Automatic verification of regular protocols in P/T nets

Citation for published version (APA):

Document status and date:
Published: 01/01/1993

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 16. Aug. 2019
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
        prof.dr.K.M.van Hee.
Automatic Verification of Regular Protocols in P/T Nets

P.M.P. Rambags

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513, 5600 MB Eindhoven
The Netherlands
e-mail: paulr@win.tue.nl

Abstract

In this article, we present two algorithms to determine whether a given place/transition (P/T) net $N$ with transitions $T$ satisfies a given protocol $\Pi$, where $\Pi$ consists of an alphabet $A \subseteq T$ and a regular language over $A$.

The first algorithm checks whether $N$ can not do more than $\Pi$ allows (trace-safety)$^1$. The second one verifies whether $N$ does not stop too early (weak progress).

Keywords: Protocols, automatic verification, P/T nets, regular languages, finite automata, decidability, coverability, reachability, liveness.

1 Introduction

Automatic protocol verification differs from other kinds of protocol verification in the sense that the entire verification is performed by a computer, without human interaction. The process of protocol verification can be very complex and only this kind of verification yields a reliable result, because human mistakes are impossible [5,15].

Up to now, some algorithms have been described for the automatic verification of protocols in Petri nets. However, these algorithms are often very specific, for most of them have been designed for just one protocol and a limited class of Petri nets. Examples can be found in [10,15,18,29,31], furthermore, in [3,8,11], invariants are used for the automatic verification and in [6] a reachability analysis.

In this article, we give algorithms for the automatic verification of a broad class of protocols: All protocols that can be described as a regular language over the transitions of a place/transition (P/T) net. We denote such protocols with an alphabet and a regular expression. Regular expressions consist of transition names, choice operators (+), concatenations, Kleene stars (*) and parentheses. For example, $\{(a, b, c), a + aa + b\}$ is the protocol where transition $c$ does not fire and where either transition $a$ fires once or twice, or transition $b$ fires once. It differs from protocol $\{(a, b, c), \varepsilon + a + aa + b\}$, because the latter allows that $a$ and $b$ do not fire at all ($\varepsilon$ is the empty string). The protocol that prescribes that one or more a's or b's fire, can be described as $\{(a, b), (a + b)(a + b)^*\}$.

We present two algorithms. The first one checks whether a given P/T net $N$ is trace-safe w.r.t. a given regular protocol $\Pi$, i.e. whether $N$ can not do more than $\Pi$ allows. The second one verifies whether $N$ does not stop prematurely (weak progress). For example, a P/T net where transition $a$ can fire infinitely often is not trace-safe w.r.t. protocol $\{(a, b), aa + b\}$, because the

$^1$In non-Petri net literature, trace-safety is usually called safety [1]. We have chosen for a different name here, because safety has already a certain meaning in Petri net terminology.
protocol forbids the firing of more than two a's. A P/T net with two transitions a and b of which only one can fire once, is trace-safe w.r.t. protocol \( \{a, b\}, aa + b \), but there is no weak progress, because stopping after a single a is not desired. Even protocols of high-level nets can be verified, as long as the high-level net can be transformed to a P/T net, i.e., as long as the high-level net has a finite number of places, transitions, arcs, and values (colours).

Both algorithms are based on the following idea: The protocol, which is given by an alphabet and regular expression, is transformed into a deterministic finite automaton which accepts the language of the regular expression. That automaton is converted to a P/T net \( N' \), which we call the test net. It has always one token. We distinguish three kinds of places in \( N' \): Forbidden, intermediate and final places. The original net \( N \) and the test net \( N' \) are composed into a P/T net \( N'' \). A characteristic of \( N' \) is that it does not restrict the behaviour of \( N \) in \( N'' \). We prove that \( N \) is trace-safe w.r.t. \( P \) iff no forbidden place of \( N'' \) can get a token. We also prove that \( N \) and \( P \) satisfy the weak progress criterion iff \( N'' \) does not stop with a token in an intermediate place.

The automatic verification of trace-safety comprises a coverability analysis on \( N'' \) and weak progress is checked with reachability analyses on \( N'' \).

This approach can be applied to several interesting areas:

- In systems engineering, usually a global specification is made first. After that, repeatedly, a part of the system (a module) is replaced by a more detailed one. Such a replacement is correct iff the environment can not detect a difference between both modules. We have derived sufficient and necessary conditions for the replacement of modules, in case the environment can interact unlimitedly with the modules [25, 26]. These conditions can be weakened if the environment and the module maintain some communication protocol.

- In Petri net literature, many so-called behaviour preserving construction rules have been described. Nets that have a certain property (liveness, deadlock freeness, divergence freeness, safeness, boundedness, covering by place invariants, etc.) are composed in such a way, that the resulting net has the same property [2, 4, 12, 13, 14, 29, 30, 34]. For protocols, similar preservation rules can be derived. This allows a bottom-up construction of the system. If the building blocks of a system have a certain protocol (which can be checked by our algorithms), and a system is built according to the protocol preservation rules, the constructed system will automatically have the same protocol. The derivation of protocol preservation rules is a topic of further research.

- P/T nets can be analysed with our algorithms. It can be checked whether a system satisfies a regular protocol. There are two different approaches here: At first, systems can be analysed as a whole. We recommend this for small nets only, because our algorithms can be quite inefficient (see Section 7). The other approach, which corresponds to the reverse of the previous item, is a partitioning the system into small parts in such a way that the overall system satisfies a certain protocol iff the parts obey that protocol. The parts can be analysed with our algorithms; which is often much easier than an analysis of the system as a whole [9].

This article is organised as follows. First, we review the definitions of finite automata and P/T nets and we introduce (regular) protocols. Then we discuss what is meant by 'a P/T net satisfies a regular protocol' and we come up with several correctness criteria. For two of them, viz. trace-safety and weak progress, we present algorithms that determine whether an arbitrary P/T net and regular protocol satisfy them. The other correctness criteria are stronger and deal with infinite loops. Language theory, however, does not support infinite strings and therefore, those correctness criteria are beyond the scope of regular protocols. But we mention some cases for which infinite loops can be handled. After that, we make some remarks on the complexity
of our algorithms and we end with some conclusions.
We introduce some notations in an appendix and in another appendix, we prove that the sub­marking reachability problem, for P/T nets with a constant number of tokens, is NP-complete.

2 Preliminaries

See Appendix A for the notations used in this article.

We use the concepts regular expression and deterministic finite automaton. These concepts are well-known from language theory (see [20]).

Definition 2.1 Regular expression
The set $S$ of regular expressions over an alphabet $\Sigma$ is defined as follows: $\varepsilon \in S$, $\Sigma \subseteq S$, if $r_1 \in S$ and $r_2 \in S$ then $(r_1 r_2) \in S$ and $(r_1 + r_2) \in S$, if $r \in S$ then $r^* \in S$, and nothing else is in $S$.

To each regular expression $\varepsilon$, a regular language $L(\varepsilon) = \{ \varepsilon \}$ for all $\varepsilon \in \Sigma$; $L(\varepsilon) = \{ \varepsilon \}$; if $r_1$ and $r_2$ are regular expressions, then $L((r_1 r_2)) = L(r_1) \cdot L(r_2)$ and $L((r_1 + r_2)) = L(r_1) \cup L(r_2)$ and if $r$ is a regular expression, then $L(r^*) = L(r)^*$.

Usually, we omit a lot of brackets in regular expressions: We use the convention that $^*$ binds stronger than concatenation and $+$ binds the weakest, and we use the associativity of concatenation and $+$. For example, $a(a + ab^* + c)^*$ corresponds to $(a((a + (ab^*) + c)^*))$ and $(a(a + ((ab^*) + c))$, and they all evaluate to the same language.

Definition 2.2 Deterministic finite automaton
A deterministic finite automaton $A$ is a quintuple $(K, \Sigma, \delta, s, F)$, where $K$ is a finite set of states, $\Sigma$ is an alphabet, $\delta : K \times \Sigma \rightarrow K$ is the transition function, $s \in K$ is the initial state and $F \subseteq K$ is the set of final states.

The transition function $\delta$ can be extended to domain $K \times \Sigma^*$ as follows: For $k \in K$, $\delta'(k, \varepsilon) = k$ and for $\sigma \in \Sigma^*$ and $a \in \Sigma$, $\delta'(k, \sigma \cdot a) = \delta(\delta'(k, \sigma), a)$.

The language accepted by $A$, $L(A)$, is $\{ \sigma \in \Sigma^* | \delta'(s, \sigma) \in F \}$.

We review the classical notion of a P/T net (see, e.g., [27]).

Definition 2.3 Place/Transition (P/T) Net
A P/T net is a quadruple $(P,T,W,M_0)$, where $P$ is a finite set of places, $T$ is a finite set of transitions, such that $P \cap T = \emptyset$, $W \in \mathcal{B}(P \times T \cup (T \times P))$ gives the bag of arcs between places and transitions and $M_0 \in \mathcal{B}(P)$ is the initial marking.

Definition 2.4 Functions $D$, $R$, and $fs$
For any P/T net $N = (P,T,W,M_0)$ and for $M \in \mathcal{B}(P)$ and $\sigma \in T^*$, $D(N, M, \sigma)$ is the resulting marking when, starting from marking $M$, all transitions in $\sigma$ have fired consecutively.

$fs(N)$ is the set of firing sequences of $N$ and $R(N)$ is the set of reachable markings of $N$.

Formally,

$D(N, M, \varepsilon) = M$ and for $\sigma \in T^*$ and $t \in T$,

$D(N, M, \sigma \cdot t)$ is defined iff
3 Correctness criteria

\[ D(N, M, \sigma) \text{ is defined and } \forall p \in P : D(N, M, \sigma)(p) \geq W(p, t). \]

If defined, \[ D(N, M, \sigma \cdot t) = \lambda p \in P : D(N, M, \sigma)(p) - W(p, t) + W(t, p). \]

\( f_s(N) = \{ \sigma \in T^* \mid D(N, M_0, \sigma) \text{ is defined} \} \)

\( R(N) = \{ D(N, M_0, \sigma) \mid \sigma \in f_s(N) \} \)

Transition \( t \in T \) is enabled (can fire) in marking \( M \in B(P) \) iff \( D(N, M, t) \) is defined.

\[ \square \]

**Definition 2.5 (Regular) protocol**

A protocol is a tuple \( \Pi = (\alpha(\Pi), \theta(\Pi)) \), where \( \alpha(\Pi) \) is an alphabet and \( \theta(\Pi) \subseteq (\alpha(\Pi))^* \) is a set of strings over \( \alpha(\Pi) \).

Protocol \( \Pi \) is regular if \( \theta(\Pi) \) can be described by a regular expression, or, if it is accepted by a finite automaton.

\[ \square \]

In this article, alphabet \( \alpha(\Pi) \) is always a subset of the transitions \( T \) of a P/T net. Not all transitions \( T \) need to be involved in \( \Pi \). We often denote \( \theta(\Pi) \) by a regular expression.

In trace theory, a protocol is called a trace structure (see [32]).

Consider a case where we have two sets \( A \) and \( B \) of transitions and we want to express that transitions of both sets fire alternately. This can be described conveniently as follows: Label the transitions of \( A \) and \( B \) with an \( a \) and \( b \), respectively, and let the protocol be \( \{(a, b), (ab)^*\} \).

Without transition labels, the protocol would have been \( (A \cup B, \{(x, y) \mid x, y \in A \cup B\})^* \).

Transition labels are a kind of syntactical sugar:

**Lemma 2.6 Transition labelling**

Let a \( P/T \) net \( N = (P, T, W, M_0) \), a labelling function \( l \) with \( \text{dom}(l) \subseteq T \), and a regular protocol \( \Pi \) with \( \alpha(\Pi) \subseteq \text{rng}(l) \) be given, i.e., the alphabet of \( \Pi \) is a subset of the labels instead of the transitions.

There is a regular protocol \( \Pi' \), with \( \alpha(\Pi') \subseteq \text{dom}(l) \subseteq T \), \( l \circ \alpha(\Pi') = \alpha(\Pi) \) and \( l \circ \theta(\Pi') = \theta(\Pi) \).

**Proof**

Put \( \alpha(\Pi') = l^{-1} \circ \alpha(\Pi) \) and let \( A = (K, \Sigma, \delta, s, F) \) be a deterministic finite automaton that accepts \( \theta(\Pi) \). Put \( A' = (K', \Sigma', \delta', s', F') \) with \( K' = K, \Sigma' = \{ t \in T \mid l(t) \in \Sigma \}, \delta' = \lambda (k, a) \in K \times \Sigma' : \delta(k, l(a)) \) and \( F' = F \), and let \( \theta(\Pi') \) be the language accepted by \( A' \), then \( \theta(\Pi') \) is a regular language over \( \alpha(\Pi') \subseteq \text{dom}(l) \subseteq T \), \( l \circ \alpha(\Pi') = \alpha(\Pi) \) and \( l \circ \theta(\Pi') = \theta(\Pi) \).

\[ \square \]

Hence, any protocol based on transition labels can be rewritten to a protocol without such labels. Transition labels are not more general.

3 Correctness criteria

Under what conditions does a \( P/T \) net \( N = (P, T, W, M_0) \) satisfy a given protocol \( \Pi \)? Different correctness criteria can be formulated. We shall illustrate them with examples.

Of course, the net may not do more than the protocol allows. Net \( N \) is called trace-safe w.r.t. protocol \( \Pi \) iff
Correctness criteria

\[ fs(N) \cap \alpha(\Pi) \subseteq \text{pref } \circ \theta(\Pi) \] (1)

Figure 1 gives five P/T nets and five protocols, which are described by regular expressions. Net \( N_1 \) is not trace-safe w.r.t. protocol \( \Pi_1 \), as \( aa \in fs(N_1) \) and \( aa \notin \text{pref } \circ \theta(\Pi_1) \).

\[
\begin{align*}
N_1: & \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} & \quad \Pi_1: & \quad \{(a), a\} \\
N_2: & \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} & \quad \Pi_2: & \quad \{(a), aa\} \\
N_3: & \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} & \quad \Pi_3: & \quad \{(a), aa\} \\
N_4: & \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} & \quad \Pi_4: & \quad \{(a, b), a^*b\} \\
N_5: & \quad \begin{array}{c}
\text{a} \\
\text{a}
\end{array} & \quad \Pi_5: & \quad \{(a, b), a^*b\}
\end{align*}
\]

Figure 1: Correctness criteria.

If a net and a protocol satisfy Eq. 1, then it is still not guaranteed that the net does not stop prematurely. For example, net \( N_2 \) and protocol \( \Pi_2 \) satisfy Eq. 1, though termination after a single \( a \) is not desired. If it were, \( \Pi_2 \) would have been \( \{(a), a + aa\} \) or \( \{(a), e + a + aa\} \). The requirement which guarantees that the net does not stop too early is called weak progress, i.e.

\[ \forall \sigma \in fs(N) : \sigma \downarrow \alpha(\Pi) \in \text{pref } \circ \theta(\Pi) \setminus \theta(\Pi) \Rightarrow \exists t \in T : \sigma \cdot t \in fs(N) \] (2)

Weak progress means that the net will not stop if the protocol has not been completed yet. But even if the net does not stop prematurely, there may be no real progress, because some transitions that do not belong to the protocol might be able to fire infinitely often, without finalising the protocol. Consider, for example, net \( N_3 \) with protocol \( \Pi_3 \). \( N_3 \) is an extension of \( N_2 \) with a kind of environment, viz. transition \( b \) and a place. There is weak progress, because transition \( b \) can always fire. But \( b \) is not part of the protocol \( (b \notin \alpha(\Pi_3)) \). After \( a \) has fired, no transition that belongs to the protocol will ever fire. Strong progress requires that there is progress even within the protocol:

\[ \forall \sigma \in fs(N) : \sigma \downarrow \alpha(\Pi) \in \text{pref } \circ \theta(\Pi) \setminus \theta(\Pi) \Rightarrow \exists \sigma' \in T^* : \sigma \cdot \sigma' \in fs(N) \land \sigma' \downarrow \alpha(\Pi) \neq \epsilon \] (3)

Please note that strong progress implies weak progress.

A P/T net that has weak progress but not strong progress w.r.t. a protocol \( \Pi \), has a set of transitions that are not part of the protocol and that can fire infinitely often. The difference between weak and strong progress is an infinite loop of firings of transitions from \( T \setminus \alpha(\Pi) \).

A net which obeys a protocol according to Eqs. 1, 2 and 3 may still not be able to complete its task. Consider, e.g., \( N_4 \) and \( \Pi_4 \). Net \( N_4 \) is trace-safe w.r.t. \( \Pi_4 \) and there is strong progress.
However, no firing sequence of $N_4$ can be completed to an element of $\theta(\Pi_4)$, because transition $b$ can never fire. The completeness requirement is:

$$fs(N) = \text{pref}\left(\{\sigma \in fs(N) \mid \sigma \in \Theta(\Pi_4) \} \in \theta(\Pi_4)\right)$$

(4)

$N_4$ and $\Pi_4$ do not satisfy Eq. 4, as $\{\sigma \in fs(N_4) \mid \sigma \in \Theta(\Pi_4) \} = \emptyset$. Please note that Eq. 4 implies Eqs. 1, 2 and 3. A net satisfying Eq. 4 may still enter an undesirable infinite loop (often referred to as livelock). For example, $N_5$ and $\Pi_5$ satisfy Eq. 4, though $N_5$ may do an infinite number of $a$'s, while $\Pi_5$ prescribes a finite number of $a$'s followed by a $b$.

If a net and a protocol do not satisfy Eq. 1 or Eq. 2, then an undesired behaviour exists that is reached within finitely many steps. This is not the case with Eq. 3. Reconsider, for example, net $N_3$ and protocol $\Pi_3$. Strong progress (Eq. 3) requires that eventually, two $a$'s will occur, but after finitely many steps of $N_3$ it can still not be concluded that something wrong has happened.

A P/T net that satisfies Eqs. 1 and 2 w.r.t. a given protocol, but not Eq. 3 (and hence, not Eq. 4), has a kind of undesired infinite behaviour, because all undesired finite behaviour has been captured by Eqs. 1 and 2. Infinite loops are beyond the scope of regular expressions. The behaviours of $N_3$, $N_4$ and $N_5$ may be described by $b^*a^* + b^*$, $a^* + a^*b$, respectively, where $a^*$ denotes an infinite sequence of $a$'s. However, language theory does not support infinite strings.

As a consequence, we shall confine ourselves to trace-safety (Eq. 1) and weak progress (Eq. 2). The first algorithm presented below checks trace-safety and the next one weak progress. We shall return to infinite strings in Section 6.

4 Trace-safety

In this section, we present an algorithm to determine whether a P/T net $N = (P, T, W, M_0)$ and a regular protocol $\Pi$ with $\alpha(\Pi) \subseteq T$ are trace-safe (Eq. 1), i.e., whether

$$fs(N) \smallsetminus \alpha(\Pi) \subseteq \text{pref} \circ \theta(\Pi).$$

First, we present the algorithm. Then we give an example and after that we prove the correctness of our algorithm.

We shall apply a coverability analysis to solve trace-safety. The general form of the coverability problem is: To determine, for a P/T net and some marking $M$, whether the P/T net has a reachable marking $M' \supseteq M$. This is known to be decidable in exponential space [24, 28].

Algorithm 4.1 Trace-safety

1. Construct a deterministic finite automaton $A$ with alphabet $\Sigma = \alpha(\Pi)$ that accepts $\theta(\Pi)$. See [20] how to construct a deterministic finite automaton $A' = (K', \Sigma', \delta', s', F')$ with alphabet $\Sigma' = \{t \in T \mid \exists x, y \in T' : x \cdot t \cdot y \in \theta(\Pi)\}$ that accepts $\theta(\Pi)$. Automaton $A = (K, \Sigma, \delta, s, F)$ can be derived from $A'$ as follows: Introduce a new state $k'$, $k' \notin K'$, and when automaton $A$ reads a symbol from $\alpha(\Pi) \smallsetminus \Sigma'$, it moves to state $k'$, which is a trap: $k'$ is not a final state, and whenever $A$ enters state $k'$, it will always remain there.

Formally:
Property: $\Sigma \subseteq T$.

2. Construct a test net $N'$ based on $A$.
   This test net is a P/T net which has always exactly one token in one of its places. We distinguish three kinds of places in $N'$:
   - **End places**, denoted by $EP$.
     Termination in such a place is allowed.
   - **Intermediate places**, denoted by $IP$.
     Termination in such a place is not allowed, but an end place is still reachable.
   - **Forbidden places**, denoted by $FP$.
     These places may never contain a token (termination here is not allowed and there is no reachable end place).
   
   $N' = \{P', T', W', M_0'\}$, with:
   
   $$
P' = K$
   $$
T' = \text{dom}(\delta) = K \times \Sigma$
   $$
W' = \lambda (k, (k', a)) \in K \times (K \times \Sigma) : \begin{cases} 
1 & \text{if } k = k' \\
0 & \text{else}
\end{cases}
we \begin{cases} 
\lambda (k, a) \in \{k'\} \times \alpha(\Pi) : k'
\end{cases}$
   $$
M_0' = \{s\}$

Moreover,

$$
EP = F$
$$
IP = \{k_0 \in K \setminus F \mid \exists n \in N_1 : \exists k_1, \ldots, k_n \in K : \exists a_1, \ldots, a_n \in \Sigma : k_n \in F \land \forall i \in \{1, \ldots, n\} : k_i = \delta(k_{i-1}, a_i)\}$
$$
FP = K \setminus (EP \cup IP)$

3. Lay $N$ over $N'$, which results in a P/T net $N'' = \{P'', T'', W'', M_0''\}$, as follows.
   We assume without loss of generality: $P \cap P' = \emptyset$ and $(T \setminus \Sigma) \cap T' = \emptyset$.
   
   $$
P'' = P \cup P' = P \cup EP \cup IP \cup FP$
   $$
T'' = (T \setminus \Sigma) \cup T'$
   $$
W'' = W' \uplus \lambda (p, t) \in P \times (T \setminus \Sigma) : W(p, t)
\uplus \lambda (t, p) \in (T \setminus \Sigma) \times P : W(t, p)
\uplus \lambda (p, (k, a)) \in P \times T' : W(p, a)
\uplus \lambda ( \{k, a\}, p) \in T' \times P : W(a, p)
\uplus \lambda (k, t) \in P' \times (T \setminus \Sigma) : 0
\uplus \lambda (t, k) \in (T \setminus \Sigma) \times P' : 0$
   $$
M_0'' = M_0 \uplus M_0'$

4. Determine, for all $k \in FP$, whether $N''$ has a reachable marking $M'' \in R(N'')$ that covers $\{k\}$, i.e., $M'' \supseteq \{k\}$.
   If such an $M''$ exists for some $k \in FP$, then $N''$ is not trace-safe w.r.t. protocol $\Pi$, i.e. $N$ and $\Pi$ do not satisfy Eq. 1. Otherwise, $N$ and $\Pi$ are trace-safe.
Please note that alphabet $\Sigma'$ of the first step of this algorithm, may really be smaller than $\alpha(\Pi)$. For example, in protocol $\{\{a, b\}, \varepsilon\}$, transitions $a$ and $b$ are not allowed to fire, while other transitions still may fire. The derived alphabet $\Sigma'$ would be empty in this case.

Some properties of test net $N'$: (these properties are easily verified, we omit the proofs)

- $N'$ is a P/T net, i.e., $P'$ and $T'$ are finite.
- There is always exactly one token in $N'$.
  I.e., $I' = (\lambda s' \in S' : 1)$ is a place invariant of $N'$ and it adds up to 1 in each reachable marking: For all $M' \in R(N')$, $\sum_{s' \in S'} I'(s')M'(s') = 1$.
- Let $M'$ be a reachable marking of $N'$ and let $k \in K$ be the place with a token (i.e. $M'(k) = 1$), then all transitions $\{k, a\} \in \Sigma$ are enabled. In other words: For all $a \in \Sigma$, there is always a $k \in K$ such that $\{k, a\}$ can fire.
  This property indicates that the test net $N'$ does not influence the behaviour of $N$, when we lay $N$ over $N'$ in step 4. $N'$ allows the firing of each transition of $N$ at any time.

Because $N'$ is contained in $N''$ and all places $P'$ of $N'$ have no additional arcs in $N''$, net $N''$ has a similar place invariant as $N'$: There is always one token in one of the places of $P'$.

Formally, $I'' = (\lambda s \in S : 0) \cup (\lambda s' \in S' : 1)$ is a place invariant of $N''$ and for all $M'' \in R(N'')$, $\sum_{s \in S} I''(s)M''(s) = 1$.

In the last step of the algorithm, $|FP|$ coverability analyses are performed. Verification of trace-safety can be easier with the aid of an additional place $x$ in $N''$. Let all transitions with an output arc to a forbidden place, have an output arc to $x$ as well. No other arcs go to or from place $x$. With this construction, $N$ is trace-safe w.r.t. $\Pi$ iff there is no reachable marking in $N''$ with one or more tokens in $x$. Thus, a single coverability analysis suffices.

Example 4.2
Let P/T net $N$ be given in Figure 2 and let regular protocol $\Pi$ be given by $\{\{a, b\}, (b + ab)^*\}$.

To determine whether $N$ is trace-safe w.r.t. $\Pi$.

1. From $\Pi$, the finite deterministic automaton $A = (K, \Sigma, \delta, s, F)$ can be derived, with:

   $K = \{k_1, k_2, k_3\}$

\[ \Pi = \{\{a, b\}, (b + ab)^*\} \]
\[\Sigma = \{a, b\}\]
\[\delta = \{(k_1, a), k_2\}, ((k_1, b), k_1), ((k_2, a), k_3), ((k_2, b), k_1), ((k_3, a), k_3), ((k_3, b), k_3)\}\]
\[s = k_1\]
\[F = \{k_1\}\].

See Figure 3. Final states are indicated by a circle, and the initial state is shown by a >.

2. From automaton \(A\), test net \(N'\) of Figure 4 is derived.

3. \(N''\) is \(N\) over \(N'\): See Figure 5.

4. \(N''\) has a reachable marking that covers \(\{k_3\}\), viz. \(s_2, k_3\). That marking is obtained after firing sequence \(c(k_1, a)(k_2, a)\). Hence, net \(N\) is not trace-safe w.r.t. protocol II.

\(\square\)
Theorem 4.3
Algorithm 4.1 is correct. In other words,
\[ fs(N) \cap \alpha(\Pi) \subseteq \text{pref} \circ \theta(\Pi) \iff \forall M'' \in R(N'') : \forall k \in FP : M''(k) = 0. \]
\[ \square \]

Before we prove this theorem, we introduce three functions, viz. \( f, \kappa \) and \( \Delta \), and we prove three lemmas.

Function \( f \in T'' \rightarrow T \) maps each transition of \( N'' \) to a transition of \( N \):
\[ f = (\lambda t \in T \backslash \Sigma : t) \cup (\lambda (k,a) \in K \times \Sigma : a) \]

The mentioned place invariants imply that there is one place \( k \in K \) in each reachable marking of \( N' \) and \( N'' \) that has a token. Function \( \kappa \) gives this place.
\[ \kappa = \lambda M \in R(N') \cup R(N'') : \text{the } k \in K \text{ with } M(k) = 1 \]

Function \( \Delta \in T* \rightarrow B(K) \) gives the marking of test net \( N' \) after a sequence of transitions of \( N \) has fired.
\[ \Delta(c) = M_0 = \{ s \}, \text{ and for } \sigma \in T* \text{ and } a \in T, \]
\[ \Delta(\sigma \cdot a) = \begin{cases} \Delta(\sigma), & a \notin \Sigma \\ D(N', \Delta(\sigma), (\kappa \circ \Delta(\sigma), a)), & a \in \Sigma \end{cases} \]

In net \( N'' \), the behaviour of test net \( N' \) is restricted: \( N' \) has to follow original net \( N \). The following lemma expresses that test net \( N' \) validates firing sequences of \( N \). As we have argued, test net \( N' \) has always one token. If this token is in an end place, the firing sequence processed so far agrees with a string of the protocol. If it is in an intermediate state, the current firing sequence corresponds to a real prefix of the protocol and else, if it resides in a forbidden place, the current firing sequence can not be completed anymore to a string of the protocol.

Lemma 4.4
\[ \forall \sigma \in fs(N) : \begin{cases} \sigma \Sigma \in \Theta(\Pi) & \Rightarrow \kappa \circ \Delta(\sigma) \in EP \\ \sigma \Sigma \in \text{pref} \circ \theta(\Pi) \setminus \Theta(\Pi) & \Rightarrow \kappa \circ \Delta(\sigma) \in IP \\ \sigma \Sigma \notin \text{pref} \circ \theta(\Pi) & \Rightarrow \kappa \circ \Delta(\sigma) \in FP \end{cases} \]

Proof
Let \( \sigma \in fs(N) \). If \( \sigma \Sigma \in \Theta(\Pi) \), then automaton \( A \) accepts \( \sigma \Sigma \), i.e., after \( \sigma \Sigma \) is a in a state \( k \in F = EP \).

If \( \sigma \Sigma \in \text{pref} \circ \theta(\Pi) \setminus \Theta(\Pi) \), then \( A \) is not in a final state after \( \sigma \Sigma \), but such a state can be reached, because there is a \( \sigma' \in \Sigma^* \) with \( (\sigma \cdot \sigma') \Sigma \in \Theta(\Pi) \).

If \( \sigma \Sigma \notin \text{pref} \circ \theta(\Pi) \), then, after \( \sigma \Sigma \), \( A \) is in a state from which no final state can be reached anymore.

Now apply functions \( \kappa \) and \( \Delta \) to \( N' \).
\[ \square \]
Lemma 4.5

\[ \text{fs}(N) = f \circ \text{fs}(N') \]

Proof

\[ \subseteq : \text{Let } \sigma \in \text{fs}(N). \text{ To be proven: } \exists \sigma'' \in \text{fs}(N') : f(\sigma'') = \sigma. \]

We define markings \( M_i, M'_i \) and \( M''_i \) (\( i \in \{1, \ldots, \#\sigma\} \)) for nets \( N, N' \) and \( N'' \), respectively, and a string \( \sigma'' \in (T'')^* \) with \#\(\sigma''\) = \#\(\sigma\).

\[
\begin{align*}
M_i &= D(N, M_0, \sigma[i]) \\
M'_i &= \Delta(\sigma[i]) \\
M''_i &= M_i \uplus M'_i \\
\sigma''[i] &= \begin{cases} 
(\kappa(M'_{i-1}), \sigma[i]) & \sigma[i] \in \Sigma \\
\sigma[i] & \sigma[i] \in T \setminus \Sigma
\end{cases}
\end{align*}
\]

Then \( f(\sigma'') = \sigma \) and by the construction of \( N'', \sigma'' \in \text{fs}(N'') \).

\[ \supseteq : \text{Let } \sigma'' \in \text{fs}(N''). \text{ Put } M''_i = D(N'', M''_{i-1}, \sigma''[i]) \text{ for all } i \in \{1, \ldots, \#\sigma\}. \]

\( M''_i \) can be split into \( M_i = M''_i \uplus P \) and \( M'_i = M''_1 \uplus P' \).

If \( \sigma''[i] \in T \setminus \Sigma \), i.e. \( \sigma''[i] \) is a transition that is not part of the alphabet of the protocol, then \( M_i = D(N, M_{i-1}, \sigma''[i]) \) and \( M'_i = M''_{i-1} \).

Otherwise, if \( \sigma''[i] \in \Sigma \), i.e. \( \sigma''[i] = (k, a) \) for some \( k \in K \) and \( a \in \Sigma \), then \( M_i = D(N, M_{i-1}, a) \) and \( M'_i = D(N', M'_{i-1}, (k, a)) \).

Hence, \( f(\sigma'') \in \text{fs}(N) \).

\[ \square \]

The following lemma indicates that each marking of net \( N'' \) consists of two separate parts, viz. an \( N' \)- and an \( N'' \)-part.

Lemma 4.6

\[ \forall \sigma'' \in \text{fs}(N'') : D(N'', M''_0, \sigma'') = D(N, M_0, f(\sigma'')) \uplus \Delta \circ f(\sigma'') \]

Proof

We use induction. If \( \sigma'' = \varepsilon \), then \( D(N'', M''_0, \sigma'') = M''_0 = M_0 \uplus M'_0 = D(N, M_0, f(\varepsilon)) \uplus \Delta \circ f(\varepsilon) \).

Otherwise, \( \sigma'' = \sigma_0 \cdot x \) for some \( \sigma_0 \in (T'')^* \) and \( x \in T'' \). Put \( X = D(N'', M''_0, \sigma_0) \), then by induction, \( X = D(N, M_0, f(\sigma_0)) \uplus \Delta \circ f(\sigma_0) \). There are two possibilities: \( x \in K \times \Sigma \) or \( x \in T \setminus \Sigma \).

\[ x \in K \times \Sigma : D(N'', M''_0, \sigma'') = \varepsilon \text{ def. } D \uplus D(N'', X, x) = \varepsilon \text{ def. } N'', \text{ def. } f \uplus \]

\[
\begin{align*}
D(N, X \uplus P, f(x)) \uplus D(N', X \uplus P', x) &= \varepsilon \text{ induction } \varepsilon \\
D(N, D(N, M_0, f(\sigma_0)), f(x)) \uplus D(N', \Delta \circ f(\sigma_0), x) &= D(N, M_0, f(\sigma_0)) \uplus \Delta \circ f(\sigma'') \\
&= \varepsilon \text{ def. } D, \text{ and because of place invariant } l'', x = (\kappa \circ \Delta \circ f(\sigma_0), f(x)) \uplus \\
D(N, M_0, f(\sigma'')) \uplus D(N', \Delta \circ f(\sigma_0), (\kappa \circ \Delta \circ f(\sigma_0), f(x))) &= \varepsilon \text{ def. } D \uplus \\
D(N, M_0, f(\sigma'')) \uplus \Delta \circ f(\sigma'').
\end{align*}
\]

\[ x \in T \setminus \Sigma : D(N'', M''_0, \sigma'') = \varepsilon \text{ def. } D \uplus D(N'', X, x) = \varepsilon \text{ def. } N'', \text{ def. } f \uplus \]

\[
\begin{align*}
D(N, X \uplus P, f(x)) \uplus D(N', X \uplus P', x) &= \varepsilon \text{ induction } \downarrow \\
D(N, D(N, M_0, f(\sigma_0)), f(x)) \uplus D(N', D(N, M_0, f(\sigma_0)), f(x)) \uplus \Delta \circ f(\sigma_0) &= \varepsilon \text{ def. } D, \text{ def. } D \uplus D(N, M_0, f(\sigma'')) \uplus \Delta \circ f(\sigma'').
\end{align*}
\]

\[ \square \]

The correctness of our trace-safety algorithm can now be proven.
5 Weak progress

Below, we present an algorithm to determine whether a P/T net \( N = (P,T,W,M_0) \) and a regular protocol \( \Pi \) with \( \alpha(\Pi) \subseteq T \) satisfy weak progress (Eq. 2), i.e., whether

\[
\forall \sigma \in \text{fs}(N) : \sigma \cap \alpha(\Pi) \in \text{pref} \circ \theta(\Pi) \setminus \theta(\Pi) \Rightarrow \exists t \in T : \sigma \cdot t \in \text{fs}(N).
\]

First, we give the algorithm and then we prove its correctness.

We have solved the trace-safety problem with the aid of a coverability analysis. For weak progress, we shall use submarking reachability analyses. The general form of the submarking reachability problem is: Given a P/T net \( N = (P,T,W,M_0) \), a subset of places \( P' \subseteq P \) and a submarking \( M' \in B(P') \), does a reachable marking \( M \in R(N) \) exist with \( \forall p \in P' : M(p) = M'(p) \)?

Van Leeuwen [33] showed that the submarking reachability problem reduces to the reachability problem (see also [23]), i.e., a P/T net \( N'' \) and a marking \( M'' \) can be constructed from \( N, T' \) and \( M' \), such that an above-mentioned reachable submarking exists iff \( M'' \) is a reachable state of \( N'' \).

The reachability problem is solvable [19, 21, 22].

For the verification of trace-safety, we have constructed an automaton \( A \) with alphabet \( \Sigma = \alpha(\Pi) \). Alphabet \( \Sigma \) does not have to contain all transitions of net \( N : \Sigma \subseteq T \), but not necessarily \( \Sigma = T \). For trace-safety, it is irrelevant whether transitions that do not appear in \( \Pi \) can fire, or not. For example, the protocol might require that two transitions \( a \) and \( b \) fire after each other all the time \( \Pi = \{(a,b),(ab)^*\} \), while the net has many more transitions than only \( a \) and \( b \). Those other transitions are irrelevant to the protocol and do not have to be part of automaton \( A \) and test net \( N' \).

For weak progress, we need to check whether any transition in \( N \) can fire, or not. We have to involve all transitions \( T \) of \( N \). To this end, we extend automaton \( A \) in such a way that \( \Sigma = T \).

Algorithm 5.1 Weak progress

1. Let \( A = (K,\Sigma,\delta,a,F) \) be the deterministic finite automaton of Step 1 of Algorithm 4.1, i.e. \( \alpha(\Pi) = \Sigma \subseteq T \) and \( A \) accepts \( \theta(\Pi) \).

   Put \( A' = (K',\Sigma',\delta',a',F') \), with
   \[
   K' = K \cup T
   \]
   \[
   \Sigma' = T
   \]
5 Weak progress

\[ \delta' = \delta \cup (\lambda (k, t) \in K \times (T \setminus \Sigma) : k) \]
\[ s' = s \]
\[ F' = F, \]

then \( A' \) is a deterministic finite automaton that accepts \( \theta(\Pi') \), where \( \Pi' = (T', \{ \sigma \in T' | \sigma \in \alpha(\Pi) \in \theta(\Pi) \}) \).

2. Construct a test net \( N' \) based on \( A' \) instead of \( A \).

See the construction in Algorithm 4.1.

3. Lay \( N \) over \( N' \).

See also the construction in Algorithm 4.1.

Remark: Because \( T \setminus E' = 0 \), \( T'' = T' \) and \( W'' = W' \).

4. Test whether \( N'' \) can stop in an intermediate state.

We use a finite number of submarking reachability tests for this.

Net \( N'' \) can stop in an intermediate state \( k \in IP \) iff there is a reachable marking \( M'' \in R(N'') \) with \( M''(k) = 1 \) and no transition that consumes tokens from \( k \), can fire, i.e.

\[ \forall t \in T : \exists p \in P : M''(p) < W''(p, (k, t)) \]

(see Figure 6 for an example). Since \( W''(p, (k, t)) = W(p, t) \), the phrase “\( N'' \) can stop in an intermediate state” can be rephrased as

\[ \exists M'' \in R(N'') \exists k \in IP : (M''(k) = 1 \land \forall t \in T : \exists p \in P : M''(p) < W(p, t)) \]  (5)

This equation can be verified with the aid of finitely many submarking reachability analyses, as follows:

Choose \( k \in IP, X \subseteq P \) and \( M \subseteq IB(X) \) such that:

- \( \forall t \in T : \exists x \in X : M(x) < W(x, t) \).
  In other words, submarking \( M \) disables all transitions of \( N \) and hence, submarking \( M \cup \{k\} \) disables all transitions of \( N'' \). For each transition \( t \in T \), there is a place \( x \in X \) that has too less tokens for \( t \) to fire.

- \( X \) is as small as possible, i.e. the previous item does not hold if one place is omitted from \( X \).
  Formally: \( \forall x' \in X : \exists t \in T : \forall x \in X \setminus \{x'\} : M(x) \geq W(x, t) \).

An upper bound for the number of ways to choose such a \( k \), \( X \) and \( M \) can be found as follows. Choose a \( k \in IP \), and for all transitions \( t \in T \), choose a place where \( t \) consumes from and let it have less tokens than \( t \) needs to fire. This can be done in at most \( |IP| \times |T| \times |P| \times \max_{w \in W} \text{rang}(W) \) different ways. That number may be large, but it is finite.

For each such \( k \), \( X \) and \( M \), apply a submarking reachability analysis on \( N'' \) to determine whether there is an \( M'' \in R(N'') \) with \( M''(k) = 1 \) and \( \forall x \in X : M''(x) = M(x) \). If so, Eq. 5 is satisfied, which means that \( N'' \) can stop in an intermediate state. Otherwise, \( N'' \) can not stop in an intermediate state.

Below we prove that \( N \) and \( \Pi \) satisfy the weak progress criterion iff \( N'' \) can not stop in an intermediate state.
Figure 6: Part of an original net $N$ and corresponding $N''$. If place $k$ in $N''$ has a token and transition $(k, t)$ cannot fire, then $p$ or $p'$ does not have enough tokens.

**Theorem 5.2**
The above algorithm is correct. In other words, the negation of Eq. 5 is equivalent with Eq. 2.

**Proof**

\[ \forall M'' \in R(N'') : \forall k \in IP : (M''(k) = 1) \Rightarrow \exists t \in T : \forall p \in P : M''(p) \geq W(p, t) \]

\[ \Leftrightarrow \text{ def. } R, \text{ def. } D \]

\[ \forall \sigma'' \in fs(N'') : \forall k \in IP : (D(N'', M_0'', \sigma'')(k) = 1) \Rightarrow \exists t \in T : \forall p \in P : D(N'', M_0'', \sigma'')(p) \geq W(p, t) \]

\[ \Leftrightarrow \text{ def. predicate calculus} \]

\[ \forall \sigma'' \in fs(N'') : (\exists k \in IP : D(N'', M_0'', \sigma'')(k) = 1) \Rightarrow \exists t \in T : \forall p \in P : D(N'', M_0'', \sigma'')(p) \geq W(p, t) \]

\[ \Leftrightarrow \text{ def. } \kappa, \text{ Lemma 4.6} \]

\[ \forall \sigma'' \in fs(N'') : (\exists k \in IP : \Delta \circ f(\sigma'')(k) = 1) \Rightarrow \exists t \in T : \forall p \in P : D(N, M_0, f(\sigma''))(p) \geq W(p, t) \]

\[ \Leftrightarrow \text{ def. } \kappa, \text{ Lemma 4.5} \]

\[ \forall \sigma \in fs(N) : \kappa \circ \Delta(\sigma) \in IP \Rightarrow \exists t \in T : \forall p \in P : D(N, M_0, \sigma)(p) \geq W(p, t) \]

\[ \Leftrightarrow \text{ def. } \kappa, \text{ Lemma 4.4 with } \Pi \text{ and } \Sigma \text{ replaced by } \Pi' \text{ and } \Sigma' = T, \text{ respectively; def. } D \text{ and } fs \]

\[ \forall \sigma \in fs(N) : \sigma \in \text{pref } \theta(\Pi') \setminus \theta(\Pi') \Rightarrow \exists t \in T : \sigma \cdot t \in fs(N) \]

\[ \Leftrightarrow \theta(\Pi') = \{ \sigma \in T^* \mid \sigma \cdot \alpha(\Pi) \in \theta(\Pi) \} \]

\[ \forall \sigma \in fs(N) : \sigma \cdot \alpha(\Pi) \in \text{pref } \theta(\Pi') \setminus \theta(\Pi) \Rightarrow \exists t \in T : \sigma \cdot t \in fs(N) \]

\[ \square \]

## 6 Infinite behaviour

We have mentioned four correctness criteria for P/T nets and regular protocols, viz. trace-safety, weak progress, strong progress and completeness. This list is not complete. For example, net $N_3$ and protocol $\Pi_3$ of Figure 1 satisfy the completeness requirement (Eq. 4), though $N_3$ can do an infinite number of $a$'s, while $\Pi_3$ prescribes a finite number of $a$'s followed by a $b$. Such undesired infinite loops might be forbidden by another correctness criterion.

We have given algorithms for the verification of the first two correctness criteria. As we have
argued in Section 3, the other correctness criteria concern undesired infinite behaviour.

In Figure 7, we distinguish four kinds of infinite behaviour. Transition \(b\) can fire unboundedly often in all four cases. In case I, after \(b\) has fired once, it can fire finitely often and then it will stop. In case IV, \(b\) will fire forever. II and III are intermediate cases: In case II, \(b\) may keep on firing until \(a\) has fired (which may never happen) and in case III, \(b\) suffers from livelock: \(b\) is stuck if \(a\) keeps on firing.

In order to automatically verify the other correctness criteria as well, we need algorithms that can distinguish among these four different kinds of infinite behaviour.

There is an algorithm that distinguishes between cases I or II on the one hand, and cases III or IV on the other hand. In I and II, transition \(b\) is not live and in III and IV, \(b\) is. The general form of the liveness problem is: To determine, for a P/T net \(N = (P, T, W, M_0)\) and a transition \(t \in T\), whether

\[
\forall M \in R(N) : \exists \sigma \in T^* : \sigma \{ t \} \neq \varepsilon \land D(N, M, \sigma) \text{ is defined}.
\]

The liveness problem is equivalent with the reachability problem [16] and therefore decidable [19, 21, 22].

7 Complexity

The input of our algorithms is a P/T net \(N\) and a regular protocol. If the protocol is given by a regular expression, the following steps are performed by both of our algorithms:

1. Convert the regular expression to a non-deterministic automaton \(A\). The construction in [20] requires polynomial time and the number of states of \(A\) is linear w.r.t. the length of the regular expression.
2. Convert the non-deterministic automaton to a deterministic one. The construction in [20] requires exponential space: If the non-deterministic automaton has \(k\) states, then the deterministic one has \(2^k\) states.
3. Convert the deterministic automaton \(A'\) to a test net \(N'\). Requires polynomial time w.r.t. the size of \(A'\).
4. Lay \(N\) over \(N'\), resulting in \(N''\). Requires polynomial time w.r.t. the size of \(N\) and \(N''\).

\(^2\)The size of a P/T net is the number of bits needed to represent it (see [17]).
Thus, due to the second step, the construction of net $N''$ requires exponential space w.r.t. the length of the regular expression. Algorithm 4.1, which checks trace-safety, performs a coverability analysis on $N''$. The coverability problem is exponential space hard and an algorithm exists that uses exponential space [24, 28]. Hence, our trace-safety algorithm uses exponential exponential space w.r.t. the length of the regular expression and exponential space w.r.t. the size of $N$.

Algorithm 5.1, which checks weak progress, performs a number of submarking reachability analyses on net $N'$. The submarking reachability problem can be reduced to the reachability problem in polynomial time [23], but the complexity of the reachability problem is unknown and at best exponential space hard [7, 22]. Hence, the algorithm for weak progress is at best exponential exponential space hard w.r.t. the length of the regular expression and at best exponential space hard w.r.t. the size of $N$.

Our algorithms may be inefficient. In fact, they give an upper bound for the complexity of the trace-safety and weak progress problems. The actual complexity of these problems may be much better than what is sketched above.

8 Concluding remarks

In this article, we have presented two algorithms to determine whether an arbitrary P/T net $N$ with transitions $T$ satisfies a protocol $\Pi$, where $\Pi$ consists of an alphabet $A \subseteq T$ and a regular language over $A$.

The first algorithm (trace-safety) tests whether $N$ does not do something wrong and the second one (weak progress) checks whether $N$ does not stop prematurely. These algorithms can be very helpful in systems engineering and analysis.

Our algorithms are more general than existing algorithms for the automatic verification of protocols, because they have not been designed for one particular protocol, nor for a very specific class of Petri nets. For example, the automatic verification of the arbiter cascade [15] boils down to a mutual exclusion check and a liveness test. The former can be handled with our trace-safety algorithm and for the latter, an algorithm already exists (see Section 6). We can handle the alternating bit protocol [5, 10, 31] as well, in a similar way.

Current research in the automatic verification of more specific protocols has not become superfluous by our algorithms. Our algorithms are not very efficient (see Section 7) and it is still very interesting to find efficient algorithms for special cases.

Topics of further research:

- An implementation of our algorithms, to gain an insight into the average-case efficiency;
- Efficiency enhancing. Our algorithms consist of several steps and we did not investigate whether each step is really necessary, nor whether each step can be replaced by a more efficient one;
- A derivation of efficient algorithms for special cases.
  For example, the reachability problem is NP-complete for P/T nets where the number of tokens never changes [17]. As a consequence, the submarking reachability problem for this kind of P/T nets is NP-complete, too (see Appendix B).
  If the original, to be analysed, net $N$ has a non-changing number of tokens, then net $N''$ (on which submarking reachability tests are performed in Algorithm 5.1) has this property, too. Each submarking reachability test is now NP-complete instead of at best exponential space w.r.t. the size of $N''$;
- The definition of protocol preserving construction rules for the engineering and analysis of systems (see the introduction);
Besides trace-safety and weak progress, there are other correctness criteria. The list of criteria mentioned in Section 3 is not complete (see Section 6). We would like to extend this list and find decision algorithms for other correctness criteria as well;

- An extension to non-regular protocols and protocols with infinite strings.

References


REFERENCES


A  

Notations

\( \mathbb{N}_0 = \{0, 1, \ldots\} \) is the set of natural numbers and \( \mathbb{N}_1 = \mathbb{N}_0 \setminus \{0\} \). For \( A \) and \( B \) sets, \( A \supseteq B \) if \( B \subseteq A \) and the set of all total functions from \( A \) to \( B \) is \( A \rightarrow B \). We denote function restriction by \( \upharpoonright \) and function concatenation by \( \circ \). The domain \( \text{dom}(f) \) of a function \( f \in A \rightarrow B \) is \( A \) and the range, \( \text{rng}(f) \), is \( \{f(a) \mid a \in A\} \). The inverse of \( f \) is \( f^{-1} \).

\( \mathcal{P}(A) \) denotes the set of all subsets of \( A \) and \( \mathcal{B}(A) = A \rightarrow \mathbb{N}_0 \) is the set of all bags (multisets) over \( A \). For \( b \in \mathcal{B}(A) \) and \( x \) some element: \( x \in b \) iff \( x \in A \) and \( b(x) > 0 \). The number of elements in bag \( b \) is denoted by \( \#b \), i.e. \( \#b = \sum_{a \in A} b(a) \). We denote bag union with \( \uplus \): For \( A_1 \) sets and \( b_1, b_2 \in \mathcal{B}(A) \) (\( \{1, 2\} \)), \( b_1 \uplus b_2 = (\lambda a \in A \setminus A_1 : b_1(a)) \cup (\lambda a \in A_1 : b_2(a)) \). Moreover, \( b_1 \subseteq b_2 \) iff \( \forall a \in b_1 : a \in b_2 \land b_1(a) \leq b_2(a) \); \( b_1 \supseteq b_2 \) iff \( b_2 \subseteq b_1 \); and \( b_1 \neq b_2 \) iff \( b_1 \subseteq b_2 \) and \( b_2 \subseteq b_1 \). Note: Any bag \( b \in \mathcal{B}(A) \) can be generalised to a larger domain \( A' \supseteq A \): Put \( b' = b \cup (\lambda a \in A' \setminus A : 0) \), then \( b' \in \mathcal{B}(A') \) and by our definition of bag equality, \( b = b' \).

Sets can be viewed as bags. For \( S \) a set, the corresponding bag \( b \in \mathcal{B}(S) \) is \( \lambda s \in S : 1 \).

\( A^* \) is the set of all finite sequences (strings) over set \( A \). The empty string is \( \epsilon \). For \( \sigma \in A^* \), \( \#\sigma \) is its length, \( \sigma[i] \) (\( 1 \leq i \leq \#\sigma \)) is the \( i \)-th element of \( \sigma \) and \( \sigma[1..i] \) (\( 0 \leq i \leq \#\sigma \)) is the prefix of \( \sigma \) with length \( i \). If \( S \) is a set of strings, then \( \text{pref}(S) \) is the set of all prefixes, i.e. \( \text{pref}(S) = \{ \sigma[1..i] \mid \sigma \in S \land 0 \leq i \leq \#\sigma \} \). We denote string concatenation with \( \cdot \), but the \( \cdot \) is often omitted. We denote string restriction, like function restriction, with \( \upharpoonright \): For \( A, B \) sets, \( \sigma \in A^* \) and \( a \in A \), \( \sigma \upharpoonright B = \epsilon \) and \( (\sigma \cdot a) \upharpoonright B = (\sigma \upharpoonright B) a \) if \( a \in B \), otherwise \( (\sigma \cdot a) \upharpoonright B = \sigma \upharpoonright B \).

We lift any function \( f \) with domain \( A \) to domains \( \mathcal{P}(A) \) and \( A^* \), as follows: For \( S \in \mathcal{P}(A) \), \( f(S) = \{ f(s) \mid s \in S \} \) and for \( a_1 \cdots a_n \in A^* \) (\( n \in \mathbb{N}_0 \)), \( f(a_1 \cdots a_n) = f(a_1) \cdots f(a_n) \).

We use \( \downarrow \) ... \( \Rightarrow \) for comment.

B  

The Submarking Reachability Problem For P/T Nets With A Constant Number of Tokens is NP-Complete

In this appendix, we consider only P/T nets where the number of tokens does not change. We denote this class with \( \mathcal{N} \). \( \mathcal{N} \) is the class of P/T nets \( N = (P, T, W, M_0) \) with

\[
\forall t \in T : \sum_{p \in P} W(p, t) = \sum_{p \in P} W(t, p).
\]

The reachability problem for class \( \mathcal{N} \) is: To determine, for \( N = (P, T, W, M_0) \in \mathcal{N} \) and \( M \in \mathcal{B}(P) \), whether \( M \in R(N) \).

The submarking reachability problem for class \( \mathcal{N} \) is: To determine, for \( N = (P, T, W, M_0) \in \mathcal{N} \), \( P' \subseteq P \) and \( M' \in \mathcal{B}(P') \), whether \( \exists M \in R(N) : \forall p \in P' : M(p) = M'(p) \).

We shall show that the reachability problem for class \( \mathcal{N} \) is equivalent to the submarking reachability problem for class \( \mathcal{N} \). It is known that the former is NP-complete [17]. Therefore, the latter is NP-complete, too.


Each reachability problem is a submarking reachability problem. Hence, the reachability problem for class $\mathcal{N}$ reduces to the submarking reachability problem for class $\mathcal{N}$ in a trivial way.

Next, let a submarking reachability problem with net $N = (P, T, W, M_0) \in \mathcal{N}$, $P' \subseteq P$ and $M' \in \mathcal{B}(P')$ be given. If $\#M_0 < \#M'$ then submarking $M'$ is not reachable. In the sequel, we assume that $\#M_0 \geq \#M'$.

We shall construct a $P/T$ net $N'' \in \mathcal{N}$ with places $P''$ and a marking $M'' \in \mathcal{B}(P'')$, such that:

$$M'' \in R(N'') \iff \exists M \in R(N) : \forall p \in P' : M(p) = M'(p).$$

(6)

The idea is to extend $N$ with a place $x$ and for each place $p \in P \setminus P'$, an extra transition is added with an input arc from $p$ and an output arc to $x$.

Formally, $N'' = (P'', T'', W'', M_0'')$, with:

$$P'' = P \cup \{x\},$$

where $x$ is a new place ($x \notin P$)

$$T'' = T \cup \text{rng}(-)$$

$$W'' = W \cup \{\{p, \,(p)\} \mid p \in P \setminus P'\} \cup \{\,(p), x\} \mid p \in P \setminus P'\}$$

$$M_0'' = M_0.$$  

where $-$ is an injective function on domain $P \setminus P'$ such that $\text{rng}(\) \text{,} T$ and $P \cup \{x\}$ are mutually disjoint. (Such a function can always be constructed.) Furthermore, $M'' = M' \cup \{(x, \#M_0 - \#M')\}$

**Example B.1**

Let $N \in \mathcal{N}$ be the net of Figure 8. It has 4 places, viz. $a, b, c$ and $d$. We would like to know whether it has a reachable marking with 4 tokens in place $a$ and 4 tokens in place $b$.

The corresponding net $N'' \in \mathcal{N}$ is shown in Figure 9. It has an additional place $x$, which is empty at start.

Net $N$ has a reachable submarking $M'$ with $M'(a) = M'(b) = 4$ iff $N''$ has a reachable marking $M''$ with $M''(a) = M''(b) = 4$, $M''(c) = M''(d) = 0$ and $M''(x) = 10 - 8 = 2$.

$$\square$$

![Figure 8: Net $N$, $N \in \mathcal{N}$](image1)

![Figure 9: Corresponding $N''$.](image2)

**Theorem B.2**

The submarking reachability problem reduces to the reachability problem in class $\mathcal{N}$.

\[\text{There is indeed such a reachable submarking. We leave it to the reader to check this out.}\]
Proof
First, we remark that the constructions of net \( N'' \) and marking \( M'' \) require linear time w.r.t. the size of \( N, P' \) and \( M' \).
What is left to prove is Eq. 6.
\( \Rightarrow \) : If \( M'' \in R(N'') \), then all places \( p \in P' \) have the amount of tokens specified by \( M' \). The other original places, i.e. the places of \( P \setminus P' \), are empty and place \( z \) has \( \#M_0 - \#M' \) tokens. Then, \( N \) must have a reachable marking with \( \#M_0 - \#M' \) tokens somehow distributed over places \( P \setminus P' \), while all places \( p \in P' \) have \( M'(p) \) tokens.
\( \Leftarrow \) : If \( N \) has a reachable marking \( M \) with in each place \( p \in P' \) exactly \( M'(p) \) tokens, then \( N'' \) has the same reachable marking \( M \) because \( N \) is contained in \( N'' \). Then, all newly added transitions can empty places \( P \setminus P' \), thus leaving \( \#M_0 - \#M' \) tokens in place \( z \).

Hence, like the reachability problem for class \( N \), the submarking reachability problem for class \( N \) is NP-complete.
**In this series appeared:**

<table>
<thead>
<tr>
<th>Year/Volume</th>
<th>Authors / Editors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H.C.M. de Swart</td>
<td></td>
</tr>
<tr>
<td>1991/03</td>
<td>J.P. Katoen</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</td>
</tr>
<tr>
<td></td>
<td>L.A.M. Schoenmakers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A.F. v.d. Stappcn</td>
<td></td>
</tr>
<tr>
<td>1991/05</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>1991/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>1991/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td></td>
<td>P.J. de Bruin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P. Hoogendijk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G. Malcolm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E. Voermonds</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J. v.d. Woude</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P.J. de Bruin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G. Malcolm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E. Voermonds</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J. van der Woude</td>
<td></td>
</tr>
<tr>
<td>1991/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
<tr>
<td>1991/16</td>
<td>A.J.J.M. Marcelis</td>
<td>An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.</td>
</tr>
<tr>
<td></td>
<td>P.M.E. de Bra</td>
<td></td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
</tbody>
</table>
91/18 Rik van Geldrop  
Transformational Query Solving, p. 35.

91/19 Erik Poll  
Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.

91/20 A.E. Eiben  
R.V. Schuwer

91/21 J. Coenen  
Assertional Data Reification Proofs: Survey and Perspective, p. 18.
W.-P. de Roever  
J. Zwiers

91/22 G. Wolf  

91/23 K.M. van Hee  
Z and high level Petri nets, p. 16.
L.J. Somers  
M. Voorhove

91/24 A.T.M. Aerts  
Formal semantics for BRM with examples, p. 25.
D. de Reus

91/25 P. Zhou  
A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.
J. Hooman  
R. Kuiper

91/26 P. de Bra  
The GOOD based hypertext reference model, p. 12.
G.J. Houben  
J. Paredaens

91/27 F. de Boer  
Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
C. Palamidessi

91/28 F. de Boer  
A compositional proof system for dynamic process creation, p. 24.

91/29 H. Ten Eikelder  
Correctness of Acceptor Schemes for Regular Languages, p. 31.
R. van Geldrop

91/30 J.C.M. Baeten  
An Algebra for Process Creation, p. 29.
F.W. Vaandrager

91/31 H. ten Eikelder  
Some algorithms to decide the equivalence of recursive types, p. 26.

91/32 P. Struik  

91/33 W. v.d. Aalst  
The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.

91/34 J. Coenen  
Specifying fault tolerant programs in deontic logic, p. 15.

91/35 F.S. de Boer  
Asynchronous communication in process algebra, p. 20.
J.W. Klop  
C. Palamidessi
<table>
<thead>
<tr>
<th>Issue</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwierts, W.-P. de Roever</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18.</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra, p. 45.</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
<td>Set theory and nominalisation, Part II, p. 22.</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
<td>The total order assumption, p. 10.</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
<td>A system at the cross-roads of functional and logic programming, p. 36.</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljée</td>
<td>Integrity checking in deductive databases; an exposition, p. 32.</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis, p. 20.</td>
</tr>
<tr>
<td>92/18</td>
<td>R. Nederpelt, F. Kamareddine</td>
<td>A unified approach to Type Theory through a refined lambda-calculus, p. 30.</td>
</tr>
<tr>
<td>92/20</td>
<td>F. Kamareddine</td>
<td>Are Types for Natural Language? P. 32.</td>
</tr>
<tr>
<td>92/21</td>
<td>F. Kamareddine</td>
<td>Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.</td>
</tr>
</tbody>
</table>
A useful lambda notation, p. 17.

Nominalization, Predication and Type Containment, p. 40.

Bottom-up Abstract Interpretation of Logic Programs, p. 33.

A Programming Logic for Fó, p. 15.

A modelling method using MOVIE and SimCon/ExSpect, p. 15.

A taxonomy of keyword pattern matching algorithms, p. 50.

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

A continuous version of the Prisoner's Dilemma, p. 17

Quicksort for linked lists, p. 8.

Deterministic and randomized local search, p. 78.

A congruence theorem for structured operational semantics with predicates, p. 18.

On the unavoidability of metastable behaviour, p. 29

Exercises in Multiprogramming, p. 97

A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.


<table>
<thead>
<tr>
<th>Paper Number</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooyman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19</td>
<td>G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p.21.</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27</td>
<td>T. Klok and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Klok, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
</tr>
<tr>
<td>93/33</td>
<td>L. Loyens and J. Moonen</td>
<td>ILIAS, a sequential language for parallel matrix computations, p. 20.</td>
</tr>
<tr>
<td>Code</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Real Time Process Algebra with Infinitesimals, p.39.</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Non Interleaving Process Algebra, p. 17.</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoef</td>
<td>A general conservative extension theorem in process algebra, p. 17.</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
<td>Temporal operators viewed as predicate transformers, p. 11.</td>
</tr>
</tbody>
</table>