Fundamenta Informaticae XX (2015) 1–33
DOI 10.3233/FI-2012-0000
IOS Press

Soundness of Timed-Arc Workflow Nets in Discrete and Continuous-Time Semantics

José Antonio Mateo
Dept. of Computer Science, University of Castilla-La Mancha, Spain
Dept. of Computer Science, Aalborg University, Denmark

Jiří Srba [C]
Dept. of Computer Science, Aalborg University, Denmark

Mathias Grund Sørensen
Department of Computer Science, Aalborg University, Denmark

Abstract. Analysis of workflow processes with quantitative aspects like timing is of interest in numerous time-critical applications. We suggest a workflow model based on timed-arc Petri nets and study the foundational problems of soundness and strong (time-bounded) soundness. We first consider the discrete-time semantics (integer delays) and explore the decidability of the soundness problems and show, among others, that soundness is decidable for monotonic workflow nets while reachability is undecidable. For general timed-arc workflow nets soundness and strong soundness become undecidable, though we can design efficient verification algorithms for the subclass of bounded nets. We also discuss the soundness problem in the continuous-time semantics (real-number delays) and show that for nets with nonstrict guards (where the reachability question coincides for both semantics) the soundness checking problem does not in general follow the approach for the discrete semantics and different zone-based techniques are needed for introducing its decidability in the bounded case. Finally, we demonstrate the usability of our theory on the case studies of a Brake System Control Unit used in aircraft certification, the MPEG2 encoding algorithm, and a blood transfusion workflow. The implementation of the algorithms is freely available as a part of the model checker TAPAAL (www.tapaal.net).

1. Introduction

Workflow nets [1, 2] were introduced by Wil van der Aalst as a formalism for modelling, analysis and verification of business workflow processes. The formalism is based on Petri nets abstracting away most

Address for correspondence: Dept. of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, Aalborg, Denmark
[C]Corresponding author

of the data while focusing on the possible flow in the system. Its intended use is in finding design errors like the presence of deadlocks, livelocks and other anomalies in workflow processes. Such correctness criteria can be described via the notion of *soundness* (see [3]) that requires the option to complete the workflow, guarantees proper termination and optionally also the absence of redundant tasks.

After the seminal work on workflow nets, researchers have invested much effort in defining new soundness criteria and/or improving the expressive power of the original model by adding new features and studying the related decidability and complexity questions (it is not in the scope of this paper to list all these works but we refer to [3] for a recent overview). In the present paper we consider a quantitative extension of workflow nets with timing features, allowing us to argue, among others, about the execution intervals of tasks, deadlines and urgent behaviour of workflow processes. Our workflow model is based on timed-arc Petri nets [7, 15] where tokens carry timing information and arcs are labelled with time intervals, restricting the available ages of tokens used for transition firing. Let us first informally introduce the model on our running example.

The timed-arc workflow net in Figure 1 describes a simple booking-payment workflow where a web-service provides a booking form followed by online payment. The workflow net consists of six places drawn as circles and nine transitions drawn as rectangles. Places can contain timed tokens, like the one of age 0 in the place in (the input place of the workflow). The collection of tokens present in the net form a marking. Places and transitions are connected by arcs such that arcs from places to transitions contain time intervals restricting the possible ages of tokens that can be consumed by transition firing. For simplicity we do not draw time intervals of the form $[0, \infty]$ as they do not restrict the ages of tokens in any way.

Normal flow of the net executes the transition *start* followed by the transitions *book* and *pay*. The whole booking-payment procedure cannot last for more than 10 minutes and the booking phase takes at least 2 minutes and must be finished within the first 5 minutes. The process can fail at any moment and the service allows for three additional attempts before it will terminate with failure.

In the initial marking of the net, the transition $start$ is enabled as it has a token in its input place. The transition is urgent (marked with an empty circle), so no time delay is allowed once it gets enabled. After the $start$ transition is fired, a new token of age 0 arrives into the place $booking$ (initiating the booking phase) and three new tokens of age 0 arrive into the place $attempts$ (in order to count the number of attempts we have before the service fails). The transition $fail1$ is not enabled as the place $attempts$, connected to $fail1$ via the so-called inhibitor arc, contains tokens, inhibiting $fail1$ from firing. The transition $book$ is not enabled either, as the token's age in the place $booking$ does not belong to the interval $[2, 5]$. However, after waiting for example 3 minutes, $book$ can fire. This consumes the token of age 3 from $booking$ and transports it to the place $payment$, preserving its age. This is signalled by the use of transport arcs that contain the diamond-shaped tips with index :1 (denoting how these arcs are paired).

At any moment, the booking-payment part of the workflow can be restarted by firing the transitions $restart1$ or $restart2$. This will bring the token back to the place $booking$, reset its age to 0, and consume one attempt from the place $attempts$. Once no more attempts are available and the age of the token in the place $booking$ or $payment$ reaches 5 resp. 10, we can fire the transition $fail1$ resp. $fail2$ and terminate the workflow by placing a token into the output place out. Note that the places $booking$ and $payment$ contain age invariants ≤ 5 resp. ≤ 10, meaning that the ages of tokens in these places should be at most 5 resp. 10. Hence if the service did not succeed within the given time bound, the workflow will necessarily fail. Finally, if the payment transition was executed within 10 minutes from the service initialization,

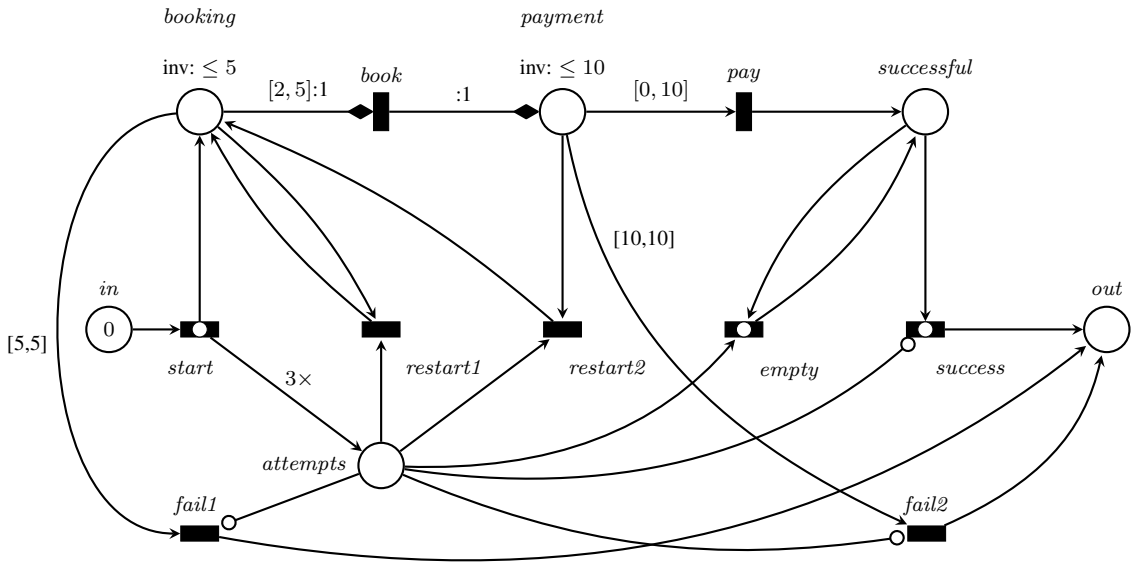


Figure 1: Booking-payment workflow with timing constraints

the transition $empty$ can now repeatedly remove any remaining tokens from the place $attempts$ and the transition $success$ terminates the whole workflow. As both the transitions $empty$ and $success$ are urgent, no further time delay is allowed in this termination phase.

We are concerned with the study of soundness and strong soundness, intuitively meaning that from any marking reachable from the initial one, it is always possible to reach a marking (in case of strong soundness additionally within a fixed amount of time), having just one token in the place out. Moreover, once a token appears in the place out, it is mandatory that the rest of the workflow net does not contain any remaining tokens. One can verify (either manually or using our tool mentioned later) that the workflow net of our running example is both sound and strongly sound.

Our contribution. We define a workflow theory based on timed-arc Petri nets, extend the notion of soundness from [1] to deal with timing features and introduce a new notion of strong soundness that guarantees time-bounded workflow termination. We study the decidability of soundness and strong soundness and conclude that even though they are in general undecidable, we can still in case of discrete-time semantics design efficient verification algorithms for two important subclasses: monotonic workflow nets (not using any inhibitor arcs, age invariants and urgent transitions) and for the subclass of bounded nets. This contrasts to the fact that for example the reachability question for monotonic workflow nets is already undecidable [27]. Moreover, our algorithms allow us to compute the minimum and maximum execution times of the workflow. As already mentioned, the theory of timed-arc workflow nets is developed for the discrete-time semantics but we also discuss the challenges connected to the continuous-time semantics, showing that even though in general there are nets sound in the discrete semantics but not in the continuous one, on the subclass of nets with no age invariants and no urgent transitions the notion of soundness is the same. We also argue that the discrete-time and continuous-time semantics both provide

the same minimum and maximum execution times. Last but not least, we implemented the algorithms given in this paper within the open-source tool TAPAAL [10, 17] and successfully demonstrate on a number of case studies the applicability of the theory in real-world scenarios.

Related work. Soundness for different extensions of Petri nets with e.g. inhibitor arcs, reset arcs and other features have been studied before, leading often to undecidability results (for a detailed overview see [3]). We shall now focus mainly on time extensions of Petri net workflow models. Ling and Schmidt [20] defined timed workflow nets in terms of Time Elementary Nets (TENs). These nets are 1-bounded by definition and a net is sound iff it is live and the initial marking is a home marking in a net that connects the output place of the workflow with the input one. Du and Jiang [13] suggested Logical Time Workflow Nets (LTWN) and their compositional semantics. Here liveness together with boundedness is a necessary and sufficient condition for soundness. Moreover, the soundness of a well-structured LTWN can be verified in polynomial time. Tiplea et al. [24] introduced a variant of timed workflow nets in terms of timed Petri nets and showed the decidability of soundness for the bounded subclass. In subsequent work [25, 26] they studied the decidability of soundness under different firing strategies. The papers listed above rely on the model of time Petri nets where timing information is associated to transitions and not to tokens like in our case. The two models are significantly different, in particular the number of timing parameters for time Petri nets is fixed, contrary to the dynamic creation of tokens with their private clocks in timed-arc Petri nets. We also see several modelling advantages of having ages associated to tokens as we can for example track the duration of sequentially composed tasks (via transport arcs) as demonstrated in our running example. We are not aware of other works developing a workflow theory and the corresponding notions of soundness based on timed-arc Petri nets. Finally, we implement the soundness checks within a user-friendly tool that permits easy GUI-based debugging of issues in workflows—something that is not that common for other workflow analysis tools (see [14] for more discussion).

Organization of the article. In Section 2 we define the syntax and semantics of extended timed-arc Petri nets with nonstrict time intervals, their monotonic subclass, we recall the extrapolation technique for discrete-time nets and we prove a monotonicity lemma, essential for several other results obtained in the article. In Section 3 we introduce timed-arc workflow nets, define the notion of soundness and prove that any sound monotonic net is bounded. Later in this section we show that soundness is in general undecidable while it is decidable (in the discrete semantics) for monotonic workflow nets and for bounded workflow nets. These results also imply that we can effectively compute the minimum execution time in these two cases. In Section 4 we suggest the notion of strong soundness that guarantees that a strongly sound workflow net terminates in a finite time and that the maximum execution time is well-defined. We conclude this section with a result showing that strong soundness is decidable for bounded timed-arc workflow nets in the discrete-time semantics. In Section 5 we then return to the continuous-time semantics, show that both discrete and continuous time semantics coincide up to reachability for our class of nets (the technical proof of this result is moved to an appendix), while this is not the case for soundness checking, unless the workflow nets do not contain any nontrivial age invariants and urgent transitions. In Section 6 we discuss the integration of our algorithms into the open-source tool TAPAAL and demonstrate the general applicability of the techniques on three case studies. Finally, in Section 7 we conclude the article and outline possible directions for future work.

2. Extended Timed-Arc Petri Nets

We shall start with the definition of extended timed-arc Petri nets and later on we recall some basic facts about the extrapolation technique applicable on this class of nets that allows us to prove the monotonicity lemma. The notion of timed-arc Petri net we define adopts the global time policy where all tokens age with the same speed.

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. Let $\mathbb{R}^{\geq 0}$ be the set of all nonnegative real numbers. A *timed transition system* (TTS) is a triple $(S, Act, \rightarrow)$ where S is the set of states, Act is the set of actions and $\rightarrow \subseteq S \times (Act \cup \mathbb{R}^{\geq 0}) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$. If $a \in Act$ then we call it a *switch transition*, if $a \in \mathbb{R}^{\geq 0}$ we call it a *delay transition*. We also define the set of *well-formed closed time intervals* as $\mathcal{I} \stackrel{\text{def}}{=} \{[a, b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$ and its subset $\mathcal{I}^{\text{inv}} \stackrel{\text{def}}{=} \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$ used in age invariants.

Definition 2.1. (Extended timed-Arc Petri Net)
An *extended timed-arc Petri net* (ETAPN) is a 9-tuple $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $P \cap T = \emptyset$,
- $T_{urg} \subseteq T$ is the set of *urgent transitions*,
- $IA \subseteq P \times T$ is a finite set of *input arcs*,
- $OA \subseteq T \times P$ is a finite set of *output arcs*,
- $g : IA \rightarrow \mathcal{I}$ is a *time constraint function* assigning guards to input arcs such that
  - if $(p, t) \in IA$ and $t \in T_{urg}$ then $g((p, t)) = [0, \infty]$,
- $w : IA \cup OA \rightarrow \mathbb{N}$ is a function assigning *weights* to input and output arcs,
- $Type : IA \cup OA \rightarrow \mathbf{Types}$ is a *type function* assigning a type to all arcs where $\mathbf{Types} = \{Normal, Inhib\} \cup \{Transport_j \mid j \in \mathbb{N}\}$ such that
  - if $Type(z) = Inhib$ then $z \in IA$ and $g(z) = [0, \infty]$,
  - if $Type((p, t)) = Transport_j$ for some $(p, t) \in IA$ then there is exactly one $(t, p') \in OA$ such that $Type((t, p')) = Transport_j$,
  - if $Type((t, p')) = Transport_j$ for some $(t, p') \in OA$ then there is exactly one $(p, t) \in IA$ such that $Type((p, t)) = Transport_j$,
  - if $Type((p, t)) = Transport_j = Type((t, p'))$ then $w((p, t)) = w((t, p'))$,
- $I : P \rightarrow \mathcal{I}^{inv}$ is a function assigning *age invariants* to places.

Remark 2.2. Note that for transport arcs we assume that they come in pairs (for each type $Transport_j$) and that their weights match. Also for inhibitor arcs and for input arcs to urgent transitions, we require that the guards are $[0, \infty]$. This restriction is important for some of the results presented in this paper and it also guarantees that we can use DBM-based algorithms in the tool TAPAAL [10].

The ETAPN model is not monotonic, meaning that adding more tokens to markings can disable time delays or transition firing. Therefore we define a subclass of ETAPN where the monotonicity-breaking features are not allowed. In the literature such nets are often considered as the standard timed-arc Petri net model [7, 15] but we add the prefix monotonic for clarity reasons.

Definition 2.3. (Monotonic timed-arc Petri net)
A *monotonic timed-arc Petri net* (MTAPN) is an extended timed-arc Petri net with no urgent transitions ($T_{urg} = \emptyset$), no age invariants ($I(p) = [0, \infty]$ for all $p \in P$) and no inhibitor arcs ($Type(z) \neq Inhib$ for all $z \in IA$).

Before we give the formal semantics of the model, let us fix some notation. Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be an ETAPN. We denote by ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in IA \cup OA,\ Type((y, x)) \neq Inhib\}$ the preset of a transition or a place x. Similarly, the postset $x^\bullet$ is defined as $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in (IA \cup OA)\}$. Let $\mathcal{B}(\mathbb{R}^{\geq 0})$ be the set of all finite multisets over $\mathbb{R}^{\geq 0}$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}^{\geq 0})$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$, in other words all tokens have to satisfy the age invariants. The set of all markings in a net N is denoted by $\mathcal{M}(N)$.

We write (p, x) to denote a token at a place p with the age $x \in \mathbb{R}^{\geq 0}$. Then $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ is a multiset representing a marking M with n tokens of ages x_i in places p_i. We define the size of a marking as $|M| = \sum_{p \in P} |M(p)|$ where $|M(p)|$ is the number of tokens located in the place p.

Definition 2.4. (Enabledness)
Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be an ETAPN. We say that a transition $t \in T$ is *enabled* in a marking M by the multisets of tokens $In = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((t,p'))}) \mid p' \in t^\bullet\}$ if

- for all input arcs except the inhibitor arcs, the tokens from In satisfy the age guards of the arcs, i.e.

$$\forall p \in {}^\bullet t.\ x_p^i \in g((p, t)) \text{ for } 1 \leq i \leq w((p, t))$$

- for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p is smaller than the weight of the arc, i.e.

$$\forall (p, t) \in IA.\, Type((p, t)) = Inhib \Rightarrow |M(p)| < w((p, t))$$

- for all input arcs and output arcs which constitute a transport arc, the age of the input token must be equal to the age of the output token and satisfy the invariant of the output place, i.e.

$$\begin{aligned} \forall (p, t) \in IA. \forall (t, p') \in OA.\, Type((p, t)) = Type((t, p')) = Transport_j \\ \Rightarrow \left(x_p^i = x_{p'}^i \wedge x_{p'}^i \in I(p')\right) \text{ for } 1 \leq i \leq w((p, t)) \end{aligned}$$

- for all normal output arcs, the age of the output token is 0, i.e.

$$\forall (t, p') \in OA.\, Type((t, p')) = Normal \Rightarrow x_{p'}^i = 0 \text{ for } 1 \leq i \leq w((t, p')).$$

A given ETAPN N defines a TTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \rightarrow)$ where states are the markings and the transitions are as follows.

- If $t \in T$ is enabled in a marking M by the multisets of tokens In and Out then t can *fire* and produce the marking $M' = (M \setminus In) \uplus Out$ where $\uplus$ is the multiset sum operator and $\setminus$ is the multiset difference operator; we write $M \xrightarrow{t} M'$ for this switch transition.
- A time *delay* $d \in \mathbb{R}^{\geq 0}$ is allowed in M if
  - $(x + d) \in I(p)$ for all $p \in P$ and all $x \in M(p)$, and
  - if $M \xrightarrow{t} M'$ for some $t \in T_{urg}$ then $d = 0$.

  By delaying d time units in M we reach the marking M' defined as $M'(p) = \{x + d \mid x \in M(p)\}$ for all $p \in P$; we write $M \xrightarrow{d} M'$ for this delay transition.

Remark 2.5. Note that time delay steps are allowed also in the situation when a certain token will become too old to fit into any time interval on the outgoing arcs. Such a time delay will create a so called *dead token* that cannot be used for firing any further transitions. We are not in any way removing such dead tokens, however, they can be prevented by the use of age invariants on places in order to guarantee a proper termination of the workflow net.

Let $\rightarrow \stackrel{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t} \cup \bigcup_{d \in \mathbb{R}^{\geq 0}} \xrightarrow{d}$. The set of all markings reachable from a given marking M is denoted by $[M\rangle \stackrel{\text{def}}{=} \{M' \mid M \rightarrow^* M'\}$. By $M \xrightarrow{d,t} M'$ we denote that there is a marking M'' such that $M \xrightarrow{d} M'' \xrightarrow{t} M'$.

A marking M is a *deadlock* if there are no $d \in \mathbb{R}^{\geq 0}$, $t \in T$ and marking M' such that $M \xrightarrow{d,t} M'$. A marking M is *divergent* (allows unbounded time delays) if for every $d \in \mathbb{R}^{\geq 0}$ we have $M \xrightarrow{d} M'$ for some M'.

The semantics defined above in terms of timed transition systems is called the *continuous-time semantics*. If we restrict the possible delay transitions to take values only from nonnegative integers and the markings to be of the form $M : P \longrightarrow \mathcal{B}(\mathbb{N}_0)$, we call it the *discrete-time semantics*. In the rest of the article, the presented results are valid both for the discrete and continuous-time semantics, unless explicitly stated otherwise.

2.1. Extrapolation of ETAPNs

In general, ETAPNs are infinite in two aspects. The number of tokens in reachable markings can be unbounded and even for bounded nets the ages of tokens can be arbitrarily large. We shall now recall a few results that allow us to make finite abstractions (in the discrete-time semantics) for bounded ETAPNs, i.e. for nets where the maximum number of tokens in any reachable marking is bounded by a constant. The theorems and lemmas in the rest of this section hold also for continuous-time semantics, however, the finiteness of the extrapolated state space is not guaranteed in this case.

Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be a given ETAPN. In [4] the authors provide an algorithm for computing a function $C_{max} : P \rightarrow (\mathbb{N}_0 \cup \{-1\})$ returning for each place $p \in P$ the maximum constant associated to this place, meaning that the ages of tokens in place p that are strictly greater than

$C_{max}(p)$ are irrelevant. The function $C_{max}(p)$ for a given place p is computed by essentially taking the maximum constant appearing in any outgoing arc from p and in the place invariant of p, where a special care has to be taken for places with outgoing transport arcs (details are discussed in [4]). In particular, places where $C_{max}(p) = -1$ are the so-called *untimed* places where the age of tokens is not relevant at all, implying that all the intervals on their outgoing arcs are $[0, \infty]$.

Let M be a marking of N. We split it into two markings $M_>$ and $M_\leq$ where $M_>(p) = \{x \in M(p) \mid x > C_{max}(p)\}$ and $M_\leq(p) = \{x \in M(p) \mid x \leq C_{max}(p)\}$ for all places $p \in P$. Clearly, $M = M_> \uplus M_\leq$.

We say that two markings M and M' in the net N are equivalent, written $M \equiv M'$, if $M_\leq = M'_\leq$ and for all $p \in P$ we have $|M_>(p)| = |M'_>(p)|$. In other words M and M' agree on the tokens with ages below the maximum constants and have the same number of tokens above the maximum constant.

The relation $\equiv$ is an equivalence relation and it is also a timed bisimulation (see e.g. [19]) where delays and transition firings on one side can be matched by exactly the same delays and transition firings on the other side and vice versa.

Theorem 2.6. ([4])
The relation $\equiv$ is a timed bisimulation.

We can now define canonical representatives for each equivalence class of $\equiv$.

Definition 2.7. (Cut)
Let M be a marking. We define its canonical marking $cut(M)$ by $cut(M)(p) = M_\leq(p) \uplus \{\underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_>(p)| \text{ times}}\}$.

Lemma 2.8. ([4])
Let M, M_1 and M_2 be markings. Then

(i) $M \equiv cut(M)$, and

(ii) $M_1 \equiv M_2$ if and only if $cut(M_1) = cut(M_2)$.

Let M and M' be two markings. We say that M' *covers* M, denoted by $M \sqsubseteq M'$, if $M(p) \subseteq M'(p)$ for all $p \in P$. We write $M \sqsubseteq_{cut} M'$ if $cut(M) \sqsubseteq cut(M')$.

For monotonic timed-arc Petri nets we will now show that adding more tokens to the net does not restrict its possible behaviour.

Lemma 2.9. (Monotonicity Lemma)
Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be an MTAPN and $M, M' \in \mathcal{M}(N)$ be two of its markings such that $M \sqsubseteq_{cut} M'$. Let $a \in \mathbb{R}^{\geq 0} \cup T$. If $M \xrightarrow{a} M_1$ then $M' \xrightarrow{a} M'_1$ such that $M_1 \sqsubseteq_{cut} M'_1$ and $|M'| - |M| = |M'_1| - |M_1|$.

Proof:
Let $M \xrightarrow{d} M_1$, resp. $M \xrightarrow{t} M_1$. As $M \equiv cut(M)$ by Lemma 2.8(i), we can by Theorem 2.6 conclude that also $cut(M) \xrightarrow{d} M_2$, resp. $cut(M) \xrightarrow{t} M_2$, such that $M_1 \equiv M_2$. Recall that $cut(M) \sqsubseteq cut(M')$ by the assumption of the lemma.

- Time delay case ($cut(M) \xrightarrow{d} M_2$). As the net does not contain any nontrivial age invariants (different from $[0, \infty)$) and there are no urgent transitions, we know that also $cut(M') \xrightarrow{d} M_3$ such that $M_2 \sqsubseteq M_3$ as time delay preserves the $\sqsubseteq$-relation.

- Transition firing case ($cut(M) \xrightarrow{t} M_2$). As the net does not have any inhibitor arcs, we can see that also $cut(M') \xrightarrow{t} M_3$ by consuming exactly the same tokens in $cut(M')$ as we did in $cut(M)$. Clearly, $M_2 \sqsubseteq M_3$.

Because $cut(M') \equiv M'$ due to Lemma 2.8(i), we know by Theorem 2.6 that $M' \xrightarrow{d} M'_1$, resp. $M' \xrightarrow{t} M'_1$, such that $M_3 \equiv M'_1$. Hence $M_1 \equiv M_2 \sqsubseteq M_3 \equiv M'_1$. By Lemma 2.8(ii) we get $cut(M_1) = cut(M_2)$ and $cut(M_3) = cut(M'_1)$. Observe now a simple fact that $M_2 \sqsubseteq M_3$ implies that $cut(M_2) \sqsubseteq cut(M_3)$. This all together implies that $cut(M_1) = cut(M_2) \sqsubseteq cut(M_3) = cut(M'_1)$ which is another way of saying that $M_1 \sqsubseteq_{cut} M'_1$ as required by the lemma. As time delays do not change the number of tokens in M and M' and transition firing adds or removes an equal number of tokens from both M and M', we can also conclude that $|M'| - |M| = |M'_1| - |M_1|$. □

3. Timed-Arc Workflow Nets

We shall now formally define timed-arc workflow nets, introduce the soundness notion for this class of nets and answer the questions about the decidability of soundness.

Timed-arc workflow nets are defined similarly as untimed workflow nets [1]. Every workflow net has a unique input place and a unique output place. After initializing such a net by placing a token into the input place, it should be guaranteed that any possible workflow execution can be always extended such that the workflow terminates with just one token in the output place (also known as the soundness property).

Definition 3.1. (Extended timed-arc workflow net)
An $ETAPN\ N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ is called an *Extended Timed-Arc WorkFlow Net* (ETAWFN) if

- there exists a unique place $in \in P$ such that ${}^\bullet in = \emptyset$ and $in^\bullet \neq \emptyset$,
- there exists a unique place $out \in P$ such that $out^\bullet = \emptyset$ and ${}^\bullet out \neq \emptyset$,
- for all $p \in P \setminus \{in, out\}$ we have ${}^\bullet p \neq \emptyset$ and $p^\bullet \neq \emptyset$, and
- for all $t \in T$ we have ${}^\bullet t \neq \emptyset$.

Remark 3.2. Notice that the conditions ${}^\bullet in = \emptyset$ and ${}^\bullet out \neq \emptyset$ necessarily imply that $in \neq out$. Moreover, we allow the postset of a transition to be empty ($t^\bullet = \emptyset$). This is just a technical detail and an equivalent workflow net where all transitions satisfy $t^\bullet \neq \emptyset$ can be constructed by introducing a new place p_{new} so that any outgoing transition from the start place in puts a token into p_{new} and every incoming transition to the final place out consumes the token from p_{new}. Now for any transition t with $t^\bullet = \emptyset$ we add the pair of arcs (p_{new}, t) and (t, p_{new}) without influencing the soundness of the net (both nets provide the same behaviour until the output place is marked).

Decidability of soundness crucially depends on the modelling features allowed in the net. Hence we define a subclass of so-called monotonic workflow nets.

Definition 3.3. (Monotonic timed-arc workflow net)
A monotonic timed-arc workflow net (MTAWFN) is an ETAWFN where the underlying net is monotonic, i.e. with no urgent transitions, no age invariants and no inhibitor arcs.

The marking $M_{in} = \{(in, 0)\}$ of a timed-arc workflow net is called *initial*. A marking M is *final* if $|M(out)| = 1$ and for all $p \in P \setminus \{out\}$ we have $|M(p)| = 0$, i.e. it contains just one token in the place out. There may be several final markings with different ages of the token in the place out.

We now provide the formal definition of soundness that formulates the standard requirement on proper termination of workflow nets [2, 3].

Definition 3.4. (Soundness of timed-arc workflow nets)
An (extended or monotonic) timed-arc workflow net $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ is sound if for any marking $M \in [M_{in}\rangle$ reachable from the initial marking M_{in}:

a) there exists some final marking M_{out} such that $M_{out} \in [M\rangle$, and

b) if $|M(out)| \geq 1$ then M is a final marking.

A workflow is sound if once it is initiated by placing a token of age 0 in the place in, it has always the possibility to terminate by moving the token to the place out (option to complete) and moreover it is guaranteed that the rest of the workflow net is free of any remaining tokens as soon as the place out is marked (proper completion). We now define a subclass of bounded workflow nets.

Definition 3.5. (Boundedness)
A timed-arc workflow net N is *k-bounded* for some $k \in \mathbb{N}_0$ if any marking M reachable from the initial marking M_{in} satisfies $|M| \leq k$. A net is *bounded* if it is k-bounded for some k.

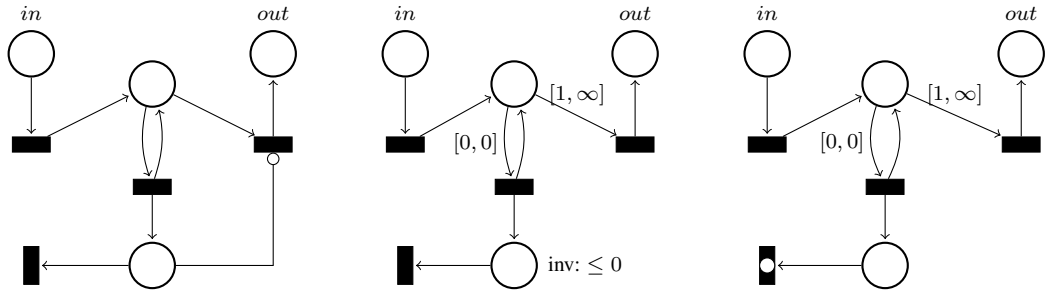
A classical result states that any untimed sound net is bounded [1]. This is not in general the case for extended timed-arc workflow nets because inhibitor arcs, age invariants and urgent transitions can model sound but unbounded workflows as demonstrated in Figure 2. In all three nets the place at the bottom can contain an arbitrary large number of tokens, however, before the nets can terminate (by marking the place out), these tokens must be removed.

Nevertheless, we recover the boundedness result for the subclass of monotonic timed-arc workflow nets.

Theorem 3.6. Let N an MTAWFN. If N is sound then N is bounded.

Proof:
By contradiction assume that N is a sound and unbounded MTAWFN. Clearly, if N is unbounded in the continuous-time semantics, it is unbounded also in the discrete-time semantics (follows directly from Theorem 5.1 proved in the appendix). Let us so assume, for the rest of the proof, that N is unbounded in the discrete-time semantics. Let M_{in} be the initial workflow marking. Now we can argue that there must exist two reachable markings $M, M' \in [M_{in}\rangle$ such that

i) $M \sqsubseteq_{cut} M'$, and



(a) Inhibitor arcs (b) Age invariants (c) Urgent transitions

Figure 2: Sound and unbounded extended timed-arc workflow nets

ii) $|M| < |M'|$.

This follows from the fact that $M \sqsubseteq_{cut} M'$ iff $cut(M) \sqsubseteq cut(M')$ and from Definition 2.7 where the cut function is given such that each token is placed into one of the finitely many places, say p, and its age is bounded by $C_{max}(p) + 1$. Thanks to Dickson's Lemma [12], saying that every set of n-tuples of natural numbers has only finitely many minimal elements, we are guaranteed that conditions i) and ii) are satisfied for some reachable markings M and M'.

Since N is a sound workflow net, we now use condition a) of Definition 3.4, implying that from M we reach some final marking M_{out}. Assume that this is achieved w.l.o.g. by the following sequence of transitions:

$$M \xrightarrow{d_1} M_1 \xrightarrow{t_1} M_2 \xrightarrow{d_2} M_3 \xrightarrow{t_2} M_4 \ldots \xrightarrow{t_n} M_{out} \ .$$

We know that $M \sqsubseteq_{cut} M'$ and hence by repeatedly applying Lemma 2.9 also

$$M' \xrightarrow{d_1} M'_1 \xrightarrow{t_1} M'_2 \xrightarrow{d_2} M'_3 \xrightarrow{t_2} M'_4 \ldots \xrightarrow{t_n} M'_{out}$$

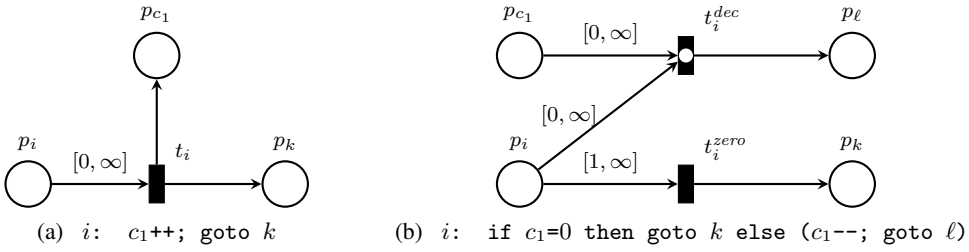
such that at the end $M_{out} \sqsubseteq_{cut} M'_{out}$. The facts that $M_{out} \sqsubseteq_{cut} M'_{out}$ and M_{out} is a final marking imply that $|M'_{out}(out)| \geq 1$. By a repeated application of Lemma 2.9 we also get $|M'|-|M| = |M'_{out}|-|M_{out}|$. By condition ii) of this lemma we know that $|M| < |M'|$, this implies that also $|M_{out}| < |M'_{out}|$. However, now the place out in M'_{out} is marked and there is at least one more token somewhere else in the marking M'_{out}. This contradicts condition b) of Definition 3.4. □

Next we show that soundness for extended timed-arc workflow nets is undecidable. The result has been known for the extension with inhibitor arcs [3], we prove it (by reduction from two counter Minsky machines) also for urgent transitions and age invariants.

Theorem 3.7. Soundness is undecidable for extended timed-arc workflow nets. This is the case also for MTAWFNs that contain additionally only inhibitor arcs, age invariants or urgent transitions but not necessarily all of them together.

Proof:
The proofs are by reduction from the reachability problem for a Minsky machine. A *Minsky machine* with two nonnegative counters c_1 and c_2 is a sequence of labelled instructions

$$1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots, n : \mathsf{inst}_n$$



(a) i: c_1++; goto k (b) i: if c_1=0 then goto k else (c_1--; goto ℓ)

Figure 3: Simulation of (Inc) and (Dec) instructions with urgent transitions

where $\mathsf{inst}_n = \mathsf{HALT}$ and each inst_i, $1 \leq i < n$, is of one of the following forms

- (Inc) i: c_j++; goto k
- (Dec) i: if c_j=0 then goto k else (c_j--; goto ℓ)

for $j \in \{1, 2\}$ and $1 \leq k, \ell \leq n$.

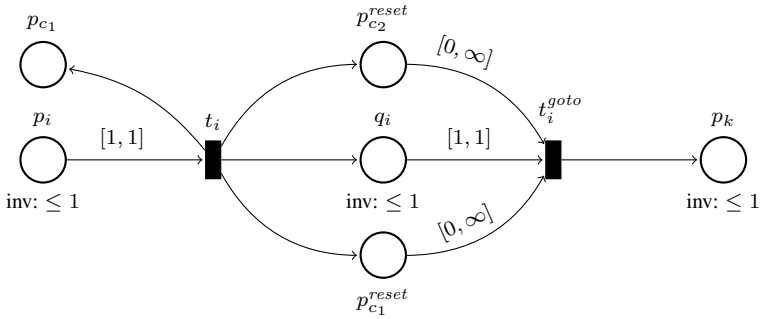
Instructions of type (Inc) are called *increment* instructions and those of type (Dec) are called *test and decrement* instructions. A configuration is a triple (i, v_1, v_2) where i is the current instruction and v_1 and v_2 are the values of the counters c_1 and c_2, respectively. A computation step between configurations is defined in the natural way. If starting from the initial configuration $(1, 0, 0)$ the machine eventually reaches the instruction HALT then we say it *halts*, otherwise it *loops*. The problem whether a given Minsky machine halts is undecidable [21]. W.l.o.g. we assume that the machine halts only when both counters are empty (instead of jumping directly to the instruction HALT, we first enter newly added instructions that will empty both counters and after that halt).

We shall now reduce reachability in Minsky machines into the soundness problem on ETAWFN. Counters c_1 and c_2 will be simulated by two places p_{c_1} and p_{c_2} such that the number of tokens in those places represent the values of the counters. For every instruction label i, $1 \leq i \leq n$, we add a new control place p_i. At any moment at most one of the p_i places will be marked by a token, representing the instruction to be executed in the next step.

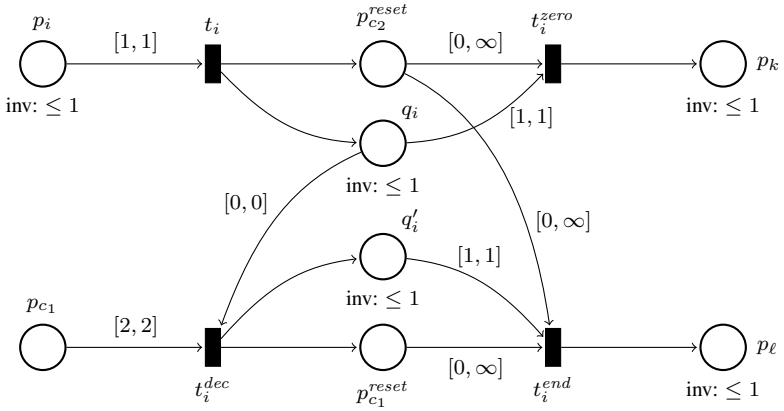
If we allow urgent transitions, we can create for any given Minsky machine a workflow net constructed according to the patterns given in Figure 3 (we only show the encoding of the instructions that manipulate the first counter; the encoding for the second counter is completely analogous). We also postulate that the input place is $in = p_1$ and the output place is $out = p_n$. Now, given the initial marking with one token in p_1, the net will faithfully simulate the (deterministic) computation of the Minsky machine. This is clear for the increment instruction as the control token moves from p_i to p_k and the number of tokens in p_{c_1} is increased by one. For the test and decrement instruction, if p_{c_1} contains at least one token then the transition t_i^{dec} will be fired with no delay (the transition is urgent), decreasing the counter by one and moving the control token to p_ℓ as required. Only if the counter c_1 is empty (there are no tokens in p_{c_1}), we are allowed to delay one time unit and fire the transition t_i^{zero} such that the control token is moved to p_k. Hence the test and decrement instruction is also faithfully simulated and there is no possibility of any deadlock situation, meaning that either t_i^{dec} or t_i^{zero} can always fire. It is now an easy observation that the workflow net is sound if and only if the Minsky machine halts.

A similar reduction can be shown for workflow nets that use inhibitor arcs instead of urgent transitions. In order to guarantee that the simulation is fair, we simply add an inhibitor arc from the counter

(a) Simulation of the counters c_1 and c_2



(b) i: c_1`++; goto` k



(c) i: `if` c_1=0 `then goto` k `else (`c_1`--; goto` ℓ`).`

Figure 4: Simulation of a Minsky machine by a workflow net with invariants

place p_{c_1} to the transition t_i^{zero} and remove all the timing information. The untimed workflow net is now sound if and only if the Minsky machine halts.

The reduction for workflow nets that contain only age invariants is more complicated. The reduction idea is based on [18], however, it had to be nontrivially modified in order to avoid the large number of possible deadlocks introduced in the reduction.

The counters are now modelled by two places that contain age invariants ensuring that no tokens can get older than 2, see Figure 4a. As before the input place is $in = p_1$ and the output place is $out = p_n$. The intuition is that before and after the simulation of any instruction, all tokens in the places p_{c_1} and p_{c_2} are exclusively of age 1.

Let us now observe that this invariant is preserved after simulating the increment instruction (Figure 4b). Assume that all tokens in the counter places are of age 1 and that the place p_i contains one token

of age 0. Before t_i can be fired, one time unit must pass and this guarantees that all tokens in the counter places will become of age 2. After firing of t_i, we also add one token of age 0 to p_{c_1} and moreover, a token of age 0 in the place q_i is created. Before we can proceed and delay one time unit and then fire t_i^{goto}, we must fire the transitions $t_{c_1}^{reset}$ and $t_{c_2}^{reset}$ once for every token of age 2 in the counter places in order to reset them all to the age 0, otherwise the age invariant ≤ 2 in the token places disables the delay of one time unit. Clearly, after the transition t_i^{goto} is fired, all counter tokens are again of age 1 (including the one added to p_{c_1}) and we argued for a faithful simulation of the increment instruction.

Let us consider now the test and decrement instruction simulated by the net in Figure 4c. Again, let us assume that all counter tokens are of age 1 and that there is one token of age 0 in p_i. First, we wait one time unit and then fire t_i, meaning again that all counter tokens are of age 2. Now all tokens in the place p_{c_2} can be reset to 0 and if p_{c_1} does not contain any tokens, we can wait one time unit and fire t_i^{zero}, implying that we place a token to p_k as expected and all counter tokens are again of age 1. If on the other hand p_{c_1} contains some tokens (all of age 2 as we already mentioned), we must without any delay fire t_i^{dec} consuming one token from p_{c_1} while marking the places q_i' and $p_{c_1}^{reset}$, allowing now all counter tokens to be reset to 0. After this we can wait one time unit and fire the transition t_i^{end}, while continuing with the execution of the instruction with label ℓ. All tokens in the counter places are now again of age 1. Notice that these two scenarios are deterministically determined by the presence or absence of tokens in p_{c_1} and that there are no deadlock situations possible during the simulation.

As a result we can see that the net is sound if and only if the Minsky machine halts. This completes the undecidability proof also for the situation where we use only age invariants. The arguments in the proof clearly hold both in discrete-time and continuous-time semantics. □

We now prove decidability of soundness in the discrete-time semantics for workflow nets without any inhibitor arcs, age invariants and urgency. This contrasts to undecidability of reachability for this subclass [27]. Algorithm 1 shows how to efficiently check soundness on this subclass and on the subclass of bounded nets. We implicitly consider only discrete-time semantics in the rest of the exposition of this algorithm. The algorithm first performs a standard forward search on the cut-extrapolated marking graph by using the sets $Waiting$ and $Reached$ for storing the discovered resp. already explored cut markings, while at the same time computing the shortest path from the initial marking to the reachable cut-markings in the net. The algorithm will terminate at line 7 if the k bound for a nonmonotonic input net N is exceeded or at line 18 if we find a marking covering another already discovered marking in case the input net N is monotonic (note that monotonicity is a simple syntactic property of the net). In case a marking with a token in the output place is discovered, we report a problem if the marking is not a final marking; otherwise we store the final marking into the set $Final$ (line 12). In case a deadlock non-final marking is discovered, we immediately return false at line 16.

If the first phase of the algorithm successfully terminates, we initiate in the second while-loop a backward search from the set $Final$, checking that all reachable states have a path leading to some final marking. If this is the case, we return at line 27 that the net is sound together with its minimum execution time.

Formally, the correctness of the algorithm is introduced by the following series of lemmas where we refer to phase one (lines 2-20) and phase two (lines 21-29) of Algorithm 1. First, we introduce some simple loop invariants.

Lemma 3.8. (Loop Invariants)
The while-loop at lines 4-19 of Algorithm 1 satisfies the following loop-invariants:

Algorithm 1: Soundness checking for timed-arc workflow nets in discrete-time semantics

Input : A positive integer bound k and an MTAWFN or ETAWFN $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ where $in, out \in P$.

Output: "Not k-bounded" if the workflow net is not monotonic and not k-bounded; else "true" together with the minimum execution time if N is sound or "false" if N is not sound.

begin
  A marking M has an (initially empty) set of its parents $M.parents$ and a minimum execution time $M.min$ (initially ∞); $M_{in} := \{(in, 0)\}$;
  $Waiting := \{M_{in}\}$; $M_{in}.min = 0$; $Reached := Waiting$; $Final := \emptyset$;
  while $Waiting \neq \emptyset$ **do**
    Remove some marking M from $Waiting$ with the smallest $M.min$;
    foreach M' *s.t.* $M \xrightarrow{1} M'$ *or* $M \xrightarrow{t} M'$ *for some* $t \in T$ **do**
      if N is not monotonic and $|M'| > k$ **then return** "Not k-bounded";
      $M'_c := cut(M')$; $M'_c.parents := M'_c.parents \cup \{M\}$;
      if $M \xrightarrow{1} M'$ **then** $M'_c.min := \texttt{MIN}(M'_c.min, M.min + 1)$;
      else $M'_c.min = \texttt{MIN}(M'_c.min, M.min)$;
      if $|M'_c(out)| \geq 1$ **then**
        if M'_c is a final marking **then** $Final := Final \cup \{M'_c\}$;
        else return false;
      else
        if $M'_c \notin Reached$ **then**
          if M'_c is a deadlock **then return** false;
          if N *is monotonic and* $\exists M'' \in Reached.\ M'' \sqsubseteq_{cut} M'_c$ **then**
            return false;
          $Reached := Reached \cup \{M'_c\}$; $Waiting := Waiting \cup \{M'_c\}$;
  $Waiting := Final$;
  while $Waiting \neq \emptyset$ **do**
    Remove some marking M from $Waiting$;
    $Waiting := Waiting \cup (M.parents \cap Reached)$;
    $Reached := Reached \setminus M.parents$;
  if $Reached = \emptyset$ **then**
    $time := \infty$; **foreach** $M \in Final$ **do** $time = \texttt{MIN}(time, M.min)$;
    return true and $time$;
  else
    return false;

a) $Waiting \subseteq Reached$,

b) for any marking $M'_c \in Reached \cup Final$, there exists a computation of the net $M_{in} \rightarrow^* M'$ such that $M'_c = cut(M')$ and the accumulated delay on the computation $M_{in} \rightarrow^* M'$ is equal to $M'_c.min$, and

c) for any marking $M'_c \in \mathit{Reached} \cup \mathit{Final}$ and any $M \in M'_c.\mathit{parents}$ there is a transition $M \to M'$ such that $M'_c = \mathit{cut}(M')$.

Proof:
The claims a), b) and c) of the invariant are trivially satisfied the first time the while-loop is entered. Let us assume the invariant holds before the execution of the body of the while-loop.

The claim a) is easily proved, as markings are only added to *Waiting* at line 19 of the loop body, and the same marking is also added to *Reached* at line 19 and no markings are removed from $\mathit{Reached}$ in the body of the loop.

For claim b) notice that at line 5 we remove a marking M from $\mathit{Waiting}$ and for any successor M' of M, the marking M is added to the set of parents of $M'_c = \mathit{cut}(M')$ (line 8). Due to the first invariant, M was already in $\mathit{Reached}$ in the previous iteration of the loop. Hence there exists a computation $M_{in} \to^* M_1$ such that $M = \mathit{cut}(M_1)$ and the accumulated delay of this computation is $M.\mathit{min}$. Because $M \to M'$ (line 6) then also $M_1 \to M_2$ such that $M'_c = \mathit{cut}(M_2)$ (line 8). Hence $M_{in} \to^* M_2$ and $M'_c = \mathit{cut}(M_2)$ as required. The accumulated delay is updated according to the type of the transition $M_1 \to M_2$ at lines 9 and 10. If the value $M'_c.\mathit{min}$ changed after this update then the computation $M_{in} \to^* M_2$ achieves this accumulated delay, otherwise the minimum delay was achieved in some previous run of the body of the while-loop and it is hence valid due to the loop invariant.

Finally, the claim c) follows from the fact that markings to $M'_c.\mathit{parents}$ are only added at line 8 and such markings clearly satisfy the invariant claim. □

Lemma 3.9. (End of Phase One)
After the first while loop (lines 4-19) of Algorithm 1 is finished, we have at line 20 that $\mathit{Reached} \cup \mathit{Final} = \{\mathit{cut}(M') \mid M_{in} \to^* M'\}$. Moreover, if $M_{in} \to^* M'$ then the accumulated delay of this computation is greater or equal to $\mathit{cut}(M').\mathit{min}$ and there is at least one such computation ending in M' where the accumulated delay is equal to $\mathit{cut}(M').\mathit{min}$.

Proof:
Let us first argue for the fact $\mathit{Reached} \cup \mathit{Final} = \{\mathit{cut}(M') \mid M_{in} \to^* M'\}$. The inclusion "$\subseteq$" follows directly from claim b) of Lemma 3.8. The inclusion "$\supseteq$" follows from the fact that we search all possible successors of M_{in}; we do not provide further arguments as this is a standard graph searching algorithm. The optimality of the computation of the minimum delay is guaranteed because we explore the graph from the nodes with the smallest min value (line 5) and this is (up to the cut-equivalence) essentially the Dijkstra's algorithm for shortest path in a graph. □

Lemma 3.10. (Not k-bounded)
Let N be an MTAWFN or ETAWFN and $k > 0$. If Algorithm 1 returns "Not k-bounded" then N is not k-bounded.

Proof:
The algorithm returns "Not k-bounded" only at line 7, provided that the net is not monotonic and there is a marking M' reachable in one step from $M \in \mathit{Waiting}$ such that $|M'| > k$. By claim b) of Lemma 3.8, we know that there is a computation from M_{in} to M_1 such that $M = \mathit{cut}(M_1)$ and we also know that $M \to M'$ (line 6). By Lemma 2.8 and Theorem 2.6 also $M_1 \to M_2$ such that $M' = \mathit{cut}(M_2)$ and this means that M_2 is reachable from M_{in} and at the same time $|M_2| > k$ as cut preserves the number of tokens in a marking. Hence if the algorithm returns "Not k-bounded" then the net is not k-bounded. □

Lemma 3.11. (Return value false)
Let N be an MTAWFN or ETAWFN and $k > 0$. If Algorithm 1 returns false then N is not sound.

Proof:
By a simple analysis of the four places where the algorithm returns false (lines 13, 16, 18 and 29). In each of these four cases, we demonstrate that the net cannot be sound.

- Starting with line 13, the algorithm returns false if it finds a marking M' reachable from the initial marking of N such that $M'_c = cut(M')$ and M'_c has at least one token in the output place out while it is not a final marking (contains some additional tokens in other places too). Clearly, it is possible to reach from M_{in} this marking (up to cut-equivalence) by claim b) of Lemma 3.8 and it breaks condition b) of the definition of soundness (Definition 3.4). Therefore the net is not sound.

- In line 16, the algorithm returns false if we have found a marking M' reachable from M_{in} (here we use implicitly claim b) of Lemma 3.8) such that $M'_c = cut(M')$ and M'_c is a deadlock. As M'_c is a deadlock, M' is also a deadlock (by Theorem 2.6). The marking M' is not a final marking and hence breaks condition a) of Definition 3.4. Therefore the net is not sound.

- Let us continue with line 18. Here, we have reached a marking M' from the initial marking of the monotonic net N such that $M'_c = cut(M')$ and there exists a marking $M'' \in Reached$ such that $M'' \sqsubseteq_{cut} M'_c$ and $|M''| < |M'_c|$. Note that $|M''| = |M'_c|$ cannot happen since it is avoided due to the if-condition at line 15. Let us suppose that N is sound. We know due to condition a) of Definition 3.4 that there is path from M'' to the final marking of N (M_{out}). However, by applying Lemma 2.9 repeatedly (again we reason up to the cut-equivalence), we can follow the same path from M'_c to a marking M'_{out} such that $|M'_c| - |M''| = |M'_{out}| - |M_{out}|$. As $|M''| < |M'_c|$, we also know that $|M_{out}| < |M'_{out}|$, which breaks condition b) of Definition 3.4, and contradicts that N is sound.

- Finally, we take a look to line 29. Let us recall that $Reached$ contains the set of cut markings of the reachable markings that are not final (Lemma 3.9). Moreover, in the second while-loop, we remove from $Reached$ the parents of all the cut markings in $Waiting$ until $Waiting$ is empty, running a backward search from the set $Final$. Thus, if $Reached$ is not empty after this backward search terminates, it means that there is $M'_c \in Reached$ such that $M'_c = cut(M')$ for some reachable marking M' from M_{in} and there is no path from M' to any final marking. This breaks condition a) of Definition 3.4 and the net is not sound.

□

Lemma 3.12. (Return value true)
Let N be an MTAWFN or ETAWFN and $k > 0$. If Algorithm 1 returns true then N is sound.

Proof:
Assume that the algorithm returned true at line 27 and we shall argue that N satisfies conditions a) and b) of Definition 3.4. Condition b) is straightforward as by Lemma 3.9 we know that $Reached \cup Final$ is the set of cut-markings of all the reachable markings of N and if some of them marks the place out then this must be a final marking, otherwise the algorithm would return false at line 13. For condition a) we realize that the set $Final$ contains all final markings reachable from M_{in} and in the second phase of the

algorithm we run a backward search and remove from the reachable state-space all markings that have a computation leading to one of the final markings. We return true only if $Reached$ is empty, meaning that all reachable markings have a computation to some final marking. This corresponds to condition a). □

Lemma 3.13. (Termination)
Algorithm 1 terminates.

Proof:
There is one while-loop in each phase of the algorithm. The loop in the first phase is executed as long as $Waiting$ is not empty. We notice that initially $Waiting$ and $Reached$ are initialized to the same value. For each iteration of the loop, we remove a marking from $Waiting$ and newly discovered cut markings are always added to both $Waiting$ and $Reached$ (line 19) but only if they are not already in $Reached$ (line 15). Markings are never removed from the set $Reached$ and each canonical marking can appear in $Waiting$ at most once.

For nonmonotonic nets, only canonical markings with at most k tokens are added (line 7) and therefore the set of canonical markings is finite. Thus the algorithm will terminate as the set $Waiting$ eventually becomes empty.

For monotonic nets, the net could be unbounded and, therefore, the set $Reached$ would grow above any bound. In this case, we know by similar arguments like in the proof of Theorem 3.6 that there must exist M, $M' \in [M_{in}\rangle$ such that $M \sqsubseteq_{cut} M'$ and $|M| < |M'|$. Moreover, we know that at some point there will be discovered such a marking M' in the situation where M is already in the set $Reached$. The algorithm will detect such a situation at line 17 and terminate.

For the loop in the second phase, notice that $Waiting = Final$. For each iteration, a marking is removed from $Waiting$ and the intersection of the set of parents of the marking M, ***M.parents*** and the set $Reached$ is then added to $Waiting$. In addition, the set ***M.parents*** is removed from $Reached$. Thus, any marking can only be added to $Waiting$ once, and as the set $Reached$ is finite when entering the loop and a marking is removed from $Waiting$ in each iteration, eventually $Waiting = \emptyset$ and the algorithm terminates. □

We can so conclude with the main result claiming decidability of soundness for workflow nets in the discrete-time semantics that are either bounded or monotonic.

Theorem 3.14. Soundness in the discrete-time semantics is decidable for monotonic timed-arc workflow nets and for bounded extended timed-arc workflow nets.

Given a sound ETAWFN $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$, we can reason about its execution times (the accumulated time that is used to move a token from the place in into the place out). Let M_{in} be the initial marking of N and $\mathcal{F}(N)$ be the set of all final markings of N. Let $\mathcal{T}(N)$ be the set of all execution times by which we can get from the initial marking to some final marking. Formally,

$$\mathcal{T}(N) \stackrel{\text{def}}{=} \left\{ \sum_{i=0}^{n-1} d_i \mid M_{in} = M_0 \stackrel{d_0,t_0}{\rightarrow} M_1 \stackrel{d_1,t_1}{\rightarrow} M_2 \stackrel{d_2,t_2}{\rightarrow} \cdots \stackrel{d_{n-1},t_{n-1}}{\rightarrow} M_n \in \mathcal{F}(N) \right\}.$$

The set $\mathcal{T}(N)$ is nonempty for any sound net N and the *minimum execution time* of N, defined by $\min \mathcal{T}(N)$, is computable by Algorithm 1 (correctness follows from Lemma 3.9).

Theorem 3.15. Let N be a sound MTAWFN or a sound and bounded ETAWFN. The minimum execution time of N is computable.

In fact, the result holds also in the continuous-time semantics as argued in Section 5. Notice also that the set $\mathcal{T}(N)$ can be infinite for general timed-arc workflow nets, meaning that the *maximum execution time* of N, given by $\max \mathcal{T}(N)$, is not always well defined. This issue is discussed in the next section.

4. Strong Soundness

Soundness ensures the possibility of correct termination in a workflow net, however, it does not give any guarantee on a timely termination of the workflow. The notion of strong soundness introduced in this section will provide us with such guarantees.

In untimed workflows, infinite behaviour can be used to model for instance repeated queries for further information until a decision can be taken. In a time setting, we usually have a deadline such that if the information is not acquired within the deadline, alternative behaviour in the net is executed (compensation). Consider the workflow nets presented in Figure 5. They represent a simple customer-complaint workflow where, before a decision is made, the customer can be repeatedly requested to provide additional information. The net in Figure 5a is sound but there is no time guarantee by when the decision is reached. On the other hand, the net in Figure 5b introduces additional timing, requiring that the process starts within 5 days and the request/provide loop takes no more than 14 days, after which a decision is made. The use of transport arcs enables us to measure the accumulated time since the place $pending$ was entered the first time. It is clear that the workflow only permits behaviours up to 19 days in total. In fact, the net enables infinite executions never reaching any final marking, however, this only happens within a bounded time interval (producing a so-called *Zeno run*) and we postulate that such a scenario is unrealistic in a real-world workflow execution. After disregarding the Zeno runs, we are guaranteed that the workflow finishes within 19 days and we can call it *strongly sound*.

A formal definition of strong soundness follows. Recall that a marking is divergent if it allows for arbitrarily long delays.

Definition 4.1. (Strong soundness of ETAWFN)
An extended timed-arc workflow net N is *strongly sound* if

a) N is sound,

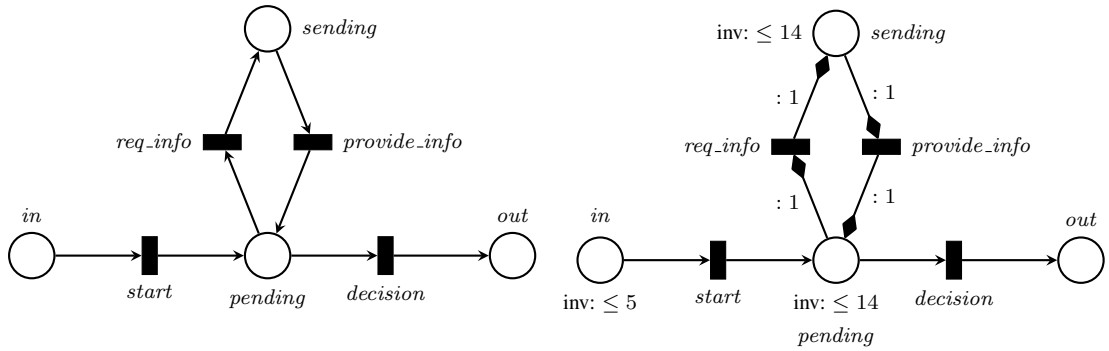
b) every divergent marking reachable in N is a final marking, and

c) there is no infinite computation starting from the initial marking $\{(in, 0)\} = M_0 \overset{d_0,t_0}{\to} M_1 \overset{d_1,t_1}{\to} M_2 \overset{d_2,t_2}{\to} \cdots$ where $\sum_{i \in \mathbb{N}_0} d_i = \infty$.

Notice that no monotonic net (even extended with inhibitor arcs) is strongly sound as all its reachable markings are always divergent.

As expected, strong soundness for general (unbounded) extended workflow nets is undecidable.

Theorem 4.2. Strong soundness of ETAWFN is undecidable.



(a) Sound but not strongly sound workflow

(b) Strongly sound workflow

Figure 5: Fragment of customer complaint workflow ($[0, \infty]$ guards are omitted)

Proof:
Similarly as in Theorem 3.7, we reduce the reachability problem for two-counter Minsky machines into checking strong soundness. We can use the same construction as in Figure 4 where each place p_i contains the age invariant ≤ 1 and hence there are no divergent markings. At the same time, before executing any instruction we have to wait exactly one time unit, hence there is no infinite computation of the workflow net that happens during a fixed time bound. As a result, the workflow net constructed in Figure 4 is sound if and only if it is strongly sound (holds both for the discrete-time and continuous-time) and the undecidability result in Theorem 3.7 is valid also for strong soundness. □

Remark 4.3. Note that urgent transitions on their own are not enough to establish undecidability of strong soundness—in order to avoid divergent markings with urgent transitions only, there has to be at least one urgent transition enabled in every reachable marking. This implies that only zero-time delays are possible and the notion of strong soundness coincides with classical soundness on untimed Petri nets that is decidable [1, 2].

We shall therefore focus on bounded ETAWFN in the discrete-time semantics where we show that strong soundness is decidable and the maximum execution time computable. Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be a given bounded ETAWFN. We can run Algorithm 1 to check for soundness of N in the discrete-time semantics. If it is not sound then N cannot be strongly sound either. Otherwise, let S be the number of non-final cut markings reachable in N (corresponding to the maximum cardinality of the set $Reached$ in Algorithm 1). Let

$$B = \max\{b \mid p \in P,\ I(p) = [0, b],\ b \neq \infty\} \tag{1}$$

be the maximum integer number used in any of the age invariants in N (where $\max \emptyset = 0$).

If the net N is strongly sound then there is no reachable divergent marking with the possible exception of final markings. Hence any reachable marking either contains some enabled urgent transition (and so no delay is possible) or the divergent behaviour is avoided by some age invariant, giving us the guarantee that no reachable marking can delay more than B units of time. As there are S reachable non-final cut markings, we know that any execution of N using more than $S \cdot B$ units of time must contain a loop

Algorithm 2: Strong soundness checking for timed-arc workflow nets

Input : A bounded ETAWFN $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ where $in, out \in P$.
Output: "true" together with the maximum execution time if N is strongly sound; "false" if N is not strongly sound.

begin
  if N *is not sound* **then**
    return *false*;
  else
    Let S be the number of non-final cut markings.
    Let B be the maximum integer bound in any of the age invariants in N.
  Let $Waiting$ be a stack and let $Passed$ be a set.
  Let $Trace$ be a stack where each element is annotated with a nonnegative integer $successors$.
  $M_{in} := \{(in, 0)\}$;
  $Waiting.push(\langle M_{in}, 0\rangle)$;
  $Passed := \{\langle M_{in}, 0\rangle\}$;
  $Trace.push(\bot)$;
  $Trace.top.successors := 1$;
  $exec_{max} := 0$;
  while $Waiting \neq \emptyset$ **do**
    $\langle M, i\rangle := Waiting.pop$;
    $Trace.top.successors{-}{-}$;
    $Trace.push(\langle M, i\rangle)$;
    $Trace.top.successors := 0$;
    foreach M'*s.t.* $M \xrightarrow{1} M'$ *or* $M \xrightarrow{t} M'$ *for some* $t \in T$ **do**
      $M'_c := cut(M')$;
      if $M \xrightarrow{1} M'$ **then** $i' := i + 1$; **else** $i' := i$;
      if $i' > S \cdot B$ **then**
        return *false*;
      if $\exists \langle M'_c, j\rangle \in Trace$ *such that* $j < i'$ **then**
        return *false*;
      if $M'_c := \{(out, x)\}$ *for some* $x \in \mathbb{N}_0$ **then**
        $exec_{max} := \texttt{MAX}(exec_{max}, i')$;
      else if $\langle M'_c, i'\rangle \notin Passed$ **then**
        $Waiting.push(\langle M'_c, i'\rangle)$;
        $Trace.top.successors{+}{+}$;
        $Passed := Passed \cup \{\langle M'_c, i'\rangle\}$;
    while $Trace.top \neq \bot \wedge Trace.top.successors = 0$ **do**
      $Trace.pop$;
  return *true* and $exec_{max}$;

with a non-zero time delay somewhere on the loop (meaning that N is not strongly sound, breaking the

condition c) of Definition 4.1) or there is a reachable divergent marking (and again the net is not strongly sound). On the other hand, if any execution of the workflow net does not take more than $S \cdot B$ units of time (until out is marked for the first time) then the set $\mathcal{T}(N)$ is finite. Now Lemma 4.4 and the fact that N is sound implies that N is strongly sound. Thanks to this observation, we can conclude with the following lemma.

Lemma 4.4. A sound and bounded ETAWFN is strongly sound if and only if the set of its execution times $\mathcal{T}(N)$ in the discrete-time semantics is finite.

We shall now prove that strong soundness for bounded workflow nets in the discrete-time semantics is decidable. The decision procedure is formalized in Algorithm 2 that gives us the following theorem.

Theorem 4.5. Strong soundness of bounded extended timed-arc workflow nets in the discrete-time semantics is decidable and the maximum execution time is computable.

Proof:
Let N be a given bounded ETAWFN, given as an input to Algorithm 2. We first run Algorithm 1 to check for soundness of N. If it is not sound, we terminate and announce that N is not strongly sound. Otherwise, by S we denote the size of the reachable state-space produced by Algorithm 1 and let B be computed using Equation (1). Now we perform a depth-first search of the reachable state space of cut-markings using the stack $Waiting$ that stores pairs consisting of a marking and the total time when it was reached (since the beginning of the execution of the workflow). The set $Passed$ stores the pairs that we already discovered and finally the additional stack $Trace$ is used to store the pairs that are at the moment on the path from the initial marking to the marking currently popped from $Waiting$ at line 16. Before the main while-loop, these data structures are initialized with the initial marking and for technical convenience we place the bottom of the stack to $Trace$. Each element on the stack $Trace$ has also the attribute $successors$, indicating how many unexplored successors of the marking are still placed on the $Waiting$ stack.

During the main while-loop at line 15, we pop a marking M with its total time stamp i from $Waiting$, update the stack $Trace$ accordingly and then examine each of M's successors in the foreach-loop at line 20. First, we create the cut-marking M'_c of the successor (line 21) and then update its total time-stamp i'. If we found a marking with a time-stamp exceeding $S \cdot B$, we terminate and announce that the net is not strongly sound at line 24. This check is strictly speaking not necessary (but allows for early termination) as exceeding the bound for a bounded net implies the existence of a loop with positive time duration that is detected at line 25. Here, if the marking M'_c is already on the stack $Trace$ but with a lower total time-stamp, we also return the negative answer. Otherwise, we either reached a final marking and update the so far computed maximum execution time at line 28 or if the marking M'_c has not been seen yet, we push it to the $Waiting$ stack (line 30) and subsequently update the number of successors of the top marking in $Trace$ and add M'_c into the $Passed$ set. Finally, before we execute again the body of the while-loop, we clean up the $Trace$ stack at line 34 by popping all the markings that have zero successors waiting on the $Waiting$ stack.

Clearly, if the algorithm at any moment returned $false$ then the workflow net is not strongly sound. Termination of the algorithm is clear as $Waiting$ only contains pairs of cut-markings (the net is bounded) together with nonnegative integers bounded by $S \cdot B$. However, if the stack $Waiting$ becomes empty, we have explored the whole state-space and can conclude that the set $\mathcal{T}(N)$ is finite and by Lemma 4.4 we

conclude that N is strongly sound (and the algorithm returns $true$ together with the computed maximum execution time in this case). □

5. Soundness for Continuous Time Workflow Nets

We have so far proved decidability of soundness and strong soundness only for discrete-time workflow nets as this is often sufficient for the practical applications. Nevertheless, we shall now also discuss how the soundness notion behaves under the assumption of continuous time.

It is well-known that for closed timed automata (without any strict inequalities in guards), the location reachability problems for continuous-time and discrete-time semantics coincide (see e.g. [5, 11]). We prove the same result also for extended timed-arc Petri nets, following the idea from [11] where the problem is reduced to an instance of linear programming, though with additional technical challenges for the Petri net case. The result also implies that the discrete time semantics is sufficient for finding the minimum and maximum execution times.

Theorem 5.1. Let N be an ETAPN and let M_0 be a marking on N where all tokens have integer age. For any computation $M_0 \stackrel{d_0,t_0}{\rightarrow} M_1 \stackrel{d_1,t_1}{\rightarrow} M_2 \stackrel{d_2,t_2}{\rightarrow} \ldots \stackrel{d_{n-1},t_{n-1}}{\rightarrow} M_n$ in N with $d_i \in \mathbb{R}^{\geq 0}$ for all i, $0 \leq i < n$, there is a computation $M_0 \stackrel{d'_0,t_0}{\rightarrow} M'_1 \stackrel{d'_1,t_1}{\rightarrow} M'_2 \stackrel{d'_2,t_2}{\rightarrow} \ldots \stackrel{d'_{n-1},t_{n-1}}{\rightarrow} M'_n$ in N such that $d'_i \in \mathbb{N}_0$ for all i, $0 \leq i < n$, and $|M_n(p)| = |M'_n(p)|$ for all $p \in P$. Moreover if N is a workflow net then there exists a computation with integer delays that achieves the minimum, and if it exists also the maximum, execution time.

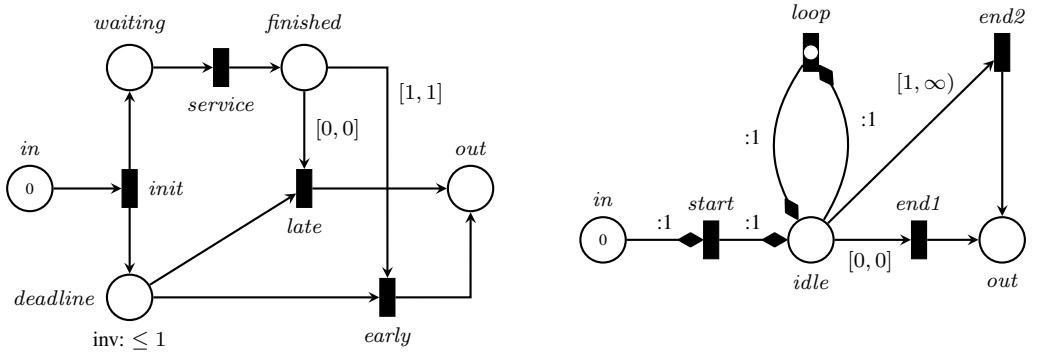
Proof:
Given a computation with real-time delays, we construct a set of linear inequalities that describe all possible delays allowed for the execution of the given trace. We only need difference constraints for this purpose and hence the corresponding matrix is totally unimodular [9]. As the instance has a real solution, it has also an optimal integral solution [16]. A detailed proof of this technical result is given in Section 8. □

As expected, continuous soundness now implies soundness also in the discrete case.

Theorem 5.2. Let N be an ETAWFN. If N is sound in the continuous semantics then it is sound in the discrete semantics.

Proof:
Let N be sound in the continuous semantics. Let M be a marking reachable from the initial marking M_{in} in the discrete semantics. As M is clearly reachable also in the continuous semantics, condition b) of Definition 3.4 is clearly satisfied. Regarding condition a) of the definition of soundness, we know that some final marking M_{out} is reachable from M in the continuous semantics. However, using Theorem 5.1 we can conclude that a marking M'_{out} that has the same distribution of tokens as M_{out} is reachable from M also in the discrete semantics, and hence N is sound w.r.t. the discrete semantics. □

If we moreover consider only workflow nets where time delays are not restricted by neither age invariants nor urgency, then both the continuous-time and discrete-time semantics coincide with respect to soundness.



(a) Using invariants (b) Using urgency

Figure 6: Sound nets in the discrete-time semantics and not sound in the continuous one

Theorem 5.3. Let N be an ETAWFN with no age invariants and no urgent transitions (inhibitor arcs are allowed). Then N is sound in the continuous-time semantics if and only if N is sound in the discrete-time semantics.

Proof:
"$\Rightarrow$": Follows from Theorem 5.2.

"$\Leftarrow$": Let N be sound in the discrete semantics. Let M be a marking reachable from the initial marking M_{in} in the continuous semantics. Now we can use Theorem 5.1 to argue that a marking M' is reachable from M_{in} in the discrete semantics (with integer delays only) such that $|M(p)| = |M'(p)|$ for all $p \in P$. As M and M' have the same number of tokens in all the places and N is sound in the discrete semantics, it follows that condition b) in Definition 3.4 holds for M' and hence also for M. Let us now argue for condition a). Let $M \xrightarrow{d} M_1$ where d is an integer greater than any constant used in any interval on any input arc. Such a delay is possible as the net does not contain any age invariants or urgent transitions. Clearly, $M' \xrightarrow{d} M'_1$ is possible also in the discrete semantics. However, now all tokens in M_1 and M'_1 are greater than any constant appearing in the net and hence these two markings are timed bisimilar. Because N is sound in the discrete semantics, we know that there is some final marking M_{out} such that $M_{out} \in [M'_1\rangle$. As M_1 is timed bisimilar with M'_1, it can also reach a final marking (bisimilar to M_{out}) and so does M. Hence condition a) of Definition 3.4 is established and we can conclude that N is sound also in the continuous semantics. □

For extended timed-arc workflow nets the notion of soundness for the discrete and continuous semantics are, perhaps surprisingly, different. Consider Figure 6 where both given workflow nets are sound in the discrete-time semantics but not in the continuous-time semantics.

Theorem 5.4. There is an ETAWFN (with either age invariants or urgent transitions) sound in the discrete-time semantics but not sound in the continuous-time semantics.

Proof:
Consider the nets in Figure 6 (as before we do not draw the $[0, \infty]$ intervals). It is easy to verify that

both of them are sound w.r.t. the discrete semantics. Indeed, in Figure 6a the age of the token in the place $finished$ can be either 1 or 0, depending on whether the service was executed early or late, and then either the transition $early$ or $late$ will be enabled and allow us to reach a final marking. Similarly in Figure 6b the age of the token in the place $idle$ will be of integer age so even though the transition $loop$ disables any time delay, we can still terminate the workflow by firing either the transition $end1$ or $end2$.

However, in the continuous semantics we can execute in the net from Figure 6a the sequence "$init$, delay 0.5, $service$, delay 0.5", bringing us into a deadlock situation. In the net from Figure 6b, we can perform the sequence "delay 0.5, $start$", ending up in a situation where only the urgent transition $loop$ is enabled and no time delay is possible. In both cases the nets are not sound w.r.t. the continuous-time semantics. □

As we just saw, the soundness checking in the discrete-time semantics does not necessarily imply soundness also in the continuous-time semantics. However, once we know that a net is sound in the continuous-time semantics, we can use Algorithm 2 to decide whether the net is strongly sound also in the continuous semantics.

Theorem 5.5. Let N be a bounded ETAWFN that is sound in the continuous-time semantics. Then N is strongly sound in the discrete-time semantics if and only if it is strongly sound in the continuous-time semantics.

Proof:
Clearly, if the net N is strongly sound in the continuous-time semantics, it is strongly sound also in the discrete-time semantics. If N is not strongly sound in the continuous-time semantics then either (i) there is a reachable, divergent and non-final marking or (ii) there is an infinite computation in N where the total accumulated time during the computation is ∞. In case (i) the net cannot be strongly sound in the discrete-time semantics either, as the divergent marking is due to Theorem 5.1 reachable also in the discrete semantics. In case (ii) we will argue that the net is not strongly sound in the discrete-time semantics due to Lemma 4.4 as the set of its execution times $\mathcal{T}(N)$ is infinite. Indeed, for any natural number m there is an execution in the continuous-time semantics where the accumulated delay is more than m and due to Theorem 5.1 there is also an execution in the discrete-time semantics that is longer than or equal to m (optimal minimal and maximal solutions can be achieved via integer delays). Hence $\mathcal{T}(N)$ cannot be finite and the net is not sound in the discrete-time semantics. □

6. Implementation and Experiments

We demonstrate the usability of our framework on three case studies. The studied workflows were modelled and verified with the help of a publicly available, open-source tool TAPAAL [10], where the algorithms presented in this paper are efficiently implemented in C++. The tool provides a convenient GUI support and one of the main advantages of our tool is the visualization of traces disproving soundness (see [14] for more discussion on this topic).

In the Brake System Control Unit (BSCU) case study, a part of a Wheel Braking System (WBS) used for the certification of civil aircrafts in the SAE standard ARP4761 [23], we discovered in less than 1 second that the workflow is not sound due to unexpected deadlocks. The authors of [23] were able to detect these problems asking a reachability query, however, the error traces contradicting soundness

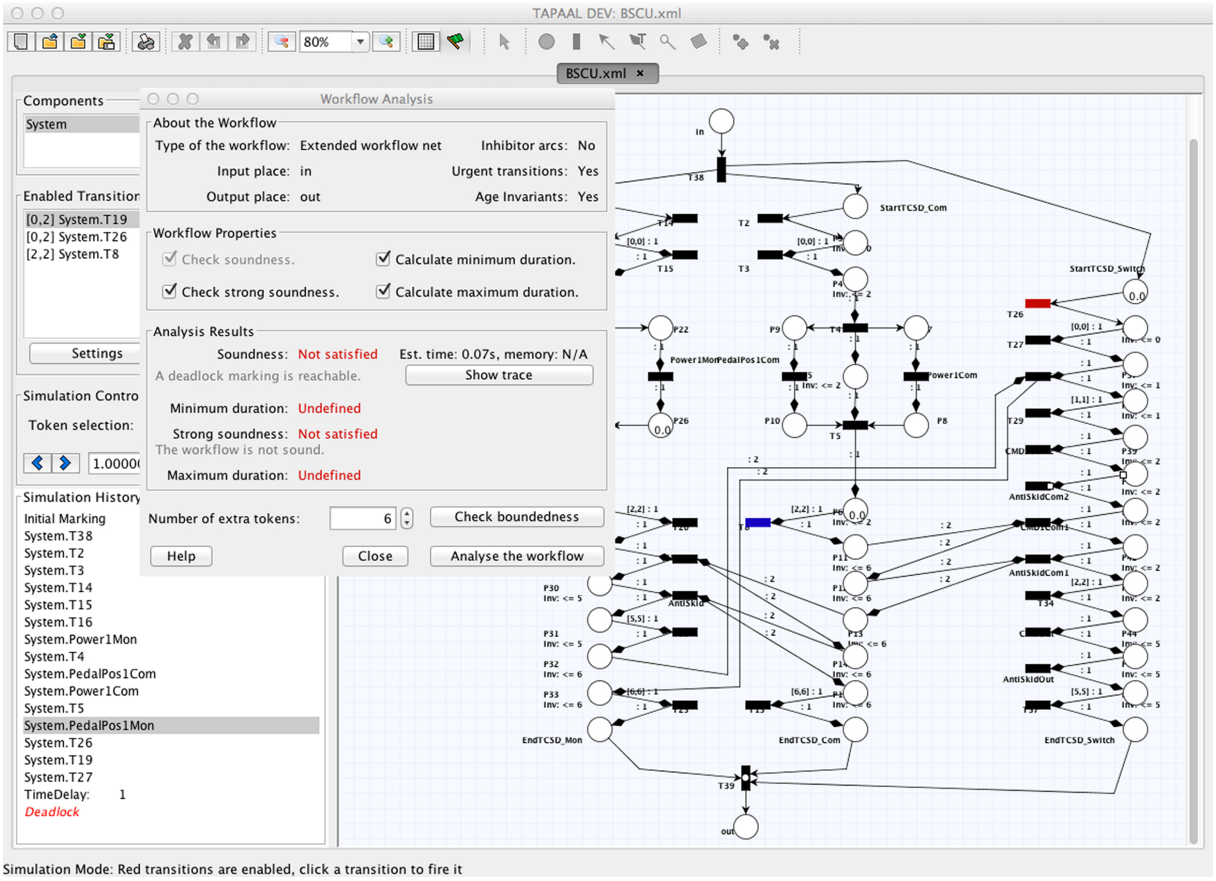


Figure 7: TAPAAL screenshot of the workflow analysis tool

were constructed manually. Our implementation allows a fully automatic detection and visualization of such situations. The workflow model contains 45 places, 33 transitions and 55 arcs.

In the second case study describing the workflow of MPEG2 encoding algorithm run on a multicore processor (Petri net model was taken from [22]), we verified in about 1 second both soundness and strong soundness, and computed the minimum and maximum encoding time for the IBBP frame sequence. The workflow model contains 44 places, 34 transitions and 82 arcs.

In the third case study, we checked the soundness of a larger blood transfusion workflow [8], the benchmarking case study of the little-JIL language. The Petri net model was suggested in [6] but we discovered several issues with improper workflow termination that were fixed and then both soundness and strong soundness was confirmed in less than 1 second, including the information about the minimum and maximum execution times. The workflow model contains 115 places, 94 transitions and 198 arcs.

TAPAAL models of all case studies can be obtained from www.tapaal.net and Figure 7 shows a screenshot of the GUI in the trace debugging mode for workflow analysis of the brake system control unit mentioned above.

7. Conclusion

We presented a framework for modelling of timed workflow processes via timed-arc workflow nets and studied the classical problem of soundness and its extension to time-bounded (strong) soundness. We provided a comprehensive analysis of decidability/undecidability of soundness and strong soundness on different subclasses of timed-arc workflow nets. We also suggested efficient algorithms for computing minimum and maximum execution times of a given workflow in the discrete-time semantics and implemented all algorithms within the tool TAPAAL [10]. As a result we have a complete theory for checking soundness on timed workflow nets in the discrete-time semantics and contrary to many other papers studying different variants of workflow processes, we took a step further by providing an efficient implementation of the algorithms, including a platform independent GUI support for modular design of timed workflow nets and visual error trace debugging. The tool is open-source and freely available at `www.tapaal.net`. The practical usability of the approach was documented on three industry-inspired case studies, demonstrating a promising potential for verification of larger timed workflows.

In our study we focused mainly on the discrete-time semantics of workflow nets that is often sufficient and allows for modelling of workflows where events can happen in discrete steps. In case of continuous time semantics (delays are from the domain of nonnegative real numbers) the situation is, perhaps surprisingly, different. Every sound net in the continuous time semantics is sound also in the discrete-time semantics but not necessarily the other way round, unless the nets do not contain any age invariants or urgent transitions that can enforce urgent behaviour. Hence the decidability of soundness in the continuous semantics cannot be derived from the results achieved in this paper. We nevertheless conjecture that soundness is decidable also in this case via the use of region-based techniques. We also note that this is a problem only for the soundness check as once we know that a given net is sound in the continuous-time semantics, checking for strong soundness can be done in the discrete-time semantics as the answers coincide here.

Acknowledgements. We thank the anonymous reviewers for their detailed comments and suggestions. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under Grant Agreement nr. 601148 (CASSTING).

8. Proof of Theorem 5.1

Let N be an extended timed-arc Petri net and let M_0 be its marking with integer ages of tokens only.

Let r be a fixed execution

$$M_0 \overset{d_0,t_0}{\rightarrow} M_1 \overset{d_1,t_1}{\rightarrow} M_2 \overset{d_2,t_2}{\rightarrow} \dots \overset{d_{n-1},t_{n-1}}{\rightarrow} M_n \tag{2}$$

where $d_i \in \mathbb{R}^{\geq 0}$ for all i, $0 \leq i < n$. Let $m = max\{|M_i| \text{ s.t. } 0 \leq i \leq n\}$ be the maximum number of tokens in any of the intermediate markings.

In order to prove that the same sequence of transitions can be executed also with integer time delays, we will first define a table with m rows and $n+1$ columns that represents how the tokens in the net (rows) are moved around in the net by each transition firing (column i represents the marking M_i that enables the firing of transition t_i and column $i+1$ the marking M_{i+1} reached after its firing). The timing information is forgotten in the table, it merely represents the consumption of tokens and whether

they were produced by normal or transport arcs. Based on such a table we will later define a difference constraint system describing all possible delays that enable the firing of the given transition sequence.

Formally, a table T for the execution r is a matrix with m rows and $n+1$ columns such that each element $T_{y,i}$ of the table, where $1 \leq y \leq m$ and $0 \leq i \leq n$, contains either the value $\bot$ (unused token) or the pair (p, f) where $p \in P$ represents the location of the token and $f \in \mathbb{N}_0 \cup \{\bullet\}$ is a flag signalling whether the age of the given token was set to some given value[1] from $\mathbb{N}_0$ or whether it has not changed (the flag value $\bullet$). We denote by $T^{place}_{y,i}$ and $T^{flag}_{y,i}$ the elements p and f of the pair of $T_{y,i}$, respectively. If one element of the pair (p, f) is not relevant for our considerations, we simply write $(p, -)$ or $(-, f)$. We let y to range over the rows in the table (tokens) and i over the columns in the table (transition firing steps).

Definition 8.1. (Valid table for run r)
A table T for the run r is *valid* if the following conditions are met.

a) Given the initial marking $M_0 = \{(p_1, x_1), (p_2, x_2), \ldots, (p_k, x_k)\}$, the zero column of T is defined as $T_{y,0} \stackrel{\text{def}}{=} (p_y, x_y)$ if $1 \leq y \leq k$, and $T_{y,0} \stackrel{\text{def}}{=} \bot$ if $k < y \leq m$.

b) For each column i in the table:

- there is a set $\mathit{Consume}_i \subseteq \{1, \ldots, m\}$ representing the y-indexes of the tokens in column i consumed by firing the transition t_i such that for all $p \in {}^\bullet t_i$ we have $w(p, t_i) = |\{y \in \mathit{Consume}_i \mid T^{place}_{y,i} = p\}|$, and for all $p \in P \setminus {}^\bullet t_i$ we have $\{y \in \mathit{Consume}_i \mid T^{place}_{y,i} = p\} = \emptyset$,
- there is a set $\mathit{Produce}_i \subseteq \{1, \ldots, m\}$ representing the y-indexes of the tokens in column $i+1$ produced by firing the transition t_i such that for all $p \in t_i^\bullet$ we have $w(t_i, p) = |\{y \in \mathit{Produce}_i \mid T^{place}_{y,i+1} = p\}|$ and for all $p \in P \setminus t_i^\bullet$ we have $\{y \in \mathit{Produce}_i \mid T^{place}_{y,i+1} = p\} = \emptyset$, and
- there is a bijection $\mathcal{P} : \{1, \ldots, m\} \to \{1, \ldots, m\}$ that relates the indexes of column i with those presented in column $i+1$ such that
  - i) if $|\mathit{Consume}_i| \leq |\mathit{Produce}_i|$ and $y \in \mathit{Consume}_i$ then $\mathcal{P}(y) \in \mathit{Produce}_i$,
  - ii) if $|\mathit{Consume}_i| \geq |\mathit{Produce}_i|$ and $\mathcal{P}(y) \in \mathit{Produce}_i$ then $y \in \mathit{Consume}_i$,
  - iii) if $y \in \mathit{Consume}_i$ and $\mathit{Type}((T^{place}_{y,i}, t_i)) = \mathit{Transport}_j = \mathit{Type}((t_i, p'))$ then $\mathcal{P}(y) = y$ and $T^{place}_{y,i+1} = p'$,
  - iv) if $y \in \{1, \ldots, m\} \setminus \mathit{Consume}_i$ and $T_{y,i} \neq \bot$ then $\mathcal{P}(y) = y$ and $T^{place}_{y,i+1} = T^{place}_{y,i}$,
  - v) if $y \in \{1, \ldots, m\} \setminus \mathit{Consume}_i$ and $T_{y,i} = \bot$ then either $\mathcal{P}(y) \in \mathit{Produce}_i$, or $\mathcal{P}(y) = y$ and $T_{y,i+1} = \bot$, and
  - vi) if $T_{\mathcal{P}(y),i+1} = \bot$ then $y \in \mathit{Consume}_i$ or $T_{y,i} = \bot$.

c) For each column i in the table:

- if $\mathit{Type}((p, t_i)) = \mathit{Inhib}$ for some $p \in P$ then $|\{y \in \{1, \ldots, m\} \mid T^{place}_{y,i} = p\}| < w(p, t_i)$

[1] The age value can only be reset to 0. The only possibility where the value can be different from zero is in the first column that encodes the marking M_0 where tokens can have arbitrary integer ages.

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
1	(in,0)	(booking,0)	(payment,•)	(successful,0)	(successful,0)	(successful,0)	(successful,0)	(out,0)
2	$\perp$	(attempts,0)	(attempts,•)	(attempts,•)	(attempts,•)	(attempts,•)	$\perp$	$\perp$
3	$\perp$	(attempts,0)	(attempts,•)	(attempts,•)	(attempts,•)	$\perp$	$\perp$	$\perp$
4	$\perp$	(attempts,0)	(attempts,•)	(attempts,•)	$\perp$	$\perp$	$\perp$	$\perp$

Table 1: A valid table for the run in Equation (3)

- for all $y \in \mathit{Produce}_i$, if $\mathit{Type}((t_i, T^{place}_{y,i+1})) = \mathit{Normal}$ then $T^{flag}_{y,i+1} = 0$ else $T^{flag}_{y,i+1} = \bullet$, and
- if $y \notin \mathit{Produce}_i$ and $T_{y,i+1} \neq \perp$ then $T^{flag}_{y,i+1} = \bullet$.

Let us now assume a computation

$$M_0 \overset{0,start}{\to} M_1 \overset{3.5,book}{\to} M_2 \overset{4.6,pay}{\to} M_3 \overset{0,empty}{\to} M_4 \overset{0,empty}{\to} M_5 \overset{0,empty}{\to} M_6 \overset{0,success}{\to} M_7 \tag{3}$$

in our running example from Figure 1, where $M_0 = \{(in, 0)\}$. One possible valid table T for this computation is given in Table 1.

One can verify that there is a valid table for any computation of the given net N. On the other hand, any valid table defines a legal computation on untimed markings represented by each column.

Definition 8.2. (Untimed marking given by column i)
Let T be a valid table and let $0 \leq i \leq n$. We define the untimed marking $M^u_i \overset{\text{def}}{=} \{\, T^{place}_{y,i} \in P \mid 1 \leq y \leq m\}$ as a multiset of all places where a token is present in the column i of the table T.

By the way in which a valid table is constructed, we can verify the validity of the following lemma.

Lemma 8.3. (Untimed consistency of a valid table T)
Let T be a valid table. Then $M^u_i \overset{t_i}{\to} M^u_{i+1}$ for all i, $0 \leq i < n$, in the classical (untimed) Petri net semantics.

We shall now proceed with defining a set of difference constraint inequalities that encode in a sound and complete way the timings aspects of the computation given in Equation (2).

Let the *execution time* of a transition t_i in Equation (2) be denoted by the variable e_i representing the total time elapsed from the initialization until the transition t_i is fired. In order to construct the system of inequalities over the variables $e_0, \ldots, e_{n-1}$, we need to define an expression describing the age of a token y just at the moment when the transition t_i is fired.

Definition 8.4. (Token-age expression)
Let T be a valid table. We define $\mathit{age}(y, i)$ where $1 \leq y \leq m$ and $0 \leq i \leq n$ as the expression

$$\text{“}e_i - e_{j-1} + d\text{”}$$

where j, $j \leq i$, is the largest number not greater than i such that $T^{flag}_{y,j} = d \in \mathbb{N}_0$. By agreement, e_{-1} is replaced with 0.

The intuition is that $age(y, i)$ expresses, in terms of the execution time variables, the current age of the token y after the time delay d_i and just before firing the transition t_i. The correctness of the definition follows from the requirements on a valid table and the fact that the first column ($i = 0$) of any valid table contains only pairs (p, f) where $f \neq \bullet$.

For example, in our running example in Table 1, the age of token 1 at the moment the transition pay is fired can be expressed as $e_2 - e_0 + 0$.

We are now ready to define, for a given valid table T, a system of inequalities over the variables e_i that expresses all the timing constraints on the firing of transitions, age invariants and urgency.

Definition 8.5. (Constraint system)
Let T be a valid table for a run r from Equation (2). The constraint system $\mathcal{C}$ for T is the set of inequations over the variables $e_0, e_1, \ldots, e_{n-1}$, containing the constraints $\{e_0 \leq e_1,\ e_1 \leq e_2, \ldots, e_{n-2} \leq e_{n-1}\}$, and constructed such that for all $p \in P$, $y \in \{1, \ldots, m\}$ and $i \in \{0, \ldots, n\}$:

a) if $T^{place}_{y,i} = p$ and $I(p) = [0, u]$ where $u \in \mathbb{N}_0$, we add the inequality $age(y, i) \leq u$ to $\mathcal{C}$,

b) if $T^{place}_{y,i} = p$ and $y \in Consume_i$ and $(p, t_i) \in IA$ and $g((p, t_i)) = [a, b]$, we add $a \leq age(y, i)$ and if $b \neq \infty$ also $age(y, i) \leq b$ to $\mathcal{C}$, and

c) if M^u_i enables some $t \in T_{urg}$ then we add $e_i - e_{i-1} = 0$ to $\mathcal{C}$ where, as mentioned above, e_{-1} is replaced with 0.

In our running example (Figure 1), the constraint system for the valid table depicted in Table 1 is the following.

- We add the inequalities
$$e_0 \leq e_1, \quad e_1 \leq e_2, \quad \ldots, \quad e_5 \leq e_6\ .$$
- For the two nontrivial age invariants, we add $age(1, 1) \leq 5$ and $age(1, 2) \leq 10$, in other words we add the constraints
$$e_1 - e_0 \leq 5, \quad e_2 - e_0 \leq 10\ .$$
- Regarding the guards on input arcs, we add for column 0 the constraint $0 \leq age(1, 0)$, for column 1 the constraints $2 \leq age(1, 1)$ and $age(1, 1) \leq 5$, for column 2 the constraints $0 \leq age(1, 2)$ and $age(1, 2) \leq 10$ and so on for the remaining columns that however produce only trivial constraints that are always satisfied. Hence the following constraints (listing only the nontrivial ones) are added to the system:
$$2 \leq e_1 - e_0, \quad e_1 - e_0 \leq 5, \quad e_2 - e_0 \leq 10\ .$$
- Finally, for each column that enables some urgent transition (columns 0, 3, 4, 5 and 6), we add (recall that $e_{-1} = 0$):
$$e_0 = 0, \quad e_3 = e_2, \quad e_4 = e_3, \quad e_5 = e_4, \quad e_6 = e_5\ .$$

Observe that the original delays in the trace from Equation (2) form a solution of the constructed constraint system: $e_0 = 0$, $e_1 = 3.5$, $e_2 = 8.1$, $e_3 = 8.1$, $e_4 = 8.1$, $e_5 = 8.1$, $e_6 = 8.1$. In fact, there is also an integer solution to the constraint system (this is not only a coincidence), e.g. $e_0 = 0$, $e_1 = 2$, $e_2 = 3$, $e_3 = 3$, $e_4 = 3$, $e_5 = 3$, $e_6 = 3$, and such a sequence is executable in our running workflow example.

Lemma 8.6. Let r be a run $M_0 \stackrel{d_0,t_0}{\rightarrow} M_1 \stackrel{d_1,t_1}{\rightarrow} M_2 \stackrel{d_2,t_2}{\rightarrow} \ldots \stackrel{d_{n-1},t_{n-1}}{\rightarrow} M_n$ with $d_i \in \mathbb{R}^{\geq 0}$ in a workflow net N. Then there is a valid table T for r and the corresponding constraint system $\mathcal{C}$ such that $e_i = \sum_{j=0\ldots i} d_j$ is a solution of $\mathcal{C}$. Moreover, $e_0, e_1, \ldots, e_{n-1}$ is a (real) solution of $\mathcal{C}$ if and only if $M_0 \stackrel{e_0,t_0}{\rightarrow} M_1 \stackrel{e_1-e_0,t_1}{\rightarrow} M_2 \stackrel{e_2-e_1,t_2}{\rightarrow} \ldots \stackrel{e_{n-1}-e_{n-2},t_{n-1}}{\rightarrow} M_n$.

Proof:
By analysing the requirements for a valid table, we can see that if a run r can be performed in the net N then we are able to design a table that satisfies all the requirements for the untimed part of the run execution and uses the right tokens in the pairing bijection such that the corresponding constraint system $\mathcal{C}$ gives the sufficient and necessary conditions for the execution time variables e_i to produce a valid computation of the net N with the given sequence of transitions firing. Hence the original execution times in the given run form one possible solution of the system but any such a solution actually provides a possible timed execution of the run (note that if a transition t_i is performed at time e_i and transition t_{i+1} is performed at time e_{i+1} then the delay between the execution of these two transitions is $e_{i+1} - e_i$). □

We can now summarise and conclude the proof of Theorem 5.1. We assumed a run of the net with real delays as in Equation (2). Based on this we know that there exists a valid table T for such a run such that the constraint system $\mathcal{C}$ for the run, representing all possible delays that can execute the transition sequence in (2), has a solution corresponding to the delays in (2). This is due to Lemma 8.6.

The constraint system $\mathcal{C}$ is an instance of linear programming problem where we used difference constraints only and we are therefore guaranteed that the matrix in the linear programming problem is totally unimodular [9]. As the system has a solution, it also has an optimal integral solutions—a general result guaranteed for any linear programming problem with totally unimodular matrix [16]. Hence using Lemma 8.6 we know that there is also an execution of N following the same transitions as in Equation (2) but with integral delays only. Moreover, the minimum and maximum execution times can be also achieved by integral delays only. Hence the proof of Theorem 5.1 is completed.

References

[1] van der Aalst, W. M. P.: Verification of Workflow Nets, *ICATPN'97*, 1248, Springer, 1997, ISBN 3-540-63139-9.

[2] van der Aalst, W. M. P.: The Application of Petri Nets to Workflow Management, *Journal of Circuits, Systems, and Computers*, **8**(1), 1998, 21–66.

[3] van der Aalst, W. M. P., van Hee, K., ter Hofstede, A. H. M., Sidorova, N., Verbeek, H. M. W., Voorhoeve, M., Wynn, M. T.: Soundness of workflow nets: classification, decidability, and analysis, *Formal Aspects of Comp.*, **23**(3), 2011, 333–363.

[4] Andersen, M., Larsen, H., Srba, J., Sørensen, M., Taankvist, J.: Verification of Liveness Properties on Closed Timed-Arc Petri Nets, *MEMICS'12*, 7721, Springer-Verlag, 2013.

[5] Asarin, E., Maler, O., Pnueli, A.: On discretization of delays in timed automata and digital circuits, *CONCUR'98*, 1466, Springer, 1998, ISBN 978-3-540-64896-3.

[6] Bertolini, C., Liu, Z., Srba, J.: Verification of Timed Healthcare Workflows Using Component Timed-Arc Petri Nets, *FHIES'12*, 7789, Springer-Verlag, 2013.

[7] Bolognesi, T., Lucidi, F., Trigila, S.: From Timed Petri Nets to Timed LOTOS, *PSTV'90*, North-Holland, Amsterdam, 1990.

[8] Christov, S., Avrunin, G., Clarke, A., Osterweil, L., Henneman, E.: A benchmark for evaluating software engineering techniques for improving medical processes, *SEHC'10*, ACM, 2010, ISBN 978-1-60558-973-2.

[9] Cong, J., Liu, B., Zhang, Z.: Scheduling with soft constraints, *ICCAD'09*, ACM, 2009, ISSN 1092-3152.

[10] David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets, *TACAS'12*, 7214, Springer-Verlag, 2012.

[11] D.B. Poulsen, J. v. V.: Concrete Traces for UPPAAL, 3rd Semester Report at Aalborg University, 2010.

[12] Dickson, L.: Finiteness of the odd perfect and primitive abundant numbers with distinct factors, *American Journal of Mathematics*, **35**, 1913, 413–422.

[13] Du, Y., Jiang, C.: Towards a Workflow Model of Real-Time Cooperative Systems, *ICFEM'03*, 2885, Springer, 2003.

[14] Flender, C., Freytag, T.: Visualizing the Soundness of Workflow Nets, *AWPN'06*, 267, Department Informatics, University of Hamburg, 2006.

[15] Hanisch, H.: Analysis of Place/Transition Nets with Timed-Arcs and its Application to Batch Process Control, *ICATPN'93*, 691, Springer, 1993.

[16] Hoffman, A. J., Kruskal, J. B.: Integral boundary points of convex polyhedra, *Linear Inequalities and Related Systems, Princeton Univ. Press*, **38**, 1956, 22–46.

[17] Jacobsen, L., Jacobsen, M., Møller, M., Srba, J.: Verification of Timed-Arc Petri Nets, *SOFSEM'11*, 2011.

[18] Jacobsen, L., Jacobsen, M., Møller, M. H.: Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants, *MEMICS'09*, 13, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[19] Larsen, K., Wang, Y.: Time-Abstracted Bisimulation: Implicit Specifications and Decidability, *Information and Computation*, **134**(2), 1997, 75–101, ISSN 0890-5401.

[20] Ling, S., Schmidt, H.: Time Petri nets for workflow modelling and analysis, *SMC'00*, 4, IEEE, 2000, ISSN 1062-922X.

[21] Minsky, M.: *Computation: Finite and Infinite Machines*, Prentice, 1967.

[22] Pelayo, F., Cuartero, F., Valero, V., Macia, H., Pelayo, M.: Applying Timed-Arc Petri Nets to improve the performance of the MPEG-2 Encoding Algorithm, *MMM'04*, IEEE, 2004.

[23] Sieverding, S., Ellen, C., Battram, P.: Sequence Diagram Test Case Specification and Virtual Integration Analysis using Timed-Arc Petri Nets, *FESCA'13*, 108, 2013.

[24] Tiplea, F., Macovei, G.: Timed Workflow Nets, *SYNASC'05*, IEEE Computer Society, 2005, ISBN 0-7695-2453-2.

[25] Tiplea, F., Macovei, G.: E-timed Workflow Nets, *SYNASC'06*, IEEE Computer Society, 2006, ISBN 0-7695-2740-X.

[26] Tiplea, F., Macovei, G.: Soundness for S- and A-Timed Workflow Nets Is Undecidable, *IEEE Trans. on Systems, Man, and Cybernetics*, **39**(4), 2009, 924–932.

[27] V.Valero, Cuartero, F., de Frutos-Escrig, D.: On non-decidability of reachability for timed-arc Petri nets, *PNPM'99*, IEEE, 1999.