Soundness of Timed-Arc Workflow Nets

José Antonio Mateo[1,2], Jiří Srba[1], and Mathias Grund Sørensen[1]

[1] Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark
[2] Department of Computer Science, University of Castilla-La Mancha,
Campus Universitario s/n, Albacete, Spain

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 explore the decidability of these 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. 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.

1 Introduction

Workflow nets [18, 19] 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 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 [20]) 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 [20] 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 [3, 9] 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.

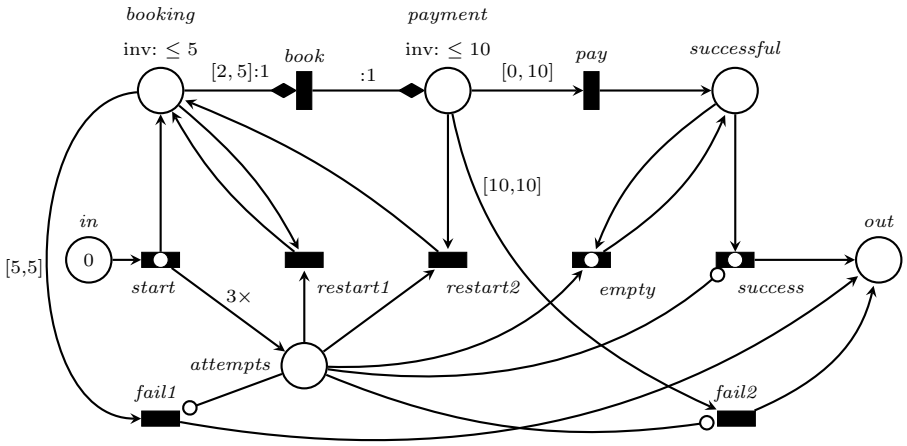


Fig. 1: Booking-payment workflow with timing constraints

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. 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. 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* (input place of the workflow). The 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.

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 a filled circle), so no time delay is possible once it gets enabled. After the *start* transition is fired, a new token of age 0 arrives to the place *booking* (initiating the booking phase) and three new tokens of age 0 arrive to 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, 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 [18] to deal with timing features and introduce a new notion of strong soundness that guarantees time-bounded workflow termination. We study the decidability/undecidability of soundness and strong soundness and conclude that even though they are in general undecidable, we can still 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 [21]. Moreover, our algorithms allow us to compute the minimum and maximum execution times of the workflow. The theory of timed-arc workflow nets is developed for discrete-time semantics. As remarked in Section 6, dealing with continuous-time semantics will require different techniques to argue about soundness of workflow nets. Last but not least, we implemented the algorithms given in this paper within the open-source tool TAPAAL [5] 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 [20]). We shall now focus mainly on time extensions of Petri net workflow models. Ling and Schmidt [11] 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 [7] 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. [15] 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 [16, 17] 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 [8] for more discussion).

2 Extended Timed-Arc Petri Nets

We shall start with the definition of extended timed-arc Petri nets in the discrete time setting and later on we recall some basic facts about the extrapolation technique applicable on this class of nets.

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. A *discrete timed transition system* (DTTS) is a triple $(S, Act, \to)$ where S is the set of states, Act is the set of actions and $\to \subseteq S \times (Act \cup \mathbb{N}_0) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \to$. If $a \in Act$ then we call it a *switch transition*, if $a \in \mathbb{N}_0$ we call it a *delay transition*. We also define the set of *well-formed 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 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,
- $g : IA \to \mathcal{I}$ *is a* time constraint function *assigning guards to input arcs,*
- $w : IA \cup OA \to \mathbb{N}$ *is a function assigning* weights *to input and output arcs,*

– *Type* : $IA \cup OA \to$ **Types** *is a* type function *assigning a type to all arcs where* **Types** $= \{Normal, Inhib\} \cup \{Transport_j \mid j \in \mathbb{N}\}$ *such that*
  - *if* $Type(a) = Inhib$ *then* $a \in IA$ *and* $g(a) = [0, \infty]$,
  - *if* $(p, t) \in IA$ *and* $t \in T_{urg}$ *then* $g((p, t)) = [0, \infty]$,
  - *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* $w((p, t)) = w((t, p'))$,
– $I : P \to \mathcal{I}^{inv}$ *is a function assigning* age invariants *to places.*

Remark 1. 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 [5].

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 [3, 9] but we add the prefix monotonic for clarity reasons.

Definition 2 (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* ($Type(a) \neq Inhib$ *for all* $a \in IA$*).*

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 $\mathcal{B}(\mathbb{N}_0)$ be the set of all finite multisets over $\mathbb{N}_0$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{N}_0)$ 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 $x \in \mathbb{N}_0$. 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 3 (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.*

$$\forall (p,t) \in IA.\mathit{Type}((p,t)) \neq \mathit{Inhib} \Rightarrow x_p^i \in g((p,t)) \text{ for } 1 \leq i \leq w((p,t))$$

– *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.*

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

– *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.*

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

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

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

A given ETAPN N defines a DTTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \rightarrow)$ 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 $M' = (M \setminus \mathit{In}) \uplus \mathit{Out}$ 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* $d \in \mathbb{N}_0$ 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.

Let $\rightarrow \stackrel{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t} \cup \bigcup_{d \in \mathbb{N}_0} \xrightarrow{d}$. 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 is no $d \in \mathbb{N}_0$, no $t \in T$ and no marking M' such that $M \xrightarrow{d,t} M'$. A marking M is *divergent* if for every $d \in \mathbb{N}_0$ we have $M \xrightarrow{d} M'$ for some M'.

In general, ETAPNs are infinite in two dimensions. 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 for bounded ETAPNs, i.e. for nets where the maximum number of tokens in any reachable marking is bounded by a constant.

Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be a given ETAPN. In [1] the authors provide an algorithm for computing a function $C_{max} : P \to (\mathbb{N}_0 \cup \{-1\})$ 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. 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 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 1 ([1]). *The relation $\equiv$ is a timed bisimulation.*

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

Definition 4 (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 1 ([1]). *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 can now show that adding more tokens to the net does not restrict its possible behaviour.

Lemma 2. *Let N be an MTAPN and $M, M' \in \mathcal{M}(N)$ be two of its markings such that $M \sqsubseteq_{cut} M'$. If $M \xrightarrow{d} M_1$ (resp. $M \xrightarrow{t} M_1$) then $M' \xrightarrow{d} M'_1$ (resp. $M' \xrightarrow{t} M'_1$) 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 1(i), we can by Theorem 1 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 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 1(i), we know by Theorem 1 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 1(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 [18]. 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 5 (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 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 behaviour of the net.

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 6 (Monotonic timed-arc workflow net). *A monotonic timed-arc workflow net (MTAWFN) is an ETAWFN 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 [19, 20].

Definition 7 (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 a 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 8 (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.*

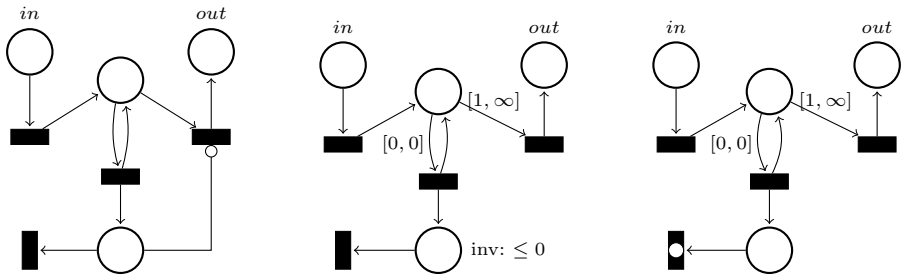
A classical result states that any untimed sound net is bounded [18]. This is not in general the case for extended timed-arc workflow nets as demonstrated in Figure 2. Nevertheless, we recover the boundedness result for the subclass of monotonic timed-arc workflow nets.

Theorem 2. *Let N an MTAWFN. If N is sound then N is bounded.*

Proof. By contradiction assume that N is a sound and unbounded MTAWFN. 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
ii) $|M| < |M'|$.

This follows from the fact that $M \sqsubseteq_{cut} M'$ iff $cut(M) \sqsubseteq cut(M')$ and from Definition 4 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 [6], 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'.



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

Fig. 2: Sound and unbounded extended timed-arc workflow nets

Since N is a sound workflow net, we now use condition a) of Definition 7, 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 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 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 7. □

Next we show that soundness for extended timed-arc workflow nets is undecidable. The result has been known for the extension with inhibitor arcs [20], we prove it (by reduction from two counter Minsky machines) also for urgent transitions and age invariants.

Theorem 3. *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.*

We now prove decidability of soundness for workflow nets without any inhibitor arcs, age invariants and urgency. This contrasts to undecidability of reachability for this subclass [21]. Algorithm 1 shows how to efficiently check soundness on this subclass and on the subclass of bounded nets. 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

Algorithm 1: Soundness checking for timed-arc workflow nets

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

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

```
1  begin
2      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)};
3      Waiting := {M_in}; M_in.min = 0;  Reached := Waiting; Final := ∅;
4      while Waiting ≠ ∅ do
5          Remove some marking M from Waiting with the smallest M.min;
6          foreach M' s.t. M -1-> M' or M -t-> M' for some t ∈ T do
7              if N is not monotonic and |M'| > k then return "Not k-bounded";
8              M'_c := cut(M'); M'_c.parents := M'_c.parents ∪ {M};
9              if M -1-> M' then M'_c.min := MIN(M'_c.min, M.min + 1);
10             else M'_c.min = MIN(M'_c.min, M.min);
11             if |M'_c(out)| ≥ 1 then
12                 if M'_c is a final marking then Final := Final ∪ {M'_c};
13                 else return false;
14             else
15                 if M'_c ∉ Reached then
16                     if M'_c is a deadlock then return false;
17                     if N is monotonic and ∃M'' ∈ Reached. M'' ⊑_cut M'_c then
18                         return false;
19                     Reached := Reached ∪ {M'_c}; Waiting := Waiting ∪ {M'_c};
20     Waiting := Final;
21     while Waiting ≠ ∅ do
22         Remove some marking M from Waiting;
23         Waiting := Waiting ∪ (M.parents ∩ Reached);
24         Reached := Reached ∖ M.parents;
25     if Reached = ∅ then
26         time := ∞; foreach M ∈ Final do time = MIN(time, M.min);
27         return true and time;
28     else
29         return false;
```

terminate at line 7 if the k bound for a non-monotonic 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 reach-

able 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. The next loop invariants can be proved in a straightforward manner.

Lemma 3 (Loop Invariants). *The while-loop in lines 4-19 of Algorithm 1 satisfies the following loop-invariants:*

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

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

Proof. Let us first argue for the fact $Reached \cup Final = \{cut(M') \mid M_{in} \to^* M'\}$. The inclusion "$\subseteq$" follows directly from claim b) of Lemma 3. 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 5 (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 $M \in Waiting$ such that $|M'| > k$. By claim b) of Lemma 3, we know that there is a computation from M_{in} to M_1 such that $M = cut(M_1)$ and we also know that $M \to M'$ (line 6). By Lemma 1 and Theorem 1 also $M_1 \to M_2$ such that $M' = cut(M_2)$ and this means that M_2 is reachable from M_{in} and at the same time $|M_2| > k$ as cut preservers the number of tokens in a marking. Hence if the algorithm returns "Not k-bounded" then the net is not k-bounded. □

Lemma 6 (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). □

Lemma 7 (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 7. Condition b) is straightforward as by Lemma 4 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 realise 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 8 (Termination). *Algorithm 1 terminates on any legal input.*

Proof. For non-monotonic nets there are only finitely many canonical markings with at most k tokens to be explored. For monotonic nets, similar arguments like in the proof of Theorem 2 imply that there cannot be infinitely many markings that are incomparable w.r.t. $\sqsubseteq_{cut}$. □

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

Theorem 4. *Soundness 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,

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

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 4).

Theorem 5. *Let N be a sound MTAWFN or a sound and bounded ETAWFN. The minimum execution time of N is computable.*

Notice 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 3. 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 3a is sound but there is no time guarantee by when the decision is reached. On the other hand, the net in Figure 3b 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 9 (Strong soundness of TAWFN). *An (extended or monotonic) timed-arc workflow net N is* strongly sound *if*

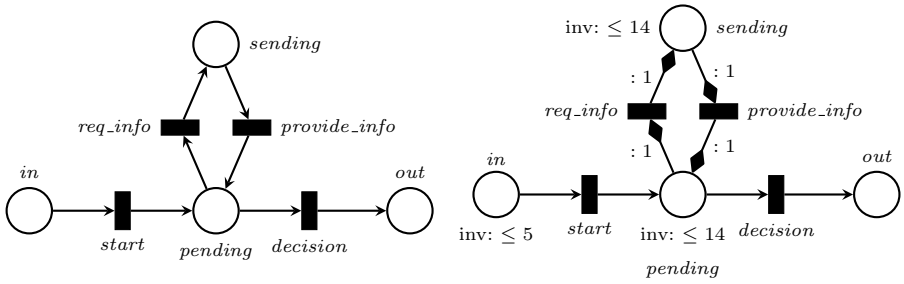
a) N is sound,
b) every divergent marking reachable in N is a final marking, and
c) there is no infinite computation starting from the initial marking $\{(in, 0)\} = M_0 \overset{d_0,t_0}{\to} M_1 \overset{d_1,t_1}{\to} M_2 \overset{d_2,t_2}{\to} \cdots$ where $\sum_{i \in \mathbb{N}_0} d_i = \infty$.

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

Theorem 6. *Strong soundness of ETAWFN is undecidable.*

The next lemma shows that strong soundness of bounded nets corresponds to the property that any execution of the workflow net is time bounded.

Lemma 9. *A sound and bounded ETAWFN is strongly sound if and only if the set of its execution times $\mathcal{T}(N)$ is finite.*



(a) Sound but strongly unsound workflow (b) Strongly sound workflow

Fig. 3: Fragment of customer complaint workflow ($[0, \infty]$ guards are omitted)

Proof. "⇐": By contradiction we assume that $\mathcal{T}(N)$ is finite while N is not strongly sound. This means that either (i) there is a reachable divergent marking of N that is not a final marking or (ii) the net contains an infinite time-divergent computation. In case (i) we can reach the divergent marking and perform an arbitrarily long delay after which (thanks to soundness of N) we can still reach some final marking. Hence $\mathcal{T}(N)$ is clearly infinite, contradicting our assumption. In case (ii) we can again follow the infinite execution for a sufficiently long time so that an arbitrary accumulated delay is achieved and again (thanks to soundness of N) we can reach some final marking, implying that $\mathcal{T}(N)$ is again infinite, contradicting our assumption.

"⇒": Let N be a strongly sound workflow net. From condition b) of Definition 9 we know that any reachable non-final marking in N cannot diverge. Moreover, there is a global bound B such that any reachable marking can delay at most B time units but not more. This is due to the fact that non-divergent behaviour is guaranteed either by age invariants (that have a fixed upper-bound limiting the maximum delay) or by urgent transitions with input arcs having $[0, \infty]$ guards only (prohibiting time delay as soon as a marking enables some urgent transition). Also, it is impossible to have a reachable marking with no tokens as the net cannot be sound in this case (Definition 5 requires that every transition has at least one input place).

Let S denote the number of reachable cut-markings in the net N. Hence any execution from the initial marking to some final one has either length of no more than S, meaning that its accumulated time duration is at most $S \cdot B$, or it contains the same cut marking twice, forming a loop on the execution. We know that there must be only zero delays on any such a loop as otherwise we would be able to repeat the cycle infinitely often, breaking condition c) of Definition 9 (of course, this loop is only on the cut markings but due to Theorem 1 it can be found also in the real execution of the net with exactly the same delays). This implies that the loop can be omitted while preserving the accumulated execution time of the path. So we are guaranteed that the set $\mathcal{T}(N)$ is bounded by $S \cdot B$ and hence it is finite. □

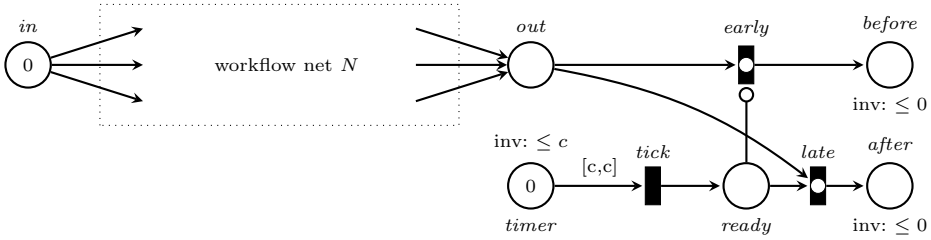


Fig. 4: Transformation of an ETAWFN N into an ETAPN $N(c)$

Lemma 9 implies that for any bounded and strongly sound net N, the maximum execution time is well defined. Notice that for monotonic nets (even extended with inhibitor arcs), the answer to the strong soundness is always negative as all reachable markings are divergent.

We shall so focus on bounded ETAWFN where strong soundness is decidable and the maximum execution time computable, relying on Lemma 9. We prove this by reducing strong soundness of a given bounded ETAWFN N into a reachability problem on a bounded ETAPN $N(c)$, where c is a nonnegative integer; the translation is given in Figure 4. The token from the place *timer* has to move to the place *ready* exactly at the time c from the start of the workflow. If the workflow can finish (by marking the place *out*) after at least c time units passed, then we can fire the transition *late* and mark the place *after*. If a token is moved to *out* earlier, then the urgent transition *early* will have to fire immediately.

Lemma 10. *Let N be a sound ETAWFN. Let $M_{after} = \{(after, 0)\}$ be a marking in $N(c)$ with one token in the place after. If $c \in \mathcal{T}(N)$ then $N(c)$ can reach the marking M_{after}. If $N(c)$ can reach the marking M_{after} then $c' \in \mathcal{T}(N)$ for some $c' \geq c$.*

Proof. If $c \in \mathcal{T}(N)$ then we perform the execution lasting exactly c time units in the net N and at the moment c we fire the transition *tick*, enabling the transition *late* and marking the place *after*. If on the other hand the place *after* can be marked then necessarily the token in the place *out* arrived at time c' such that $c' \geq c$, otherwise the urgent transition *early* had to be fired instead. □

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. 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 $B = \max\{b \mid p \in P,\ I(p) = [0, b],\ b \neq \infty\}$ be the maximum integer number used in any of the age invariants in N.

Lemma 11. *A sound and bounded ETAWFN N is strongly sound if and only if $N(S \cdot B + 1)$ cannot reach the marking $\{(after, 0)\}$.*

Proof. 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 with a non-zero time delay somewhere on the loop. Hence if $N(S \cdot B + 1)$ can mark the place *after*, then either there is a reachable divergent marking (and the net is not strongly sound) or there exists an execution with a non-zero delay loop and by repeating the loop infinitely often, we get an execution breaking the condition c) of Definition 9 and the net is not strongly sound either.

On the other hand, if the place *after* is not reachable in $N(S \cdot B + 1)$ then it is surely not reachable also for any other $c \geq S \cdot B + 1$, meaning that the set $\mathcal{T}(N)$ is finite by Lemma 10. Now Lemma 9 and the fact that N is sound implies that N is strongly sound. □

Theorem 7. *Strong soundness of bounded extended timed-arc workflow nets is decidable and the maximum execution time is computable.*

Proof. Let N be a given bounded ETAWFN. 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, we check whether $N(S \cdot B + 1)$ can reach a marking containing just one token in the place *after* (this check is decidable for bounded ETAPN [1]). If this is the case, we return that N is not strongly sound due to Lemma 11. Otherwise the net is sound and we return the maximum accumulated delay in any marking discovered during the check as the maximum execution time (correctness follows from Lemma 10, soundness of N and the fact that once a token appears in the place *out* in $N(S \cdot B + 1)$, no further delay is possible). □

5 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 [5], 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 [8] 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 [14], we discovered in less than 1 second that the workflow is not sound due to unexpected deadlocks. The authors of [14] were able to detect these problems asking a reachability query, however, the error traces contradicting soundness 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 [13]), we

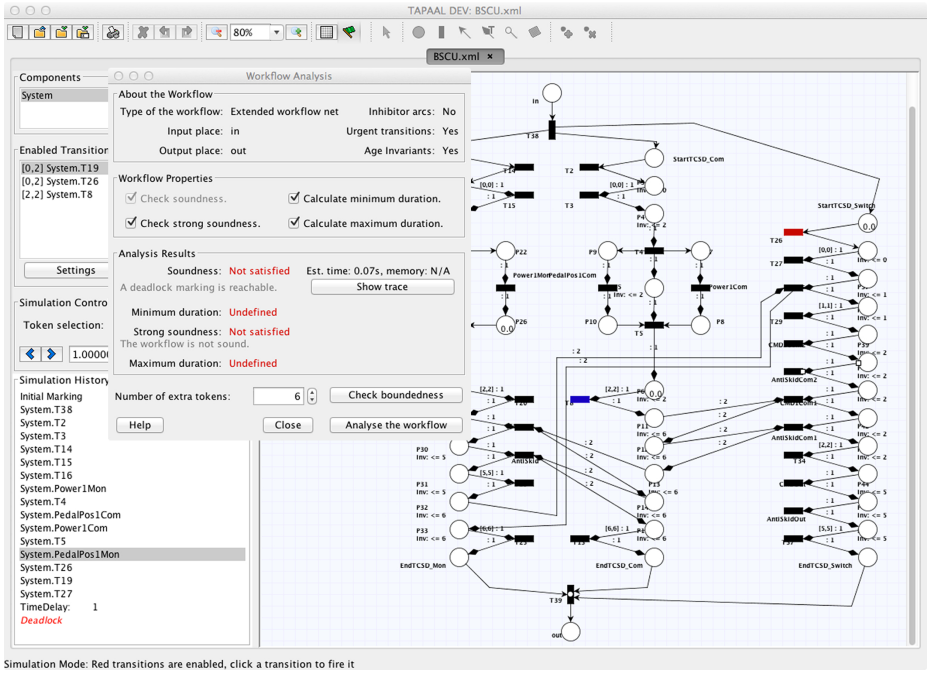


Fig. 5: TAPAAL screenshot of the workflow analysis tool

verified in about 10 seconds 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 [4], the benchmarking case study of the little-JIL language. The Petri net model was suggested in [2] but we discovered several issues with improper workflow termination that were fixed and then both soundness and strong soundness was confirmed in about 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 5 shows a screenshot of the GUI in the trace debugging mode for workflow analysis of the brake system control unit mentioned above.

6 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

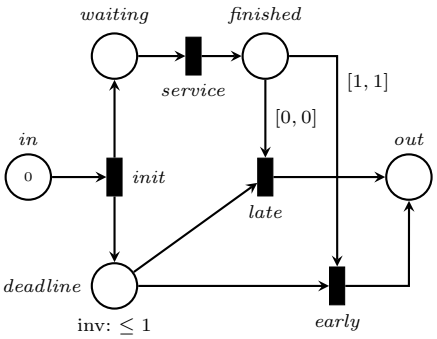


Fig. 6: A net sound in the discrete semantics but unsound in the continuous one

subclasses of timed-arc workflow nets. We also suggested efficient algorithms for computing minimum and maximum execution times of a given workflow and implemented all algorithms within the tool TAPAAL [5]. As a result we have a complete theory for checking soundness on timed workflow nets and contrary to many other papers studying different variants of workflow processes, we took a step further by providing 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 on the discrete 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. Consider the workflow net in Figure 6 (as before we do not draw the $[0, \infty]$ intervals). The workflow is clearly sound w.r.t. the discrete semantics as 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* is enabled and allows us to always reach a final marking. However, in the continuous semantics we can execute the sequence "*init*, delay 0.5, *service*, delay 0.5", bringing us into a deadlock situation. 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 and the details are left for future work.

References

1. M. Andersen, H.G. Larsen, J. Srba, M.G. Sørensen, and J.H. Taankvist. Verification of liveness properties on closed timed-arc Petri nets. In *MEMICS'12*, volume

7721 of *LNCS*, pages 69–81. Springer-Verlag, 2013.
2. C. Bertolini, Zh. Liu, and J. Srba. Verification of timed healthcare workflows using component timed-arc Petri nets. In *FHIES'12*, volume 7789 of *LNCS*, pages 19–36. Springer-Verlag, 2013.
3. T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *PSTV'90*, pages 1–14. North-Holland, Amsterdam, 1990.
4. S.C. Christov, G.S. Avrunin, A.L. Clarke, L.J. Osterweil, and E.A. Henneman. A benchmark for evaluating software engineering techniques for improving medical processes. In *SEHC'10*, pages 50–56. ACM, 2010.
5. A. David, L. Jacobsen, M. Jacobsen, K.Y. Jørgensen, M.H. Møller, and J. Srba. TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In *TACAS'12*, volume 7214 of *LNCS*, pages 492–497. Springer-Verlag, 2012.
6. L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.
7. Y. Du and C. Jiang. Towards a workflow model of real-time cooperative systems. In *ICFEM'03*, volume 2885 of *LNCS*, pages 452–470. Springer, 2003.
8. C. Flender and T. Freytag. Visualizing the soundness of workflow nets. In *AWPN'06*, volume 267. Department Informatics, University of Hamburg, 2006.
9. H.M. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *ICATPN'93*, volume 691 of *LNCS*, pages 282–299. Springer, 1993.
10. L. Jacobsen, M. Jacobsen, and M. H. Møller. Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants. In *MEMICS'09*, volume 13 of *OASICS*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.
11. S. Ling and H. Schmidt. Time Petri nets for workflow modelling and analysis. In *SMC'00*, volume 4, pages 3039–3044. IEEE, 2000.
12. M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice, 1967.
13. F.L. Pelayo, F. Cuartero, V. Valero, H. Macia, and M.L. Pelayo. Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. In *MMM'04*, pages 49–56. IEEE, 2004.
14. S. Sieverding, Ch. Ellen, and P. Battram. Sequence diagram test case specification and virtual integration analysis using timed-arc Petri nets. In *FESCA'13*, volume 108 of *EPTCS*, pages 17–31, 2013.
15. F.L. Tiplea and G. Macovei. Timed workflow nets. In *SYNASC'05*, pages 361–366. IEEE Computer Society, 2005.
16. F.L. Tiplea and G. Macovei. E-timed workflow nets. In *SYNASC'06*, pages 423–429. IEEE Computer Society, 2006.
17. F.L. Tiplea and G. Macovei. Soundness for s- and a-timed workflow nets is undecidable. *IEEE Trans. on Systems, Man, and Cybernetics*, 39(4):924–932, 2009.
18. Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN'97*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.
19. Wil M. P. van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
20. Wil M. P. van der Aalst, K. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Comp.*, 23(3):333–363, 2011.
21. V.Valero, F. Cuartero, and D. de Frutos-Escrig. On non-decidability of reachability for timed-arc Petri nets. In *PNPM'99*, pages 188–196. IEEE, 1999.

Appendix

A Proofs from Section 3

Proof (Theorem 3). The proofs are by reduction from the Minsky machine. A *Minsky machine* with two nonnegative counters c_1 and c_2 is a sequence of labelled instructions

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

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

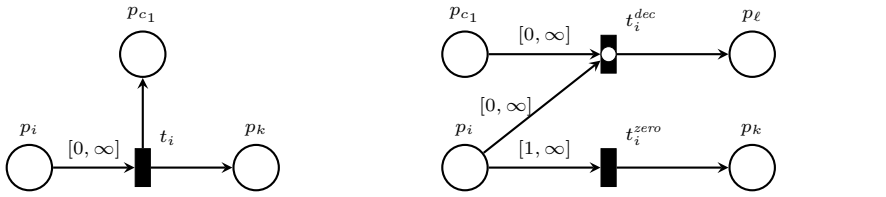
- (Inc) i: c_j++; goto k
- (Dec) i: if c_j=0 then goto k else (c_j--; goto ℓ)

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 reaches the instruction HALT then we say it *halts*, otherwise it *loops*. The problem whether a given Minsky machine halts is undecidable [12]. W.l.o.g. we assume that the machine halts only when both counters are empty (we can add a few instructions that will always empty the counters before reaching the halting instruction).

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 represents the value of the counters. For every instruction label i, $1 \leq i \leq n$, we add a new control place p_i. At any moment exactly 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 7 (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



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

Fig. 7: Simulation of (Inc) and (Dec) instructions with urgent transitions

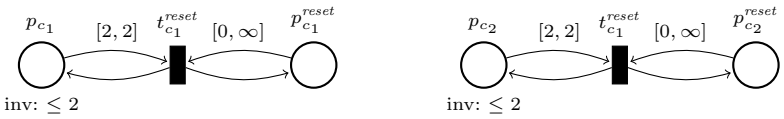
that either t_i^{dec} or t_i^{zero} can always fire. It is now an easy observation that the workflow net is sound if and only if the Minsky machine halts.

The reduction for workflow nets that contain only age invariants is more complicated. The reduction idea is based on [10], however, it had to be nontrivially modified in order to avoid the large number of possible deadlocks introduced in the reduction.

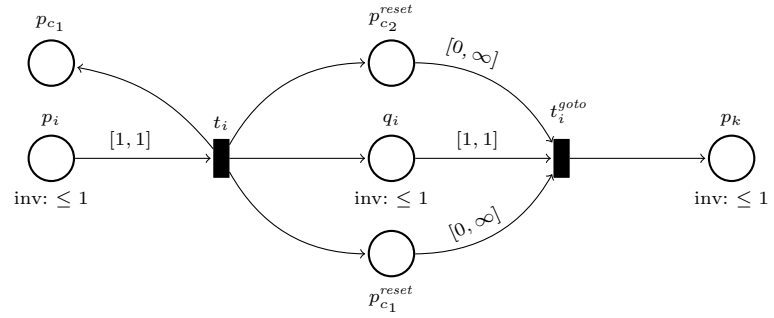
The counters are now modelled by two places that contain age invariants ensuring that no tokens can get older than 2, see Figure 8a. The intuition is that before and after the simulation of any instruction, all tokens in the places p_{c_1} and p_{c_2} are exclusively of age 1. As before the input place is $in = p_1$ and the output place is $out = p_n$.

Let us now observe that this invariant is preserved after simulating the increment instruction (Figure 8b). 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.

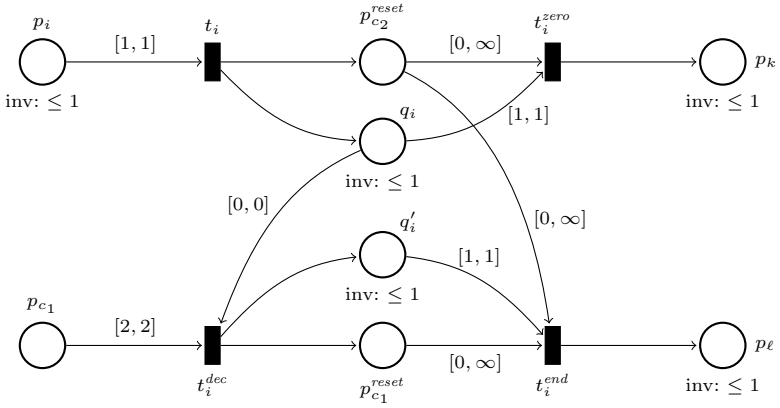
Let us consider now the test and decrement instruction simulated by the net in Figure 8c. Again, let us assume that all counter tokens are of age 1 and that there is one token of age 0 in p_i. First we wait one time unit and then fire t_i, meaning again that all counter tokens are of age 2. Now all tokens 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



(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`)`.

Fig. 8: Simulation of a Minsky machine by a workflow net with invariants

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. □

B Proofs Related to Algorithm 1

We will refer to phase 1 (lines 2-20) and phase 2 (lines 21-29) of Algorithm 1.

Proof (Lemma 3). 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* in line 19 of the loop body, and the same marking is also added to *Reached* in line 19 and no markings are removed from *Reached* in the body of the loop.

For claim b) notice that in line 5 we remove a marking M from *Waiting* and for any successor M' of M, the marking M is added to the set of parents of $M'_c = cut(M')$ (line 8). Due to the first invariant, M was already in *Reached* in the previous iteration of the loop. Hence there exists a computation $M_{in} \rightarrow^* M_1$ such that $M = cut(M_1)$ and the accumulated delay of this computation is $M.min$. Because $M \rightarrow M'$ (line 6) then also $M_1 \rightarrow M_2$ such that $M'_c = cut(M_2)$ (line 8). Hence $M_{in} \rightarrow^* M_2$ and $M'_c = cut(M_2)$ as required. The accumulated delay is updated according to the type of the transition $M_1 \rightarrow M_2$ at lines 9 and 10. If the value $M'_c.min$ changed after this update then the computation $M_{in} \rightarrow^* M_2$ 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 $M'_c.parents$ are only added at line 8 and such markings clearly satisfy the invariant claim. □

Proof (Lemma 6). Reading the algorithm, one can notice that it returns false in four lines (lines 13, 16, 18 and 29). Therefore, we have to demonstrate that the net is not sound in any of those cases.

- Starting with line 13, the algorithm returns false if it finds a marking M' 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 and it breaks condition b) of the definition of soundness (Definition 7). 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) 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 1). The marking M' is not a final marking and hence breaks condition a) of Definition 7. 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 $|M''| < |M'_c|$. Important to remark here that this situation $|M''| = |M'_c|$ 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 7 that there is path

from M'' to the final marking of N (M_{out}). However, by applying Lemma 2 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 7, 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 4). 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 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 7 and the net is not sound. □

Proof (Lemma 8). 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 initialised 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 non-monotonic 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 2 that there must exist M, $M' \in [M_{in}\rangle$ such that $M \sqsubseteq_{cut} M'$ and $|M| < |M'|$. However, 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. □

C Proofs from Section 4

Proof (Theorem 6). Similarly as in Theorem 3, we reduce the reachability problem for two-counter Minsky machines into checking strong soundness. We can use the same construction as in Figure 8 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 8 is sound if and only if it is strongly sound and the undecidability result in Theorem 3 is valid also for strong soundness. □