Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 223 (2008) 153–165

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Dynamic Slicing Techniques for Petri Nets[1]

M. Llorens[2] J. Oliver[2] J. Silva[2] S. Tamarit[2] G. Vidal[2]

DSIC, Technical University of Valencia
Valencia, Spain

Abstract

Petri nets provide a means for modelling and verifying the behavior of concurrent systems. Program slicing is a well-known technique in imperative programming for extracting those statements of a program that may affect a given program point. In the context of Petri nets, computing a net slice can be seen as a graph reachability problem. In this paper, we propose two slicing techniques for Petri nets that can be useful to reduce the size of the considered net, thereby simplifying subsequent analysis and debugging tasks by standard Petri net techniques.

Keywords: Petri nets, program slicing, reachability analysis

1 Introduction

Program slicing is a method for decomposing programs in order to extract parts of them—called program *slices*—which are of interest. This technique was first defined by Mark Weiser [20] in the context of program debugging. In particular, Weiser's proposal was aimed at using program slicing for isolating the program staments that may contain a bug, so that finding this bug becomes simpler for the programmer. In general, slicing extracts the statements that may affect some point of interest, referred to as *slicing criterion*.

Let us illustrate this technique with an example taken from [19]. Figure 1(a) shows a simple program which requests a positive integer number n and computes the sum and the product of the first n positive integer numbers. Figure 1(b) shows a slice of this program w.r.t. the slicing criterion (`10`,`product`), i.e., variable `product` in line `10`. As can be seen in the figure, all the computations that do not contribute to the final value of the variable `product` have been removed from the slice.

[1] This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grants TIN2005-09207-C03-02, TIN2008-06622-C03-02, and *Acción Integrada* HA2006-0008.

[2] Email: {`mllorens`,`fjoliver`,`jsilva`,`stamarit`,`gvidal`}`@dsic.upv.es`

doi:10.1016/j.entcs.2008.12.037

```
(1)  read(n) ;
(2)  i := 1 ;
(3)  sum := 0 ;
(4)  product := 1 ;
(5)  while i <= n do
        begin
(6)        sum := sum + i ;
(7)        product := product * i ;
(8)        i := i + 1 ;
        end ;
(9)  write (sum) ;
(10) write (product) ;
```

(a) Example program.

```
read(n) ;
i := 1 ;

product := 1 ;
while i <= n do
   begin

      product := product * i ;
      i := i + 1 ;
   end ;

write (product) ;
```

(b) Program slice w.r.t. (10,product).

Fig. 1. Sub-figures 1(a) and 1(b) show an example of program slicing.

The work by Weiser has inspired a lot of different approaches to compute slices which include generalizations and concretizations of the initial approach. In general, all of them are classified into two classes: *static* and *dynamic*. 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.

In this work, we propose the use of slicing techniques to produce subnets of a Petri net. A Petri net [13,14] is a graphic, mathematical tool used to model and verify the behavior of systems that are concurrent, asynchronous, distributed, parallel, non-deterministic and/or stochastic. As a graphic tool, they provide a visual understanding of the system and the mathematical tool facilitates its formal analysis. State space methods are the most popular approach to automatic verification of concurrent systems. In their basic form, these methods explore the transition system associated with the concurrent system. The transition system is a graph, known as the *reachability graph*, that represents the system's reachable states as nodes: there is an arc from one state s to another s', whenever the system can evolve from s to s'. In the worst case, state space methods have to explore all the nodes and transitions in the transition system. This makes the method useless in practice, even though it is simple in concept, due to the state-explosion problem that occurs when a Petri net is applied to nontrivial real problems. The technique is costly even in bounded nets with a finite number of states since, in the worst case, the reachable states are multiplied beyond any primitive recursive function. For this reason, various approaches have been proposed to minimize the number of system states to be studied in a reachability graph [17].

Program slicing has a great potential here since it allows us to syntactically reduce a model in such a way that the reduced model is composed only of those parts that may influence the slicing criterion. Since it was originally defined by Weiser, program slicing has been applied to different formalisms which are not strictly programming languages, like attribute grammars [18], hierarchical state machines [9], Z and CSP-OZ specifications [5,2,3], etc. Unfortunately, very little work has been carried out on slicing for Petri nets (some notable exceptions are [4,11,15,16]). For instance, Chang and Wang [4] present a static slicing algorithm for Petri nets that slices out all sets of paths, known as concurrence sets, so that all paths within the same set should be executed concurrently. In [11], a static slicing technique for Petri nets is proposed in order to divide enormous P/T nets into manageable modules so that the divided model can be analyzed by a compositional reachability analysis technique. A Petri net model is partitioned into concurrent units (Petri net slices) using minimal invariants. In order to preserve all the information in the original model, uncovered places should be added into minimally-connectable concurrent units since minimal invariants may not cover all the places. Finally, in [15,16], Rakow presents another static slicing technique to reduce the Petri net size and, thus, lessen the problem of state explosion that occurs in the *model checking* [6] of Petri nets [1]. From the best of our knowledge, there is no previous proposal for *dynamic* slicing of Petri nets. This is surprising because considering an initial marking and/or a particular sequence of transition firings would allow us to further reduce the size of the slices and focus on a particular use of the considered Petri net.

In this work, we explore two different alternatives for dynamic slicing of Petri nets. Firstly, we present a slicing technique that extends the slicing criterion in [15,16] in order to also consider an initial marking. We show that this information can be very useful when analyzing Petri nets and, moreover, it allows us to significantly reduce the size of the computed slice. Furthermore, we show that our algorithm is, in the worst case, as precise as Rakow's algorithm. This can still be seen as a lightweight approach to slicing since its cost is bounded by the number of transitions in the Petri net. Then, we present a second approach that further reduces the size of the computed slice by only considering a particular execution—here, a sequence of transition firings. Clearly, in this case the computed slice is only useful to analyze the considered firing sequence. We illustrate both techniques with examples.

2 Petri Nets

A Petri net [13,14] is a directed bipartite graph, whose two essential elements are called *places* (represented by circles) and *transitions* (represented by bars or rectangles). The edges of the graph form the *arcs*, which are labelled with a positive integer known as *weight*. Arcs run from places to transitions and vice versa. The *state* of the system modeled by the net is represented by assigning non-negative integers to places. This is known as a *marking*, and is shown graphically by adding small black circles to the places, known as *tokens*. The *dynamic behavior* of the system is simulated by changes in the markings of a Petri net, a process which is carried out by the firing of the transitions. The basic concepts of Petri nets are summarized as follows:

Definition 2.1 A *Petri net* [13,14] is a tuple $\mathcal{N} = (P, T, F)$, where:

- P is a set of *places*.

- T is a set of *transitions*, such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.
- F is the *flow relation* that assigns weights to arcs: $F : P \times T \ \cup \ T \times P \to \mathbb{N}$.

The *marking* M of a Petri net is defined over the set of places P. For each place $p \in P$ we let $M(p)$ denote the number of tokens contained in p.

A *marked Petri net* Σ is a pair $(\mathcal{N}, M)$ where $\mathcal{N}$ is a Petri net and M is a marking. We denote by M_0 the *initial marking* of the net.

In the following, given a marking M and a set of places P, we denote by $M|_P$ the restriction of M over P, i.e., $M|_P(p) = M(p)$ for all $p \in P$ and $M|_P$ is undefined otherwise.

Definition 2.2 [14] Given a Petri net $\mathcal{N} = (P, T, F)$, we say that a marking M' *covers* a marking M if $M' \geq M$, i.e., $M'(p) \geq M(p)$ for each $p \in P$.

Given a Petri net $\mathcal{N} = (P, T, F)$, we say that a place $p \in P$ is an *input (resp. output) place* of a transition $t \in T$ iff there is an *input (resp. output) arc* from p to t (resp. from t to p). Given a transition $t \in T$, we denote by ${}^\bullet t$ and $t^\bullet$ the set of all input and output places of t, respectively. Analogously, given a place $p \in P$, we denote ${}^\bullet p$ and $p^\bullet$ the set of all input and output transitions of p, respectively.

Definition 2.3 Let $\Sigma = (\mathcal{N}, M)$ be a marked Petri net, with $\mathcal{N} = (P, T, F)$. We say that a transition $t \in T$ is *enabled* in M, in symbols $M \xrightarrow{t}$, iff for each input place $p \in P$ of t, we have $M(p) \geq F(p, t)$. A transition may only be fired if it is enabled.

The *firing* of an enabled transition t in a marking M eliminates $F(p, t)$ tokens from each input place $p \in {}^\bullet t$ and adds $F(t, p')$ tokens to each output place $p' \in t^\bullet$, producing a new marking M', in symbols $M \xrightarrow{t} M'$.

We say that a marking M_n is *reachable* from an initial marking M_0 if there exists a *firing sequence* $\sigma = t_1 t_2 \ldots t_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$. In this case, we say that M_n is reachable from M_0 through σ, in symbols $M_0 \xrightarrow{\sigma} M_n$. This notion includes the empty sequence ϵ; we have $M \xrightarrow{\epsilon} M$ for any marking M. We say that a firing sequence is *initial* if it starts from an initial marking.

The set of all possible markings which are reachable from an initial marking M_0 in a marked Petri net $\Sigma = (\mathcal{N}, M_0)$ is denoted by $R(\mathcal{N}, M_0)$ (or simply by $R(M_0)$ when $\mathcal{N}$ is clear from the context).

The following notion of *subnet* will be particularly relevant in the context of slicing (roughly speaking, we will identify a slice with a subnet). Let $P' \times T' \ \cup \ T' \times P' \subseteq P \times T \ \cup \ T \times P$, we say that a flow relation $F' : P' \times T' \ \cup \ T' \times P' \to \mathbb{N}$ is a restriction of another flow relation $F : P \times T \cup T \times P \to \mathbb{N}$ over P' and T', in symbols $F|_{(P',T')}$, if F' is defined as follows: $F'(x, y) = F(x, y)$ if $(x, y) \in P' \times T' \ \cup \ T' \times P'$ and F' is not defined otherwise.

Definition 2.4 [8] A *subnet* $\mathcal{N}' = (P', T', F')$ of a Petri net $\mathcal{N} = (P, T, F)$ is a Petri net such that $P' \subseteq P$, $T' \subseteq T$ and F' is a restriction of F over P' and T', i.e., $F' = F|_{(P',T')}$.

3 Dynamic Slicing of Petri Nets

In this section, we introduce our first approach to dynamic slicing of Petri nets. We say that our slicing technique is *dynamic* since an initial marking is taken into account (in contrast to previous approaches, e.g., [4,11,15,16]).

Using an initial marking can be useful, e.g., in debugging. Consider for instance that the user is analyzing a particular trace for a marked Petri net (using a simulation tool [7], which we assume correct), so that an erroneous state is reached. Here, by *erroneous* state, we mean a marking in which some places have an incorrect number of tokens. In this case, we are interested in extracting the set of places and transitions (more formally, a subnet) that may erroneously contribute tokens to the places of interest, so that the user can more easily locate the bug.

Therefore, our first notion of *slicing criterion* is formalized as follows:

Definition 3.1 Let $\mathcal{N} = (P, T, F)$ be a Petri net. A *slicing criterion* for $\mathcal{N}$ is a pair $\langle M_0, Q\rangle$ where M_0 is an initial marking for $\mathcal{N}$ and $Q \subseteq P$ is a set of places.

Roughly speaking, given a slicing criterion $\langle M_0, Q\rangle$ for a Petri net $\mathcal{N}$, we are interested in extracting a subnet with those places and transitions of $\mathcal{N}$ which can contribute to change the marking of Q in any execution starting in M_0.

Our notion of *dynamic* slice is defined as follows. In the following, we say that σ' is a *subsequence* of a firing sequence σ w.r.t. a set of transitions T if σ' contains all transitions of σ that belong to T and in the same order.

Definition 3.2 Let $\mathcal{N} = (P, T, F)$ be a Petri net and let $\langle M_0, Q\rangle$ be a slicing criterion for $\mathcal{N}$. Given a Petri net $\mathcal{N}' = (P', T', F')$, we say that $\mathcal{N}'$ is a slice of $\mathcal{N}$ w.r.t. $\langle M_0, Q\rangle$ if the following conditions hold:

- the Petri net $\mathcal{N}'$ is a subnet of $\mathcal{N}$ and
- for each firing sequence $\sigma = t_1 \dots t_n$, for $\mathcal{N}$, with $M_0 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n$ such that $M_{n-1}(p) < M_n(p)$ for some $p \in Q$, there exists a firing sequence σ' for $(\mathcal{N}', M'_0)$, with $M'_0 = M_0|_{P'}$, such that
  · σ' is a subsequence of σ w.r.t. T',
  · $M'_0 \xrightarrow{\sigma'} M'_m$, $m \leq n$, and
  · M'_m covers $M_n|_{P'}$ (i.e., $M'_m \geq M_n|_{P'}$).

Intuitively speaking, a Petri net $\mathcal{N}'$ is a slice of another Petri net $\mathcal{N}$ if $\mathcal{N}'$ is a subnet of $\mathcal{N}$ (i.e., no additional places nor transitions are added) and the behaviour of $\mathcal{N}$ is preserved in $\mathcal{N}'$ for the restricted sets of places and transitions. In order to formalize this second condition, we require that, for all firing sequences $\sigma = t_1 \dots t_n$ that may *move* tokens to the places of the slicing criterion, i.e.,

$$M_0 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n \text{ and } M_{n-1}(p) < M_n(p),\ p \in Q$$

the *restriction* of this firing sequence can also be performed on the slice $\mathcal{N}'$, i.e.,

$$M'_0 \xrightarrow{\sigma'} M'_m \text{ and } M'_m \geq M_n$$

Trivially, given a Petri net $\mathcal{N}$, the complete net $\mathcal{N}$ is always a correct slice w.r.t. any slicing criterion. The challenge then is to produce a slice as small as possible.

Algorithm 1 Dynamic slicing of a marked Petri net.

Let $\mathcal{N} = (P, T, F)$ be a Petri net and let $\langle M_0, Q\rangle$ be a slicing criterion for $\mathcal{N}$. First, we compute a *backward slice* similar to that of [15]. This is obtained from $\mathsf{b\_slice}_{\mathcal{N}}(Q, \{\,\})$, where function $\mathsf{b\_slice}_{\mathcal{N}}$ is defined as follows:

$$\mathsf{b\_slice}_{\mathcal{N}}(W, W_{done}) = \begin{cases} \{\,\} & \text{if } W = \{\,\} \\ T \cup {}^{\bullet}T \cup \mathsf{b\_slice}_{\mathcal{N}}(W \setminus W'_{done}, W'_{done}) & \\ \qquad \text{if } W \neq \{\,\}, \text{ where } T = {}^{\bullet}p, \text{ and } W'_{done} = W_{done} \cup \{p\} & \\ \qquad\qquad \text{for some } p \in P & \end{cases}$$

Now, we compute a *forward slice* from

$$\mathsf{f\_slice}_{\mathcal{N}}(\{p \in P \mid M_0(p) > 0\}, \{\,\}, \{t \in T \mid M_0 \xrightarrow{t}\})$$

where function $\mathsf{f\_slice}_{\mathcal{N}}$ is defined as follows:

$$\mathsf{f\_slice}_{\mathcal{N}}(W, R, V) = \begin{cases} W \cup R & \text{if } V = \{\,\} \\ \mathsf{f\_slice}_{\mathcal{N}}(W \cup V^{\bullet}, R \cup V, V') & \\ \qquad \text{if } V \neq \{\,\}, \text{ where } V' = \{t \in T \setminus (R \cup V) \mid {}^{\bullet}t \subseteq W \cup V^{\bullet}\} & \end{cases}$$

Then, the dynamic slice is finally obtained from the intersection of the backward and forward slices. Formally, let

$$P' \cup T' = \mathsf{b\_slice}_{\mathcal{N}}(Q, \{\,\}) \cap \mathsf{f\_slice}_{\mathcal{N}}(\{p \in P \mid M_0(p) > 0\}, \{\,\}, \{t \in T \mid M_0 \xrightarrow{t}\})$$

with $P' \subseteq P$ and $T' \subseteq T$, the computed slice is

$$\mathcal{N}' = (P', T', F|_{(P', T')})$$

Algorithm 1 describes our method to extract a dynamic slice from a Petri net. Intuitively speaking, Algorithm 1 constructs the slice of a Petri net (P, T, F) for a set of places $Q \subseteq P$ as follows. The key idea is to capture a possible token flow relevant for places in Q. For this purpose,

- we first compute the possible paths which lead to the slicing criterion,
- then we also compute the paths that may be followed by the tokens of the initial marking.

This can be done by taking into account that (i) the marking of a place p depends on its input and output transitions, (ii) a transition may only be fired if it is enabled, and (iii) the enabling of a transition depends on the marking of its input places. The algorithm is divided in three steps:

- The first step is a backward slicing method (which is similar to the *basic slicing algorithm* of [15]) that obtains a slice $\mathcal{N}_1 = (P_1, T_1, F_1)$ defined as the subnet of

$\mathcal{N}$ that includes all input places of all transitions connected to any place p in P_1, starting with $Q \subseteq P_1$.

- The core of this method is the auxiliary function $\mathsf{b\_slice}_{\mathcal{N}}$, which is initially called with the set of places Q of the slicing criterion together with an empty set of places.
- For a particular non-empty set of places W and a particular place $p \in W$, function $\mathsf{b\_slice}_{\mathcal{N}}$ returns the transitions T in ${}^\bullet p$ and the input places of these transitions ${}^\bullet T$. Then, function $\mathsf{b\_slice}_{\mathcal{N}}$ moves backwards adding the place p to the set W_{done} and removing from W the updated set W_{done} until the set W becomes empty.

- The second step is a forward slicing method that obtains a slice $\mathcal{N}_2 = (P_2, T_2, F_2)$ defined as the subnet of $\mathcal{N}$ that includes all transitions initially enabled in M_0 as well as those transitions connected as output transitions of places in P_2, starting with $p \in P$ such that $M_0(p) > 0$.
  - We define an auxiliary function $\mathsf{f\_slice}_{\mathcal{N}}$, which is initially called with the places that are marked at M_0, an empty set of transitions and the enabled transitions in M_0.
  - For a particular set of places W, a particular set of transitions R and a particular non-empty set of transitions V, function $\mathsf{f\_slice}_{\mathcal{N}}$ moves forwards adding the places in $V^\bullet$ to W, adding the transitions in V to R and replacing the set of transitions V by a new set V' in which are included the transitions that are not in $R \cup V$ and whose input places are in $W \cup V^\bullet$.
  - Finally, when V is empty, function $\mathsf{f\_slice}_{\mathcal{N}}$ returns the accumulated set of places and transitions $W \cup R$.

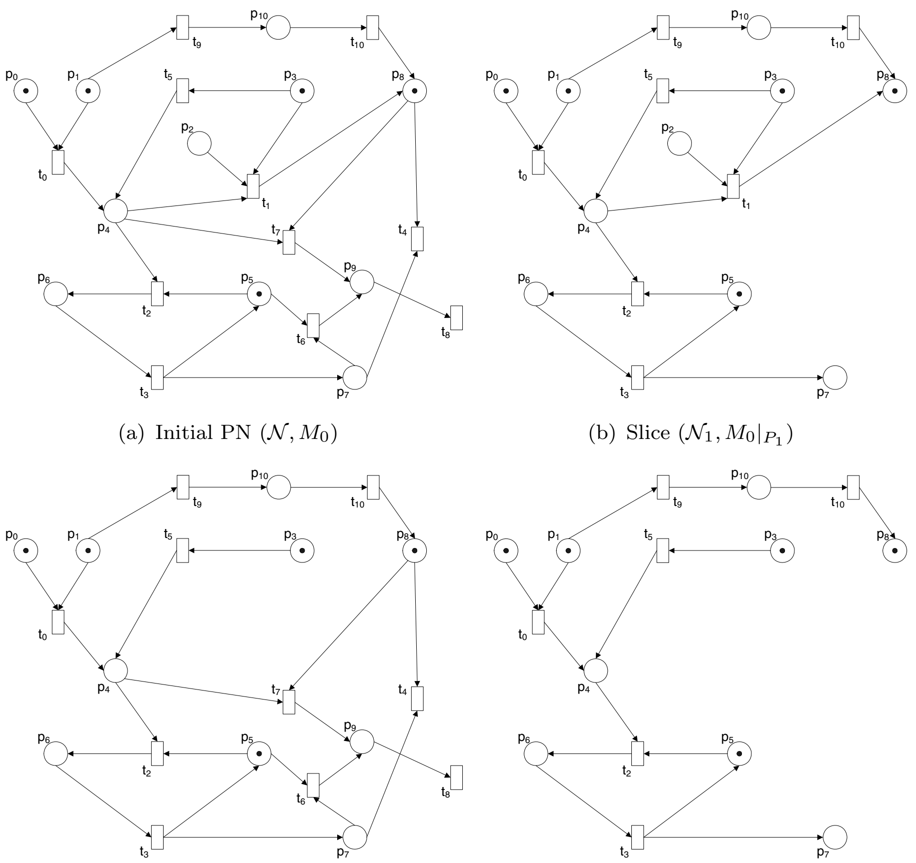
- Finally, the third step obtains the slice $\mathcal{N}' = (P', T', F')$ defined as the subnet of $\mathcal{N}$ where P' is the intersection of P_1 and P_2, T' is the intersection of T_1 and T_2, and F' is the restriction of F over P' and T', i.e., the intersection of backward and forward slices.

The following result states the completeness of our algorithm for computing Petri net slices. The proof of this result follows easily by induction on the length of the firing sequences considered in Definition 3.2.

Theorem 3.3 *Let $\mathcal{N}$ be a Petri net and $\langle M_0, Q\rangle$ be a slicing criterion for $\mathcal{N}$. The dynamic slice $\mathcal{N}'$ computed in Algorithm 1 is a correct slice according to Definition 3.2.*

We will now show the usefulness of the technique with a simple example.

Example 3.4 Consider the Petri net $\mathcal{N}$ of Fig. 2(a) where the user wants to produce a slice w.r.t. the slicing criterion $\langle M_0, \{p_5, p_7, p_8\}\rangle$. Figure 2(b) shows the slice $\mathcal{N}_1$ obtained in the first part of Algorithm 1. Figure 2(c) shows the slice $\mathcal{N}_2$ obtained in the second part of Algorithm 1. The subnet shown in Fig. 2(d) is the final result of Algorithm 1 (the intersection of $\mathcal{N}_1$ and $\mathcal{N}_2$). This slice contains all the places and transitions of the original Petri net which can transmit tokens to the slicing criterion.



(a) Initial PN $(\mathcal{N}, M_0)$

(b) Slice $(\mathcal{N}_1, M_0|_{P_1})$

(c) Slice $(\mathcal{N}_2, M_0|_{P_2})$

(d) Slice result of Algorithm 1

Fig. 2. Example of an application of Algorithm 1

Clearly, using an initial marking allows us to produce smaller slices. Surprisingly, previous approaches completely ignored the marking of the net, and thus their slices are often rather big. For instance, the slice of Fig. 2(b) is a subset of the slice produced by Rakow's algorithm [15] (this algorithm would also include transitions t_4, t_6 and t_7). Clearly, this slice contains parts of the Petri net that cannot be reached with the given initial marking (e.g., transition t_1 which could never be fired because place p_2 is empty). Rakow's algorithm computes all the parts of the Petri net which could transmit tokens to the slicing criterion and, thus, the associated slicing criterion is just $\langle Q \rangle$, where $Q \subseteq P$ is a set of places. In contrast, we compute all the parts of the Petri net which could transmit tokens to the slicing criterion from the initial marking. Therefore, our technique is essentially a generalization of Rakow's technique because the slice produced with Rakow's algorithm w.r.t. $\langle Q \rangle$ is the same as the slice produced w.r.t. $\langle M_0, Q \rangle$ if $M_0(p) > 0$ for all $p \in P$ and all $t \in T$ are enabled transitions at M_0.

Our slicing technique is more general than Rakow's technique but, at the same time, it keeps its simplicity and efficiency because we still use the Petri net structure to produce the slice. Therefore, our first approach can be considered *lightweight* because its cost is bounded by the number of transitions T of the original Petri net; namely, the cost of our algorithm is $\mathcal{O}(2T)$.

4 Extracting Slices from Traces

In this section, we present an alternative approach to dynamic slicing that generally produces smaller slices by also considering a particular firing sequence.

In principle, Algorithm 1 should consider all possible executions of the Petri net starting from the initial marking. This approach can be useful in some contexts but it is too imprecise for debugging when a particular simulation has been performed. Therefore, in our second approach, we refine the notion of slicing criterion so as to also include the firing sequence that represents the erroneous simulation. By exploting this additional information, the new slicing algorithm will usually produce smaller slices. Formally,

Definition 4.1 Let $\mathcal{N} = (P, T, F)$ be a Petri net. A *slicing criterion* for $\mathcal{N}$ is a triple $\langle M_0, \sigma, Q\rangle$ where M_0 is a marking for $\mathcal{N}$, σ is an initial firing sequence (i.e., starting from M_0) and $Q \subseteq P$ is a set of places.

Roughly speaking, given a slicing criterion $\langle M_0, \sigma, Q\rangle$ for a Petri net, we are interested in extracting a subnet with those places and transitions which are necessary to move tokens to the places in Q.

Our notion of *dynamic* slice is defined as follows:

Definition 4.2 Let $\mathcal{N} = (P, T, F)$ be a Petri net. Let $\langle M_0, \sigma, Q\rangle$ be a slicing criterion for $\mathcal{N}$, with $\sigma = t_1 t_2 \ldots t_n$. Given a Petri net $\mathcal{N}' = (P', T', F')$, we say that $\mathcal{N}'$ is a slice of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma, Q\rangle$ if the following conditions hold:

- the Petri net $\mathcal{N}'$ is a subnet of $\mathcal{N}$,
- the set of places Q appears in P' (i.e., $Q \subseteq P'$), and
- there exists a firing sequence σ' for $(\mathcal{N}', M'_0)$, with $M'_0 = M_0|_{P'}$, such that
  - · σ' is a subsequence of σ w.r.t. T',
  - · $M'_0 \xrightarrow{\sigma'} M'_m$, $m \leq n$, and
  - · M'_m covers $M_n|_{P'}$ (i.e., $M'_m \geq M_n|_{P'}$).

Trivially, given a marked Petri net $(\mathcal{N}, M_0)$, the complete net $\mathcal{N}$ is always a correct slice w.r.t. any slicing criterion. The challenge then is to produce a slice as small as possible.

Intuitively speaking, given a slicing criterion $\langle M_0, \sigma, Q\rangle$, the slicing algorithm proceeds as follows:

- The core of the algorithm lies in the auxiliary function slice, which is initially called with the marking M_n which is reachable from M_0 through σ, together with the firing sequence σ and the set of places Q of the slicing criterion.
- For a particular marking M_i, $i > 0$, a firing sequence σ and a set of places W, function slice just moves "backwards" when no place in W increased its tokens by the considered firing.

Algorithm 2 Extracting slices from traces.

Let $\mathcal{N} = (P, T, F)$ be a Petri net and let $\langle M_0, \sigma, Q\rangle$ be a slicing criterion for $\mathcal{N}$, with $\sigma = t_1 t_2 \ldots t_n$. Then, we compute a dynamic slice $\mathcal{N}'$ of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma, Q\rangle$ as follows:

- We have $\mathcal{N}' = (P', T', F')$, where $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$, $P' \cup T' = \mathsf{slice}(M_n, \sigma, Q)$, $P' \subseteq P$, $T' \subseteq T$, and $F' = F|_{(P',T')}$. Auxiliary function slice is defined as follows:

$$\mathsf{slice}(M_i, \sigma, W) = \begin{cases} W & \text{if } i = 0 \\ \mathsf{slice}(M_{i-1}, \sigma, W) & \text{if } \forall p \in W.\ M_{i-1}(p) \geq M_i(p),\ i > 0 \\ \{t_i\} \cup \mathsf{slice}(M_{i-1}, \sigma, W \cup {}^{\bullet}t_i) & \text{if } \exists p \in W.\ M_{i-1}(p) < M_i(p),\ i > 0 \end{cases}$$

- The initial marking M'_0 is the restriction of M_0 over P', i.e., $M'_0 = M_0|_{P'}$.

- Otherwise, the fired transition t_i increased the number of tokens of some place in W. In this case, function slice already returns this transition t_i and, moreover, it moves backwards also adding the places in ${}^{\bullet}t_i$ to the previous set W.
- Finally, when the initial marking is reached, function slice returns the accumulated set of places (which includes the initial places in Q).

We will now show the utility of the technique with a simple example.

Example 4.3 Consider the Petri net $\mathcal{N}$ of Example 3.4 shown in Fig. 2(a), together with the firing sequence σ shown in Fig. 3(b). The firing sequence $\sigma = t_5 t_2 t_3 t_0 t_2 t_3$ corresponds to the branch of the reachability graph shown in Fig. 3(a) that goes from the root to the node M_{45}. Then, the user can define the slicing criterion $\langle M_0, \sigma, \{p_5, p_7, p_8\}\rangle$ for $\mathcal{N}$; where M_0 is the initial marking for $\mathcal{N}$ defined in Fig 2(a).

Clearly, this slicing criterion focus on a particular execution and thus the slice produced is more precise than the one produced by Algorithm 1. In this case, the slice of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma, \{p_5, p_7, p_8\}\rangle$ is the Petri net shown in Fig. 3(c).

The following result states the completeness of our algorithm for computing Petri net slices.

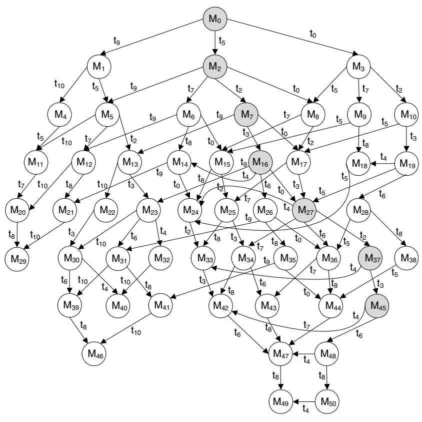
Theorem 4.4 *Let $\mathcal{N} = (P, T, F)$ be a Petri net and let $\langle M_0, \sigma, Q\rangle$ be a slicing criterion for $\mathcal{N}$. The dynamic slice $\mathcal{N}'$ computed in Algorithm 2 is a correct slice according to Definition 4.2.*

Proof. (Sketch) We prove the claim by induction on the number n of transitions in σ.

If $n = 0$, then $\mathsf{slice}(M_0, \sigma, Q) = \bigcup_{p \in Q} \mathsf{slice}(M_0, \sigma, \{p\}) = Q$ and the claim follows trivially for $\mathcal{N}' = (Q, \{\}, \{\})$ and $M'_0 = M_0|_Q$.

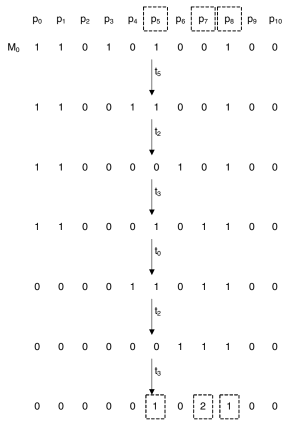
If $n > 0$, then we distinguish two cases:

- If $M_{n-1}(p) \geq M_n(p)$ for all $p \in Q$, then $\mathsf{slice}(M_n, \sigma, Q) = \mathsf{slice}(M_{n-1}, \sigma, Q)$ and the claim follows by induction.



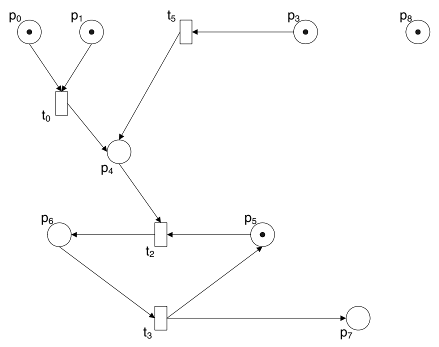
M_0	1 1 0 1 0 1 0 0 1 0 0	M_{17}	0 0 0 0 1 0 1 0 1 0 0	M_{34}	0 0 0 0 0 1 0 1 0 1 0
M_1	1 0 0 1 0 1 0 0 1 0 1	M_{18}	0 0 0 1 0 1 0 0 0 0 0	M_{35}	1 1 0 0 0 0 0 0 1 0 0
M_2	1 1 0 0 1 1 0 0 1 0 0	M_{19}	0 0 0 1 0 1 0 1 1 0 0	M_{36}	0 0 0 0 1 0 0 0 1 1 0
M_3	0 0 0 1 1 1 0 0 1 0 0	M_{20}	1 0 0 0 0 1 0 0 1 1 0	M_{37}	0 0 0 0 0 0 1 1 1 0 0
M_4	1 0 0 1 0 1 0 0 2 0 0	M_{21}	1 0 0 0 0 1 0 0 0 0 1	M_{38}	0 0 0 1 0 0 0 0 1 0 0
M_5	1 0 0 0 1 1 0 0 1 0 1	M_{22}	1 0 0 0 0 0 1 0 2 0 0	M_{39}	1 0 0 0 0 0 0 0 2 1 0
M_6	1 1 0 0 0 1 0 0 0 1 0	M_{23}	1 0 0 0 0 1 0 1 1 0 1	M_{40}	1 0 0 0 0 1 0 0 1 0 0
M_7	1 1 0 0 0 0 1 0 1 0 0	M_{24}	0 0 0 0 1 1 0 0 0 0 0	M_{41}	1 0 0 0 0 0 0 0 1 0 1
M_8	0 0 0 0 2 1 0 0 1 0 0	M_{25}	0 0 0 0 0 0 1 0 0 1 0	M_{42}	0 0 0 0 0 1 0 1 0 0 0
M_9	0 0 0 1 0 1 0 0 0 1 0	M_{26}	1 1 0 0 0 0 0 0 1 1 0	M_{43}	0 0 0 0 0 0 0 0 0 2 0
M_{10}	0 0 0 1 0 0 1 0 1 0 0	M_{27}	0 0 0 0 1 1 0 1 1 0 0	M_{44}	0 0 0 0 1 0 0 0 1 0 0
M_{11}	1 0 0 0 1 1 0 0 2 0 0	M_{28}	0 0 0 1 0 0 0 0 1 1 0	M_{45}	0 0 0 0 0 1 0 2 1 0 0
M_{12}	1 0 0 0 0 1 0 0 0 1 1	M_{29}	1 0 0 0 0 1 0 0 1 0 0	M_{46}	1 0 0 0 0 0 0 0 2 0 0
M_{13}	1 0 0 0 0 0 1 0 1 0 1	M_{30}	1 0 0 0 0 1 0 1 2 0 0	M_{47}	0 0 0 0 0 0 0 0 0 1 0
M_{14}	1 1 0 0 0 1 0 0 0 0 0	M_{31}	1 0 0 0 0 0 0 0 1 1 1	M_{48}	0 0 0 0 0 0 0 1 1 1 0
M_{15}	0 0 0 0 1 1 0 0 0 1 0	M_{32}	1 0 0 0 0 1 0 0 0 0 1	M_{49}	0 0 0 0 0 0 0 0 0 0 0
M_{16}	1 1 0 0 0 1 0 1 1 0 0	M_{33}	0 0 0 0 0 0 1 0 0 0 0	M_{50}	0 0 0 0 0 0 0 1 1 0 0

(a) Reachability graph



	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
M_0	1	1	0	1	0	1	0	0	1	0	0
t_5											
	1	1	0	0	1	1	0	0	1	0	0
t_2											
	1	1	0	0	0	0	1	0	1	0	0
t_3											
	1	1	0	0	0	1	0	1	1	0	0
t_0											
	0	0	0	0	1	1	0	1	1	0	0
t_2											
	0	0	0	0	0	0	1	1	1	0	0
t_3											
	0	0	0	0	0	1	0	2	1	0	0

(b) Firing sequence σ



(c) Slice of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma, \{p_5, p_7, p_8\}\rangle$

Fig. 3. Example of an application of Algorithm 2

- Otherwise, there exists some $p \in Q$ with $M_{n-1}(p) < M_n(p)$ and, therefore, $\mathsf{slice}(M_n, \sigma, Q) = \{t_n\} \cup \mathsf{slice}(M_{n-1}, \sigma, Q \cup {}^\bullet t_n)$. Let $\mathcal{N}' = (P', T', F')$, $F' = F|_{(P',T')}$, and $M'_0 = M_0|_{P'}$. Now, we prove that $\mathcal{N}'$ is a slice of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma, Q\rangle$:
  · Trivially, $\mathcal{N}'$ is a subnet of $\mathcal{N}$, M'_0 is a restriction of M_0 and $Q \subseteq P'$.
  · Let $\mathcal{N}''$ be the slice of $\mathcal{N}$ w.r.t. $\langle M_0, \sigma_{n-1}, Q \cup {}^\bullet t_n\rangle$, with $\sigma_{n-1} = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1}$ and $\mathcal{N}'' = (P'', T'', F'')$.

    By the inductive hypothesis, there exists a firing sequence σ'' for $(\mathcal{N}'', M''_0)$, with $M''_0 = M_0|_{P''}$, such that

    σ'' is a subsequence of σ_{n-1} w.r.t. T'',
    $M''_0 \xrightarrow{\sigma''} M''_k$, $k \leq n-1$, and
    M''_k covers M_{n-1} (i.e., $M''_k \geq M_{n-1}$).

    Now, we consider a firing sequence σ' for $(\mathcal{N}', M'_0)$ that mimicks σ'' (which is safe since $P'' = P'$ and $T'' \subseteq T'$) and then adds one more firing depending on whether $t_n \in T'$ or not. If $\sigma' = \sigma''$ then the claim follows by induction. Otherwise, it follows trivially by the inductive hypothesis and the fact that M''_k

covers M_n.

□

5 Conclusions and Future Work

In this work, we have introduced two different techniques for dynamic slicing of Petri nets. To the best of our knowledge, this is the first approach to dynamic slicing for Petri nets. The first approach takes into account the Petri net and an initial marking, 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. In general, our slices are smaller than previous (static) approaches where no initial marking nor firing sequence were considered.

As a future work, we plan to carry on an experimental evaluation of our slicing techniques in order to test its viability in practice. We also find it useful to extend our slicing technique to other kind of Petri nets (e.g., coloured Petri nets [10] and marked-controlled reconfigurable nets [12]).

References

[1] A. Bell and B.R. Haverkort. Sequential and distributed model checking of Petri nets. *Int. Journal on Software Tools for Technology Transfer*, 7(1):43–60, 2005.

[2] I. Brückner. Slicing CSP-OZ Specifications. In P. Pettersson and W. Yi, editors, *Proc. of the 16th Nordic Workshop on Programming Theory*, number 2004-041 in Technical Reports of the Department of Information Technology, pages 71–73. Uppsala University, Sweden, 2004.

[3] I. Brückner and H. Wehrheim. Slicing Object-Z Specifications for Verification. In *Proc. of the 4th Int'l Conf. of B and Z Users (ZB 2005)*, pages 414–433. Springer LNCS 3455, 2005.

[4] C.K. Chang and H. Wang. A Slicing Algorithm of Concurrency Modeling Based on Petri Nets. In *Proc. of the Int'l Conf. on Parallel Processing (ICPP'86)*, pages 789–792. IEEE Computer Society Press, 1986.

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

[6] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.

[7] Petri Nets Tool Database. Available at http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html.

[8] J. Desel and J. Esparza. *Free choice Petri nets*. Cambridge University Press, New York, NY, USA, 1995.

[9] M.P.E. Heimdahl and M.W. Whalen. Reduction and Slicing of Hierarchical State Machines. In M. Jazayeri and H. Schauer, editors, *Proc. of the 6th European Software Engineering Conference (ESEC/FSE'97)*, pages 450–467. Springer LNCS 1301, 1997.

[10] K. Jensen. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1: Basic Concepts, 1992. Volume 2: Analysis Methods, 1994. Volume 3: Practical Use, 1997. Monographs in Theoretical Computer Science, Springer-Verlag.

[11] W.J. Lee, S.D. Cha, Y.R. Kwon, and H.N. Kim. A Slicing-based Approach to Enhance Petri Net Reachability Analysis. *Journal of Research and Practice in Information Technology*, 32(2):131–143, 2000.

[12] M. Llorens and J. Oliver. Introducing Structural Dynamic Changes in Petri Nets: Marked-Controlled Reconfigurable Nets. In Farn Wang, editor, *Proc. of the 2nd Int'l Conf. on Automated Technology for Verification and Analysis (ATVA'04)*, pages 310–323. Springer LNCS 3299, 2004.

[13] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

[14] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[15] A. Rakow. Slicing Petri Nets. Technical report, Department für Informatik, Carl von Ossietzky Universität, Oldenburg, 2007.

[16] A. Rakow. Slicing Petri Nets with an Application to Workflow Verification. In *Proc. of the 34th Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, pages 436–447. Springer LNCS 4910, 2008.

[17] M. Rauhamaa. *A Comparative Study of Methods for Efficient Reachability Analysis.* Licentiate's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Digital Systems Laboratory, 1990.

[18] A.M. Sloane and J. Holdsworth. Beyond traditional program slicing. In *Proc. of the Int'l Symp. on Software Testing and Analysis*, pages 180–186, San Diego, CA, 1996. ACM Press.

[19] F. Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3:121–189, 1995.

[20] M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.