An SMT-based Approach to Fair Termination Analysis

Javier Esparza
Institut für Informatik
Technische Universität München
Garching bei München, Germany
Email: esparza@in.tum.de

Philipp J. Meyer
Institut für Informatik
Technische Universität München
Garching bei München, Germany
Email: meyerphi@in.tum.de

***Abstract*—Algorithms for the coverability problem have been successfully applied to safety checking for concurrent programs. In a former paper (An SMT-based Approach to Coverability Analysis, CAV14) we have revisited a constraint approach to coverability based on classical Petri net analysis techniques and implemented it on top of state-of-the-art SMT solvers. In this paper we extend the approach to fair termination; many other liveness properties can be reduced to fair termination using the automata-theoretic approach to verification. We use T-invariants to identify potential infinite computations of the system, and design a novel technique to discard false positives, that is, potential computations that are not actually executable. We validate our technique on a large number of case studies.**

I. Introduction

In recent years, verification problems for concurrent shared-memory or asynchronous message-passing software have been attacked by means of Petri net techniques. In particular, it has been shown that safety properties or fair termination can be solved by constructing and analyzing the coverability graph of a Petri net, or some related object [1]–[5]. This renewed interest on the coverability problem has led to numerous algorithmic advances for the construction of coverability graphs [4], [6]–[9].

Despite this success, the coverability problem remains computationally expensive [10], since it involves exhaustive state-space exploration. This motivates the study of cheaper incomplete procedures: algorithms much faster than the construction of the coverability graph, which may prove the property true, but also answer “don’t know”. In a recent paper, the authors, together with other colleagues, have revisited and further developed tests based on the marking equation and traps, two classical Petri net analysis techniques [11]. These techniques allow one to efficiently compute program invariants expressed as constraints of linear arithmetic [12]–[14]. Ifthe states violating the property also corresponds to those satisfying a linear constraint, unsatisfiability of the complete constraint system proves the property true. In the test suite analyzed in [11], 83% of the positive problem instances (that is, the instances for which the property holds) could be proved in this way. Moreover, due to advances in SMT-solving, the constraint systems could be solved at a fraction of the cost of state-exploration techniques. So the technique makes sense as a preprocessing that allows to prove many of easy cases at low cost; if the technique fails, then we can always resort to complete state-exploration methods.

In this paper we extend the approach to liveness properties. As in [11], which revisited and expanded previous work, we revisit an idea initially presented in [15], based on the use of transition invariants. Since liveness is typically harder than safety, and the constraint technology of 1997 was very primitive compared to state-of-the-art SMT-solvers, the work of [15] only explored a rather straightforward test, and only considered one case study. In this paper we improve the test of [15], design different implementations, compare their performance, and validate them on numerous case studies coming from different areas: distributed algorithms, workflow processes, Erlang programs, and asynchronous programs.

We conclude this introduction with a brief outline of our technique. Given an infinite execution σ of a Petri net model, let $\mathit{inf}(\sigma)$ be the set of transitions that occur infinitely often in σ. We consider liveness properties such that whether σ satisfies the property or not depends only on $\mathit{inf}(\sigma)$. (This is not an important restriction because, by taking the product of the Petri net model with a suitable Büchi automaton, every LTL property can be reduced to a property of this kind.) We say that a set T of transitions is *feasible* if $T = \mathit{inf}(\sigma)$ for some σ. We use T-invariants (more precisely, T-surinvariants) to extract boolean constraints that must be satisfied by every feasible set of transitions. However, these constraints are typically quite weak, and have spurious solutions, that is, they are satisfied by unfeasible sets of transitions. So we design a refinement loop that, given a solution, tries to construct an additional constraint that excludes it. If the refinement procedure terminates, then the model satisfies the property.

The paper is structured as follows. Section II contains basic definitions. Section III introduces the main technique. In Section IV and V, we describe two methods to refine the main technique. Section VI contains the experimental evaluation. Finally, Section VII presents conclusions.

II. Preliminaries

A *net* is a triple (P, T, F), where P is a set of *places*, T is a (disjoint) set of *transitions*, and $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ is the *flow function*. For $x \in P \cup T$, the *pre-set* is ${}^\bullet x = \{y \in P \cup T \mid F(y, x) = 1\}$ and the *post-set* is $x^\bullet = \{y \in P \cup T \mid$

```
      procedure PROCESS 1
      begin
        $bit_1$ := $false$
        while $true$ do
$p_1$:      $bit_1$ := $true$
$p_2$:      while $bit_2$ do skip od
$p_3$:      (∗ critical section ∗)
            $bit_1$ := $false$
        od
      end
```

```
      procedure PROCESS 2
      begin
        $bit_2$ := $false$
        while $true$ do
$q_1$:      $bit_2$ := $true$
$q_2$:      if $bit_1$ then
$q_3$:        $bit_2$ := $false$
$q_4$:        while $bit_1$ do skip od
              goto $q_1$
            fi
$q_5$:      (∗ critical section ∗)
            $bit_2$ := $false$
        od
      end
```

Fig. 1. Lamport's 1-bit algorithm for mutual exclusion [16].

$F(x, y) = 1\}$. We extend the pre- and post-set to a subset of $P \cup T$ as the union of the pre- and post-sets of its elements. A *subnet* of a Petri net (P, T, F) is a triple (P', T', F') such that $P' \subseteq P$, $T' \subseteq T$, and $F' : (P' \times T') \cup (T' \times P') \to \{0, 1\}$ with $F'(x, y) = F(x, y)$. Since F' is completely determined by F,P', and T', we often speak of the subnet (P', T').

A *marking* of a net (P, T, F) is a function $m : P \to \mathbb{N}$. Assuming an enumeration $p_1, \ldots, p_n$ of P, we often identify m and the vector $(m(p_1), \ldots, m(p_n))$. For a subset $P' \subseteq P$ of places, we write $m(P') = \sum_{p \in P'} m(p)$. A *Petri net* is tuple $N = (P, T, F, m_0)$, where (P, T, F) is a net and m_0 is a marking called the *initial marking*. Petri nets are represented graphically as follows: places and transitions are represented as circles and boxes, respectively. For $x, y \in P \cup T$, there is an arc leading from x to y iff $F(x, y) = 1$. The initial marking is represented by putting $m_0(p)$ black tokens in each place p.

A transition $t \in T$ is *enabled at* m iff $m(p) \geq 1$ for every $p \in {}^\bullet t$. A transition t enabled at m may *fire*, yielding a new marking m' (denoted $m \xrightarrow{t} m'$), where $m'(p) = m(p) + F(t, p) - F(p, t)$.

A sequence of transitions, $\sigma = t_1 t_2 \ldots t_r$ is an *occurrence sequence* of N iff there exist markings $m_1, \ldots, m_r$ such that $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_r} m_r$. The marking m_r is said to be *reachable* from m_0 by the occurrence of σ (denoted $m_0 \xrightarrow{\sigma} m_r$).

An infinite sequence of transitions, $\sigma = t_1 t_2 \ldots$ is an *infinite occurrence sequence* of N iff every finite prefix of σ is an occurrence sequence of N. This is denoted by $m_0 \xrightarrow{\sigma}$. The set $inf(\sigma)$ contains the transitions occuring infinitely often in σ.

A. Liveness properties

We consider a restricted notion of liveness property. Section II-C briefly sketches how to handle general LTL properties.

A *liveness property* φ of a net $N = (P, T, F, m_0)$ is a boolean constraint over the free variables T. The property φ holds for an infinite occurrence sequence σ (denoted by $\sigma \models \varphi$) iff $I_\sigma \models \varphi$, where $I_\sigma(t) = 1$ if $t \in inf(\sigma)$ else 0.

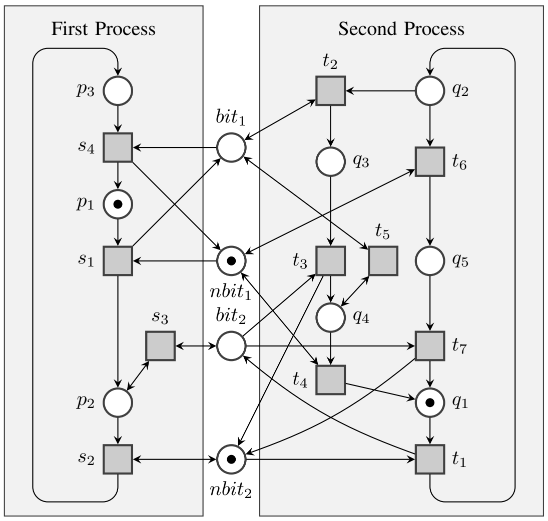


Fig. 2. Petri net for Lamport's 1-bit algorithm.

A Petri net N satisfies a property φ (denoted by $N \models \varphi$) iff $\sigma \models \varphi$ for every infinite occurrence sequence $m_0 \xrightarrow{\sigma}$. Note that a liveness property is always satisfied if the Petri net has no infinite occurrence sequences. Therefore the property $\varphi = false$ is equivalent to termination of the Petri net. Fair termination properties can be expressed by means of more complex formulas φ.

B. Two examples

As a first example, consider Lamport's 1-bit algorithm for mutual exclusion [16], shown in Fig. 1. Fig. 2 shows a Petri net model for the code. The two grey blocks model the control flow of the two processes. For instance, the token in place p_1 models the current position of process 1 at program location p_1. The four places in the middle of the diagram model the current values of the variables. For instance, a token in place $nbit_1$ indicates that the variable bit_1 is currently set to *false*.

The main liveness property for the processes states that, assuming a fair scheduler that allows both processes to execute actions infinitely often, each process enters the critical section infinitely often. For the first process, this corresponds to the property that every infinite occurrence sequence in which at least one of $s_1, \ldots, s_4$ and one of $t_1, \ldots, t_7$ occur infinitely often, contains infinitely many occurrences of s_2. As a boolean formula, we get

$$\left(\bigvee_{i=1}^{4} s_i \right) \wedge \left(\bigvee_{j=1}^{7} t_j \right) \Rightarrow s_2$$

For the second process we obtain a similar property.

As a second example, consider the fairly terminating asynchronous program [17] given in Fig. 3. Here, the **post** command is a non-blocking operation for launching a process in parallel. Initially, the process INIT is executed, which sets x to *true* and launches H. Process H launches new instances of H and G until G sets x to *false*. Assuming a fair scheduler,

```
    procedure H           procedure G           procedure INIT
    begin                 begin                 begin
h:    if $x$ then       g:   $x$ := false          $x$ := true
        post H            end                     post H
        post G                                  end
      fi
    end
```

Fig. 3. Asynchronous program [17].

i.e., one that will execute each process eventually, the program should terminate. This fair termination is the liveness property we want to prove.

Transforming the program into a Petri net gives us the net in Fig. 4. The place s models the scheduler, ph and pg are pending instances of H and G, respectively, and h and g are program locations. The transitions t_1 and s_1 dispatch the processes, while the other transitions exit the processes depending on the value of x. Note that the net is unbounded, as repeatedly firing $s_1 s_2$ puts arbitrarily many tokens in pg.

If the scheduler is fair and continues dispatching instances of H and G infinitely often, the program should terminate, giving us the liveness property $s_1 \wedge t_1 \implies \mathit{false}$, equivalent to $\neg(s_1 \wedge t_1)$.

C. LTL properties

To check general LTL properties we can use the automata-theoretic approach. Given a property φ, we construct the product of the Petri net model of the system and a Büchi automaton for $\neg\varphi$. The product yields a new Petri net with a set of accepting places. The initial net violates the property iff the product net has an infinite sequence σ such that $\mathit{inf}(\sigma)$ contains at least one of the input transitions of the accepting places. A detailed construction can be found in [15].

III. T-SURINVARIANTS

We present a procedure, called LIVENESS, which checks a sufficient condition for a given Petri net to satisfy a liveness property. The condition is unsatisfiability of an appropriate linear arithmetic formula.

Definition 1 (Incidence matrix)**.** The *incidence matrix* C of a Petri net N is a $|P| \times |T|$ matrix given by

$$C(p,t) = F(t,p) - F(p,t)$$

Definition 2 (T-surinvariant)**.** A vector $X : T \to \mathbb{Z}$ is a *T-surinvariant* of a Petri net N iff $C \cdot X \geq 0$. If moreover $C \cdot X = 0$, then X is a *T-invariant*.

A T-surinvariant X is *semi-positive* iff $X \geq 0$ and $X \neq 0$. The *support* of a T-surinvariant X is given by $\|X\| = \{t \in T \mid X(t) > 0\}$.

Loosely speaking, X is a surinvariant if for every place p and for every occurrence sequence $m \xrightarrow{\sigma} m'$, if σ fires each transition t exactly $X(t)$ times, then $m(p) \leq m'(p)$, that is, the number of tokens in p can only increase. The following theorem, where we identify X with the multiset of transitions containing each $t \in T$ exactly $X(t)$ times, shows that the T-surinvariants of a Petri net provide information about its infinite runs.

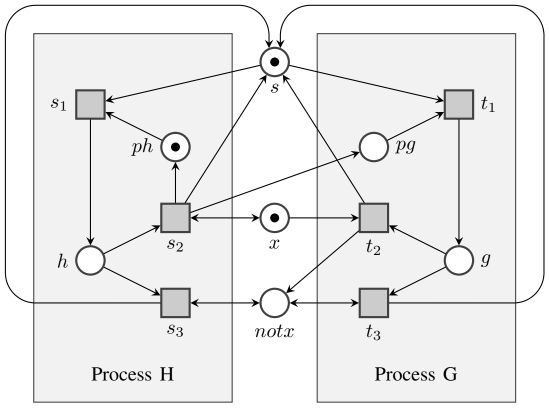


Fig. 4. Petri net for the asynchronous program.

Theorem 1. *[13], [14] Let σ be an infinite sequence of transitions and N a Petri net. If σ is an infinite occurrence sequence of N, then there is a semi-positive T-surinvariant X satisfying $\|X\| = \mathit{inf}(\sigma)$.*

Proof. Let σ' be a suffix of σ containing only transitions of $\inf(\sigma)$, and let $\sigma' = \sigma'_1\sigma'_2\sigma'_3 \ldots$ such that each σ'_i contains every transition of $\inf(\sigma)$ at least once. Since σ is an occurrence sequence of N, there exist markings $m_1, m_2, m_3, \ldots$ such that $m_1 \xrightarrow{\sigma'_1} m_2 \xrightarrow{\sigma'_2} m_3 \xrightarrow{\sigma'_3} \ldots$. By Dickson's lemma, there exist indices $i < j$ such that $m_i \leq m_j$. Let X be the Parikh vector of $\sigma'_i \ldots \sigma'_{j-1}$, i.e., the vector assigning to each transition its number of occurrences in the sequence. By the definition of the firing rule and the incidence matrix C, for every place p we have $m_j(p) - m_i(p) = \sum_{t \in T} C(p,t)X(t)$ or, in matrix form, $m_j - m_i = C \cdot X$. Since $m_j \geq m_i$, we have $m_j - m_i \geq 0$, and so X is a semi-positive T-surinvariant. Since $\sigma'_i \ldots \sigma'_{j-1}$ contains all transitions of $\inf(\sigma)$, we have $\|X\| = \inf(\sigma)$. ☐

However, a T-surinvariant does not guarantee the existence of a corresponding occurrence sequence. Consider the net in Fig. 4. The multiset $X = \{s_1, s_2, t_1, t_3\}$ is a semi-positive T-invariant, but, as we will see later, no infinite occurrence sequence σ satisfies $\mathit{inf}(\sigma) = \{s_1, s_2, t_1, t_3\}$. We say that a T-surinvariant X is *realizable* if there is an infinite occurence sequence σ with $\|X\| = \mathit{inf}(\sigma)$.

For a T-surinvariant X and a liveness property φ, we denote by $\varphi(X)$ the constraint of linear arithmetic obtained by substituting $X(t) > 0$ for every occurrence of t in φ. So, for instance, if $\varphi = t_1 \vee t_2$, then $\varphi(X) = X(t_1) > 0 \vee X(t_2) > 0$. By Theorem 1, if there is an infinite sequence σ such that $\sigma \models \varphi$, then there is also a semi-positive T-surinvariant X such that $\varphi(X)$ holds. Taking the contrapositive, we have: if no semi-positive T-surinvariant X satisfies $\neg\varphi(X)$, then no

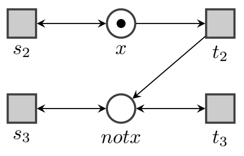


Fig. 5. Subnet of the net of Fig. 4.

sequence σ satisfies $\neg\varphi$, and so $N \models \varphi$. This directly leads to a semi-decision procedure for checking if a liveness property φ is a property of a Petri net N: If the following constraints are unsatisfiable, then $N \models \varphi$.

$$\mathcal{C}(N,\varphi) :: \begin{cases} CX \geq 0 & \text{T-surinvariant condition} \\ X \geq 0 & \text{non-negativity condition} \\ X \neq 0 & \text{non-zero condition} \\ \neg\varphi(X) & \text{property condition} \end{cases} \tag{1}$$

In practice, the procedure is very efficient, but often fails to prove the property. As an example, consider Lamport's algorithm. The negation of the fairness property for the first process yields

$$(s_1 \vee s_2 \vee s_3 \vee s_4) \wedge (t_1 \vee t_2 \vee t_3 \vee t_4 \vee t_5 \vee t_6 \vee t_7) \wedge \neg s_2$$

which corresponds to runs of the system where both processes are executed infinitely often, but where the first process never enters the critical section. However, $X = \{s_3, t_5\}$ is a solution to the constraints (1). The solution corresponds to the processes being stuck in the locations p_2 and q_4 while executing the **skip** commands. For this reason, in the next section we revisit an idea of [15] which leads to a more precise set of constraints.

IV. Refining T-surinvariants with P-components

The method Liveness can be strengthened by discarding T-surinvariants which are not realizable.

Consider again the net of Fig. 4. Recall that $X = \{s_1, s_2, t_1, t_3\}$ is a semi-positive T-invariant. We prove that it is not realizable. Consider the subnet $N' = (P', T')$, where $P' = \{x, notx\}$ and $T' = \{s_2, t_2, s_3, t_3\}$, shown in Fig. 5.

Inspection of the subnet shows that firing a transition does not change the total number of tokens in P'. For example, firing t_2 takes a token from x, but adds a token to $notx$. So this number is always equal to 1, and so it makes sense to speak of "the" token of N'. Assume now that X is realized by some infinite sequence σ, i.e., $inf(\sigma) = \|X\|$. Since both s_2 and t_3 occur infinitely often in σ, there are sequences $\sigma_1, \sigma_2, \sigma_3$ such that $\sigma = \sigma_1\, s_2\, \sigma_2\, t_3 \sigma_3$, and $\sigma_2 \in \|X\|^*$. After the occurrence of s_2 the token of N' is on x, and before the occurrence of t_3 it is on $notx$. But σ_2 cannot "move" the token from x to $notx$, as it does not contain any occurrence of t_2 (because $t_2 \notin \|X\|$). So we reach a contradiction, and σ does not exist.

In the rest of the section we show how to automatically search for proofs of non-realizability like this. We need the notion of a P-component of a net.

Definition 3 (P-component)**.** A *P-component* of a net $N = (P, T, F)$ is a subnet $N' = (P', T')$ such that $P' \neq \emptyset$ and $|t^\bullet \cap P'| = |{}^\bullet t \cap P'| = 1$ for all $t \in T'$ and $T' = P'^\bullet \cup {}^\bullet P'$ (where pre- and postsets are taken with respect to N).

The subnet of Fig. 5 is a P-component. Note that the number of tokens in a P-component never changes, i.e., $m_0(P) = m(P)$ for all $m_0 \xrightarrow{\sigma} m$. Therefore, if initially a P-component only contains one token, then we know that the token will stay in the P-component.

Lemma 2. *Let X be a T-surinvariant of a Petri net N. If N has a P-component (P', T') such that $m_0(P') = 1$, and the subnet $(P', T' \cap \|X\|)$ is not strongly connected, then X is not realizable.*

Proof. (Sketch.) If $(P', T' \cap \|X\|)$ is not strongly connected, then by the definition of P-component there are two transitions $t_1, t_2 \in T' \cap \|X\|$ such that no path of $(P', T' \cap \|X\|)$ leads from t_1 to t_2. So the token of (P', T') cannot be transported from the output place of t_1 in $(P', T' \cap \|X\|)$ to the input place of t_2 in $(P', T' \cap \|X\|)$ by firing transitions of X only. Since every sequence realizing X must fire both t_1 and t_2 infinitely often, no such sequence exists. □

Lemma 2 provides a refinement condition. To find such a refinement, we encode the condition as a conjunction of linear arithmetic constraints. A pair (P', T') is the set of places and transitions of a P-component such that $m_0(P') = 1$ iff it satisfies these constraints:

$$\begin{array}{ll} \forall t \in T' : |t^\bullet \cap P'| = 1 & \qquad P'^\bullet \cup {}^\bullet P' = T' \\ \forall t \in T' : |{}^\bullet t \cap P'| = 1 & \qquad m_0(P') = 1 \end{array}$$

For the strong connectedness condition, we use that a graph (V, E) is not strongly connected iff there is a partition $V = V_1 \uplus V_2$ such that no edge $(v, v') \in E$ satisfies $v \in V_1, v' \in V_2$. In our case, V is the set $T' \cap \|X\|$, and E is the set of pairs (t_1, t_2) such that some place $p \in P'$ satisfies $(t_1, p), (p, t_2) \in F'$. So $(P', T' \cap \|X\|)$ is not strongly connected iff the following constraints are satisfiable:

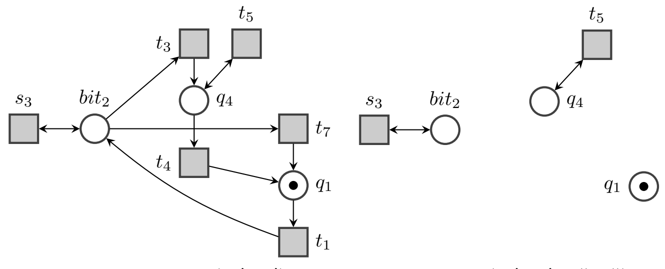
$$\begin{array}{rl} T' \cap \|X\| = T_1 \uplus T_2 & \qquad T_1 \neq \emptyset \\ (T_1^\bullet \cap P')^\bullet \cap \|X\| \subseteq T_1 & \qquad T_2 \neq \emptyset \end{array}$$

These constraints can be encoded by introducing an array of variables with range $\{0, 1\}$ for each set of places or transitions. For example, the constraint $\forall t \in T' : |t^\bullet \cap P'| = 1$ translates to the linear arithmetic constraint

$$\bigwedge_{t \in T} \left[T'(t) = 1 \implies \sum_{p \in t^\bullet} P'(p) = 1 \right]$$

where $(T'(t1), \ldots, T'(t_n))$ is the array of boolean variables for the set T'.

If the constraints above are satisfiable for a given T-surinvariant X, then X is not realizable. We can exclude X (and any other T-surinvariant whose support has the same intersection with the P-component as $\|X\|$) by adding the constraint:



(a) P-component (P', T') (b) Subnet $(P', T' \cap \|X\|)$

Fig. 6. P-component and subnet of the Petri net for Lamport's algorithm.

$$\delta ::= \left[\bigvee_{t \in T_1} t\right] \wedge \left[\bigvee_{t \in T_2} t\right] \implies \bigvee_{t \in T' \setminus \|X\|} t \tag{2}$$

to the set of constraints (1). We can iterate the process, until either the constraints are unsatisfiable, which means successfully proving the property, or no further P-components can be found to discard a T-surinvariant, which means failure.

For example, for Lamport's algorithm and the fairness property for the first process, the constraints (1) have the solution $X = \{s_3, t_5\}$. However, since $(P', T') = (\{bit_2, q_1, q_4\}, \{s_3, t_1, t_3, t_4, t_5, t_7\})$ is a P-component and $T_1 = \{s_3\}$, $T_2 = \{t_5\}$ satisfy the constraints above, we get that X is not realizable. The P-component (P', T') and the subnet $(P', T' \cap \|X\|)$ are shown in Fig. 6. We can immediately see that the token cannot be transported from the output place of s_3 to the input place of t_5.

We add the refinement constraint

$$s_3 \wedge t_5 \implies t_1 \vee t_3 \vee t_4 \vee t_7$$

to the set (1) and check again for satisfiability. The new set is still satisfiable with solution $X = \{s_3, t_1, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. In a second refinement step we find a P-component with $\{nbit_1, p_2, p_3\}$ as set of places, and add the refinement constraint

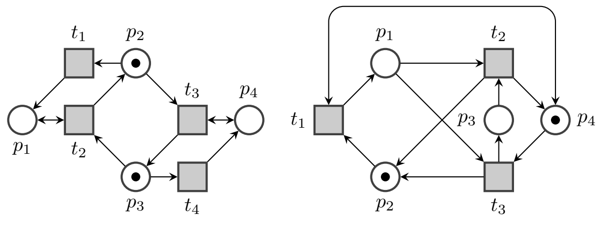
$$s_3 \wedge (t_4 \vee t_6) \implies s_1 \vee s_2 \vee s_4,$$

after which the constraints (1) are unsatisfiable, and we conclude that the fairness property for the first process holds.

For the second example (Fig. 3 and 4), we considered the fair termination property $\varphi = \neg(s_1 \wedge t_1)$. After adding $\neg\varphi = s_1 \wedge t_1$ to the constraints (1), we obtain a solution $X = \{s_1, s_2, t_1, t_3\}$. With the P-component $(P', T') = (\{x, notx\}, \{s_2, s_3, t_2, t_3\})$ and the partition $T_1 = \{s_2\}$ and $T_2 = \{t_3\}$, we can discard this T-invariant as unrealizable and obtain the refinement constraint

$$s_2 \wedge t_3 \implies s_3 \vee t_2,$$

after which the constraints (1) are unsatisfiable and we can prove fair termination.



(a) Net without P-components (b) Net without unmarked traps

Fig. 7. Terminating Petri nets for which refinement is insufficient.

V. Refining T-surinvariants with Traps

For some Petri nets, refinement with P-components is not sufficient for discarding unrealizable T-surinvariants. For example, we cannot prove the properties for the leader election algorithm by Dolev, Klawe and Rodeh [18] or the mutual exclusion algorithm by Szymanski [19]. These nets are too big to give as an example, but consider instead the net in Fig. 7a, which is similar to a subnet of the net for the leader election algorithm. The net has no infinite occurence sequences and we would like to prove termination. The multiset $X = \{t_2, t_3\}$ is a T-surinvariant, but the net has no P-components, so we cannot refine the constraints. To solve this problem we develop a refinement technique based on traps.

Definition 4 (Trap). A *trap* is a set of places $S \subseteq P$ such that $S^\bullet \subseteq {}^\bullet S$.

It follows immediately from the definition that marked traps stay marked: if a trap S is marked at some marking m, i.e. $m(S) > 0$, then it is also marked at all markings m' reachable from m, because every transition taking tokens from S also adds at least one token to S.

Given a T-surinvariant X, we consider the subnet $(P', T') = (\|X\|^\bullet, \|X\|)$. In the example of Fig. 7a, (P', T') is obtained by removing transitions t_1 and t_4, together with their input and output arcs. Assume $\|X\|$ is realized by an infinite ocurrence sequence σ. Then there are sequences σ', σ'' such that $\sigma = \sigma'\sigma''$ and $\sigma'' \in \|X\|^\omega$. Since every place P' has an input transition in $\|X\|$, every place of P' gets marked during the execution of σ'', and therefore every trap of (P', T') becomes eventually marked. So we have the following lemma:

Lemma 3. *Let $N = (P, T, F, m_0)$ be a net and let $\|X\|$ be a realizable T-surinvariant. Then some marking m reachable from m_0 in N marks every trap of the subnet $(P', T') = (\|X\|^\bullet, \|X\|)$.*

By this lemma, if we show that no reachable marking marks every trap of (P', T'), then X is unrealizable. We use an iterative approach. Given a set of traps $\mathcal{Q}$, using the techique of [11] we can construct a set of constraints satisfied by every reachable marking that marks every trap of $\mathcal{Q}$.[1] If the constraints are satisfiable, then we extract from the solution a

[1]The constraints express that a solution m satisfies the marking equation and that $m(S) > 0$ for every trap $S \in \mathcal{Q}$.

marking m that marks all traps in $\mathcal{Q}$. Since m may not mark all traps, we search for a new trap $S \notin \mathcal{Q}$ not marked at m. If we find such S, we set $\mathcal{Q} = \mathcal{Q} \cup \{S\}$ and iterate, otherwise we give up. If the constraints are unsatisfiable, then no reachable marking marks all traps in $\mathcal{Q}$, which implies that X is not realizable. We can then add a new constraint excluding any solution with the same support as X. However, we can do better, and add a stronger constraint. Since we have shown that no infinite occurrence sequence σ can reach a marking that simultaneously marks all traps of $\mathcal{Q}$, we choose a constraint expressing that if $\mathit{inf}(\sigma)$ contains transitions marking all traps of (P', T'), then it must also contain at least one transition that empties a trap of (P', T'). (Of course, such a transition cannot belong to T', it must be a transition of $T \setminus T'$.)

$$\delta ::= \bigwedge_{S \in \mathcal{Q}} \left[\bigvee_{t \in {}^\bullet S} t \right] \implies \bigvee_{S \in \mathcal{Q}} \left[\bigvee_{t \in S^\bullet \setminus {}^\bullet S} t \right] \qquad (3)$$

For example, for the Petri net in Fig. 7a, the method LIVENESS returns $X = \{t_2, t_3\}$ as a T-surinvariant. The corresponding subnet is $(P', T') = (\{p_1, p_2, p_3, p_4\}, \{t_2, t_3\})$. Initially, for $\mathcal{Q} = \emptyset$, we can take the inital marking m_0. In m_0, the trap $S_1 = \{p_1\}$ of (P', T') is unmarked. We search for a marking m satisfying the marking equation and $m(p_1) \geq 1$, and obtain as solution $m_1 = (1, 0, 1, 0)$. At this marking the trap $S_2 = \{p_4\}$ is unmarked. So we search for a marking m satisfying the marking equation, $m(p_1) \geq 1$ and $m(p_4) \geq 1$, and obtain as solution $m_2 = (1, 0, 0, 1)$. At this marking the trap $S_3 = \{p_2, p_3\}$ is unmarked. We search for a marking m satisfying the marking equation, $m(p_1) \geq 1$, $m(p_4) \geq 1$, and $m(p_2) + m(p_3) \geq 1$, and obtain that the constraints are unsatisfiable. So we generate the refinement constraint

$$(t_1 \vee t_2) \wedge (t_3 \vee t_4) \wedge (t_2 \vee t_3) \implies t_1 \vee t_4,$$

which excludes $\{t_2, t_3\}$. In fact, the additional constraint turns out to exclude not only $\{t_2, t_3\}$, but all T-surinvariants, which proves termination of the Petri net.

The refinement with traps is a generalization of the refinement with P-components. For a T-surinvariant X, assume there is refinement with a P-component (P', T') and a partition $T_1 \uplus T_2 = T' \cap \|X\|$. In the subnet $(\|X\|^\bullet, \|X\|)$, $S_1 = P' \cap T_1^\bullet$ and $S_2 = P' \cap T_2^\bullet$ are two different traps, and as the P-component has only one token, we can show with the marking equation that S_1 and S_2 cannot be marked at the same time.

Generally, refinement with P-components requires fewer calls to the SMT solver, and is therefore more efficient. In our experiments it is also sufficient for most cases, and if it fails, refinement with traps can be applied afterwards. So we always start with a refinement with P-components, and apply then a refinement with traps if necessary.

Even with both refinements, the method is still incomplete. Consider the Petri net in Fig. 7b, which appears in a Petri net model of the drinking philosopher's problem [20]. The net is terminating, but $X = \{t_1, t_1, t_2, t_3\}$ is a surinvariant (observe that X is a genuine multiset with two copies of t_1). The subnet corresponding to X is the complete net, and every trap is initially marked, so no refinement can be found.

If the property does not hold, our method fails and returns a surinvariant that cannot be excluded by our refinements. For example, for Lamport's algorithm, the fairness property for the second process is not satisfied. The negation of the property is

$$(s_1 \vee s_2 \vee s_3 \vee s_4) \wedge (t_1 \vee t_2 \vee t_3 \vee t_4 \vee t_5 \vee t_6 \vee t_7) \wedge \neg t_6.$$

After two refinement steps, our method returns the T-surinvariant $X = \{s_1, s_2, s_3, s_4, t_1, t_2, t_3, t_4, t_5\}$, which satisfies $\neg\varphi$ and cannot be further refined. In this case we can use guided state-space exploration [21] to try to identify permutations of X that are actual repeatable occurrence sequences. In this case, $\sigma = s_1 t_1 s_3 t_2 t_3 s_2 t_5 s_4 t_4$ is indeed a matching occurrence sequence which can be repeated infinitely and violates the property.

VI. Experimental Evaluation

We extended our tool *Petrinizer* [11], implemented on top of the SMT solver Z3 [22], with the method LIVENESS. The method can be used without refinement, with only P-component or trap refinement, or with P-component refinement followed by trap refinement. In addition, the refinement structures can be minimized.

For our evaluation, we had three goals. First, we wanted to measure the success rates on a large number of case studies. The second goal was to investigate the usefulness and necessity of P-components, traps and minimization of them. As a third goal, we wanted to measure the performance of the method and compare it with the model checker SPIN[2] [23].

A. Benchmarks

For the evaluation, we used five different benchmark suites from various sources. The first two suites are workflow nets coming from business processes [24]. One is a collection of SAP reference models [25] and the other consists of IBM business process models [26]. We examined the nets for termination. In total, these suites contains 1976 models, out of which 1836 are terminating.

The third suite contains 50 examples that come from the analysis of Erlang programs [5], found on the website of the Soter tool[3]. Out of these, 33 are terminating.

For the fourth suite, we used classic asynchronous programs that can be scaled in the number of processes. These include a leader election algorithm [18], a snapshot algorithm [27] and three mutual exclusion algorithms [16], [19], [28]. Each of the 5 algorithms is scaled from $n = 2$ to 6 processes, resulting in 25 examples. For the former two distributed algorithms, the property is repeated liveness, i.e., infinitely often electing a leader or taking a snapshot infinitely often, while for the latter three mutual exclusion algorithms it is non-starvation for the first process. These properties all contain a fairness assumption for the scheduler, and they hold for all examples.

[2]http://spinroot.com/

[3]http://mjolnir.cs.ox.ac.uk/soter/

TABLE I
FAIRLY TERMINATING EXAMPLES WITH RATE OF SUCCESS BY DIFFERENT REFINEMENT METHODS.

Benchmark	No ref.	Ref. w/P-co.	Ref. w/traps	Terminating
SAP	1263	1263	1264	1264
IBM	571	571	572	572
Erlang	27	27	27	33
Asynchronous	0	14	20	25
Literature	0	3	5	5
Total	1861	1878	1888	1899

Finally, as the fifth suite, we collected 5 examples from the literature on termination and liveness analysis and modeled them as Petri nets. These are the programs from Fig. 2 in [29], Fig. 3 in [30], Fig. 1(b) in [17] and two variants of the Windows NT Bluetooth driver from [31]. These are all terminating programs.

The Petri nets for these benchmarks vary largely in size. The number of places ranges from 4 to 66950, with a mean of 116 and a median of 38. The number of transitions ranges from 3 and 3 to 213626, with a mean of 163 and a median of 30.

We try to prove the fairness property of the asynchronous programs and termination for the examples from the other benchmark suites. In total, we have 1899 examples where the property holds.

B. Rate of success on terminating examples

In Table I, the rate of success with different refinement methods is shown. Even without refinement, we can prove termination of all but 2 of the SAP and IBM examples, and of 27 of the 33 Erlang examples. However, without refinement we can prove none of the 30 examples from the other two suites. Refinement with P-components and traps allows us to prove the 2 remaining SAP and IBM examples, 6 more Erlang examples, and 25 of the 30 examples from the other two suites. of them. Additional refinement with traps allows us to prove th remaining In total, we can prove termination for 1888 of the 1899 terminating examples, and at least 80% of the terminating examples of each suite.

C. Usefulness of refinement methods and minimization

Table II presents results on the asynchronous benchmark suite and refinement with and without minimization. Minimization of the refinement components can result in better refinement constraints that exclude more T-surinvariants, at the price of a time overhead, since repeated calls to the SMT solver are needed until a minimal component is found. The default method, R_1, is refinement with P-components and traps without any minimization. Refinement method R_2 minimizes P-components (P', T') by $|P'|$ and traps S by $|S|$. Other criteria were also tested, but there was no optimal one working for all benchmarks. For each method, the number of P-components $|\mathcal{R}|$, number of trap refinements $|\mathcal{Q}|$ and total execution time in seconds for proving the property are given.

TABLE II
COMPARISON OF REFINEMENT WITH AND WITHOUT MINIMIZATION AND RUNTIME COMPARISON WITH SPIN. FOR AN EXECUTION, TO DENOTES EXCEEDING THE TIME LIMIT AND MO EXCEEDING THE MEMORY LIMIT.

		Refinement R_1			Ref. w/ min. R_2			SPIN
Benchmark	n	$\|\mathcal{R}\|$	$\|\mathcal{Q}\|$	T (s)	$\|\mathcal{R}\|$	$\|\mathcal{Q}\|$	T (s)	T (s)
Leader election by Dolev, Klawe and Rodeh [18]	2	0	4	2.53	0	4	2.30	0.69
	3	0	6	8.45	0	6	9.03	0.74
	4	0	8	35.5	0	8	38.4	15.7
	5	0	13	206	0	10	154	MO
	6	0	17	1104	0	12	728	MO
Snapshot algorithm by Bougé [27]	2	2	0	0.35	2	0	0.30	0.31
	3	3	0	0.50	3	0	0.81	0.72
	4	4	0	0.60	4	0	0.91	10.3
	5	5	0	0.73	5	0	1.41	218
	6	6	0	1.82	6	0	1.63	MO
Lamport's 1-bit algorithm for mutual exclusion [16]	2	2	0	0.50	3	0	0.43	0.69
	3	6	0	1.26	6	0	1.63	0.69
	4	12	0	2.83	13	0	5.50	0.92
	5	27	0	9.34	18	0	11.3	10.4
	6	26	0	13.4	23	0	20.6	MO
Peterson's mutual exclusion algorithm [28]	2	1	0	0.37	1	0	0.41	0.69
	3	13	0	6.57	7	0	8.55	0.71
	4	21	0	65.9	18	0	92.5	1.16
	5	285	0	2289	36	0	911	43.5
	6	-	-	TO	-	-	TO	MO
Szymanski's mutual exclusion algorithm [19]	2	21	6	10.9	26	6	17.6	0.70
	3	Property cannot be proven with refinement for $n \geq 3$.						0.80
	4							5.83
	5							347
	6							MO

We observe cases where we need refinement only with P-components (Snapshot), only with traps (Leader election) or with both (Szymanski with $n = 2$). For Szymanksi and $n \geq 3$ we cannot prove the property even with both refinement methods.

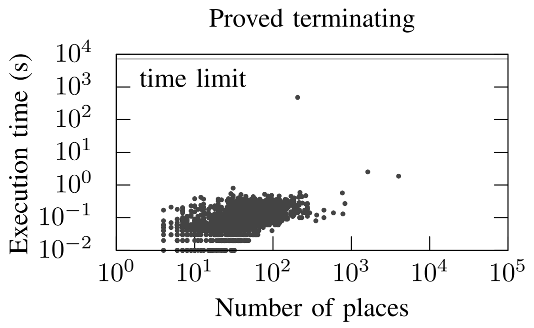
Minimization with method R_2 saves many refinement steps for Peterson and a few for Lamport and Leader election, while for Szymanski the number of steps increases. The time overhead when no steps are saved is not very large (up to 2×).

Our method produces a certificate for fair termination consisting of the P-components $\mathcal{R}$ and traps $\mathcal{Q}$, and of the T-surinvariants they exclude. One can use independent methods to check that $\mathcal{R}$ and $\mathcal{Q}$ are indeed P-components and traps, and that the constraints (1) are unsatisfiable. The size of each P-component and trap is limited by the size of the net. The size of the whole certificate depends on the number of refinement steps, however it is usually much more compact than the whole state space.

D. Performance

All experiments were performed on the same machine, equipped with a Intel Core i7-4810MQ CPU at 2.8 GHz and 16 GB of memory, running Linux 3.18.6 in 64-bit mode. Execution time was limited to 2 hours and memory to 8 GB.

Table II shows the execution times of Petrinizer for the asynchronous benchmark suite with and without refinement and a comparison with SPIN. SPIN was used with a fairness



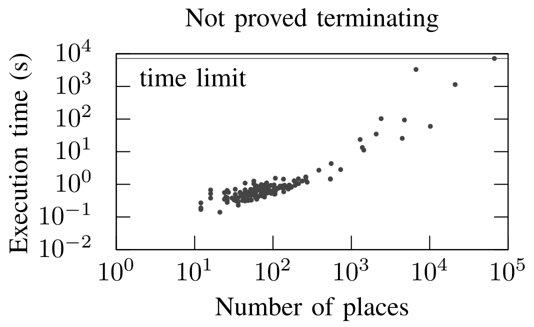


Fig. 8. Execution time in dependence on the number of places for the examples from the benchmark suites SAP, IBM, Erlang and Literature, depending on whether Petrinizer succeeds in proving termination.

strategy enforced and partial order reduction. Only for the snapshot algorithm, partial order reduction was turned off, as it is not supported together with fairness and the rendezvous operations used in the algorithm. For small examples, SPIN is usually faster. However, as n grows to 5 or 6, SPIN quickly reaches the memory limit. Here, Petrinizer outperforms SPIN significantly on the examples Leader election, Snapshot and Lamport.

For the other four benchmark suites, Fig. 8 shows the performance of Petrinizer. For the positive examples (i.e., those where we can prove the property), we can prove all but one of the 1868 examples in under 3 seconds. The outlier is from the SAP suite, for which we need 320 refinement steps and 8 minutes. Even the largest positive example from the Erlang suite with 4014 places only needs 1.86 seconds. For the negative examples, Petrinizer performs worse, usually because it performs more refinement steps. However, it terminates in under 10 seconds in all but four cases, and within the time limit of 2 hours in all but one case (our largest example with 66950 places).

We only need more than 3 refinement steps in one case (an outlier with 320 steps). The number of steps is not correlated to the net size.

VII. Conclusion

Transition invariants and P-components are a classical analysis techniques for Petri nets. We have demonstrated that, combined with a state-of-the-art SMT solver, these techniques are very effective in proving fair termination for a large number of common benchmark examples. We have further developed a novel technique based on traps, which allows us to reach a high degree of completeness on these benchmarks. The constraint systems produced by our tool can be used as a certificate of fair termination.

Acknowledgments

We thank Filip Niksic, Corneliu Popeea, and Karsten Wolf for kindly providing examples for our experimental evaluation.

References

[1] A. Kaiser, D. Kroening, and T. Wahl, “Dynamic cutoff detection in parameterized concurrent programs,” in *CAV*, 2010, pp. 645–659.
[2] P. Ganty and R. Majumdar, “Algorithmic verification of asynchronous programs,” *ACM Trans. Program. Lang. Syst.*, vol. 34, no. 1, p. 6, 2012.
[3] A. Bouajjani and M. Emmi, “Bounded phase analysis of message-passing programs,” in *TACAS*, 2012, pp. 451–465.
[4] A. Kaiser, D. Kroening, and T. Wahl, “Efficient coverability analysis by proof minimization,” in *CONCUR*, 2012, pp. 500–515.
[5] E. D’Osualdo, J. Kochems, and C.-H. L. Ong, “Automatic verification of Erlang-style concurrency,” in *SAS*, 2013, pp. 454–476.
[6] G. Geeraerts, J.-F. Raskin, and L. V. Begin, “Expand, enlarge and check: New algorithms for the coverability problem of WSTS,” *J. Comput. Syst. Sci.*, vol. 72, no. 1, pp. 180–203, 2006.
[7] P. Ganty, J.-F. Raskin, and L. Van Begin, “From many places to few: Automatic abstraction refinement for Petri nets,” *Fundam. Inform.*, vol. 88, no. 3, pp. 275–305, 2008.
[8] A. Valmari and H. Hansen, “Old and new algorithms for minimal coverability sets,” in *Petri Nets*, 2012, pp. 208–227.
[9] J. Kloos, R. Majumdar, F. Niksic, and R. Piskac, “Incremental, inductive coverability,” in *CAV*, 2013, pp. 158–173.
[10] C. Rackoff, “The covering and boundedness problems for vector addition systems,” *Theor. Comput. Sci.*, vol. 6, pp. 223–231, 1978.
[11] J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic, “An SMT-based approach to coverability analysis,” in *CAV*, 2014, pp. 603–619.
[12] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[13] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge University Press, 1995.
[14] W. Reisig, *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.
[15] J. Esparza and S. Melzer, “Model checking LTL using constraint programming,” in *ICATPN*, 1997, pp. 1–20.
[16] L. Lamport, “The mutual exclusion problem - part II: Statement and solutions,” *J. ACM*, vol. 33, no. 2, pp. 327–348, 1986.
[17] P. Ganty, R. Majumdar, and A. Rybalchenko, “Verifying liveness for asynchronous programs,” in *POPL*, 2009, pp. 102–113.
[18] D. Dolev, M. M. Klawe, and M. Rodeh, “An o(n log n) unidirectional distributed algorithm for extrema finding in a circle,” *J. Algorithms*, vol. 3, no. 3, pp. 245–260, 1982.
[19] B. K. Szymanski, “A simple solution to lamport’s concurrent programming problem with linear wait,” in *ICS*, 1988, pp. 621–626.
[20] K. M. Chandy and J. Misra, “The drinking philosopher’s problem,” *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 4, pp. 632–646, 1984.
[21] H. Wimmel and K. Wolf, “Applying CEGAR to the Petri net state equation,” *Logical Methods in Computer Science*, vol. 8, no. 3, 2012.
[22] L. M. de Moura and N. Bjørner, “Z3: An efficient smt solver,” in *TACAS*, 2008, pp. 337–340.
[23] G. J. Holzmann, “The model checker SPIN,” *IEEE Trans. Software Eng.*, vol. 23, no. 5, pp. 279–295, 1997.
[24] W. M. P. van der Aalst, “Challenges in business process management: Verification of business processing using petri nets.” *Bulletin of the EATCS*, vol. 80, pp. 174–199, 2003.
[25] B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek, and W. M. P. van der Aalst, “Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants,” *Computers in Industry*, vol. 58, no. 6, pp. 578–601, 2007.

[26] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf, “Instantaneous soundness checking of industrial business process models,” in *BPM*, 2009, pp. 278–293.
[27] L. Bougé, “Repeated snapshots in distributed systems with synchronous communications and their implementation in CSP,” *Theor. Comput. Sci.*, vol. 49, pp. 145–169, 1987.
[28] G. L. Peterson, “Myths about the mutual exclusion problem,” *Inf. Process. Lett.*, vol. 12, no. 3, pp. 115–116, 1981.
[29] B. Cook, A. Podelski, and A. Rybalchenko, “Proving thread termination,” in *PLDI*, 2007, pp. 320–330.
[30] A. Podelski and A. Rybalchenko, “Transition invariants,” in *LICS*, 2004, pp. 32–41.
[31] S. Qadeer and D. Wu, “KISS: keep it simple and sequential,” in *PLDI*, 2004, pp. 14–24.