m_{|\sigma'|}$ does not enable any transition $T' \in T'$, If σ' is infinite it obviously fires infinitely often a transition $t' \in T'$ and thus is slice-fair. □

Proposition 6. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Slice(σ) is maximal firing sequence of N'.*

Proof. Let $\sigma = t_1 t_2 \ldots$ with $m_i[t_{i+1}\rangle m_{i+1}, \forall i, 0 \leq i < |\sigma|$. By Proposition 2, $slice(\sigma)$ is a firing sequence of N'. Let $slice(\sigma)$ be $\sigma' = t'_1 t'_2 \ldots$ with $m'_i[t'_{i+1}\rangle m'_{i+1}$, $\forall i, 0 \leq i < |\sigma|$. Let us assume σ' is not a maximal firing sequence of N'. Thus σ' is finite and there is a transition $t' \in T'$ with $m'_{|\sigma'|}[t'\rangle$. Let σ_1 be the smallest prefix of σ such that $slice(\sigma_1)$ equals σ'.

By Proposition 2 $(m_{|\sigma_1|} \mid_{P'} = m'_{|\sigma'|}$. By Lemma 2, and the state equation it follows, that $(m_{|\sigma_1|} \mid_{P'} = m'_{|\sigma'|+1} = \ldots$. So t' stays enabled for all markings m_j with $|\sigma_1| \leq j \leq |\sigma|$ but is fired finitely many times only. This is a contradiction to the assumption that σ is slice-fair. □

Theorem 1. *Let N be a marked APN and let ϕ be an LTL formula such that $scope(\phi) \subseteq P$. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let Ψ be an LTL_{-X} formula with $scope(\Psi) \subseteq P$.*

$N \models \phi$ slice-fairly $\Rightarrow N' \models \phi$, for an LTL formula ϕ.

$N \models \Psi$ slice-fairly $\Leftarrow N' \models \Psi$, for an LTL_{-X} formula Ψ.

Proof. We first show " $N \models \phi$ slice-fairly $\Rightarrow N' \models \phi$ ". Let us assume that $N \models \phi$ slice-fairly holds. Let σ' be a maximal firing sequence of N'. Since σ' is a slice-fair firing sequence of N by Proposition 5 $\mathcal{M}(\sigma') \models \phi$. Let us now assume $N' \models \Psi$. Let σ be a slice-fair firing sequence of N. By Proposition 6, $slice(\sigma)$ is maximal firing sequence of N' and thus satisfies Ψ. By Proposition 4, it follows that σ satisfies Ψ. □

Verification is possible under interleaving semantics if we assume slice-fairness. A firing sequence σ is fair w.r.t T' , if σ is either maximal and if σ eventually permanently enables a $t' \in T'$, a transition $t \in T'$ will be fired infinitely often, t may not equal t'. Unfolded APN $\models \varphi$ fairly w.r.t. T' holds if all fair firings sequences of N, more precisely, their corresponding traces satisfy φ.

4 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 recent years [4, 9, 13–16, 20]. Mark Weiser introduced the slicing term in [21], and presented slicing as a formalization of an abstraction technique that experienced programmers (unconsciously) use during debugging to minimize the program. Broadly, we can divide slicing into types, which are *static slicing* and *dynamic slicing*. A slice is said to be static if the input of the program is unknown (this is the case of Weiser's approach). On the other hand, it is said to be dynamic if a particular input for the program is provided, i.e., a particular computation is considered. The first algorithm about Petri net slicing was presented by Chang et al [4]. 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. Lee et al. proposed the Petri nets slice approach in order to partition huge place transition net models into manageable modules, so that the partitioned model can be analyzed by compositional reachability analysis technique [12]. Llorens et al. introduced two different techniques for dynamic slicing of Petri nets [13]. In the first technique of Llorens et al. the Petri net and an initial marking is taken into account, but produces a slice w.r.t. any possibly firing sequence. The second approach further reduces the computed slice by fixing a particular firing sequence. Wangyang et al presented a backward dynamic slicing algorithm [20]. The basic idea of proposed algorithm is similar to the algorithm proposed by Lloren et al, [13]. At first for both algorithms, a static backward slice 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 forward slice is computed for the resultant Petri net model obtained from static backward slice.

Astrid Rakow developed two flavors of Petri net slicing, CTL^*_{-X} slicing and *Safety slicing* in [16]. The key idea behind the construction is to distinguish between reading and non-reading transitions. A reading transition $t \in T$ can not change the token count of place $p \in P$ while other transitions are non-reading transitions. For CTL^*_{-X} slicing, a subnet is built iteratively by taking all non-reading transitions of a place P together with their input places, starting with given criterion place. And for the *Safety slicing* a subnet is built by taking only transitions that increase token count on places in P and their input places. CTL^*_{-X} slicing algorithm is fairly conservative. By assuming a very weak fairness assumption on Petri net it approximates the temporal behavior quite accurately by preserving all CTL^*_{-X} properties and for safety slicing focus is on the preservation of stutter-invariant linear safety properties only.

We notice that all the approaches are limited to low-level Petri nets. The main difference between High-level and low-level Petri net is that in high-level Petri nets tokens are no longer black dots, but complex structured data. Whereas in case of low-level Petri nets, all (black) tokens correspond to the same data object. The idea of *reading and non-reading transitions* introduced in [16] deals only with the token count of places in low-level Petri nets. In Algebraic Petri nets there are properties that may concern the values of tokens. The main difference between the existing slicing constructions such as, CTL^*_{-X}, *Safety slicing* and ours is that in CTL^*_{-X}, *Safety slicing* only transitions are included that change the token count, whereas in *APNSlicing*, we include transitions that change the token values together with the transitions that change the token count.

Let us study the results summarized in the comparison table.1, the first column shows different slicing algorithms under observation (Remark that we only include PN slicing algorithms, which are designed for improving the model checking process). For each algorithm, table lists:

- i) The design context refers to the application of slicing algorithm with respect to Petri nets formalism; there are two major variants of Petri nets that

are low-level Petri nets and high-level Petri nets,

- ii) Properties that are preserved by the slicing constructions. As we can notice that some of the slicing algorithms preserve particular propertes. Particular properties here refer to examine the bound etc.,

- iii) Slicing type refers to the construction methodology i.e., either it is static or dynamic slice and is it following backward propagation or forward (or both),

- iv) Whether the algorithm has been implemented or not.

Table 1. Comparison of PN slicing Algorithms

Agorithm	*Design context*	*Preserved properties*	*Type slicing*	*Implementation*
Rakow CTL^*_{-X} *slicing*	*Designed for low-level PN*	CTL^*_{-X}	*Static backward slicing*	*Own*
Rakow Safety slicing	*Designed for low-level PN*	*Safety*	*Static backward slicing*	*Own*
Khan et al slicing	*Designed for high-level PN*	LTL_{-X}	*Static backward slicing*	*Own*
Llorens et al slicing	*Designed for low-level PN*	*Particular*	*Dynamic forward/backward slicing*	*No*
Wangyang et al slicing	*Designed for low-level PN*	*Particular*	*Dynamic backward slicing*	*No*

5 Case Study

We took a small case study from the domain of crisis management systems (car crash management system) for the experimental investigation of the proposed approach. In a car crash management system (CCMS); reports on a car crash are received and validated, and a `Superobserver` (i.e., an emergency response team) is assigned to manage each crash.

The APN Model can be observed in Fig. 7, it represents the semantics of the operation of a car crash management system. This behavioral model contains labeled places and transitions. There are two tokens of type `Fire` and `Blockage` in place `Recording Crisis Data`. These tokens are used to mention

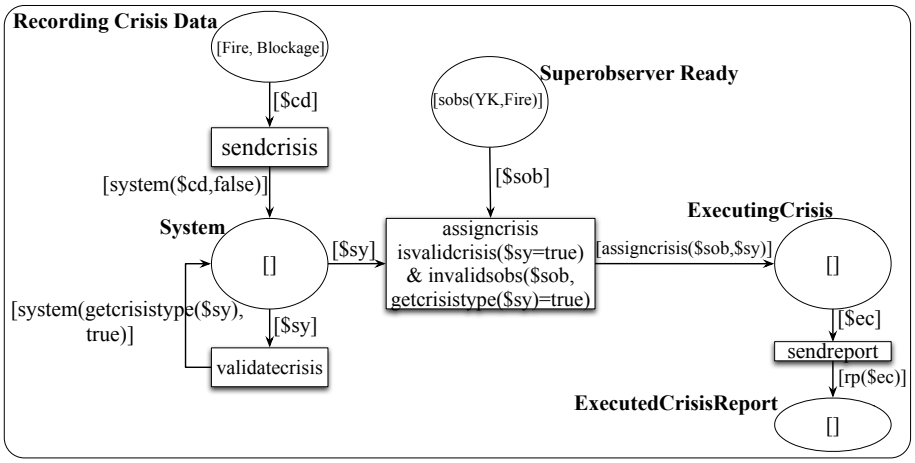

Fig. 7. Car crash APN model

which type of data has been recorded. The input arc of transition `sendcrisis` takes the `cd` variable as an input from the place `Recording Crisis Data` and the output arc contains term `system(cd,false)` of sort `sys`. Initially, every recoded crisis is set to false. The `sendcrisis` transition passes recorded crisis to system for further operations. The output arc of `validatecrisis` contains `system(getcrisistype(sy),true)` term which sends validated crisis to system. The transition `assigncrisis` has two guards, first one is `isvalid(sy)=true` that enables to block invalid crisis reporting to be executed for the mission and the second one is `isvalid(sob,getcrisestype(sy))=true` which is used to block invalid `Superobserver` (a skilled person for handling crisis situation) to execute the crisis mission. The `Superobserver YK` will be assigned to handle `Fire` situation only. The transition `assigncrisis` contains two input arcs with `sob` and `sy` variables and the output arc contains term `assigncrisis(sob,sy)` of sort `crisis`. The output arc of transition `sendreport` contains term `rp(ec)`. This enables to send a report about the executed crisis mission. We refer the interested reader to [8] for the algebraic specification of car crash management system.

An important safety threat, which we will take into an account in this case study is that the invalid crisis reporting can be hazardous. The invalid crisis reporting is the situation that results from a wrongly reported crisis. The execution of crisis mission based on the wrong reporting can waste both human and physical resources. In principle, it is essential to validate the crisis that it is reported correctly. Another, important threat could be to see the number of crisis that can be sent to place `System` should not exceed from a certain limit. Informally, we can define the properties:

φ_1 : All the crises inside place `System` are validated eventually.
φ_2 : Place `System` never contains more than two crises.

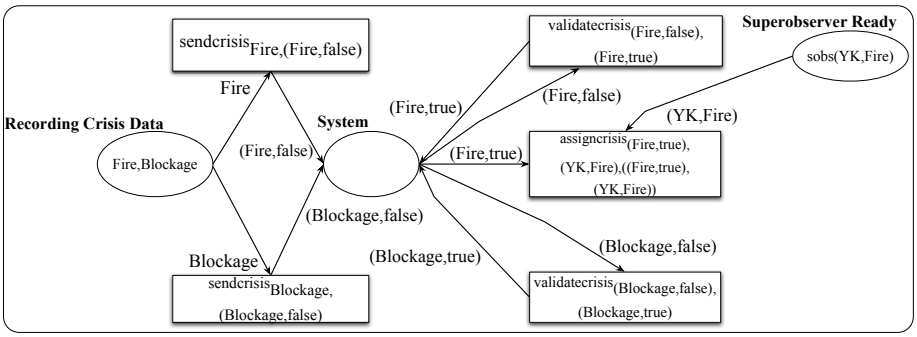

Fig. 8. The unfolded car crash APN model

Formally we can specify the properties as, let `Crises` be a set representing recorded crisis in car crash management system. Let $isvalid : Crises \rightarrow BOOL$, is a function used to validate the recorded crisis.

$\varphi_1 = \mathbf{F}(\forall crisis \in Crises | isvalid(crisis) = true)$

$\varphi_2 = \mathbf{G}(|Crises| \leq 2)$

In contrast to generate the complete state space for the verification of φ_1 *and* φ_2, we alleviate the state space by applying our proposed algorithm. For both φ_1, φ_2 LTL formulas, $scope(\varphi_1 \wedge \varphi_2) \subseteq Q$. The criterion place for both properties is `System`.

The unfolded car crash APN model is shown in Fig. 8. The slicing algorithm *APNSlicing(Unfolded car crash APN model,System)* takes *the unfolded car crash APN model* and *System (an input criterion place)* as an input and iteratively builds a sliced net. The sliced unfolded car crash APN model is shown in Fig. 9, places named `ExecutedCrisis and ExecutedCrisisreporting` together with transition named `sendreport` are sliced away. From the initial marking of Car Crash APN Model `36` states are reachable, whereas sliced car crash APN model has `27` reachable states. The resultant sub net is sufficient to verify both properties (see proof in Theorem 1).

6 Evaluation

In this section, we evaluate our slicing algorithm with the existing benchmark case studies. We measure the effect of slicing in terms of savings of the reachable state space, as the size of the state space usually has a strong impact on time and space needed for model checking. Instead of presenting case studies where our methods work best, it is equally interesting to see where it gives an average or worst case results, so that we will present a comparative evaluation on the benchmark case studies (we refer the interested reader to [10] for APN models of case studies used in the evaluation).

To evaluate our approach, we made the follwing assumptions:

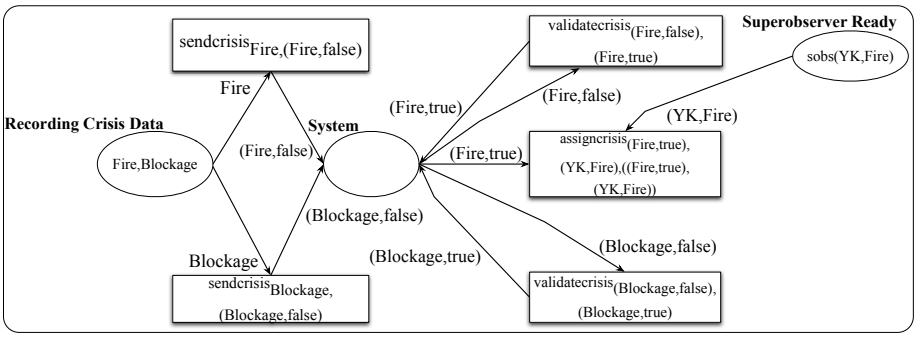

Fig. 9. Sliced and unfolded car crash APN model

– The evaluation procedure is independent of the temporal properties. In general, it is not feasible to determine which places correspond to the interesting properties. Therefore, we generated slices for each place in the given APN model (later, we take some specific temporal properties about the APN models under observation) .

– We abandoned the initially marked places (we follow [16] to assume that there are not interesting properties concerning to those places).

Table 2. Results on different APN models

System	*T.States*	*Bst.Reduct*	*Avg.Reduct*	*Worst Places* (*no reduction*)	*N.Type*
Complaint Handling	*2200*	*98.01%*	*40.54%*	*2*	*Weak.Connect*
Divide & Conquer	*117863*	*99.09%*	*14.22%*	*1*	*Weak.Connect*
Beverage Vending & Machine	*136*	*80.14%*	*02.15%*	*2*	*Weak.Connect*
Insurance Claim	*889*	*99.05%*	*24.52%*	*1*	*Weak.Connect*
A Customer support Production system	*471*	*99.01%*	*37.79%*	*zero*	*Weak.Connect*
Electronic Trade System	*260*	*97.56%*	*62.34%*	*2*	*Weak.Connect*
Daily Routine of 2 Employees & Boss	*80*	*93.75%*	*86.12%*	*zero*	*Str.Connect*
Simple Protocol	*1861*	*95.91%*	*39.01%*	*1*	*Str.Connect*
Producer Consumer	*372*	*0.00%*	*0.00%*	*5*	*Str.Connect*

Let us study the results summarized in the Table.3, the first column shows different APNs models under observation. Based on the initial markings, total number of states is shown in the second column. Best reduction and average reduction (shown in the third and fourth column) refers to the biggest and an average achievable reduction in the state space among all possible properties. The fifth column reports how many places (related to the properties) in the model lead to no reduction using our slicing method. Finally, the structure of APN models under observation is given. Results clearly indicate the significance of slicing; the proposed *APNSlicing* algorithm can alleviate the state space even for some strongly connected nets.

To show that the state space could be reduced for the practically relevant properties, let us take some specific examples of the temporal properties from the APN models shown in Table.3 and compare the reduction in terms of states by applying the *APNSlicing* algorithm. For the *Daily Routine of two Employees and Boss APN model*, for example, we are interested to verify that: *"Every time the boss does not schedule a meeting, he will be at home eventually"*. Formally, we can specify the property:

$\varphi_1 = \mathbf{G}(NM \Rightarrow \mathbf{F}B1)$, *where* "NM" (resp. B1) means "place NM (resp. B1) is not empty".

For a *Producer Consumer APN model* an interesting property could be to verify that: *"Buffer place is never empty"*. Formally, we can specify the property:

$\varphi_2 = \mathbf{G}(|Buffer| > 0)$.

And for a *Complaint Handling APN model*, we are interested to verify: *"All the registered complaints are collected eventually"*. Formally, we can specify the property:

$\varphi_3 = \mathbf{G}(RecComp \Rightarrow \mathbf{F}CompReg)$, *where* "RecComp" (resp. CompReg) means "place RecComp (resp. CompReg) is not empty".

Table 3. Results with different properties concerning to APN models

System	*Tot.States*	*Property*	*Crit.Place(s)*	*Percent.Reduction*
Daily Routine of 2 Employees & Boss	*80*	φ_1	*{NM,B1}*	*75.00%*
Producer Consumer	*372*	φ_2	*{Buffer}*	*0.00%*
Complaint Handling	*2200*	φ_3	*{RecComp,RegComp}*	*50.54%*

Let us study the results summarized in the table shown in Table. 3, the first column represents the system under observation whereas in the second column total number of states are given based on the initially marked places. The third column refers the property that we are looking for the verification. In the fourth column, places are given that are considered as criterion places, and for those

places slices are generated. The fifth column represents the number of states that are reduced (in percentage) after applying *APNSlicing* algorithm.

We can draw the following conclusions from the evaluation results such as:

- The choice of the 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 *non-reading transitions* together with their input places. The fewer *non-reading transitions* are attached to the criterion place, the more reduction is possible.

- Reduction 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 property, remaining part may have more places and transitions that increase the overall number of states. If slicing removes parts of the net that expose highly concurrent behavior, 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.

- For certain strongly connected nets slicing may produce a reduced number of states. For all the strongly connected nets that contain *reading transitions* slicing can produce noteworthy reductions.

- Slicing produces best results for not strongly connected nets. By definition work-flow nets are not strongly connected and since they model work flows, slicing can effectively reduce such nets.

7 Conclusion and Future Work

In this work, we developed an Algebraic Petri net reduction approach to alleviate the state space explosion problem for model checking. The proposed work is based on slicing. The presented slicing algorithm (*APNSlicing*) for Algebraic Petri net guarantees that by construction the state space of sliced net is at most as big as the original net. We showed that the slice allow verification and falsification if Algebraic Petri net is slice fair. Our results show that slicing can help to alleviate the state space explosion problem of Algebraic Petri net model checking.

The future work has twofold objectives; first to implement the proposed slicing construction in AlPiNA (Algebraic Petri net analyzer) a symbolic model checker [6]. As discussed in the section 3.1, we are using the same unfolding approach for APNs as AlPiNA. Obviously, this will reduce the effort in terms of implementation. Secondly, we aim to utilize the sliced net when verifying the evolutions of the net. Slicing can serve as a base step to identify those evolutions that do not require re-verification.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
2. D. Buchs, S. Hostettler, A. Marechal, A. Linard, and M. Risoldi. Alpina: A symbolic model checker. *Springer Berlin Heidelberg*, pages 287–296, 2010.
3. J. R. Burch, E. Clarke, K. L. McMillan, D. Dill, and L. J. Hwang. Symbolic model checking: 1020 states and beyond. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439, 1990.
4. J. Chang and D. J. Richardson. Static and dynamic specification slicing. In *In Proceedings of the Fourth Irvine Software Symposium*, 1994.
5. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.
6. S. Hostettler, A. Marechal, A. Linard, M. Risoldi, and D. Buchs. High-level petri net model checking with alpina. *Fundamenta Informaticae*, 113(3-4):229–264, Aug. 2011.
7. K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer Berlin Heidelberg, 1987.
8. Y. I. Khan. A formal approach for engineering resilient car crash management system. Technical Report TR-LASSY-12-05, University of Luxembourg, 2012.
9. Y. I. Khan. Optimizing verification of structurally evolving algebraic petri nets. In A. Gorbenko, A. Romanovsky, and V. Kharchenko, editors, *Software Engineering for Resilient Systems*, volume 8166 of *Lecture Notes in Computer Science*, pages 64–78. Springer Berlin Heidelberg, 2013.
10. Y. I. Khan. Optmizing algebraic petri net model checking by slicing. Technical Report TR-LASSY-13-02, University of Luxembourg, 2013.
11. L. Lamport. What good is temporal logic. *Information processing*, 83:657–668, 1983.
12. 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*, 32:131–143, 2000.
13. M. Llorens, J. Oliver, J. Silva, S. Tamarit, and G. Vidal. Dynamic slicing techniques for petri nets. *Electron. Notes Theor. Comput. Sci.*, 223:153–165, Dec. 2008.
14. 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*, SOFSEM'08, pages 436–447, Berlin, Heidelberg, 2008. Springer-Verlag.
15. A. Rakow. *Slicing and Reduction Techniques for Model Checking Petri Nets*. PhD thesis, University of Oldenburg, 2011.
16. A. Rakow. Safety slicing petri nets. In S. Haddad and L. Pomello, editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 268–287. Springer Berlin Heidelberg, 2012.
17. W. Reisig. Petri nets and algebraic specifications. *Theor. Comput. Sci.*, 80(1):1–34, 1991.
18. K. Schmidt. T–invariants of algebraic petri nets. *Informatik– Bericht*, 1994.
19. A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, UK, 1998. Springer-Verlag.

20. Y. Wangyang, Y. Chungang, D. Zhijun, and F. Xianwen. Extended and improved slicing technologies for petri nets. *High Technology Letters*, 19(1), 2013.
21. M. Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, ICSE '81, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

Acknowledgment

This work has been supported by the National Research Fund, Luxembourg, Project MOVERE, ref.C09/IS/02.

8 Models

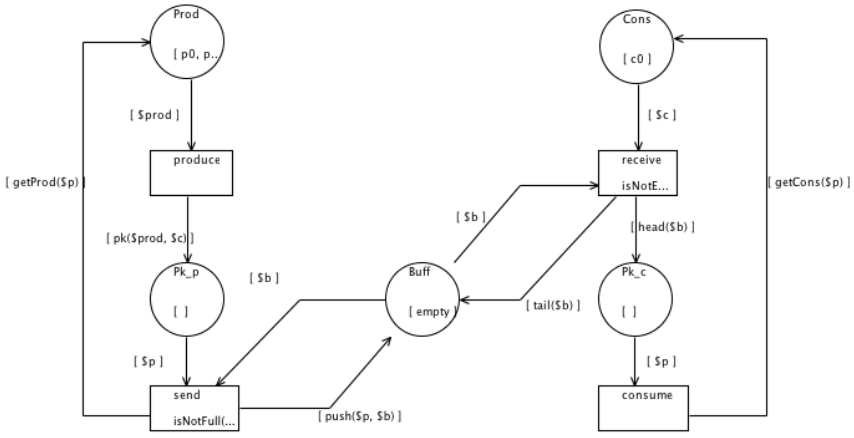

Fig. 10. Producer Consumer APN-MODEL

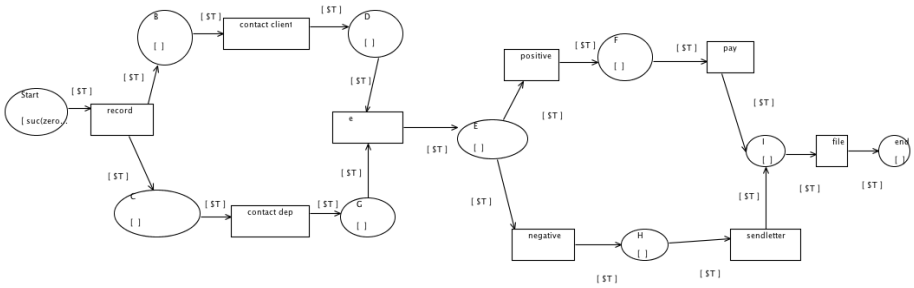

Fig. 11. Complaint Handling APN-MODEL

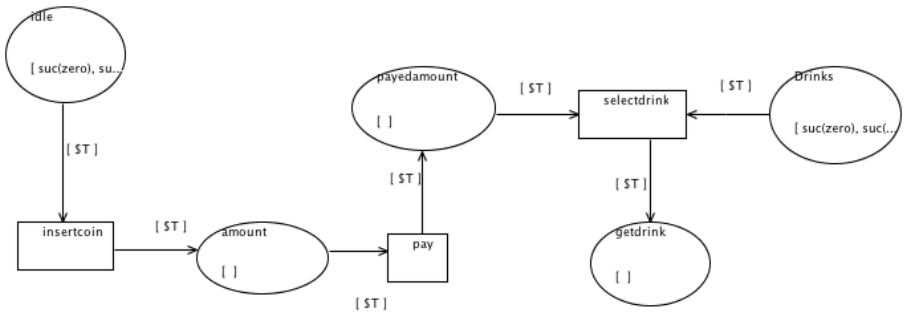

Fig. 12. Bevarage Vending Machine APN-MODEL

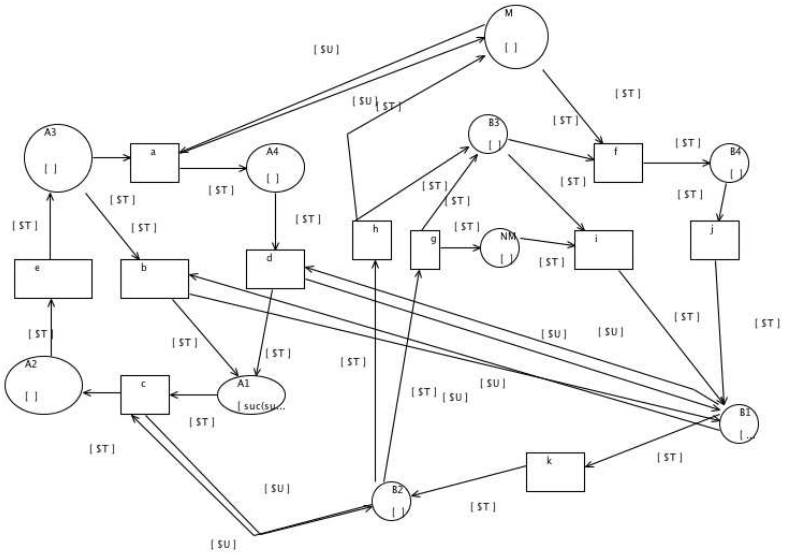

Fig. 13. Daily Routine of 2 employees and Boss APN-MODEL

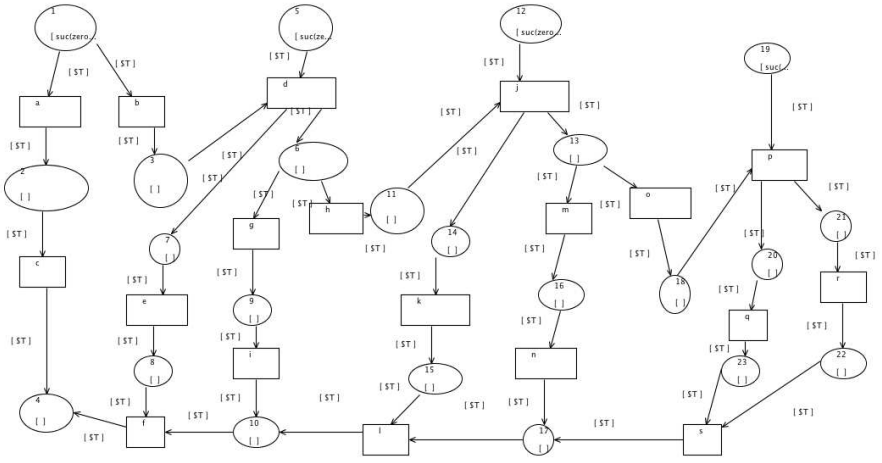

Fig. 14. Divide and Conqure APN-MODEL

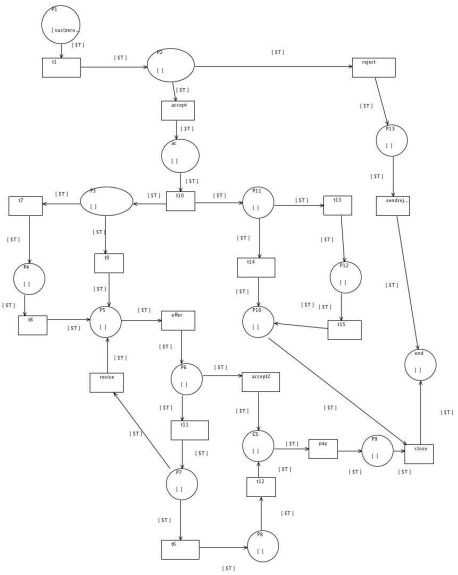

Fig. 15. Insurance Claim APN-MODEL

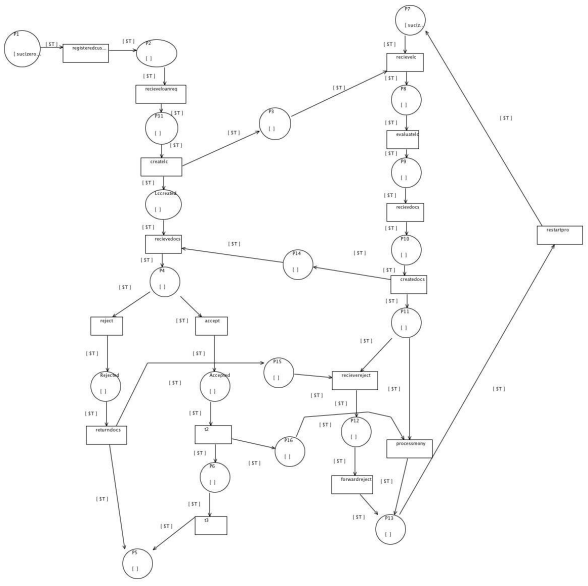

Fig. 16. Electronic Trade System APN-MODEL