Automatic Abstraction Refinement for Petri Nets Verification

Zhenyu Chen, Conghua Zhou, Decheng Ding
Department of Mathematics, Nanjing University, P. R. China
Email: {zchen, chzhou, dcding}@mail.edu.cn

Abstract—Model Checking has emerged as a promising and powerful approach to analyze Petri nets automatically, but a main challenge is the state explosion problem. To obtain an efficient state space, we implement a series of formalisms based on Petri nets to achieve automatically predicate abstraction and refinement via new predicate discovery. However, the complicated predicates would slow down the abstraction refinement process. A novel feature of our approach is minimizing the support of predicates, via diagnosing the failure reasons of transitions and projecting places on predicates to eliminate the dumb variables. In addition, a demonstrative example shows that our techniques could work efficiently on Petri nets.

I. INTRODUCTION

Petri net [1] is a graph-based mathematical formalism that allows to describe the causality, concurrency and conflict relations in systems modeling. Many different application areas have considered Petri nets for the modeling and analysis of their systems, such as operating systems, communication protocols, distributed systems, workflows, etc. In particular, Petri nets play an important role in the specification of concurrent systems.

In the past two decades, model checking [2] has emerged as a promising and powerful approach to automatic verification of systems. By constructing the occurrence (reachable) graphs, we could use model checking techniques to analyze Petri nets. The advantage of model checking is that it is possible to algorithmically reason about the behavior of a system, i.e., verify that the system possesses certain desired properties in the system. Unfortunately, a main challenge of model checking is the state explosion problem [3]: even small systems may have an astronomical or even infinite number of reachable states, and it is a serious limitation on the use of model checking in practice.

There are also many efforts to deal with the state explosion problem. Two popular approaches for Petri nets are the occurrence graphs with equivalence classes (OE-Graphs) [4] and stubborn sets [3], [5]. The latter reduces the size of the state space by eliminating a number of interleaving of independent processes. Here, the full state space cannot be recovered, but it is guaranteed that the desirable properties are not affected by the reduction in the state space. The closest to our work is the former, which identify sets of equivalence states and store only one representative from each set. However, this effect is strong preservation [6]: a property holds on OE-Graphs if and only if it holds on the original one. The disadvantage of strong preservation is that it might lose many significant reductions. Therefore, we introduce a weak preservation approach based on predicate abstraction for Petri nets verification.

Abstraction [6], [7] is probably the most important technique for reducing the state explosion problem. The idea of abstraction is similar to the OE-Graphs, but abstraction is weak preservation [6] for enhanced reduction. Roughly speaking, this reduction is achieved by abstracting away some of the system details that might be irrelevant for the checked property. Verifying the simplified models is in general more efficient than verifying properties of the original ones. Predicate abstraction [8], a special form, has emerged as a promising technique for constructing abstraction automatically. In predicate abstraction, the concrete states are mapped to abstract states with a finite set of predicates, and these predicates are generated from verification conditions and guarded commands of systems by automatic algorithms. Predicate abstraction has been used to verify protocol [9], hardware [10], and software [11]. The SLAM project has also explored predicate abstraction for programming languages, such as C [12]. However, most of existing work only discuss state-based models. Petri nets combine state-based with action-based for convenient expressions [3]. Although we can use state-based methods for action-based verification tasks and vise versa, it still brings on some difficulties for traditional predicate abstraction. In this paper, we define a series of formalisms based on Petri nets for efficient verification.

In many cases, the abstraction may be coarse and could fail to prove the desired property. Choosing a suitable abstraction is not trivial for large systems and requires considerable creativity. To address it, refinement has been used in predicate abstraction via discovering new predicates. A generally useful approach is counterexample guided refinement [13]. It analyzes the spurious abstract counterexample to find a refinement of the current abstraction, such that the spurious counterexample is eliminated from the refined abstract model. In [10], an approach of SAT based conflict clauses analysis is presented to produce new predicates. Discovering new predicate by diagnosing failure proofs in predicate abstraction is described in [14]. These efforts make predicate abstraction be a promising technique to implement model checking for a large system. However, there are several challenges must be overcome before it. The first one is how to efficiently analyze...
a spurious counterexample and discover new predicates. The second one is how to minimize the set of predicates (with fewer predicates) to reduce the size of abstraction. The last, closely related challenge is how to minimize the support of predicates (with fewer clauses and fewer variables) to construct the refined abstract model efficiently.

In this paper, we present a framework based on predicate abstraction and counterexample guided refinement for Petri nets verification. Profiting from the features of firing transitions, we implement an efficient technique for automatic abstraction and discovering new predicates. A novel feature is replacement of a successive state \(s'\) with an assignment of firing transition \(t\) (see in equations 8, 11, 12, etc.). It is convenient for computation and minimization of new predicates. Another important contribution of our approach is to minimize the support of predicates to accelerate abstraction refinement process. We analyze the failure reasons to eliminate the useless transitions and project places on the predicates to eliminate the dumb variables. A demonstrative example shows that our approach could work efficiently on Petri nets. Moreover, although we discuss the classical Petri nets in this paper for simplicity, our ideas could work well to high level Petri nets.

The paper is organized as follows. In the next section, we introduce some basic definitions of Petri nets and model checking. Section 3 presents the framework of abstraction refinement on Petri nets, including construction of the abstract model, identifying the abstract counterexamples, and discovering new predicates. In section 4, we describe how to minimize the support of predicates in detail. A simple example is demonstrated in section 5. The conclusion is drawn in the last section.

II. BASIC DEFINITIONS

A. Petri nets

A Petri net [1] is a 5-tuple \(PN = (P, T, F, W, S_0)\), where \(P = \{p_1, ..., p_m\}\) is a finite set of places, \(T = \{t_1, ..., t_n\}\) is a finite set of transitions \((P \cap T = \emptyset)\), \(F \subseteq (P \times T) \cup (T \times P)\) is a set of arcs (flow relation), \(W : F \rightarrow \mathbb{N}\) is a weight function, \(s_0 : P \rightarrow \mathbb{N}\) is the initial marking. The marking is also called state.

A place \(p\) is called an input place of a transition \(t\) iff there exists a directed arc from \(p\) to \(t\). Place \(p\) is called an output place of transition \(t\) iff there exists a directed arc from \(t\) to \(p\). We use \(\bullet t\) to denote the set of input places for a transition \(t\). The notations \(\bullet t, \bullet p\) have similar meanings. A pair of a place \(p\) and a transition \(t\) is called a self-loop if \(p\) is both an input and output place of \(t\). A Petri net is said to be pure if it has no self-loops. For simplicity, we consider pure Petri nets in this paper. And we could extend to define the weight function:

\[
\begin{align*}
\text{w}(t, p) = \begin{cases} 
  +\text{w}(t, p): & p \in \bullet t \\
  -\text{w}(p, t): & p \in \bullet t \\
  0 & \text{otherwise}
\end{cases}
\end{align*}
\]

(1)

\((w(t, p_1), ..., w(t, p_n))\) is denoted by \(w(t)\). A transition \(t\) is said to be enabled iff each input place \(p\) of \(t\) contains at least \(w(p, t)\) tokens. An enabled transition may fire. If transition \(t\) fires, then \(t\) consumes \(w(p, t)\) tokens from each input place \(p\) of \(t\) and produces \(w(t, p)\) tokens for each output place \(p\) of \(t\).

We can formalize the firing rule with the following formula.

\[
\forall p \in \bullet t, s(p) \geq w(p, t) \quad \text{t is enabled. And firing t can get a next-step marking} \quad s' = s + w[t], \quad \text{that is:}
\]

\[
\forall p \quad s'(p) = s(p) + w(t, p)
\]

(2)

A sequence of firings will result in a sequence of markings. A marking \(s_n\) is said to be reachable from a marking \(s_0\) if there exists a sequence of firings that transforms \(s_0\) to \(s_n\). A firing or occurrence sequence is denoted by \(\sigma = s_0 t_1 s_1 t_2 s_2 ... t_n s_n\). It is simplified to \(\sigma = t_1 ... t_n\) if we only focus on the transitions and to \(\sigma = s_1 s_2 ... s_n\) if we do not care the transitions. In this case, \(s_n\) is reachable from \(s_0\) by \(\sigma\) and we write \(s_0 \rightarrow^\sigma s_n\). The set of all possible markings reachable from \(s_0\) in a Petri net \(PN\) is denoted by \(R(PN, s_0)\) or simply \(R(PN)\).

A place \(p\) is called \(k\)-bounded iff \(\forall s \in R(PN), s(p) \leq k\). A Petri net is \(k\)-bounded iff every place is \(k\)-bounded. A Petri net is bounded iff it is \(k\)-bounded for some \(k\). A Petri net is safe iff it is 1-bounded. A Petri net is unbounded iff it is not \(k\)-bounded for any \(k\).

B. Model Checking

Generally speaking, model checking [2] is to decide whether a given structure \(M\) is a model of a logical formula \(f\), i.e., whether \(M\) satisfies \(f\), abbreviated as \(M \models f\). Intuitively, \(M\) is a model of the system and \(f\) is a desirable property, which is typically drawn from temporal logic. The model checker then provides a fully automatic approach for proving whether the system modeled by \(M\) enjoys this property \(f\).

Model checking typically depends on a discrete model of a system represented by a graph structure. Graphs alone are too weak to provide an interesting description, so they are annotated with more specific information. Two approaches in common use are Kripke structures (KS) [2] and Labelled Transition Systems (LTS) [3]. We combine them to a definition for Petri nets:

Definition 1 (Concrete Model): Given a Petri net \(PN = (P, T, F, W, S_0)\). A state space (concrete model) over \(PN\) is a 5-tuple \(M = (S, T, R, L, I)\). Where:

1. \(S = R(PN)\) is a set of states (may be infinite).
2. \(T\) is a finite set of transitions.
3. \(R(s, t, s') = s \rightarrow t s'\) is a transition relation.
4. \(L : S \rightarrow \mathbb{N}^{|P|}\) is an interpretation, i.e. \(L(s) = (s(p_1), ..., s(p_n))\), or denoted by \(s = (p_1, ..., p_n)\) for simplicity.
5. \(I \subseteq S\) is a set of initial states.

In order to make \(M\) be consistent with the semantics of temporal logics, we require that \(R\) must be total, that is, for every state \(s \in S\) there is a state \(s' \in S\) such that \(R(s, t, s')\). \(s \rightarrow^t s'\) means that \(s\) enables \(t\) and if firing \(t\), we get \(s'\). Formally, we define two predicates as follows:

\[
\text{En}(s, t) := s + w[t] \geq 0
\]

(3)
\[ R(s, t, s') := E_n(s, t) \land (s' = s + w[t]) \]  

(4)

\[ R(s, t, s') \] could denote by \( R(s_1, s_2) \) for simplicity if we do not care the transition \( t \). \( R(s, s') \) is formalized as:

\[ R(s, s') := \exists t. R(s, t, s') \]  

(5)

A path is an infinite sequence \( \pi = s_0s_1s_2 \ldots \) such that for every \( i \geq 0 \), \( R(s_i, s_{i+1}, s_{i+1}) \). It is simplified to \( \pi = t_1t_2 \ldots \) if we only focus on the transitions and to \( \pi = s_0s_1 \ldots \) if we do not care the transitions.

Temporal logic is a formalism for describing sequences of transitions between states in a reactive system. A powerful logic is called CTL* [2]. CTL* formulas are composed of path quantifiers \( A, E \) and temporal operators \( X, F, G, U, R \). ACTL* [2] is the universal fragment of CTL* without existential path quantify \( E \) in negation normal form (NNF).

### III. ABSTRACTION REFINEMENT ON PETRI NETS

Now, we introduce an overview of abstraction refinement framework for Petri nets. Given a Petri net \( PN \) and a property specification \( f \):

1. Extract a set of predicates \( \Phi = \{ \phi_1, ..., \phi_m \} \) including the atomic propositions of \( f \), construct an initial abstract model \( M^a \) from \( PN \) (details in section 3.1).
2. Check whether \( f \) holds on \( M^a \). If \( M^a \models f \), then return TRUE, i.e., the concrete model \( M \) satisfies \( f \).
3. If \( M^a \not\models f \), then check counterexample on the Petri net \( PN \) (details in section 3.2).
   3.1. If the counterexample is real, then return FALSE, i.e., the concrete model \( M \) does not satisfy \( f \).
   3.2. Else, add a new predicate to refine the model \( M^a \) (details in section 3.3), go to step 2.

For general Petri nets, model checking problem (even for the fragment of CTL allowing only \( EF \)) is known to be undecidable. For simplicity, we only consider bounded Petri nets in this paper. For such finite state systems, the above procedure is complete, that is, this process is repeated until the property is proved or a real counterexample is generated. The details of this framework are presented in the following subsections.

#### A. Construction of Abstract Model

Predicate abstraction can be regarded as a special case of conservative abstraction [14]. The key idea in conservative abstraction is that if the property specification \( f \) holds in abstract model then it holds in the concrete model. In predicate abstraction, a set of predicates \( \Phi = \{ \phi_1, ..., \phi_m \} \) includes those in the property to be verified. \( \Phi = \{ \phi_1, ..., \phi_m \} \) is the set of predicates with the variables \( p_1, ..., p_n \), which are the places of Petri nets \( PN \). Since \( s = (p_1, ..., p_n) \), \( \phi_i(s) \) denotes \( \phi_i(p_1, ..., p_n) \) and \( \Phi(s) \) denotes \( \phi_1(s), ..., \phi_m(s) \) for simplicity. Let \( \{ b_1, ..., b_m \} \) be the corresponding set of boolean variables. A literal \( l_i \) is either \( b_i \) or \(-b_i \) for some \( 1 \leq i \leq n \).

In the framework of predicate abstraction, an abstract state is a bit-vector \( s^a = (l_1, ..., l_m) \), and \( s^a(i) = l_i \). On the other hand, an abstract state could be viewed as a set of concrete states.

In [14], the abstraction is formalized as a standard Galois connection, having an abstraction function, \( \alpha \) which maps concrete states to bit-vectors and a concretization function \( \gamma \) which is essentially the inverse image of \( \alpha \). In this paper, we use an abstraction function \( \alpha \) to determine the relations between abstract states and concrete states. The abstraction function \( \alpha : S \times S^a \rightarrow \{0, 1\} \) is defined as:

\[ \alpha(s, s^a) := s \geq 0 \land \Phi(s) = s^a \]  

(6)

Please notice that \( s \geq 0 \) is the enabling condition of its predecessor \( s' \), because of \( s = s' + w[t] \) and \( E_n(s', t) = s' + w[t] \geq 0 \). Following the abstraction function \( \alpha \), each abstract state represents an equivalence class of concrete states. Similar to [7], [6], [14], we define the abstract model as follows:

**Definition 2 (Abstract Model):** Given a Petri net \( PN \), let \( M = (S, T, R, L, I) \) be the corresponding concrete model and \( \Phi = (\phi_1, ..., \phi_m) \) be a set of predicates. An abstract model of \( PN \) over \( \Phi \) is a 4-tuple \( M^a = (S^a, R^a, L^a, I^a) \). Where:

1. \( S^a \) is a finite set of abstract states.
2. \( R^a : S^a \times S^a \rightarrow \{0, 1\} \) is an abstract transition relation:
   \[ R^a(s^a, s'^a) := \exists s, s'. \alpha(s, s^a) \land \alpha(s', s'^a) \land R(s, s') \]  
   (7)
3. \( L^a : S \rightarrow 2^{I^a} \) is an interpretation, i.e. \( L^a(s^a) = (l_1, ..., l_m) \), or denoted by \( s^a = (l_1, ..., l_m) \) for simplicity.
4. \( I^a = \{ s^a \exists s \ I(s) \land \alpha(s, s^a) \} \) is a set of initial abstract states.

\( R^a \) is total, since \( R \) is total. A path is an infinite sequence \( \pi^a = s_0^a s_1^a s_2^a \ldots \) such that for each \( i \geq 0 \), \( R^a(s_i^a, s_{i+1}^a) \). We could simplify the abstract transition relation (7). Obviously, \( R^a(s^a, s'^a) \) can be stated equivalently as \( \exists t, s, s'. \alpha(s, s^a) \land \alpha(s', s'^a) \land R(s, t, s') \). Since \( R(s, t, s') = E_n(s, t) \land (s' = s + w[t]) \), if \( s' \) is replaced with \( s + w[t] \), we get an equivalent definition of abstract transition relation as follows:

\[ R^a(s^a, s'^a) := \exists s, t. \alpha(s, s^a) \land \alpha(s + w[t], s'^a) \]  

(8)

In order to reason about the relation between \( M \) and \( M^a \), we introduce simulation preorder [2], [15]. A relation \( H \subseteq S \times S^a \) is a simulation relation if and only if for all \( s, s' \), if \( H(s, s') \) then the following conditions hold: (1) \( \alpha(s, s^a) = 1 \). (2) For every state \( s' \) such that \( R(s, s') \), there is a state \( s^a \) satisfying \( R^a(s^a, s'^a) \) and \( H(s', s^a) \).

If there is a simulation preorder \( H \) for every initial state \( s_0 \in I \), there exists \( s_0^a \in I^a \) such that \( H(s_0, s_0^a) \), we say that \( M^a \) simulates \( M \), denoted by \( M \preceq M^a \). We prove \( M^a \) is a conservative abstraction of \( M \) informally. Considering \( H(s, s^a) \), if \( R(s, s') \), then there is an abstract state \( s'^a \), such that \( \alpha(s', s'^a) = 1 \). Since \( R(s, s') \) and \( \alpha(s, s^a) = 1 \), following item 2 in definition 2, \( R^a(s^a, s'^a) \) holds. It is formalized as the following preservation theorem [2], [7].

**Theorem 3:** If \( M \) is a concrete model and \( M^a \) is a corresponding abstract model, then

\[ M \preceq M^a \]  

(9)
Furthermore, if $M \preceq M^a$, then for every $ACTL^*$ formula $f$, $M^a \models f \Rightarrow M \models f$ [2], [7]. For simplicity, we consider the property specification as an $ACTL^*$ formula in the following sections.

B. Checking Finite Prefix Counterexample

When checking the abstract model $M^a$, if $M^a \models f$, then it returns TRUE. However, if $M^a \not\models f$, we could not draw any conclusions, since the abstract counterexample may be a spurious path. A spurious path is an abstract path which has no corresponding concrete path.

In this paper, we consider the counterexample generated by the model checker as a finite path $\pi^a = s_0^a, ..., s_k^a$. $\pi^a$ is a spurious prefix iff the following concretization condition is unsatisfiable [10], [14].

$$CC(\pi^a) := \exists s_0, ..., s_k. I(s_0) \land \bigwedge_{i=0}^{k-1} \alpha(s_i, s_{i+1}) \land R(s_i, s_{i+1})$$

A spurious prefix is a prefix of abstract path which has no corresponding concrete prefix. A prefix $\pi^a = s_0^a, ..., s_k^a$ is a minimal spurious prefix iff $CC(\pi^a)$ is unsatisfiable but $CC(\pi^a_{k-1})$ is satisfiable. A spurious path has a unique minimal spurious prefix and the number $k$ is called the failure index. Following the definition, we could increase the prefix from $\pi^a_0$ one state by one state until $CC(\pi^a)$ is unsatisfiable to get the minimal spurious prefix.

Since $R(s, s') = \exists t. R(s, t, s')$ and $R(s, t, s') = En(s, t) \land (s' = s + w[t])$, if $s'$ is replaced with $s + w[t]$, we get an equivalent definition of concretization condition as follows:

$$CC(\pi^a) := \exists s_0, t_1, ..., t_k. I(s_0) \land \bigwedge_{i=0}^{k-1} \alpha(s_0 + \sum_{j=1}^{i} w[t_j], s_i^a)$$

C. Refinement for Abstract Model

In the previous two subsections, we show how to construct an abstract model and verify it. Unfortunately, choosing a suitable abstraction is not trivial for large systems and requires considerable creativity. In this section, we introduce an automatic approach to refine the abstract model via discovering new predicates.

The deadend and bad sets of states are important terms in the refinement technique applied in [6], [13]. Now, we introduce them here to produce new predicates. Suppose $\pi^a_k$ (in Fig.1) is a minimal spurious prefix of an abstract counterexample. Since $\pi^a_{k-1}$ is real, there exists a corresponding concrete path $x_0, x_1, ..., x_{k-1}$. Let $D$ denote all such states $x_k-1$. We call $D$ the set of deadend states. On the other hand, since $R(s_k, s_{k-1}, s^a_k)$, there exists a corresponding concrete transition $R(y_{k-1}, y_k)$. Let $B$ denote all such states $y_k-1$. We call $B$ the set of bad states. $D \cap B = \emptyset$, otherwise, there is a concrete path $x_0, x_1, ..., x_{k-1}, y_k$, i.e., $\pi^a_k$ is real. Obviously, the existing set of predicates does not distinguish between $D$ and $B$, because the abstractions of the two are the same abstract state $s^a_k$.

Abstraction refinement is to produce a new predicate $\phi_{\text{new}}$. It splits the abstract state $s^a_{k-1}$ into two, $s^a_D$ and $s^a_B$, such that $D \subseteq s^a_D, B \subseteq s^a_B$, $s^a_D \cap s^a_B = \emptyset$, and $s^a_D \cup s^a_B = s^a_{k-1}$. A successful refinement discovers a new predicate $\phi_{\text{new}}$ to separate the deadend set $D$ from the bad set $B$, that is, $s \in D \Rightarrow \phi_{\text{new}}(s) = 0$ and $s \in B \Rightarrow \phi_{\text{new}}(s) = 1$.

Notice that, $B$ is the preimage of abstract state $s^a_k$, i.e., $B = \{s|\exists s'. \alpha(s, s^a_{k-1}) \land \alpha(s', s^a_k) \land R(s, s')\}$. Similar to (8), it is equivalent to $B = \{s|\exists t. \alpha(s, s^a_{k-1}) \land \alpha(s + w[t], s^a_k)\}$. We define a new predicate as follows:

$$\phi_{\text{new}}^1(s) := \exists t. \alpha(s, s^a_{k-1}) \land \alpha(s + w[t], s^a_k)$$

Obvioulsy, $\phi_{\text{new}}^1(s) = 1 \Leftrightarrow s \in B$, that is, $\phi_{\text{new}}^1$ splits the state $s^a_{k-1}$ into $B$ and $s^a_D - B$. Since $D \cap B = \emptyset$, $D \subseteq s^a_B \subseteq B$, i.e., $s \in D \Rightarrow \phi_{\text{new}}^1(s) = 0$. Otherwise, there exists a corresponding concrete path $x_0, x_1, ..., x_{k-1}, y_k$, i.e., $\pi^a_k$ is a real abstract path. Therefore, given the minimal spurious prefix $\pi^a_k$, we could produce new predicates $\phi_{\text{new}}^1$, automatically.

IV. MINIMIZING THE SUPPORT OF PREDICATES

In the previous section, we introduce how to construct abstract model and analyze spurious counterexample to discover new predicates. Another important goal of our abstraction refinement is to compute a predicate that can be represented compactly (with fewer clauses and fewer variables). The more compact predicates are, the more efficient we construct the abstract models.

A. Projection on $\Phi$ to Eliminate Dumb Places

In some industrial cases, the places involved in property specification $f$ are far less than the places of Petri nets. Notice that $s = (p_1, ..., p_n)$ and $\alpha(s, s')$ is $(s \geq 0) \land (\Phi(s) = s')$. $\Phi$ is the set of predicates. $s|_{\Phi}$ denotes the projection of $(p_1, ..., p_n)$ on $\Phi$, i.e., omitting the dumb place variables in $\Phi$. For a predicate $h(x)$, we define $h|_{\Phi}(x) = h(x|_{\Phi})$. If $x = s$, then $x|_{\Phi} = s|_{\Phi}$, otherwise, $x|_{\Phi} = x$.

By the above definition, the following claims hold:
1. $\Phi|_{\Phi}(s) = \Phi(s)$
2. $\alpha|_{\Phi}(s, s') \equiv (s|_{\Phi} \geq 0 \land \Phi(s) = s')$
3. $\alpha|_{\Phi}(s, s') \Rightarrow \alpha|_{\Phi}(s, s')$
4. $R^a(s^a, s'^a) \Rightarrow R^a|_{\Phi}(s^a, s'^a)$.

The claims 1-3 hold obviously and we prove the fourth one.
Proof: \( R^\ast(s^a, s^{\alpha}) \Rightarrow R^\ast[\Phi](s^a, s^{\alpha}) \), since \( \alpha(s, s^{\alpha}) \Rightarrow \alpha[\Phi](s, s^a) \). If \( R^\ast[\Phi](s^a, s^{\alpha}) \), that is, \( \exists s, t, \alpha[\Phi](s, s^a) \land \alpha(\Phi(s + w[t], s^{\alpha})) \). For each place \( p \) not involved in \( \Phi \), if \( s(p) < 0 \) or \( s(p) < w[t] \), we let \( s(p) = |w[t]| \), then the new state \( s \) satisfies \( s \geq 0 \) and \( s \geq w[t] \). And this assignment does not change the values of predicates in \( \Phi \). Thus \( \Phi(s) = s^a \land \Phi(s + w[t]) = s^{\alpha} \). Therefore, \( R^\ast[\Phi](s^a, s^{\alpha}) \Rightarrow R^\ast(s^a, s^{\alpha}). \)

Claim 1 shows that two different abstraction functions, \( \alpha \) and \( \alpha[\Phi] \), derive the same set of abstract states. Claim 4 shows that the abstract transition relation is invariant, no matter using \( \alpha \) or \( \alpha[\Phi] \). That is, we could construct an abstract model by the projection abstraction function \( \alpha[\Phi] \) the same as \( \alpha \). It is formalized as the following theorem.

**Theorem 4:** Given a Petri net \( PN \) and a set of predicates \( \Phi \), the abstract model \( M^\alpha \) is constructed by \( \alpha \) and the abstract model \( M^\alpha[\Phi] \) is constructed by \( \alpha[\Phi] \), then \( M^\alpha = M^\alpha[\Phi] \).

In general, the places involved in \( \alpha[\Phi] \) are far less than the places in \( PN \), thus \( \alpha[\Phi] \) is simpler than \( \alpha \). Therefore, the construction of abstract model based on \( \alpha[\Phi] \) would be more efficient.

**B. Eliminating Redundant Clauses of New Predicates**

For a minimal spurious prefix \( \pi_k^a \), the new predicate of refinement \( \phi_{new}^1(s) \) is \( \alpha(s, s_{k-1}^a) \land \exists t. \alpha(s + w[t], s_{k-1}^a) \). As we know, \( s \in s_{k-1}^{a} \) if and only if \( \alpha(s, s_{k-1}^a) = 1 \). That is, when \( s \in s_{k-1}^{a} \), \( \alpha(s, s_{k-1}^a) \) is a redundant clause of \( \phi_{new}^1 \). Therefore, we could define a new compact predicate as follows:

\[
\phi_{new}^3(s) := \bigvee_{t \in T} \alpha(s + w[t], s_{k-1}^a)
\]

Since \( \alpha(s, s_{k-1}^a) \land \phi_{new}^2 = \phi_{new}^1 \), \( \phi_{new}^2 \Rightarrow \phi_{new}^1 \). That is \( \phi_{new}^2 \) splits \( s_{k-1}^a \) the same as \( \phi_{new}^1 \).

On the other hand, if \( s \not\in s_{k-1}^{a} \), then \( \phi_{new}^1(s) = 0 \). That is, \( \phi_{new}^1 \) does not split any abstract state, except \( s_{k-1}^{a} \). Actually, \( \phi_{new}^2 \) may split other abstract states beside \( s_{k-1}^{a} \). For any new predicate \( \phi_{new} \), conjunction with \( \alpha(s, s_{k-1}^{a}) \) is to restrict it only to split \( s_{k-1}^{a} \) and keep other abstract states intact. This could reduce the size of refined abstract model. However, it would lose the advantages of compact predicates and some potential splits.

**C. Diagnosing the Failure Levels of Transitions for Minimizing**

Given a minimal spurious prefix \( \pi_k^a \), consider \( \phi_{new}^2 \) and expand it as \( \bigvee_{t \in T} \alpha(s + w[t], s_{k-1}^a) \land \exists t. \alpha(s + w[t], s_{k-1}^a) \). The size of new predicate \( \phi_{new}^2 \) lies on the number of transitions. \( \mathcal{CC}(\pi_k^a) \) is unsatisfiable, that is, all transitions are fail to move concrete state \( s_{k-1} \) (which comes from \( s_0 \) along \( \pi_k^a \)) to the next abstract state \( s_{k}^a \).

Intuitively, there may be some different failure reasons of transition \( t \): (1) \( t \) could not work between \( s_{k-1}^{a} \) and \( s_{k}^{a} \). (2) Any concrete state in \( s_{k-1}^{a} \), which comes from \( s_0 \) along \( \pi_k^a \), could not enable \( t \). (3) There exists \( s_{k-1} \) enabling \( t \), but \( s_{k-1} + w[t] \) does not belong to \( s_{k}^{a} \). We could diagnose the failure reasons of transitions to reduce the support of \( \phi_{new}^2 \). Given a minimal spurious prefix \( \phi_{new}^2 \), we define the failure levels of transitions as follows:

- **Failure Level 1:** \( FL1(\pi_k^a) = \{ t(\alpha(s, s_{k-1}^a) \land \alpha(s + w[t], s_{k}^a) \text{ unsatisfiable} \} \).
- **Failure Level 2:** \( FL2(\pi_k^a) = \{ t(\mathcal{CC}(\pi_k^a) \land (s_{k-1} + w[t] \geq 0) \text{ unsatisfiable} - FL1(\pi_k^a) \}. \)
- **Failure Level 3:** \( FL3(\pi_k^a) = T - FL1(\pi_k^a) \cup FL2(\pi_k^a) \).

Following the failure levels transitions, we define a new compact predicate as follows:

\[
\phi_{new}^3(s) := \bigvee_{t \in FL2(\pi_k^a)} (s + w[t] \geq 0) \lor \bigvee_{t \in FL3(\pi_k^a)} \alpha(s + w[t], s_{k}^a)
\]

**Theorem 5:** Given a minimal spurious prefix \( \pi_k^a \), \( D \) is the deadend set and \( B \) is the bad set of \( s_{k-1}^{a} \). Then \( s \in B \Rightarrow \phi_{new}^1(s) = 0 \) and \( s \in B \Rightarrow \phi_{new}^1(s) = 1 \). That is, \( \phi_{new}^1 \) separates \( D \) from \( B \).

**Proof:** Since \( D = \{ s|\exists t. \alpha(s, s_{k-1}^a) \land \alpha(s + w[t], s_{k}^a) \} \) if \( s \in B \), there is a transition \( t \notin FL1(\pi_k^a) \), \( \alpha(s, s_{k-1}^a) \land \alpha(s + w[t], s_{k}^a) \) is \( \text{TRUE} \). Then, if \( t \notin FL2(\pi_k^a) \), then \( s + w[t] \geq 0 \), else \( t \in FL3(\pi_k^a) \), then \( \alpha(s + w[t], s_{k}^a) \) is \( \text{TRUE} \). Thus \( \phi_{new}^3(s) = 1 \). Thus, \( s \in B \Rightarrow \phi_{new}^3(s) = 1 \).

If \( s \notin D \), then there is no transition \( t \) to satisfy \( \alpha(s, s_{k-1}^a) \land \alpha(s + w[t], s_{k}^a) \) (otherwise, \( \pi_k^a \) is real). Since \( \alpha(s, s_{k-1}^a) \) is \( \text{TRUE} \), \( \alpha(s + w[t], s_{k}^a) \) is \( \text{FALSE} \). Thus \( \bigvee_{t \in FL2(\pi_k^a)} (s + w[t] \geq 0) \) is \( \text{FALSE} \). Therefore, \( \phi_{new}^3(s) = 0 \).

Consider \( \phi_{new}^3 \) and expand it as \( \bigvee_{t \in FL2(\pi_k^a)} (s + w[t], p_0 \geq 0) \lor \bigvee_{t \in FL3(\pi_k^a)} (s + w[t], p \geq 0) \lor \bigvee_{(t, p_0 \in *)} \phi_{new}^1(s + w[t], p) \).

From the failure levels transitions, we know that the transitions involved in \( \phi_{new}^3 \) is minimized. But the places \( p \in * \) and predicates \( \phi_1 \) in \( \phi_{new}^3 \) have not been done. We could reduce the items \( (s + w[t], p_0 \geq 0) \) and \( \phi_{new}^1(s + w[t], p) \) greedily and keep the minimal new predicates \( \phi_{new}^3 \) separate deadend set \( D \) from bad set \( B \).

**V. A DEMONSTRATIVE EXAMPLE**

In this section, we give a simple example based on Petri net to demonstrate our framework of predicate abstraction and refinement.

**A. Original System**

Fig.2 shows a simple example based on Petri net \( PN = (P, T, W, s_0) \) (\( F \) is ignored), where:

- \( P = \{ x, y, z \} \) is a finite set of places.
- \( T = \{ t_1, t_2, t_3 \} \) is a finite set of transitions.
- The vectors of weights are \( w[t_1] = (-2, 1, 2), w[t_2] = (2, -1, 0), w[t_3] = (4, -1, -4) \), where \( w[t] \) denotes \( (w[t, x], w[t, y], w[t, z]) \).
- \( s_0 = (2, 1, 2) \) is the initial state.

Obviously, it is an unbounded Petri net. The number of reachable states is infinite and we could not get the whole concrete model \( M \) of \( PN \). However, we want to decide
whether there is a reachable state, such that the tokens in y are more than the ones in z, i.e., EF(y > z).

And since $EF(y > z) \Leftrightarrow \neg AG(y \leq z)$, we just need to check whether $M \models f(f = AG(y \leq z))$.

**B. Predicate Abstraction**

$f = AG(y \leq z)$, and it involves only one predicate $y \leq z$. Let $\phi_1 = y \leq z$. Fig. 3 shows the abstract model $M^a$ of $PN$ over $\phi_1$.

There are two abstract states $s^a_0$ and $s^a_1$. The labels of them are $L(s^a_0) = \phi_1$ and $L(s^a_1) = \neg \phi_1$. It means that $s^a_0$ is the abstraction of all concrete states $s$, such that $\phi_1(s) = TRUE$, and, $s^a_1$ is the abstraction of all concrete states $s$, such that $\phi_1(s) = FALSE$. And $s^a_0$ is the initial abstract state, because $\phi_1(2,1,2) = TRUE$.

For Petri nets, the abstraction function $\alpha[\phi(s, s^a)]$ is formalized as $(s \models \phi \geq 0) \wedge (\Phi(s) = s^a)$. The abstract transition relation is defined as $R^a[\phi(s^a, s^a')] \Leftrightarrow \exists s, s'. \alpha[\phi(s, s')] \wedge \alpha[\phi(s^a, s^a')] \wedge R[\phi(s, s')]$, the right part of this formula is equivalent to

$$\exists s, t. \alpha[\phi(s, s^a)] \wedge \alpha[\phi(s + w[t], s^a')]$$

(15)

For $R^a(s^a_0, s^a_1)$, there is a transition $t_1$ to satisfy (15). That is, for a concrete state $s_0 \in s^a_0$, i.e., $y \leq z = TRUE$, if firing $t_1$, we get the next state $s_1$, such that $s_0 \in s^a_0$. Similarly, for $R^a(s^a_1, s^a_0)$, there are $t_1$, $t_2$, and $t_3$; for $R^a(s^a_0, s^a_1)$, there is $t_3$; and for $R^a(s^a_1, s^a_0)$, there are $t_1$ and $t_2$.

**C. Checking Counterexample**

When model checking $M^a$ with $f$, it returns $FALSE$ and outputs a counterexample $\pi^a_2 = s^a_0s^a_1$. Obviously, $s_0 = (2,1,2)$ is the concrete state of $s^a_0$. However, there is no transition $t$, such that firing $t$ then we can get a concrete state $s_1$ of $s^a_1$. There is only $t_3$ satisfying (15) for $R^a(s^a_0, s^a_1)$, but $s_0$ cannot enable $t_3$. Therefore, $\pi^a_2 = s^a_0s^a_1$ is a spurious counterexample, actually it is a minimal spurious prefix. This could be done with checking concretization condition $CC$ in (10) or (11). Now, we need to eliminate the spurious counterexample to refine the abstract model.

**D. Discovering and Minimizing New Predicates**

$\pi^a_2 = s^a_0s^a_1$ is the minimal spurious prefix. Different from the SAT-based approaches in [10], [14], we could produce the new predicates $\phi^3_{new}$ as $\alpha(s, s^a_0) \wedge \bigvee_{t \in T} \alpha(s + w[t], s^a_1)$ efficiently and directly. Since $T = \{t_1, t_2, t_3\}$, expand $\phi^3_{new}$ as follows:

$((x, y, z) \geq 0) \wedge ((y + 1 \leq z + 2))$

$\lor ((x + 2, y - 1, z) \geq 0 \wedge (y - 1 \leq z))$

$\lor ((x + 4, y - 1, z - 4) \geq 0 \wedge (y - 1 \leq z - 4))$

By diagnosing the failure levels of transitions, we find that $t_1$ and $t_2$ belong to FL1 and $t_3$ belongs to FL2. Therefore, we get the $\phi^2_{new}$ as follows:

$((x, y, z) \geq 0) \wedge ((y + 1 \leq z + 2))$

$\lor ((x + 2, y - 1, z) \geq 0 \wedge (y - 1 \leq z))$

$\lor ((x + 4, y - 1, z - 4) \geq 0 \wedge (y - 1 \leq z - 4))$

If it needs to restrict new predicate only to split $s^a_0$, $\phi^4_{new}$ conjuncts with $\phi_1$ to get a new predicates $\phi^5_{new}$ as follows:

$\phi^5_{new} = (y \leq z) \wedge (z - 4 \geq 0)$

The comparison between $\phi^4_{new}$ and $\phi^5_{new}$ will be demonstrated in the following subsection.

**E. Refinement of Abstract Model**

We construct a refined abstract model (Fig. 4) of $PN$ over the predicates $\phi_1 = y \leq z$ and $\phi_2 = \phi^5_{new}$. The abstract states are $s^a_0, s^a_1, s^a_2$ and $s^a_3$. When $\neg \phi_2$, i.e., $z < 4$, it can not change $\phi_1$ into $\neg \phi_1$. That is, when $\neg \phi_2$, there is no transition satisfying (15), for $R^a(s^a_0, s^a_2)$ and $R^a(s^a_0, s^a_3)$. Similarly, if $\phi_2 = \phi^5_{new}$, we could construct a refined abstract model in Fig. 5.

Now, the spurious counterexample $\pi^a_3 = s^a_0s^a_1s^a_2$ is eliminated. Model checking the refined abstract model (Fig. 4 or Fig. 5) with $f = AG\phi_1$, it returns $FALSE$ and output a new counterexample $\pi^a_3 = s^a_0s^a_1s^a_2$. This is a real counterexample, because there is a corresponding concrete path $\pi = (2,1,2)(0,2,4)(4,1,0)\ldots$. Therefore, the Petri net $PN$ does not satisfy $AG(y \leq z)$, i.e., $PN$ satisfies $EF(y > z)$.
example shows that our approach could work efficiently on Petri nets. However, it needs further experimental results to compare with other abstraction refinement approaches. In our future work, we would implement tools to automatic abstraction refinement based on this framework, and investigate some case studies.

ACKNOWLEDGMENT

The authors would like to thank all members of the business integration technologies team in IBM China Research Lab for their valuable discussions. The authors are also grateful to the referees for their helpful comments and suggestions.

REFERENCES