Compositional Analysis of Timed-arc Resource Workflows with Communication

Sine Viesmose Birch

Christoffer Moesgaard

des105f15, Spring 2015, Aalborg University

1 Resume

Workflow analysis is a relatively new and widely researched topic often used to find design errors like deadlocks and livelocks in business workflows by abstracting away the data and focusing on the flow of the system. The notion of soundness is used to verify that workflows do not have the mentioned design errors. Soundness for workflows is an area which has been studied, redefined, and extended, both concerning new features but also different classes of workflows. We propose timed-arc resource workflows (TARWFN), which have the possibility to save information in a workflow between executions, extended with the ability to communicate with other workflows. We introduce a notion of local soundness and strong local soundness for TARWFN, sequential composition, and an algorithm for checking (strong) local soundness by reducing it to a formalism for which efficient soundness checking algorithms already exist. We make it possible to analyse workflows individually, while still retaining a notion of soundness for sequential composition. For strong locally sound workflows we can determine minimum- and maximum execution times of the workflow, making it possible to approximate the entire workflow with a single transition having the minimum- and maximum execution times as its guards. The algorithm is implemented in the publicly available, open-source model checker TAPAAL. We demonstrate the usability of the workflows on a real-world smart house. The smart house has 16 lights, 16 buttons, a motion sensor and a buzzer and a controller. The controller gets the button pushes and translates that into events for the house. The house is modeled in TAPAAL and ends up with a controller consisting of 89 TARWFNs run sequentially together with 35 components to model the environment. The size of the state space of the house renders normal state space exploration infeasible. With TARWFNs we are able to approximate the smart house such that we are able to conclusively answer questions about the house within reasonable time.

Compositional Analysis of Timed-arc Resource Workflows with Communication

Sine Viesmose Birch, Christoffer Moesgaard

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

Abstract

We propose timed-arc resource workflows (TARWFN), which have the possibility to save information in a workflow between executions, as well as being able to communicate with other workflows. We introduce a notion of local soundness and strong local soundness for TARWFN, sequential composition, and an algorithm for checking (strong) local soundness by reducing it to a formalism for which efficient soundness checking algorithms already exist. We make it possible to analyse workflows, still retaining a notion of soundness for sequential composition. For strong locally sound workflows we can determine minimum- and maximum execution time of the workflow, making it possible to approximate the entire workflow with a single transition having the minimum- and maximum execution times as its guards. The algorithm is implemented in the publicly available, open-source model checker TAPAAL. We demonstrate the usability of the workflows on a real-world smart house. The size of the state space of the house renders normal state space exploration infeasible. With TARWFNs we are able to approximate the smart house such that we are able to conclusively answer questions about the house within reasonable time.

2 Introduction

Workflow analysis is a relatively new and widely researched topic often used to find design errors like deadlocks and livelocks in business workflows by abstracting away the data and focusing on the flow of the system. One way to model workflows is based on Petri nets [1] and also has various extensions, including time and resources. A lot of research has already been done in the area of workflows, which was introduced in the late 90s by Van der Aalst [2, 3]. Petri nets [4] are a commonly used formalism to describe workflows, because it gives a way to formally prove properties about workflows with well-defined semantics while still providing a graphical representation [1]. The anomalies of workflows can be found through the notion of *soundness* [2], which is a criterion to guarantee different properties including proper termination and the absence of deadlocks and livelocks.

Soundness for workflows is an area which has been studied, redefined, and extended, both concerning new features but also different classes of workflows. Many research groups have looked into extending workflows with different kinds of resources. Both K. v. Hee et al. [5] and G. Juhás et al. [6] have looked into non-consumable resources, where the resources must be returned before the workflow ends. G. Juhás et al. further looks into the parallel composition of identical workflows. Some researchers have also extended the workflow formalism with communication places, often called interface places [7]. There exists many real-world examples where the nature of the net does not correspond to the already defined extensions of workflows. An example is a smart house with a controller running in a loop, always checking for new events to handle and ensuring that the house is up to date. The controller has a lot similarities with a workflow with communication, but it also needs to store information between runs of the subparts.

Our contribution We introduce a workflow formalism making it possible to save information in a workflow between executions, as well as being able to communicate with other workflows through special interface places. We introduce a notion of local soundness and strong local soundness for our workflow as well as an algorithm for checking soundness by reducing our workflow to a formalism for which efficient soundness checking algorithms already exist. The local soundness allows workflows to be analysed individually, and sequential compositions of workflows will still be sound. Checking strong local soundness allows us to determine a minimum- and maximum execution time of the workflow, making it possible to over-approximate the entire workflow with a single transition having the minimum- and maximum execution times as its guards. Our notion of soundness gives the opportunity to decompose workflows, analysing them individually and only analyse the over-approximations together. The algorithm is implemented in the publicly available, open-source model checker TAPAAL [8]. We demonstrate the formalism with a case study of a smart house with 16 lights, 16 relays, 16 buttons, a motion sensor and a buzzer. The smart house is modelled as a timed-arc Petri net with 124 components, 89 of them timed-arc resource workflow nets running sequentially in a loop. Verifying the smart house model is a non-trivial problem as the state space is simply too large. Our methods show promising results, and we are able to decompose the model and analyse each workflow separately, over-approximating the cycle, making it possible answer questions about the smart house which otherwise would be impossible to verify. Our proposed model is scalable, since the verification decomposes the model and verifies individual workflows and never the net as a whole. Furthermore, the individual workflows can be over-approximated further in order to speed up verification, by using scaling techniques developed by Birch et al. [9]

Related Work We have chosen to focus on Petri net based workflow and how to expand the notion of soundness to save information in the workflow together with communication with other workflows. Soundness refers to a set

of properties that ensures that the workflow will not deadlock and always has the possibility to terminate, and when it does it will always terminate properly, which in the classical sense includes that the net will be empty of tokens. Frank Puhlmann and Mathias Weske introduce the notion of lazy soundness [10, 11], which allows tokens to stay behind, when the workflow ends as long as the tokens left behind stay behind and can never reach the output place. Our approach exploits the same idea, but differs in that we only allow tokens in a subset of places, and allow tokens to already be present at the beginning of the execution, as well as the fact that our workflows include timing. Other researchers have extended the classical workflow model and provide efficient analysis algorithms. Sidorova, Stahl, and Trčka [12] introduce data into the workflow which allows the workflow to read, write and delete data during the execution. Like them, we also store information in the workflow, but we only use tokens to represent the extra information. Several researchers have introduced resources [13, 14] and have developed resource constraint nets. Common for all the work is that the resources cannot be created nor destroyed during the execution. Juhás, Kazlov, and Juhásová [6] use the resources to look at soundness, when multiple instances of the same workflow execute in parallel. For parallel execution of workflows, researchers have also looked at workflows that can communicate with each other [7, 15, 16]. Their soundness consists of global soundness, which requires that the structure of all workflows is known beforehand. Kindler, Martens, and Reisig [17] use scenarios to describe the communication, allowing local soundness to be proven without knowing the rest of the workflows. This differs from our approach in that we focus on sequential composition of workflows, and that we also include workflows with timing. Earlier it has been proved that for sound workflow nets it is possible to abstract it to a single transition, see Chapter 4 in [18] and for workflows with inhibitor arcs the work has been extended in [19]. For strongly sound timed-arc workflows this is not the case and we can only make an approximation of it. For a more complete overview of soundness and workflows, we refer to the overview paper [20] by Van der Aalst. We utilize the approximation technique for explicit state space exploration implemented in TAPAAL [9]. The technique is a bit similar to [21], where the greatest common divisor is used to reduce the constants of a timed arc Petri net.

3 Motivation

When parallelism is introduced into a computational model, an issue known as state-space explosion can occur. State-space explosion is when the size of the state space increases exponentially, and as such makes it very difficult or even impossible to search through the entire state space. A way to remedy this problem is to reduce the size of the state space by creating a simpler model that approximates the behavior of a more complex system, while still retaining a number of properties.

One way of modeling systems is in the form of a series of workflows where

control passes through each workflow in sequence. Using known analysis techniques it is possible to determine an upper- and lower bound for the execution time of each workflow, making it possible to replace the workflow entirely with a single transition emulating the time bounds of the workflow. Such an approximation is beneficial in cases where we would like to determine the overall time bounds of a sequence of workflows. However, the existing workflow formalisms can be very limiting and many real life systems cannot be modeled as workflows directly, but can have parts which look very similar to workflows.

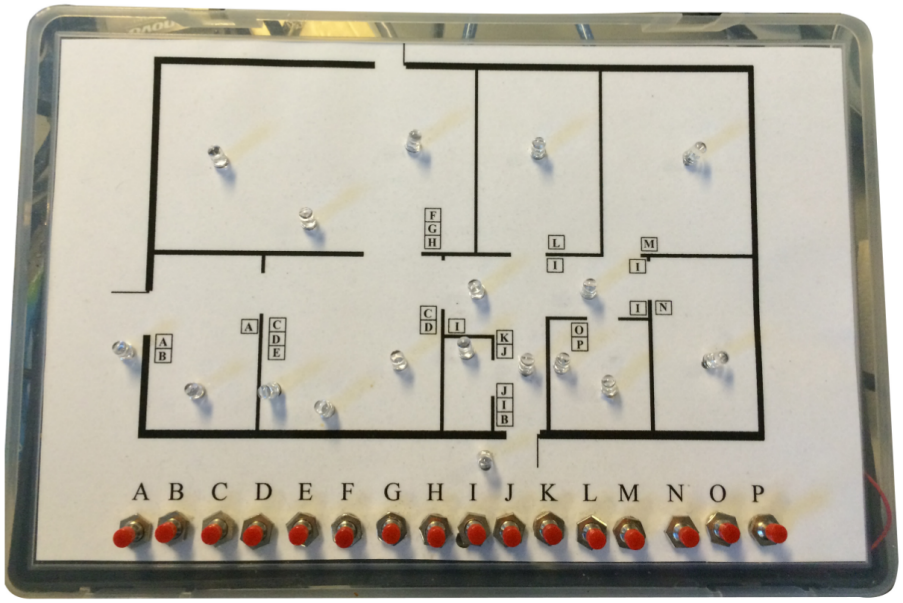


Figure 1: Simulation model of the smart house

In this report we have chosen to focus on a case study regarding home automation in a real house [22]. A picture of a physical simulation model of the house can be seen in Figure 1 [22]. All lights and buttons of the home are managed by a controller, for which we have some constraints. The controller has a program loop with a driver for each component running sequentially, where each cycle determines the state of the different components and reacts according to a set of rules. The home automation system can implement different actions for different durations of button presses, so the drivers would need to save information regarding the duration of a button press and whether a button press is currently taking place. The relays for the lights need to receive an impulse of a certain length in order to change state and if the impulse is too long the relay will burn out. This could be modeled with workflows, but we need to be able to "remember" information in a workflow between executions, as well as being able to allow a simple form of communication between workflows

and other components, e.g. a component for a button communicating with its corresponding driver. The communication between different components can be modeled with interfaces — shared places which can be used to send events, be resources or act like boolean variables. For a normal sized house, these kind of timed-arc Petri nets are too big to be verified directly and so identifying these sub-workflows and replacing them with a much simpler over-approximation will provide a way to verify these kinds of real-world applications.

We would like to determine whether or not, given the different rules for the buttons in the house, if it is possible, with the right combination of button presses, to reach a state where a relay can receive an impulse resulting in it burning out. This could for instance happen if an impulse is initiated in a button driver, but the time it takes for the program loop to reach the button driver again to end the impulse is too long. We would like to determine if it is possible to send an impulse that is too short, which would result in unreliable behavior of the relays. We would also like to determine the overall responsiveness of the buttons in the house.

We have identified three research questions, which we will try to answer in the report.

a) How can we exploit the notion of workflows, while still being able to remember information between different initializations of the workflow process?

b) What kind of information do we need to remember and what kind do we need to be able to communicate to other components in the net?

c) Will this formalism provide a way to verify real-world examples of large nets, and which constraints are there?

4 Definitions

4.1 Petri nets

Petri nets were developed by Carl Adam Petri [4], as a modeling language for the description of distributed systems. A Petri net is a directed bipartite graph, and contains two different types of nodes *places* and *transitions*. The nodes are connected via *arcs*, connecting places with transitions and transitions with places. An example of a Petri net can be seen in Figure 2. The net consists of six places represented by circles, and five transitions represented by rectangles. The net models a driver for a button controlling a light, determining whether or not the light should turn on when the button is pressed. The control flow in the model is represented by a token in the *in* place. In the initial marking, only the transition *pushButton* is enabled because of the presence of the token in *in*. Firing the transition consumes the token in the *in* place, and produces a token in each of the places *ButtonPushed* and *Pushing*. The following formal definition is based on work by Wil van der Aalst [3].

Definition 1 (Petri net [3])**.** A Petri net is a 5-tuple (P, T, IA, OA, w) where

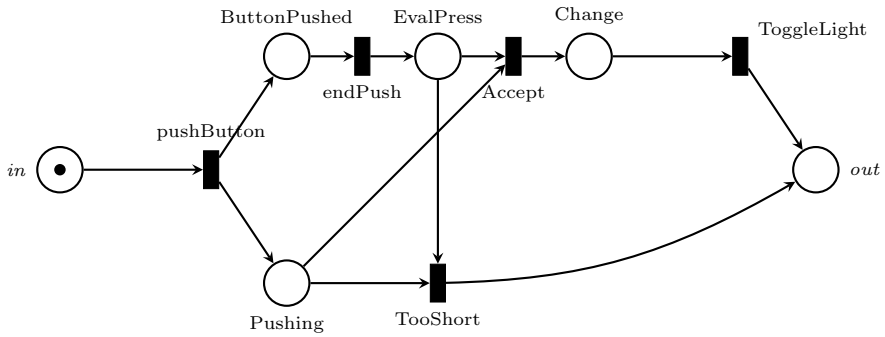


Figure 2: Model of a driver for a button controlling a light

- P is a finite set of places,
- T is a finite set of transitions. ($P \cap T = \emptyset$),
- $IA \subseteq P \times T$ is a finite set of input arcs,
- $OA \subseteq T \times P$ is a finite set of output arcs, and
- $w : IA \cup OA \to \mathbb{N}$ is a function assigning weights to input and output arcs.

The preset of input places of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p,t) \in IA\}$. Similarly, the postset of output places of t is defined as $t^\bullet = \{p \in P \mid (t,p) \in OA\}$. Places in the net may contain zero or more *tokens*. A state or *marking* in the net is a distribution of tokens over places, a marking M on N is a function $M : P \to \mathbb{N}$. To compare markings we define a partial ordering. For any two markings M_1 and M_2 we write $M_1 \leq M_2$ iff for all $p \in P : M_1(p) \leq M_2(p)$. The set of all markings over N is denoted by $\mathcal{M}(N)$.

A given Petri net $N = (P, T, IA, OA, w)$ defines a LTS $T(N) = (\mathcal{M}(N), T, \to)$, where the states are the markings of N and the transitions are as follows.

1. A transition t is *enabled* in marking M iff for each input place $p \in {}^\bullet t$ we have $M(p) \geq w((p,t))$.
2. An enabled transition t may fire, consuming tokens according to the weights from each input place of t and producing tokens according to the weights in each output place of t, producing a new marking M' where $M'(p) = M(p) - w((p,t)) + w((t,p))$ for every place $p \in P$. If (p,t) or (t,p) is not an arc, assume that $w((p,t)) = w((t,p)) = 0$.

Given a Petri net (P, T, IA, OA, w) and a marking M, we use the following notation:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in marking M_1 and firing t in M_1 results in marking M_2
- $M_1 \to M_2$: There is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from marking M_1 to marking M_n, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A marking M_n is called *reachable* from M_1, written $M_1 \rightarrow *M_n$, iff there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. The set of all markings reachable from a given marking M is denoted by $[M\rangle \stackrel{def}{=} \{M' \mid M \rightarrow^* M'\}$

4.2 Workflow Nets

The following is based on definitions by Wil van der Aalst [2] with a few alterations. Workflow nets (WFNs) is a workflow representation based on a subclass of Petri nets. WFNs differ from Petri-nets by having two special places *in* and *out*, which denote the beginning and the end of a workflow procedure. As such, processing of a case in a workflow begins with placing a token in *in*, and ends when a token is placed in *out*.

Definition 2 (Workflow net [2])**.** A *WFN* is a 7-tuple $(P, T, IA, OA, w, in, out)$ where (P, T, IA, OA, w) is a Petri net and $in \in P$ and $out \in P$ are places such that

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

The example Petri net seen in Figure 2 is also an example of a valid workflow. A marking M is an initial marking M_{in} for $N = (P, T, IA, OA, w, in, out)$ if for all $p \in P \setminus \{in\}$, $M(p) = 0$ and $M(in) = 1$. Likewise a marking is a final marking M_{out} of N if for all $p \in P \setminus \{out\}$, $M(p) = 0$ and $M(out) = 1$.

When working with workflow nets it is beneficial to be able to ensure proper termination of a procedure. A notion of workflow soundness is therefore introduced. A workflow is said to be sound if the workflow always has the option to reach a final marking, and at termination there is only a token in the *out* place. The following definition of soundness is a modified version taken from [2].

Definition 3 (Soundness of WFNs)**.** A Workflow net $N = (P, T, IA, OA, w, in, out)$ where M_{in} is the initial marking and M_{out} is the final marking, is sound iff

1. for every marking $M \in [M_{in}\rangle$ reachable from the initial marking M_{in}, there exists a firing sequence leading from marking M to marking M_{out}

$$\forall M.(M_{in} \rightarrow^* M) \Rightarrow (M \rightarrow^* M_{out}), \text{ and}$$

2. marking M_{out} is the only marking reachable from marking M_{in}, with at least one token in place *out*

$$\forall M.(M_{in} \rightarrow^* M \wedge M \geq M_{out}) \Rightarrow (M = M_{out}).$$

Wil van der Aalst has an extra third condition for soundness in [2], which states that there are no dead transitions in (N, M_{in}). The condition makes soundness even stronger, but for the purpose of this report, it does not matter and removing it makes it possible for us to add the different extensions of workflows directly without introducing new formalities.

4.3 Workflow Nets with Resources

Workflow nets usually abstract away resources such as machines or manpower, which may restrict the nets. *Resource-Constrained Workflow nets (RCWFNs)* are used to consider the influence of resources on the behavior of a workflow net.

Definition 4 (Resource-Constrained Workflow Net [6])**.** a Resource-Constrained Workflow Net (RCWFN) is a 7-tuple $(P, T, IA, OA, w, in, out)$, where (P, T, IA, OA, w) is a Petri net, with $P = P_{static} \uplus P_{normal}$ where P_{static} is a set of static-places (shared resource holders) and P_{normal} is a set of normal places, and $in \in P$ and $out \in P$ are places, such that

- there exists a unique place $in \in P$, such that ${}^\bullet in = \emptyset$,
- there exists a unique place $out \in P$, such that $out^\bullet = \emptyset$,
- 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 ${}^\bullet t \neq \emptyset$.

The initial marking M_{in} is redefined to be a marking, where for all $p \in P_{normal} \setminus \{in\}$ we have $M_{in}(p) = 0$, and $M_{in}(in) = 1$. Similarly for M_{out} where for all $p \in P_{normal} \setminus \{out\}$ we have $M_{out}(p) = 0$, $M_{out}(out) = 1$ and $M_{out}(s) = M_{in}(s)$ for all $s \in P_{static}$.

An example of a workflow net with resources can be seen in Figure 3. The workflow describes the fact that at most four relays must be turned on at the same time. For a workflow to start there must be less than four active relays. When it turns a relay on, it consumes a token from the resource place *Relays*, and hold that token for as long as the relay is turned on. When the impulse ends, the token is again freed and the relay turned off again and the workflow ends.

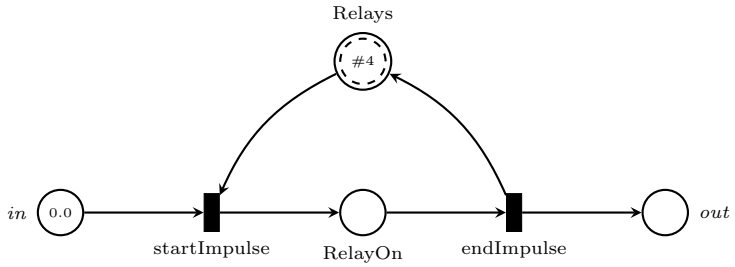


Figure 3: A RCWFN of a system controlling the number of active relays

The soundness is redefined such that a RCWFN can be sound for a single instance or multiple instances of the same workflow running in parallel. Here we only look at single instance soundness.

Definition 5 (Single instance soundness of RCWFN [6])**.** An RCWFN (N, M_{in}) is sound for a single instance if

1. for every marking $M \in [M_{in}\rangle$ reachable from the initial marking M_{in}, there exists a firing sequence leading from marking M to marking M_{out}

$$\forall M.(M_{in} \rightarrow^* M) \Rightarrow (M \rightarrow^* M_{out}),$$

2. marking M_{out} is the only marking reachable from marking M_{in}, with at least one token in place *out*

$$\forall M.(M_{in} \rightarrow^* M \wedge M(out) > 0) \Rightarrow (M = M_{out}), \text{ and}$$

3. for every marking M reachable from marking M_{in}, the number of tokens in static places are never increased above M_{in}:

$$\forall M.(M_{in} \rightarrow^* M) \Rightarrow \forall d \in P_{static}.M(d) \leq M_{in}(d).$$

The RCWFN in Figure 3 is sound for a single instance. The soundness definition has a consequence that all resources are non-consumable and must be returned before the workflow ends. That kind of resources still has merit, and there exists many real-work examples, where the limitation does not matter. An example could be doctors in a hospital, who can be used to treat patients, and after treating a patient can go on to treat other patients. Treating a patient would never result in the loss of a doctor.

4.4 Timed-Arc Petri Nets

Petri nets can also be extended with time instead of resources to better model a reality where timing matters. Before we can extend workflow nets, we must introduce extended timed-arc Petri nets in the discrete time setting, henceforth just timed-arc Petri nets (TAPNs). The following formal definition of timed-arc Petri nets and enabledness is taken from work by Birch et al. [9]. The definitions are for a discrete time setting, but are applicable for a continuous time setting as well.

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. We 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 a subset of $\mathcal{I}$ used in invariants as $\mathcal{I}^{\text{inv}} = \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$.

Definition 6 (Timed-Arc Petri Net)**.** A TAPN is a 9-tuple $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ where P, T, IA, OA and w is defined as in the earlier sections.

- T_{Urgent} is a finite set of urgent transitions such that $T_{Urgent} \subseteq T$,

- $g : IA \to \mathcal{I}$ is a time constraint function assigning guards to input 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 $t \in T_{Urgent}$ 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 $Type((t, p')) = Transport_j$ and $w((p, t)) = w((t, p'))$,
  - if $Type((t, p')) = Transport_j$ for some $(t, p') \in OA$ then there is exactly one $(p, t) \in IA$ such that $Type((p, t)) = Transport_j$ and $w((p, t)) = w((t, p'))$, and
- $I : P \to \mathcal{I}^{inv}$ is a function assigning age invariants to places.

An example of a timed-arc Petri net can be seen in Figure 4. The TAPN models the same situation as earlier, now just with added time intervals. From *in* it cannot delay and it must immediately fire the transition *pushButton*. The button can only be pushed for at most 20 time units, and it takes at most 1 time unit to evaluate the push. If the button where pushed for more than 10 time units, the light is toggled. The changing of the light takes exactly 20 time units.

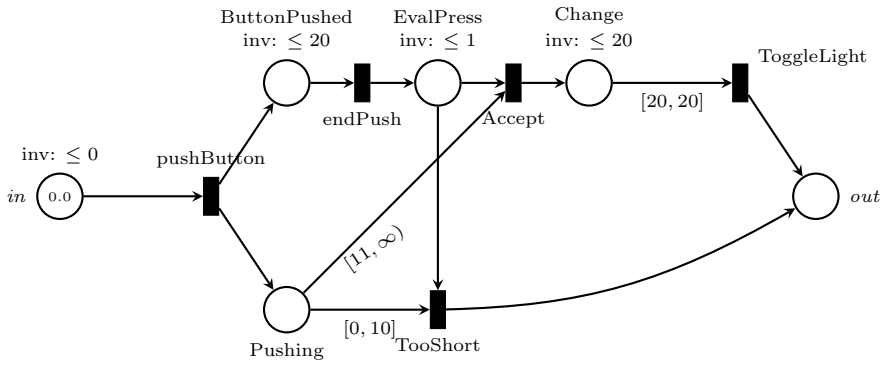


Figure 4: Timed model of a driver for a button controlling a light

Let $\mathcal{B}(\mathbb{N}_0)$ be the set of all finite multisets over $\mathbb{N}_0$. A marking M on a TAPN N is now a function $M : P \to \mathcal{B}(\mathbb{N}_0)$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$. The set of all markings over N is denoted by $\mathcal{M}$.

We use the notation (p, x) to denote a token at a place p with the age $x \in \mathbb{N}_0$. We write $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ for a marking with n tokens of ages x_i located at places p_i where $1 \leq i \leq n$ and we define $size(M) = \sum_{p \in P} |M(p)|$. A marked TAPN (N, M_0) is a TAPN N together with an initial marking M_0 with all tokens of age 0.

Definition 7 (Enabledness)**.** Let $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ be a TAPN. 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 $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((p',t))}) \mid p' \in t^\bullet\}$ if

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

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

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

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

(c) for all input 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.

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

(d) for all output arcs that are not part of a transport arc the age of the output token is 0, i.e.

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

A given TAPN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$, defines a TTS $T(N) = (\mathcal{M}(N), T, \rightarrow)$ where the 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 be fired 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 $d \in \mathbb{N}_0$ is allowed in M if $(x + d) \in I(p)$ for all $p \in P$ and all $x \in M(p)$ and there does not exist any $t \in T_{Urgent}$ and any $d' \in [0, d)$ such that t becomes enabled after the time delay d', i.e. by delaying d time units no token violates any of the age invariants and the delay can at most last until an urgent transition becomes enabled. 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.

We write $M \xrightarrow{d,t} M'$ for a delay transition $M \xrightarrow{d} M''$ followed by a switch transition $M'' \xrightarrow{t} M'$.

A marking M is called *divergent* if for every $d \in \mathbb{N}_0$ we have $M \xrightarrow{d} M'$ for some 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 $M \xrightarrow{d,t} M'$.

Definition 8 (Logic)**.** Let M be a marking and φ be a logical formula on the form

$$\varphi ::= p \bowtie n \mid p \bowtie p \mid \mathit{deadlock} \mid \neg\varphi \mid \varphi \wedge \varphi$$

where $p \in P$, $\bowtie \in \{=, <, >, \leq, \geq\}$ and $n \in \mathbb{N}_0$. The semantics of φ is

- $M \models p \bowtie m$ iff $|M(p)| \bowtie m$.
- $M \models \mathit{deadlock}$ iff for all t no d exists such that $M \xrightarrow{d,t} M'$.
- $M \models \neg\varphi$ iff $M \not\models \varphi$.
- $M \models \varphi_1 \wedge \varphi_2$ iff $M \models \varphi_1$ and $M \models \varphi_2$.

Definition 9 (Closed Timed-arc Petri Net)**.** A TAPN $N = (P, T, T_{Urgent}, IA, OA, g, w, \mathit{Type}, I)$ is called closed if and only if for all $(p,t) \in IA$ we have $g((p,t)) = [a,b]$ and for all $p \in P$ we have $I(p) = [0,b]$, for some $a, b \in \mathbb{N}_0^\infty$.

The general TAPN can be infinite in the number of tokens and thus have an infinite state space. But even if we only look at bounded TAPN with respect to the number of tokens in the net, the state space can still be infinite because of the ages of the tokens. The following presents results taken from [23, 24] to reduce the state space to a finite state space, where the corresponding LTS will be timed bisimilar.

Assume C_{max} is a function to compute the maximum constant associated with a place. The function is $C_{max} : P \to (\mathbb{N} \cup \{-1\})$. If the place is untimed, the place has the associated value -1. The consequence of C_{max} is that all ages of tokens in place p, which are strictly greater than $C_{max}(p)$ are irrelevant and can be discarded when computing the state space. C_{max} can be computed as in [23].

Let M be a marking of a TAPN N. Then $M_>$ and $M_\leq$ be defined as $M_> = \{x \in M(p) \mid x > C_{max}(p)\}$ and $M_\leq = \{x \in M(p) \mid x \leq C_{max}(p)\}$. From the definition we have $M = M_> \uplus M_\leq$. 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 $|M_>(p)| = |M'_>(p)|$. For all tokens in place p with age below $C_{max}(p)$ the two markings are the same, and for all tokens above, the number of tokens are the same, but they do not necessarily have the same age.

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 4.1 ([23])**.** *The relation $\equiv$ is a timed bisimulation.*

Definition 10 (Cut [24])**.** Let M be a marking. We define its canonical marking $\mathit{cut}(M)$ by $\mathit{cut}(M) = M_\leq \uplus \underbrace{\{C_{max}(p)+1, \ldots, C_{max}(p)+1\}}_{M_>(p) \text{ times}}$

Lemma 1 ([23])**.** *Let M, M_1 and M_2 be markings. Then (i) $M \equiv \mathit{cut}(M)$, and (ii) $M_1 \equiv M_2$ if and only if $\mathit{cut}(M_1) = \mathit{cut}(M_2)$*

From Lemma 1 it is clear that for a TAPN N the corresponding LTS will be bisimilar with N, where all markings $M \in [M_{in}\rangle$ reachable from M_{in} have been replaced with $\mathit{cut}(M)$.

4.5 Timed-Arc Workflow Nets

Timed-Arc Workflow Nets (TAWFN) have the same definitions as workflow nets, the only difference being the class of Petri Nets they are defined on.

Definition 11 (Timed-Arc Workflow Net [24])**.** A TAWFN is an 11-tuple $N' = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$, where $(P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ is a TAPN and $in \in P$ and $out \in P$ are places such that

1. $in \in P$ and ${}^\bullet in = \emptyset$,
2. $out \in P$ and $out^\bullet = \emptyset$,
3. for all $p \in P \setminus \{in, out\}$ we have ${}^\bullet p \neq \emptyset$ and $p^\bullet \neq \emptyset$, and
4. for all $t \in T$ we have ${}^\bullet t \neq \emptyset$.

The example seen in Figure 4 is also an example of a valid timed-arc workflow net, where *in* is the input place and *out* is the output place. All other places have both non-empty pre- and postset, and all transitions have a non-empty preset.

For TAPN the initial marking M_{in} is $M_{in} = \{(in, 0)\}$. A final marking M_{out} for TAPN is a marking, where $|M_{out}(out)| = 1$ and for all $p \in P \setminus \{out\}$ we have $|M_{out}(p)| = 0$.

The following definitions of soundness, strong soundness, minimum- and maximum execution times come from J. A. Mateo et al. [24].

Definition 12 (Soundness of TAWFN [24])**.** A timed-arc workflow net $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$, where M_{in} is the initial marking, is sound if

1. for every marking $M \in [M_{in}\rangle$ reachable from the initial marking M_{in}, there exists a firing sequence leading from marking M to some final marking M_{out}
$$\forall M.(M_{in} \rightarrow^* M) \Rightarrow (M \rightarrow^* M_{out}), \text{ and}$$
2. Final markings M_{out} are the only markings reachable from marking M_{in}, with at least one token in place *out*
$$\forall M.(M_{in} \rightarrow^* M \wedge |M(out)| \geq 1) \Rightarrow M \text{ is a final marking.}$$

With soundness it is possible to calculate the minimum execution time of a workflow, but not the upper bound — nor does it give a guarantee that the workflow will actually terminate. Strong soundness overcomes these problems.

Definition 13 (Strong Soundness of TAWFN [24])**.** A timed-arc workflow net $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ is strongly sound if

1. N is sound,
2. every divergent marking reachable in N is a final marking, and
3. there is no infinite computation starting from the initial marking.

$$\{(in, 0)\} = M_0 \xrightarrow{d_0, t_0} M_1 \xrightarrow{d_1, t_1} M_2 \ldots \text{ where } \sum_{i \in \mathbb{N}_0} d_i = \infty$$

Given a TAWFN N, let M_{in} be the initial marking and $\mathcal{M}_{final}(N)$ be the set of final markings of N. Then we have that $\mathcal{T}(M_{in})$ is the set of all execution times from the initial marking to the final marking, formally defined as:

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

Lemma 2 (Minimum execution time)**.** *Given a sound TAWFN N, the minimum execution time of N in marking M_{in}, defined as* $\min \mathcal{T}(M_{in})$ *is computable.*

Proof. The proof for this is given in [24]. □

Lemma 3 (Maximum execution time)**.** *Given a strongly sound TAWFN N, the maximum execution time of N in marking M_{in}, defined as* $\max \mathcal{T}(M_{in})$ *is computable.*

Proof. The proof for this is given in [24]. □

5 Timed-Arc Workflow Nets with Resources

We introduce our own notion of timed-arc workflow nets with communication and stored information, inspired by the previously defined principles of timed-arc workflow nets, resource-constrained workflow nets and workflows with communication.

Often when working with workflows, it is useful to be able to interact with an outside environment and even save information in the workflow between executions, e.g. if the workflow is part of a larger net and the workflow is in idling. However, these additions usually violate the rules of proper workflows, which means that common workflow analysis techniques are not applicable. We therefore introduce an extended definition of workflows with the addition of interface and status places, and define a notion of soundness.

The set of places are split into three sets of disjoint places. Status, interface and normal places. *Status places* are used to save information in the workflow between executions, and these places must be internal in the workflow. *Interface places* are used to interact with the outside environment in a workflow. We only allow interface places to act as untimed places with binary information. As such, no timing information may be associated with the interface places. Throughout this report we will use an abstract representation of this type of workflows, as can be seen in Figure 5. Status places are represented as dashed circles and interface places as solid circle within a dashed circle. The normal places are abstracted away within the gray diamond of the model.

Definition 14 (Timed-Arc Resource Workflow Net)**.** A Timed-Arc Resource Workflow Net (TARWFN) is a TAPN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ with $P = P_{interface} \uplus P_{status} \uplus P_{normal}$ where $P_{interface}$ is a set of interface places, P_{status} is a set of status places and P_{normal} is a set of normal places such that

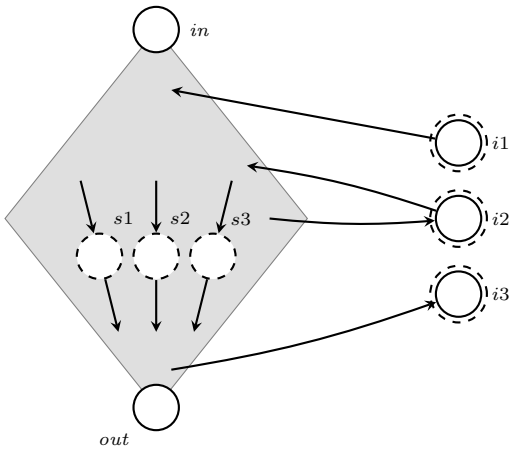


Figure 5: Abstract reprensentation of a timed-arc resource workflow net

1. there exists a unique place $in \in P_{normal}$, such that ${}^\bullet in = \emptyset$,
2. there exists a unique place $out \in P_{normal}$, such that $out^\bullet = \emptyset$,
3. for all $p \in P_{normal} \cup P_{status} \setminus \{in, out\}$ we have ${}^\bullet p \neq \emptyset$ and $p^\bullet \neq \emptyset$,
4. for all $t \in T$, we have ${}^\bullet t \neq \emptyset$,
5. for all $p \in P_{interface}$, $I(p) = [0, \infty)$,
6. for all $(p, t) \in IA$ where $p \in P_{interface}$, $g((p, t)) = [0, \infty)$,
7. for all arcs $a = (p, t) \in IA$ or $a = (t, p) \in OA$ where $p \in P_{interface}$, $Type(a) \neq Transport$, and
8. for all $(t, out) \in OA$, we have $Type((t, out)) \neq Transport$.

An example of a timed-arc resource workflow net can be seen in Figure 6. The TARWFN models the same situation as Figure 4 with the exclusion that the pressing of the button and changing the lights are now communicated in/out of the workflow. This allows the workflow to simply focus on the controller aspects of the process. The place *PushStarted* is a status place, where information about the press of the button is saved, when the controller has acknowledged a push. The button press is saved for the future runs of the controller. *ButtonPushed* and *ToggleLight* is interface places used for communication with other TAPNs.

Definition 15. Given a TARWFN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ the sets of markings $\mathcal{M}_{passive}, \mathcal{M}_{initial}$ and $\mathcal{M}_{final}$ are defined as:

- $\mathcal{M}_{passive}$ is the set of all markings such that the places $P_{status} \cup P_{interface}$ contain an arbitrary distribution of tokens, and for every $p \in P_{normal}$ we have $|M(p)| = 0$
- $\mathcal{M}_{initial}$ is a set of initial markings, defined as the set of passive markings with an added token of age 0 in in.
- $\mathcal{M}_{final}$ is a set of final markings, defined as the set of passive markings with an added token of age 0 in out.

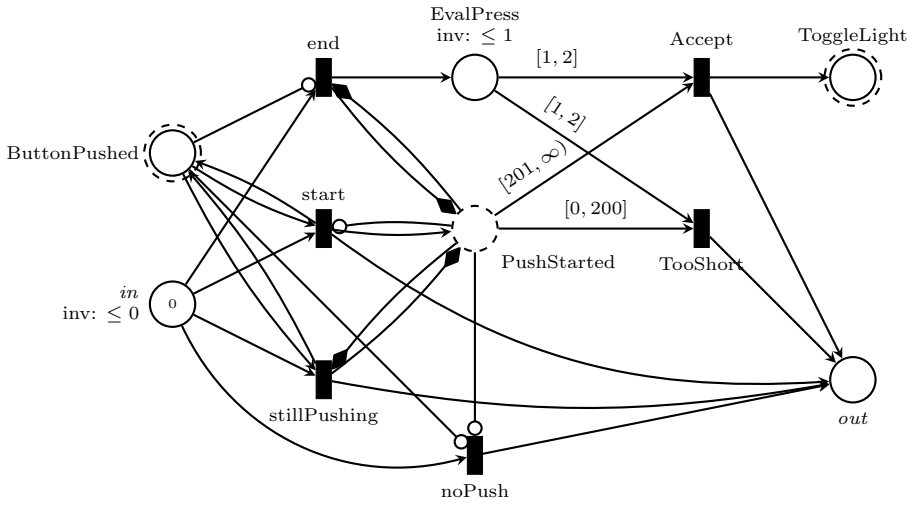


Figure 6: Timed model of a driver for a button controlling a light with communication.

A marking M is called active, if for at least one $p \in P_{normal}$, $|M(p)| \geq 1$. As we focus on models consisting of a sequential composition of several smaller workflows, we would like to define a notion of well-behavedness for the workflows when they are not currently involved in an active marking. Well-behavedness is to ensure that the workflows do not introduce deadlocks or change the state of interface places unexpectedly. So intuitively all reachable markings reachable from a passive marking should also be a passive marking, the distribution of tokens in interface places should stay the same for the reached marking and the initial passive marking and lastly from every reached marking it should be possible to make an arbitrarily long delay.

A marking M is *time computation divergent* if for every $d > 0$ there exists a combination of switch- and delay transitions $M \rightarrow^* M'$ such that the accumulated delay on the computation is at least d.

Definition 16 (Well-behaved timed-arc resource workflow net)**.** A TARWFN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ is well-behaved if for any marking $M \in [M_{passive}\rangle$ reachable from some $M_{passive} \in \mathcal{M}_{passive}$ of N:

1. $|M(p)| = 0$ for all places $p \in P_{normal}$,
2. $|M(p)| = |M_{passive}(p)|$ for all places $p \in P_{interface}$, and
3. M is time computation divergent.

With the set of possible initial markings we can define local soundness for TARWFN with status places and interface places. The soundness is very similar to soundness for TAWFN with the difference that it has to be sound for all the

initial markings, and that the workflow can leave tokens behind as long as the tokens are only present in status and interface places.

Definition 17 (Local soundness of timed-arc resource workflow net)**.** A well behaved TARWFN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$, is locally sound if for all initial markings $M_{in} \in \mathcal{M}_{initial}$:

1. for every marking $M \in [M_{in}\rangle$ reachable from marking M_{in}, there exists a firing sequence leading from marking M to some final marking $M_{out} \in \mathcal{M}_{final}$:

$$\forall M_{in} \in \mathcal{M}_{initial} \forall M.(M_{in} \rightarrow^* M) \Rightarrow (M \rightarrow^* M_{out}), \text{ and}$$

2. final markings $M_{out} \in \mathcal{M}_{final}$ are the only markings reachable from marking M_{in}, with at least one token in place out:

$$\forall M_{in} \in \mathcal{M}_{initial} \forall M.(M_{in} \rightarrow^* M \wedge |M(out)| \geq 1) \Rightarrow M \in \mathcal{M}_{final}.$$

Likewise we can define strong local soundness.

Definition 18 (Strong local soundness of timed-arc resource workflow net)**.** A TARWFN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ is strong locally sound if:

1. N is locally sound,
2. every divergent marking reachable in N from some $M_{in} \in \mathcal{M}_{initial}$ is a final marking, and
3. there is no infinite computation starting from a marking $M_{in} \in \mathcal{M}_{initial}$

$$M_{in} = M_0 \xrightarrow{d_0,t_0} M_1 \xrightarrow{d_1,t_1} M_2 \ldots \text{ where } \sum_{i \in \mathbb{N}_0} d_i = \infty.$$

The TARWFN in Figure 6 is strong locally sound.

Definition 19 (Minimum execution time)**.** Given a locally sound TARWFN N, the minimum execution time of N is defined as $\min_{M_{in} \in \mathcal{M}_{initial}} \min \mathcal{T}(M_{in})$.

Definition 20 (Maximum execution time)**.** Given a strong locally sound TARWFN N, the maximum execution time of N is defined as $\max_{M_{in} \in \mathcal{M}_{initial}} \max \mathcal{T}(M_{in})$.

5.1 Sequential Composition

If we execute two workflows sequentially after each other, we would like to establish that the composition too is a TARWRN and give a bound on the execution time of the composition. Informally, the composition merges the interface places of the two TARWFNs and executes the first workflow followed by the immidiate execution of the second workflow. An example of sequential composition of workflows can be seen in Figure 7. In Definition 21 the formal definition of the sequential composition is given.

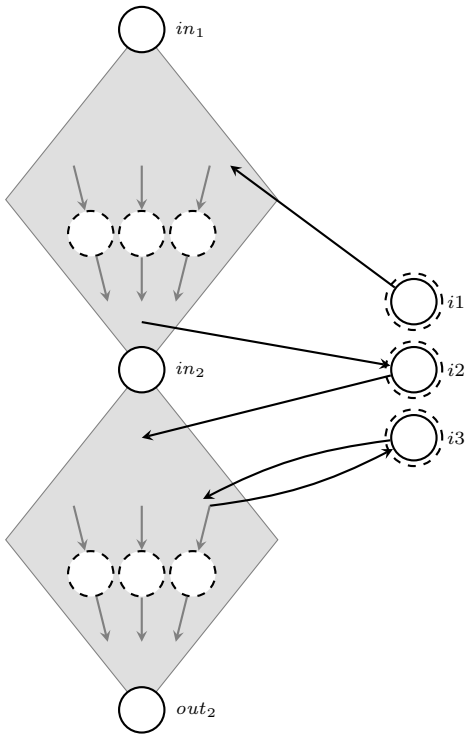


Figure 7: Example of sequential composition of two workflows

Definition 21 (Sequential composition of TARWFN)**.** Let $N_1 = (P_1, T_1, T_{Urgent1}, IA_1, OA_1, g_1, w_1, Type_1, I_1, in_1, out_1), N_2 = (P_2, T_2, T_{Urgent2}, IA_2, OA_2, g_2, w_2, Type_2, I_2, in_2, out_2)$ be two TARWFN with $(P_1 \setminus P_{interface1}) \cap (P_2 \setminus P_{interface2}) = \emptyset$ and $T_1 \cap T_2 = \emptyset$. Then the sequential composition $N_1; N_2$ is a TARWFN $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ where

- $P = P_1 \cup P_2 \setminus \{out_1\}$, where $P_{normal} = P_{normal1} \cup P_{normal2}$, $P_{interface} = P_{interface1} \cup P_{interface2}$ and $P_{status} = P_{status1} \cup P_{status2}$,
- $T = T_1 \cup T_2$,
- $T_{Urgent} = T_{Urgent1} \cup T_{Urgent2}$,
- $IA = IA_1 \cup IA_2$,
- $OA = OA_1 \setminus \{(t, out_1) \mid t \in T\} \cup OA_2 \cup \{(t, in_2) \mid (t, out_1) \in OA_1\}$,
- $g((p,t)) = \begin{cases} g_1((p,t)) & \text{if } (p,t) \in IA_1 \\ g_2((p,t)) & \text{if } (p,t) \in IA_2 \\ [0, \infty) & \text{otherwise} \end{cases}$
- $w(a) = \begin{cases} w_1(a) & \text{if } a \in IA_1 \cup OA_1 \\ w_2(a) & \text{if } a \in IA_2 \cup OA_2 \\ 1 & \text{otherwise} \end{cases}$,

- $Type(a) = \begin{cases} Type_1(a) & \text{if } a \in IA_1 \cup OA_1 \\ Type_2(a) & \text{if } a \in IA_2 \cup OA_2 \\ Normal & \text{otherwise} \end{cases}$,
- $I(p) = \begin{cases} I_1(p) & \text{if } p \in P_1 \\ I_2(p) & \text{if } p \in P_2 \end{cases}$,
- $in = in_1$, and
- $out = out_2$.

The sequential composition of TARWFN is local soundness preserving, as stated in the following theorem.

Theorem 5.1 (Sequential composition of TARWFN preserves local soundness)**.** *Let* $N_1 = (P_1, T_1, T_{Urgent1}, IA_1, OA_1, g_1, w_1, Type_1, I_1, in_1, out_1), N_2 = (P_2, T_2, T_{Urgent2}, IA_2, OA_2, g_2, w_2, Type_2, I_2, in_2, out_2)$ *be locally sound well-behaved TARWFNs. Then the sequential composition* $N_1; N_2$ *is locally sound.*

Proof. For $N_1; N_2$ to be locally sound we need to establish that the composition is indeed a TARWFN and that it satisfies the soundness constraints.
Given that N_1, N_2 are valid TARWFNs, the composition is also a valid TARWFN, because it satisfies the following constraints:

The requirements of the composition states that only interface places can be shared. From this we get that $P_{interface} \cap P_{status} = \emptyset$, $P_{interface} \cap P_{normal} = \emptyset$ and $P_{normal} \cap P_{status} = \emptyset$ also hold for the composition, because N_1 and N_2 both are valid TARWFN.

1. $in = in_1 \in P_{normal}$ is a unique place with ${}^\bullet in = \emptyset$.
2. $out = out_2 \in P_{normal}$ is a unique place with $out^\bullet = \emptyset$.
3. As out_1 is merged with in_2, in_2 gets the preset of out_1 and so every place $p \in P \setminus \{in, out\}$ has a non-empty pre- and postset.
4. We do not add new transitions, so for every $t \in T$ the preset is non-empty.
5. We do not add new interface places, so for all $p \in P_{interface}$, the invariant makes the place untimed.
6. We do not add arcs from interface places, so for all $(p, t) \in IA, p \in P_{interface}$, the guard is untimed.
7. We do not add arcs to interface places either, so no arcs to or from interface places are transport arcs.
8. We do not add arcs to $out = out_2$, so for all $(t, out) \in OA$, $Type(t, out) \neq Transport$.

For the composition to satisfies the local soundness constraints defined in Definition 17, we need to show that it is also well-behaved. The composition is well-behaved because it satisfies the following constraints for all markings $M \in [M_{passive}\rangle$reachable from a passive marking $M_{passive} \in \mathcal{M}_{passive}$ taken from Definition 16.

- Since N_1 and N_2 are both well-behaved M cannot have a token in a normal place, since all behavior is local in each of the nets. Only interface places are shared.

- Again, neither N_1 nor N_2 is capable of altering the distribution of tokens in interface places in a passive marking and so the composition cannot either.
- Both parts can delay independently of each other, so the composition is capable of delaying in a passive marking as well.

Lastly we need to prove that the composition satisfy the two local soundness constraints for every marking $M \in [M_{in}\rangle$ reachable from an initial marking $M_{in} \in \mathcal{M}_{initial}$.

- We know that N_1 is locally sound, so from every initial marking $M_{in} \in \mathcal{M}_{initial}$ we can reach a marking with a token in out_1 (corresponding to a final marking of N_1), and since the composition is sequential and out_1 and in_2 is merged after reaching a final marking of N_1 the execution of N_2 begins with a restricted set of initial markings. Now we have that the restricted set of initial markings is a subset of $\mathcal{M}_{initial2}$. Since N_2 as it is locally sound, all initial markings in $\mathcal{M}_{initial2}$ must be able to reach a final marking of N_2, and so also for all initial markings in the subset. Lastly as $out_2 = out$ this is also a final marking for the composition of the two workflows. And as such from every marking $M \in [M_{in}\rangle$ reachable from an initial marking $M_{in} \in \mathcal{M}_{initial}$ we can reach a final marking.
- And likewise if $|M(out)| \geq 1$ then it must be a final marking. We have just showed that N_2 will get enabled with a subset of the possible initial markings, and since N_2 is locally sound, then for all markings M reachable in the composition if $|M(out)| \geq 1 \Rightarrow M \in \mathcal{M}_{final}$.

□

Theorem 5.2 (Sequential composition of TARWFN preserves strong local soundness)**.** *Let* $N_1 = (P_1, T_1, T_{Urgent1}, IA_1, OA_1, g_1, w_1, Type_1, I_1, in_1, out_1)$, $N_2 = (P_2, T_2, T_{Urgent2}, IA_2, OA_2, g_2, w_2, Type_2, I_2, in_2, out_2)$ *be strong locally sound TARWFNs. Then the sequential composition* $N_1; N_2$ *is strong locally sound.*

Proof. The proof follows the same arguments as the proof of Theorem 5.1. □

The composition is soundness preserving and we would like to be able to calculate the bounds for the execution times. The following theorem gives those bounds.

Theorem 5.3. *Let* N_1, N_2 *be strongly sound TARWFNs with minimum execution times* min_1, min_2 *and maximum execution times* max_1, max_2*, respectively. Then for the sequential composition* $N_1; N_2$ *the minimum execution time min is bounded by* $min \geq min_1 + min_2$ *and the maximum execution time max by* $max \leq max_1 + max_2$*.*

Proof. min:
Since the composition is sequential, the minimum execution time reduces to the minimum execution time of N_1 followed by the minimum execution time of N_2 given the restricted set of initial markings. We are given min_1 for N_1 and the

composition does not alter the behavior of N_1, so this will stay the same. The set of initial markings $\mathcal{M}'_{initial2}$ for N_2 will be a subset of $\mathcal{M}_{initial2}$ for N_2, which might remove runs with the minimum execution time of N_2 and so the minimum execution time of N_2 will be at least min_2. Formally, the minimum execution time of the composition comprises to

$$min = min_1 + \min_{M \in \mathcal{M}'_{initial2}} (M \rightarrow^* M_{out})$$

$$min \geq min_1 + min_2$$

max:
The proof for the upper bound follows the same arguments as above. □

Beware that these bounds may not be tight since some initial markings in N_2 might be disabled because of the final markings of N_1. An example hereof

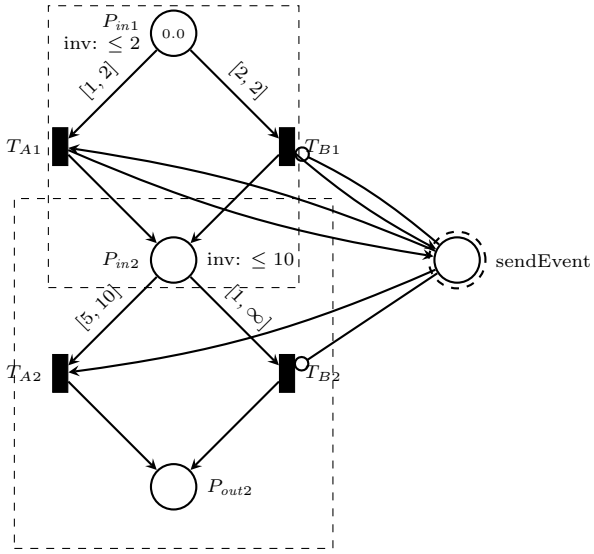


Figure 8: A locally sound composition with an higher minimum execution time than the sum of the minimum execution times of the two components.

can be seen in Figure 8. The workflows each have minimum execution times of 1, when run independently, while their composition has a minimum execution time of 6.

Our workflow composition operator is not complete though. See Figure 9, where the composition is locally sound, but the second TARWFN illustrated by the boxes is not locally sound. As an initial marking is defined by having an arbitrary distribution of tokens in interface places, the initial marking with no tokens in $p_{interface}$ does not lead to a final marking, and as such, the workflow is not locally sound. This situation can never occur in the sequential composition of the two workflows, as every final marking of the first workflow leaves a token in the $p_{interface}$ for the initial marking of the second workflow.

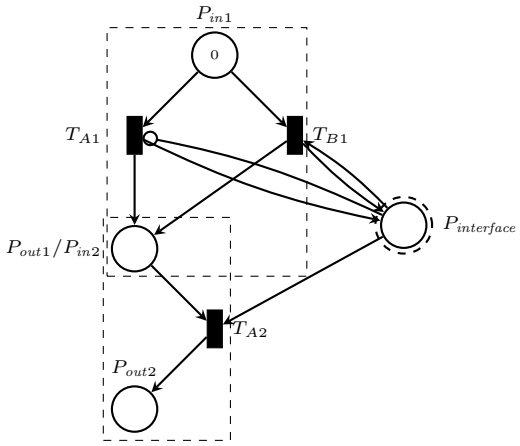


Figure 9: Example of a locally sound sequential composition, where the second workflow is not locally sound on its own.

6 Soundness Checking

Checking (strong) local soundness for the class of TARWFN is undecidable. To get to this result, we can reduce the problem to checking (strong) soundness for the class of TAWFN.

Theorem 6.1. *Local soundness is undecidable for timed-arc resource workflow nets.*

Proof Idea. The set of all TARWFNs is a superset of the set of all TAWFNs. Let $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ be a simplified TARWFN with $P_{status} \cup P_{interface} = \emptyset$. Then N is simply a TAWFN and therefore checking local soundness for N is the same as checking soundness for N. Soundness for the class of TAWFN is undecidable [24] and so it is also undecidable for the superclass of TARWFN. □

We will therefore in the rest of the report restrict the TARWFN to be 1-safe TARWFN.

Definition 22 (1-safe marking)**.** Let $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ be a TARFWFN. A marking M of N is 1-safe iff for all places $p \in P$, we have $|M(p)| \leq 1$.

We call the set of all 1-safe markings $\mathcal{M}_{safe}$.

Definition 23 (1-safe workflow)**.** A TARWFN $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ is 1-safe iff for all reachable markings $M \in [M_{in}\rangle$, where $M_{in} \in \mathcal{M}_{safe} \cap \mathcal{M}_{initial}$, M is 1-safe.

In the following section we will also restrict the status places such that only one status place can be *active* at a time[1].

Definition 24 (1-active marking)**.** Let $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ be a TARWFN. A marking M of N is said to be 1-active iff $\sum_{p \in P_{status}} |M(p)| \leq 1$.

We call the set of all 1-active markings $\mathcal{M}_{active}$.

Definition 25 (1-active workflow)**.** A TARWFN $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ is 1-active iff for all reachable markings $M \in [M_{in}\rangle$, where $M_{in} \in \mathcal{M}_{active} \cap \mathcal{M}_{initial}$, M is 1-active.

For the subclass of 1-safe, 1-active well-behaved TARWFN (strong) local soundness checking is decidable. We will in this section prove decidability for this subclass by reduction to (strong) soundness for bounded TAWFN for which there exists efficient soundness checking algorithms [24].

To reduce the problem to soundness checking of bounded TAWFN, the original TARWFN needs to be transformed to a TAWFN. Following this, there are several challenges to overcome in order to use the existing algorithms directly:

(A) **1-safe check.** How do we check if the TARWFN is 1-safe in all possible configurations, as defined in Definition 23?

(B) **1-active check.** How do we check if the TARWFN is 1-active in all possible configurations, as defined in Definition 25?

(C) **Well-behaved TARWFN.** How do we check that the TARWFN is well-behaved, as defined in Definition 16?

(D) **Soundness.** How do we check soundness and strong soundness, as defined in Definition 17 and Definition 18?

6.1 Preliminaries

We begin by introducing two transformations used to solve the problems of how to simulate initial and final markings of a TARWFN in a TAWFN. The transformations are used to check (A)–(D) as stated in the previous section.

6.1.1 Solution to initial markings

As an initial marking in a TAWFN can only contain a token in the *in* place, we need to simulate running the original workflow with each of the different initial markings. The initial marking of a TARWFN can contain an arbitrary distribution of tokens in interface places and a token in either of the status

[1]For multiple status places it proved difficult in practice to model locally sound workflows if it was not restricted to 1-active.

places. Given a TARWFN N, let $N_{initial}$ be a TAPN where we simulate initial markings in the following way. An illustration of this process can be seen in Figure 10. We define $\mathcal{C}$ to be $\mathcal{C} = \max_{p \in P_{status}} C_{max}(p) + 1$. For each possible

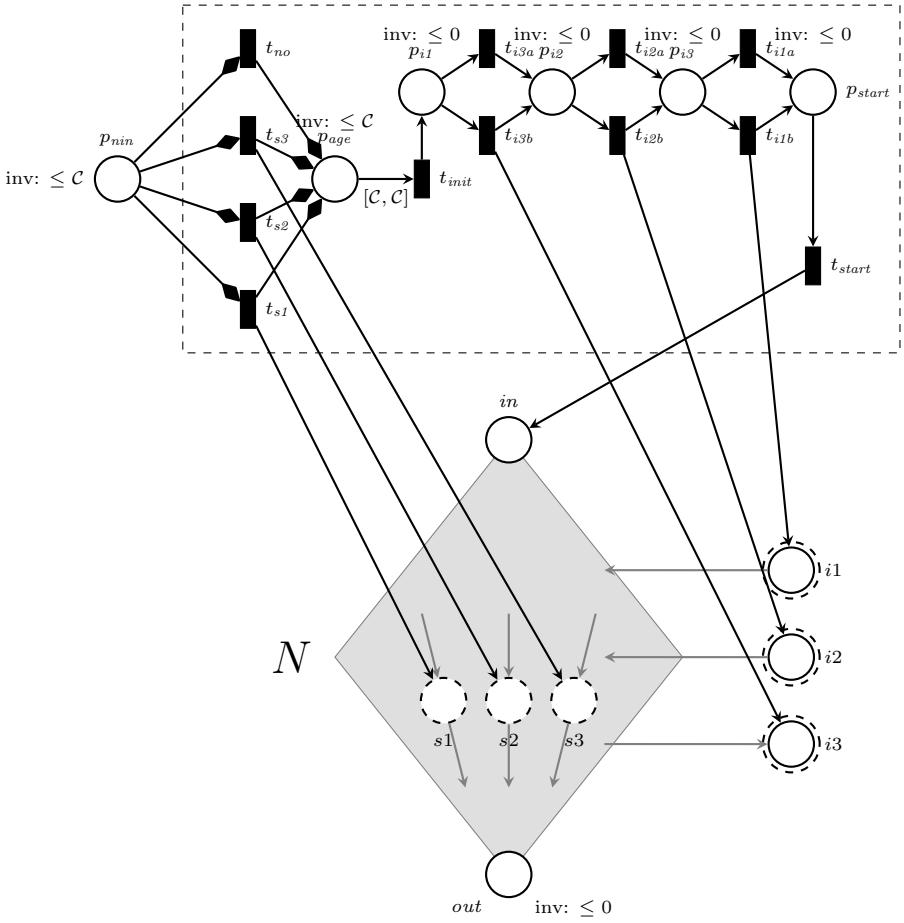


Figure 10: The net $N_{initial}$

combination of status and interface place we simulate the corresponding initial marking as follows. We add two places to the beginning of the net, p_{nin} and p_{age} and a transition for each of the status places. Each of the places have invariants corresponding to $\mathcal{C}$ and are connected with transport arcs through each of the status place transitions. This setup allows us to be able to place every possible age of token into every status place. Following this, we add a place and two transitions for every interface place. The places and transitions are chained, allowing the net to place tokens into the interface places by choosing between the two transitions for each interface place. Additionally, as interface places do not contain timing information, we simply add a ≤ 0 invariant to each of the

corresponding places. Finally, we add an arc to the old *in* place triggering the start of the actual workflow.

Definition 26. We define $\mathcal{M}_{generated} = \{M \mid M \in \mathcal{M}_{safe} \cap \mathcal{M}_{active} \cap \mathcal{M}_{initial} \wedge \forall p \in P_{interface}.|M(p)| \geq 1 \Rightarrow M(p) = \{(p,0)\} \wedge \forall p \in P_{status} \forall x \in M(p).x \leq \mathcal{C}\}$.

For a well-behaved, 1-safe, and 1-active TARWFN, $\mathcal{M}_{generated}$ corresponds to the set of simulated initial markings in $N_{initial}$.

Lemma 4. *Let N be a well-behaved TARWFN. Then $\mathcal{M}_{generated} \subseteq \mathcal{M}_{initial}$.*

Proof. From Definition 26 and the construction in Figure 10 we have that $\mathcal{M}_{generated} \subseteq \mathcal{M}_{initial}$. □

Lemma 5. *Let N be a well-behaved TARWFN. Then for all $M \in \mathcal{M}_{initial}$ there is $M' \in \mathcal{M}_{generated}$ such that $cut(M) = cut(M')$.*

Proof. From Definition 14 we have that interface places must be untimed and so for all $p \in P_{interface}$ we have $C_{max}(p) = -1$, and so all tokens in interface places will have the age reduced to 0 by *cut*, which are exactly the ages of tokens in markings in $\mathcal{M}_{generated}$.
For status places we know *cut* will reduce any token in p of ages above $C_{max}(p)+1$ to $C_{max}(p)+1$ and since for all $p \in P_{status}$ we have $\mathcal{C} \geq C_{max}(p)+1$, we have that all token ages are at most $C_{max}(p)+1$ for all $p \in P_{status}$ and these are all possible by definition of $\mathcal{M}_{generated}$.
Both sets have a last token, $(in, 0)$. This token will always have an age, which is at most $C_{max}(in)+1$ and so stays the same for the *cut* marking. □

6.1.2 Solution to final markings

A final marking of a TAWFN can only contain tokens in the *out* place and as such, we need to be able to remove the tokens in status and interface places when the workflow reaches one of the final markings from the TARWFN. This is achieved by creating a *cleanup* phase, where we remove all tokens from the status and interface places in sequence. The cleanup phase starts when the net reaches the "old" final marking. We ensure with inhibitor arcs that the status or interface place is cleaned before going advancing in the process. The cleanup phase ends when all status and interface places have been cleaned. Given a TARWFN N, let N_{final} be a net where we remove tokens in status and interface places when the final marking has been reached. An illustration of this process can be seen in Figure 11. Additionally to the illustration we also add a new place p_{block}, which has inhibitor arcs to all transitions in N. A token is placed in p_{block}, when the cleanup phase begins and is consumed again when $t_{cleanup}$ fires.

6.1.3 Transformed model

We define N_{trans} as a TARWFN N where both transformations in Subsubsection 6.1.1 and Subsubsection 6.1.2 have been applied. The final transformed

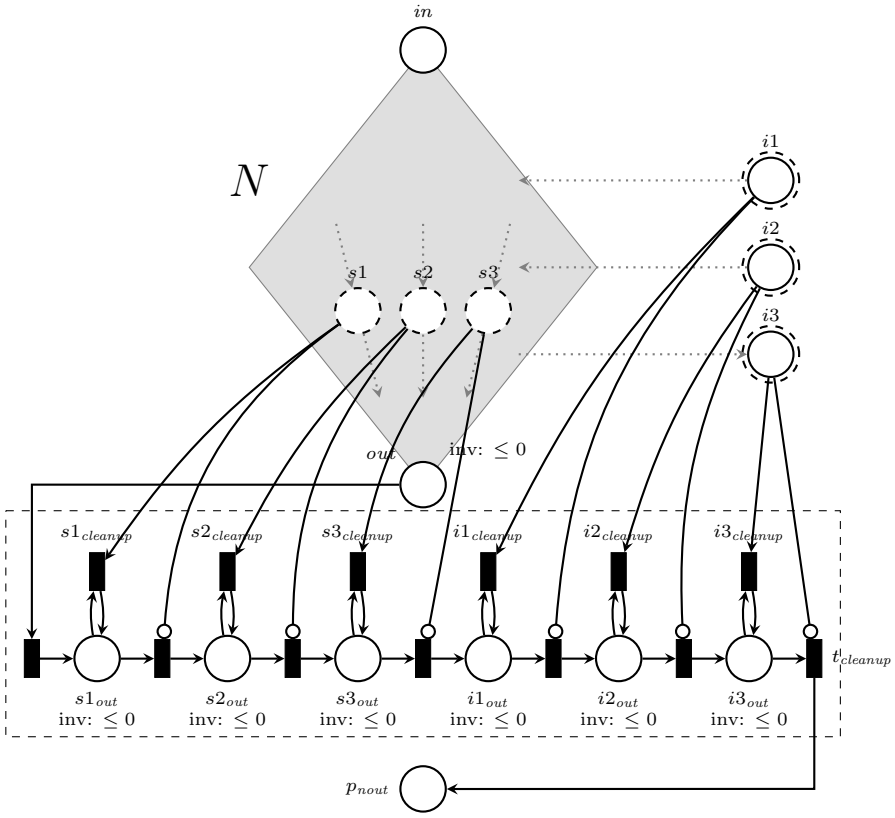


Figure 11: The net N_{final}

model can be seen in Figure 12, where *Initial* and *Final* contains the transformations contained within the dashed boxes in Figure 10 and Figure 11 respectively. The construction of the transformed workflow can be seen in Definition 27

Definition 27 (Transformation)**.** Let $N = (P_{interface} \cup P_{status} \cup P_{normal}, T, T_{Urgent}, IA, OA, g, w, Type, I, in, out)$ be a TARWFN. The workflow $\mathcal{W}$ is an algorithm that on input N constructs $N_{trans} = (P_{trans}, T_{trans}, T_{transUrgent}, IA_{trans}, OA_{trans}, g_{trans}, w_{trans}, Type_{trans}, I_{trans}, in_{trans}, out_{trans})$ and $M_{in-trans}$ such that

- $P_{trans} = P \cup \{p_{nin}, p_{age}, p_{nout}, p_{start}, p_{block}\} \cup \{p_i \mid i \in P_{interface}\} \cup \{s_{out} \mid s \in P_{status}\} \cup \{i_{out} \mid i \in P_{interface}\}$,
- $T_{trans} = T \cup \{t_{init}, t_{start}, t_{cleanup}, t_{no}\} \cup \{t_{i_a} \mid i \in P_{interface}\} \cup \{t_{i_b} \mid i \in P_{interface}\} \cup \{t_s \mid s \in P_{status}\} \cup \{i_{cleanup} \mid i \in P_{interface}\} \cup \{s_{cleanup} \mid s \in P_{status}\} \cup \{i_{clean} \mid i \in P_{interface}\} \cup \{s_{clean} \mid s \in P_{status}\}$,
- $T_{Urgent-trans} = T_{Urgent}$,
- $IA_{trans} = IA \cup$
  $\{(p_{nin}, t_{no}), (p_{age}, t_{init}), (p_{start}, t_{start}), (out, s1_{clean}), (p_{block}, t_{cleanup})\} \cup \{(p_{nin}, t_s) \mid$

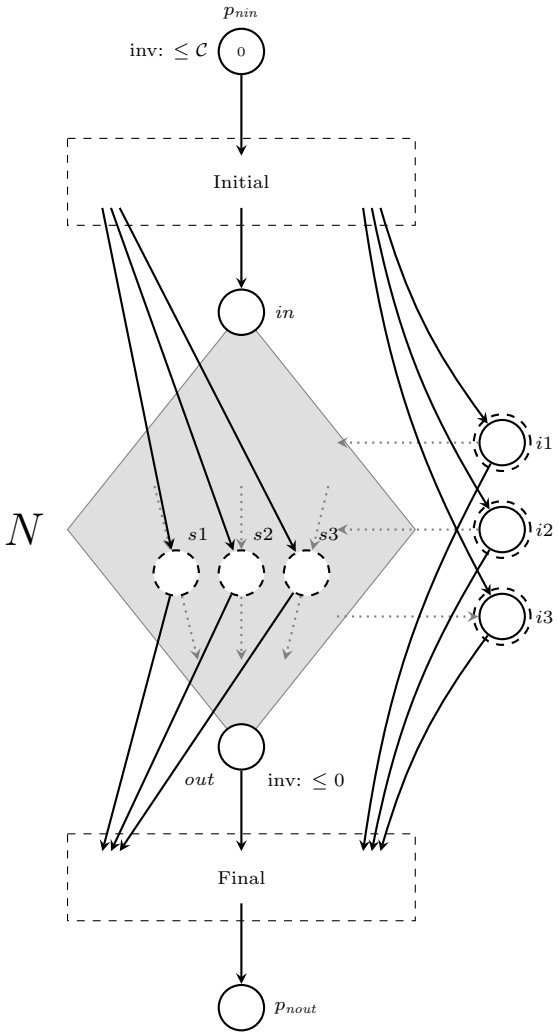


Figure 12: The net N_{trans}

$s \in P_{status}\} \cup \{(p_i, t_{p_{i_a}}) \mid i \in P_{interface}\} \cup \{(p_i, t_{p_{i_b}}) \mid i \in P_{interface}\} \cup \{(p_i, i_{cleanup}) \mid i \in P_{interface}\} \cup \{(p_s, s_{cleanup}) \mid s \in P_{status}\} \cup \{(s_{out}, i_{cleanup}) \mid i \in P_{interface}\} \cup \{(i_{out}, s_{cleanup}) \mid s \in P_{status}\} \cup \{((s+1)_{out}, i_{cleanup}) \mid i \in P_{interface} \wedge s \neq |P_{status}|\} \cup \{((i+1)_{out}, s_{cleanup}) \mid s \in P_{status} \wedge i \neq |P_{interface}|\} \cup \{(s, i1) \mid s = |P_{status}|\} \cup \{(i, t_{cleanup}) \mid i = |P_{interface}|\} \cup \{(p_{block}, t) \mid (p, t) \in IA\}$,

- $OA_{trans} = OA \cup$
  $\{(t_{init}, p_{i1}), (t_{cleanup}, p_{nout}), (t_{start}, p_{in}), (t_{no}, p_{age}), (s1_{clean}, p_{block})\} \cup$
  $\{(t_s, p_{age}) \mid s \in P_{status}\} \cup \{(t_{i_a}, p_{i+1}) \mid i \in P_{interface} \wedge i \neq |P_{interface}|\} \cup$

$\{(t_{i_b}, p_{i+1}) \mid i \in P_{interface} \wedge i \neq |P_{interface}|\} \cup \{(t_{i_a}, p_{start}) \mid i \in P_{interface} \wedge i = |P_{interface}|\} \cup \{(t_{i_b}, p_{start}) \mid i \in P_{interface} \wedge i = |P_{interface}|\} \cup \{(t_{i_b}, i) \mid i \in P_{interface}\} \cup \{(t_s, s) \mid s \in P_{status}\} \cup \{(i_{cleanup}, i_{out}) \mid i \in P_{interface}\} \cup \{(s_{cleanup}, s_{out}) \mid s \in P_{status}\} \cup \{(i_{clean}, i_{out}) \mid i \in P_{interface}\} \cup \{(s_{clean}, s_{out}) \mid s \in P_{status}\}$,

- $g_{trans}((p,t)) = \begin{cases} [\mathcal{C}, \infty) & \text{if } (p,t) = (p_{age}, t_{init}) \\ g((p,t)) & \text{if } (p,t) \in IA \\ [0, \infty) & \text{otherwise} \end{cases}$
- $w_{trans} = w$,
- $Type_{trans}((p,t)) = \begin{cases} Inhib & \text{if } p \in P_{interface} \cup P_{status} \text{ and } t = s_{clean} \\ Inhib & \text{if } p \in P_{interface} \cup P_{status} \text{ and } t = i_{clean} \\ Inhib & \text{if } p \in P_{interface} \cup P_{status} \text{ and } t = t_{cleanup} \\ Inhib & \text{if } p = p_{block} \text{ and } (p,t) \in IA \\ Transport & \text{if } (p,t) = (p_{nin}, t_s) \text{ where } s \in P_{status} \\ Type((p,t)) & \text{if } (p,t) \in IA \\ Normal & \text{otherwise} \end{cases}$,
- $Type_{trans}((t,p)) = \begin{cases} Transport_m & \text{if } (t,p) = (t_s, p_{age}) \text{ where } s \in P_{status} \\ Type((t,p)) & \text{if } (t,p) \in OA \\ Normal & \text{otherwise} \end{cases}$,
- $I_{trans}(p) = \begin{cases} [0, \mathcal{C}] & \text{if } p = p_{nin} \text{ or } p = p_{age} \\ I(p) & \text{if } p \in P \text{ and } p \neq p_{out} \\ [0,0] & \text{otherwise} \end{cases}$

Lemma 6. *The transformed net N_{trans} is a TAWFN.*

Proof. From Definition 11 we have four requirements that needs to be fulfilled for the constructed TAPN to be a valid TAWFN.

1. $p_{nin} \in P'$ and ${}^\bullet p_{nin} = \emptyset$.
2. $p_{nout} \in P'$ and $p_{nout}^\bullet = \emptyset$.
3. No arcs are removed so all $p \in P \setminus \{in, out\}$ still have ${}^\bullet p \neq \emptyset$ and $p^\bullet \neq \emptyset$ and for the added places which are not the new input and output places they have nonempty pre- and postset by the construction, see Figure 10 and Figure 11, and the old input place now has an input arc such that ${}^\bullet in = t_{start}$, and the old output place an output arc such that $out^\bullet = t_{cleanup}$.
4. No arcs are removed, so all transitions $t \in T$ have a nonempty presets. For the newly added transitions no presets are empty. This is clear from looking at Figure 10, Figure 11 and Figure 12.

□

6.2 Solutions to problem A & B

In order to be able to give a conclusive answer regarding soundness, we need to ensure that the transformed workflow is both 1-active and 1-safe. We therefore construct a TAPN $N_{AB} = (P_{AB}, T_{AB}, T_{UrgentAB}, IA_{AB}, OA_{AB}, g_{AB}, w_{AB},$

$Type_{AB}, I_{AB}, in_{AB}, out_{AB}$) that uses the initialization phase introduced in Subsubsection 6.1.1, where the net $N_{initial}$ is modified by adding a single place, p_{count}, with input and output arcs such that a token is added to the place p_{count} whenever a token is added to a status place, and removed whenever a token is removed from a status place. For the construction see Figure 13.

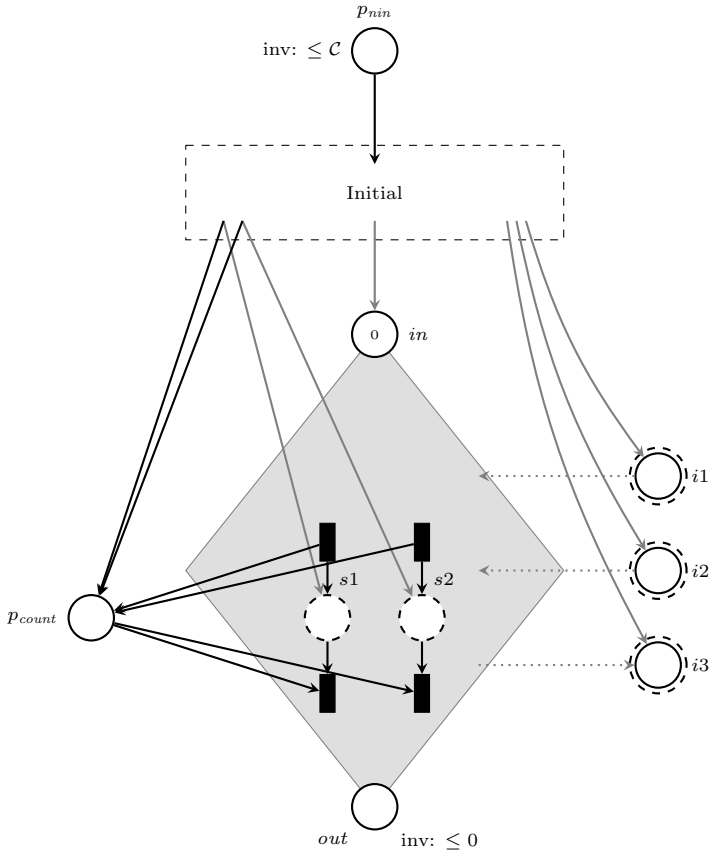


Figure 13: The TAPN N_{AB} used to check if a TARWFN N is 1-safe and 1-active.

After the construction it is simply a matter of exploring the state space to find the bound for each place in the net. If every place has a bound smaller or equal to 1, the net is 1-safe. Additionally, if the bound for p_{count} is less or equal to 1, the net is 1-active.

Lemma 7. *The TARWFN N is 1-safe if and only if for all $p \in P_{AB} \setminus \{p_{count}\}$ in N_{AB} the bound for p is less or equal to* 1.

Proof. "$\Rightarrow$"
Let the TARWFN N be 1-safe. Then for all markings $M \in [M_{in}\rangle$, where $M_{in} \in \mathcal{M}_{initial}$ we have that for all places $p \in P_{AB}$, $|M(p)| \leq 1$ and so must the

bound for p be less or equal to 1 for all $p \in P_{AB} \setminus \{p_{count}\}$. By Lemma 4 and construction of N_{AB} we have that for all $p \in P$ of N, we have that the bound for p is less or equal to 1.
"$\Leftarrow$"
Let the bound for p be less or equal to 1 for all $p \in P_{AB} \setminus \{p_{count}\}$. Then by Lemma 5 we have that all the behavior from N is still possible in N_{AB}. That means that for all markings $M \in [M_{in}\rangle$ reachable from some $M_{in} \in \mathcal{M}_{initial}$ and for all $p \in P$, we have $|M(p)| \leq 1$ and so N is 1-safe by Definition 23. □

Theorem 6.2. *Checking if a TARWFN is 1-safe is decidable.*

Proof. Start searching the state space for a marking M such that $M(p) \geq 2$ for some place $p \in P_{AB} \setminus \{p_{count}\}$. If such a marking exists the search terminates and by Lemma 7 we have that the net is not 1-safe. If such a marking does not exists the TARWFN must be 1-safe. Since the TARWFN is 1-safe and the number of status places finite, the place p_{count} is bounded and therefore N_{AB} is bounded. Because N_{AB} is bounded the state space of N_{AB} is finite and at some point the whole net is searched and the search terminates [25]. □

Lemma 8. *A TARWFN N is 1-active if and only if the bound for p_{count} is less or equal to 1 in N_{AB}.*

Proof. "$\Leftarrow$"
Let the bound for p_{count} be less than or equal 1. By construction $|M(p_{count})| = \sum_{p \in P_{status}} |M(p)|$, see Figure 10, for all reachable markings $M \in [M_0\rangle$. Since the bound for p_{count} is less than or equal 1 then $\sum_{p \in P_{status}} |M(p)| \leq 1$ for all reachable markings M and so must N_{AB} be 1-active. By Lemma 5 we have that then N is also 1-active.
"$\Rightarrow$"
Let the TARWFN N be 1-active. Then $\sum_{p \in P_{status}} |M(p)| \leq 1$ for all markings $M \in [M_{in}\rangle$, where $M_{in} \in \mathcal{M}_{initial}$, and by the construction of N_{AB} and Lemma 4 we have that $\sum_{p \in P_{status}} |M(p)| = |M(p_{count})|$ for every reachable marking M, and so the bound for p_{count} is less than or equal 1. □

Theorem 6.3. *Checking if a 1-safe TARWFN is 1-active is decidable.*

Proof. The net is 1-safe and therefore bounded. It then follows from [25] that searching through the whole state space is deciable and the rest of the argument follows from Lemma 8. □

6.3 Solution to problem C

For a TARWFN to be well-behaved, it has to satisfy three conditions as defined in Definition 16.

The conditions state that every marking reachable from a passive marking must also be a passive marking, that in every marking reachable from

a passive marking the distribution of tokens in interface places must remain unchanged, and that every marking reachable from a passive marking must be time computation divergent. We construct a timed-arc Petri net, $N_C = (P_C, T_C, T_{UrgentC}, IA_C, OA_C, g_C, w_C, Type_C, I_C, in_C, out_C)$ that allows us to verify these three conditions. The construction can be seen in Figure 14.

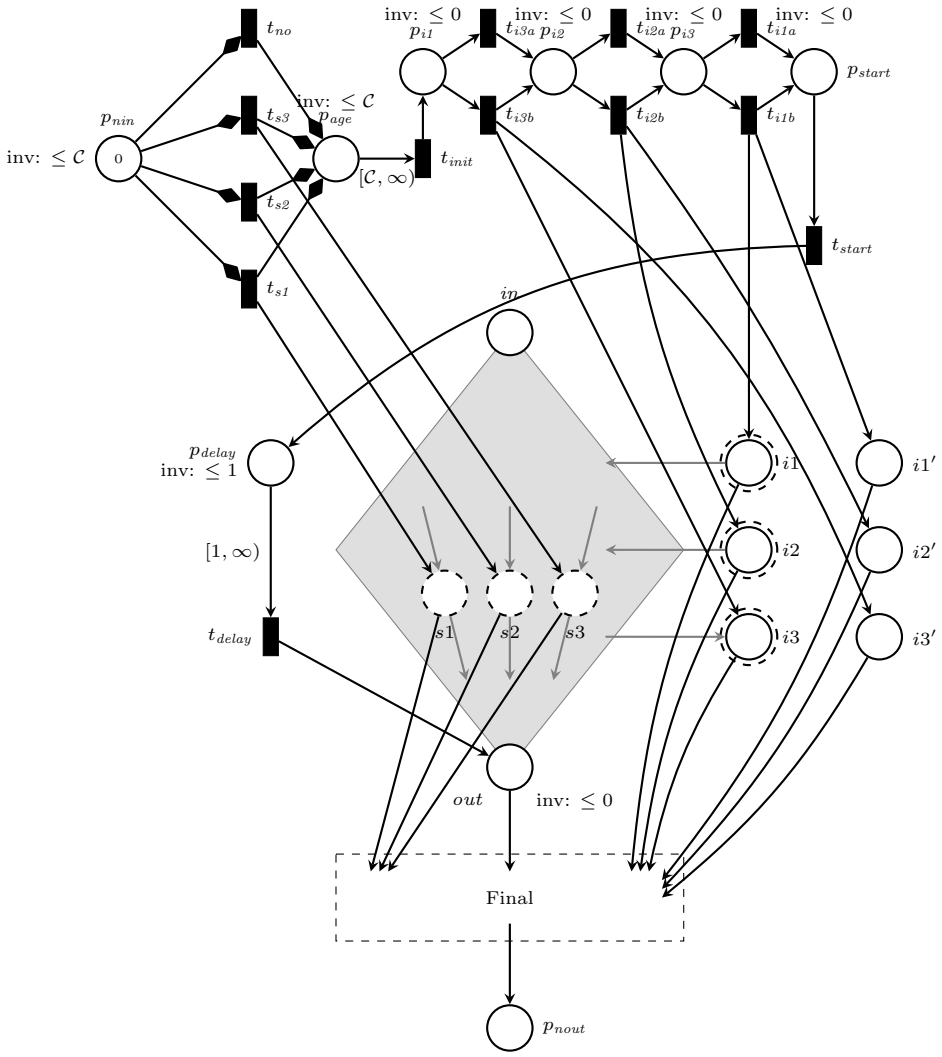


Figure 14: The TAWFN N_C, used to verify that a TARWFN N is well-behaved.

The initial net follows from N_{trans} from Figure 12, with the addition of an added place for each interface place, in which a token is placed whenever a token is placed in the original interface place during the initial phase, together with

ensuring a time delay of 1. The net also shortcuts the execution of the net going directly to *out* from p_{delay}. Note that an additional transition has been added, with input arc from p_{delay} with a guard of $[2, \infty]$ and an output arc to *in* in order for the workflow to be structurally correct, so ${}^\bullet in \neq \emptyset$. The transition is never active due to the invariant, and as thus, the transition has been omitted from the drawing.

As a token is placed in the p_{delay} place, we have placed tokens in status and interface places and the original TARWFN is in a passive marking from $\mathcal{M}_{generatedPassive=}\{M \setminus \{(in, 0)\} \mid M \in \mathcal{M}_{generated}\}$.

To verify the first two conditions, we search through the state space from the point where there is a token in p_{delay}, and by keeping the token there, we can visit every marking reachable from some initial passive marking $M_{passive} \in \mathcal{M}_{generatedPassive}$. From that point it is then a matter of determining whether or not a token is placed in a normal place, and whether the amount of tokens in an interface place differs from its copy. Formally the net is verified with the query $\phi = \neg\text{EF}(p_{delay} = 1 \wedge ((p_1 = 1 \vee \cdots \vee p_n = 1) \vee (i_1 \neq i'_1 \vee \cdots \vee i_m \neq i'_m))$ where $\{p_1, \ldots, p_n\} = P_{normal}$ and $\{i_1, \ldots, i_m\} = P_{interface}$.

Finally, the third condition states that every passive marking must be time computation divergent. To validate this we use the fact that before the workflow starts, we must always wait at least $C_{max}(p_{status}) + 1$, where p_{status} is the currently active status place. Since $C_{max} + 1$ preserves the full behavior given token ages and that interface places must be untimed, it is sufficient to check if a time delay of 1 can be done in all simulated passive markings, which corresponds to checking that from all markings reachable from some initial marking it should be true that we can reach a marking with a token in p_{delay} until we reach a marking M where $(p_{delay}, 1) \in M$. This cannot be checked with the logic defined in Definition 8, but it can be checked with a soundness check of N_C.

Lemma 9. *The TARWFN N is well behaved if and only if ϕ is satisfied and N_C is sound.*

Proof. "$\Rightarrow$"
Let the TARWFN N be well-behaved. Then no passive markings will be able to move tokens to normal places and the distribution of tokens will remain unchanged in interfaces. Then ϕ must be true, because it is not possible to have a token in p_{delay}, which corresponds to a passive marking and have a token placed in a normal place, and neither is it possible to add or remove tokens to interface places. Likewise it should always be possible to make a delay of any length and therefore also of length 1 in a passive marking and so N_C must be sound by [24], because from the construction of N_C it is always possible to reach p_{delay}, and since the delay is always possible to reach *out* and from *out* it is always possible to reach p_{nout}.
"$\Leftarrow$"
Let ϕ hold for N_C and let N_C be sound. Then no marking in $\mathcal{M}_{generatedPassive}$ could reach a marking in which there still is a token in p_{delay} and a token in a normal place nor alter the distribution of tokens in interface places. This satisfy the first two conditions of well-behavedness in Definition 16 by Lemma 5.

Since N_C is sound then from all reachable markings it is possible to reach a final marking, which have a token in p_{nout} by [24]. Since all traces must go through p_{delay}, where we must delay 1, we have that from every marking $M \in \mathcal{M}_{generatedPassive}$ it is possible to make a delay of 1. Since all tokens in status places have ages that are at most $\mathcal{C}$ and interface places must be untimed it follows from Lemma 1 that an arbitrary longer delay is also possible from all M, such that $(p_{delay}, 0) \in M$. This satisfies the third condition of Definition 16 and so must the TARWFN N be well-behaved. □

Theorem 6.4. *Checking if a 1-safe, 1-active TARWFN is well-behaved is decidable.*

Proof. The proof follows from Lemma 7 and [26] that states that reachability checking for bounded marked TAPNs is decidable and we can search the whole state space with the TAPAAL query ϕ. Together with [24] that states that soundness checking of bounded TAWFNs is decidable, and a 1-safe TARWFN N can produce a bounded TAWFN N_C. □

6.4 Solution to problem D

If N is 1-active, bounded (1-safe) and well-behaved we are able to check that the TAWFN N_{trans} is (strongly) sound, using the (strong) soundness algorithm by Mateo et al. [24], which provides us with a soundness result and if (strongly) sound a minimum (and maximum) execution time. As can be seen in Subsection 6.1, the transformation includes an added delay of $\mathcal{C} = \max_{p \in P_{status}} C_{max}(p)$ to every execution in order to create tokens in status places of the right ages, so in order to determine the actual minimum- and maximum execution times of the N, we subtract $\mathcal{C}$ from the soundness results from the analysis of N_{trans}.

Lemma 10. *If in N_{trans} there is a computation starting from the initial marking $M_{in} = \{(p_{nin}, 0)\}$, $M_{in} \rightarrow^* M_{final}$, where M_{final} is a final marking of N_{trans}, then the computation can be written as $M_{in} \rightarrow^* M \xrightarrow{t_{start}} M'_{in} \rightarrow^* M'_{final} \xrightarrow{cleanup}{}^* M_{final}$, such that $M'_{in} \rightarrow^* M'_{final}$ is a computation in N and M'_{final} is a final marking of N.*

Proof. By construction of N_{trans}, see Figure 12, we can always reach a marking M from M_{in}, where we can fire t_{start} and reach a marking M'_{in}. From Lemma 4 we know that $M'_{in} \in \mathcal{M}_{initial}$. Furthermore we have by construction of N_{trans} that there exists a marking M'_{final} in between M'_{in} and M_{final} such that $M'_{final}(out) = 1$ and for all places $p \in P_{normal}$, $|M'_{final}(p)| = 0$, because only clean up transitions from interface and status places exists, and so M'_{final} must be a final marking of N. Since the construction of N_{trans} is simply N with added behavior before and after the execution of N, $M'_{in} \rightarrow^* M'_{final}$ must also be a computation of N. □

Lemma 11. *Let N be a 1-safe, 1-active well-behaved TARWFN and M_{in} be some initial marking for N. Then if in N there is a computation $cut(M_{in}) \rightarrow^*$*

M_{final}, then there exists a run in N_{trans}, starting from the initial marking of N_{trans}, $M'_{in} = \{(p_{nin}, 0)\}$, $M'_{in} \rightarrow^* cut(M_{in}) \rightarrow M_{final} \rightarrow M'_{final}$, such that M'_{final} is a final marking of N_{trans}.

Proof. From Lemma 5 we have that for every marking $M_{in} \in \mathcal{M}_{initial}$ there exists a marking $M''_{in} \in \mathcal{M}_{generated}$ such that $cut(M_{in}) = cut(M''_{in})$, and by construction of $\mathcal{M}_{generated}$ we know that there exists a marking $M_{gen} \in \mathcal{M}_{generated}$ such that $cut(M''_{in}) = M_{gen}$. And by construction of N_{trans}, see Figure 12, we have that there exists a computation $M'_{in} \rightarrow M_{gen}$. And since $M_{gen} = cut(M_{in})$, there exists a computation $M_{gen} \rightarrow M_{final}$ in N_{trans}. And from there only cleanup transitions is possible until all interface and status places are cleaned and $t_{cleanup}$ can be fired and reach M'_{final}. M'_{final} must be a final marking of N_{trans}, because all places except interface and status places cannot have tokens in them, otherwise M_{final} would not have been a final marking of N. □

Lemma 12. *Let a TARWFN N be well-behaved, 1-active, 1-safe. Then N is locally sound if and only if the transformed TAWFN N_{trans} is sound.*

Proof. Because of Lemma 4, Lemma 5, Lemma 10, Lemma 11, we know that N_{trans} is a TAWFN where we simulate N and we can use [24] to decide soundness of N_{trans}. □

Lemma 13. *Let a TARWFN N be well-behaved, 1-active, 1-safe. Then N is strong locally sound if and only if the transformed TAWFN N_{trans} is strongly sound.*

Proof. Follows from Lemma 4, Lemma 5, Lemma 10, Lemma 11 and [24]. □

Theorem 6.5. *Checking local soundness and strong local soundness is decidable for 1-active, 1-safe TARWFN.*

Proof. Follows directly from Lemma 12 and Lemma 13. □

7 Implementation

Algorithms for verifying that nets are 1-safe, 1-active and well-behaved, as well as algorithms for verifying (strong) local soundness have been implemented in Java in the model checker TAPAAL. A GUI menu for local workflow analysis has been implemented as well, which can be seen in Figure 15. Nets in TAPAAL can be split into several components which can contain *shared places*, places which can exist in multiple components, while still structurally functioning as a single place. As such, the components and the shared places is not a syntactical addition to TAPNs, but serves merely as a means of organizing large nets. Our implemented algorithm works on the assumption that each workflow has its own component, where each interface place is a shared place. The menu allows the user to specify the status and interface places in the workflows, as well as specify an approximate maximum constant used for scaling the constants in the workflow, which can be toggled with the "Exact" option. The "Prelim." toggles

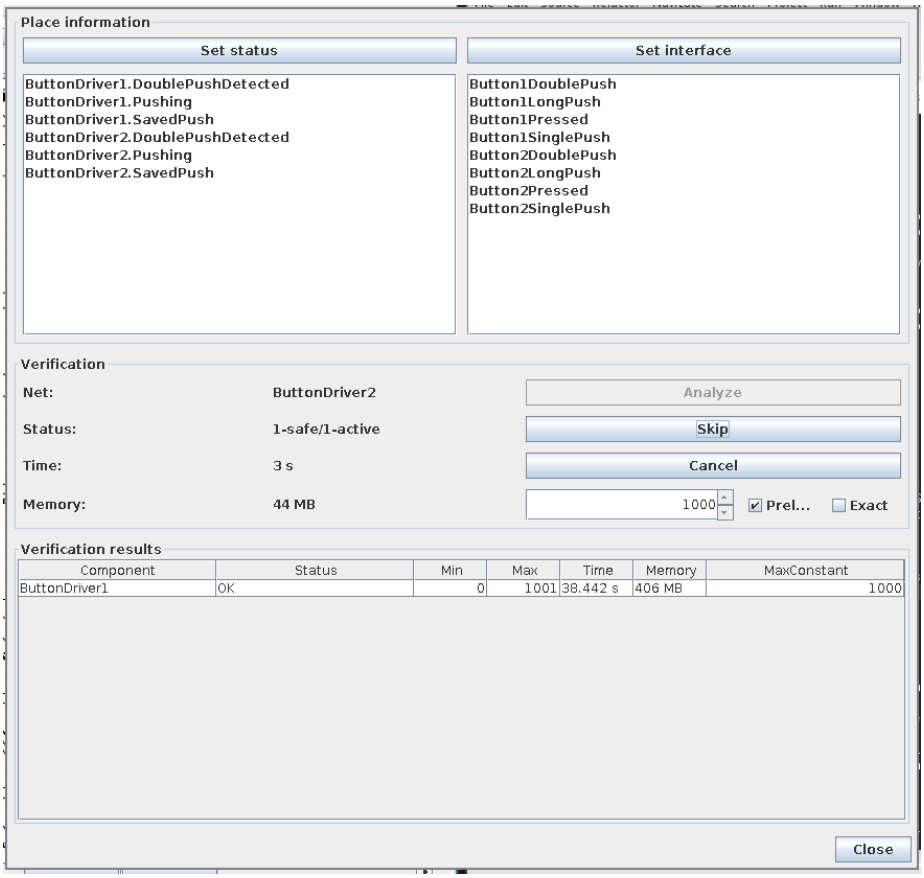


Figure 15: Screenshot of the GUI

whether the analysis should include a check to determine whether the workflow is 1-safe, 1-active and well-behaved. The user is able to monitor the verification, see which net is currently being analysed, which part of the analysis is currently active, as well as the elapsed time and the memory usage. The menu is found in TAPAAL, under the "Tools" menu, labeled "Local workflow analysis". An overview of the different status and interface places associated with the different workflows in TAPAAL, can be seen in Appendix A.

The source code for TAPAAL including our implemented algorithms and GUI is hosted on Launchpad and can be found at:

`https://goo.gl/CpumWj`

The case study TAPAAL models are located in the "models" folder in the root of the repository. In order to obtain C_{max} values for the different workflows,

we have made a modification to the discrete TAPN verification engine used by TAPAAL. The source code for the modified verification engine is similarly hosted on Launchpad and can be found at:

`https://goo.gl/NnrWlB`

8 Case Study: A Smart House

We have chosen to focus on a smart house and its components based on [22]. The smart house consists of a number of different hardware components. 16 buttons, 16 relays, 16 lights, a motion sensor, a buzzer and a controller. The controller in the system runs a program loop where it in each cycle determines the states of the different hardware components producing events, evaluates a number of rules given the events, and sends impulses to turns the lights on or off. The controller also has an alarm state that can be toggled, as well as the ability to differentiate between night and day. The relays in the house are impulse relays, which need an impulse of a specific duration in order for them to change their state between on and off. The impulse must not be too long otherwise the relays will burn out. Due to restrictions on the amount of power the controller is able to supply, a maximum of four relays can receive an impulse at the same moment [22].

We have chosen to model the smart house in TAPAAL including its various hardware components and the different rules, also integrating a number of measured worst case execution times in microseconds into the model [22]. As no information regarding best case execution times was available, we have integrated best case execution times for each model as 25 % of the worst case execution times. All constants in the model are in microseconds. A general overview of the control flow in the modeled controller can be seen in Figure 16. Each of the different components in the controller have been modeled as TAR-WFN and are explained in more detail in the following sections. For each of the examples we have divided the set of interface places into input and output, according to how the interface place is used. This is not an extension to the workflow formalism, but merely a way for us to illustrate our usage of the interface place in the design. In addition to the controller, we have also modeled each of the hardware components in the house, including the buttons and the actual relays connected to the lights and the buzzer.

Event drivers The house features a total of 17 event drivers, 16 for each of the buttons and one for the motion sensor. Each button driver monitors the state of a physical button and determines whether or not a button press has occurred in the system. The movement driver monitors the state of the motion sensor and sends a movement event to the controller. The movement driver has been modeled in such a way that it can only produce an event every two

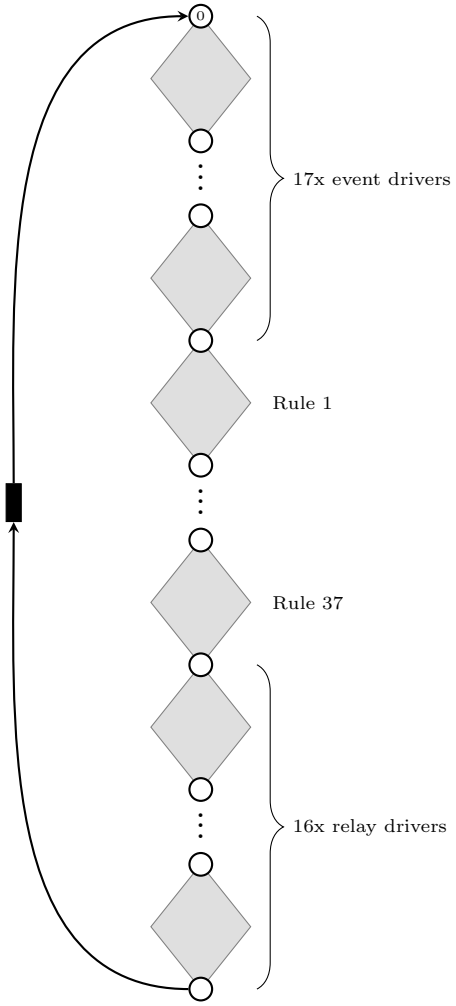


Figure 16: Overview of the controller in the case study smart house

seconds [22]. Table 1 shows the status and interface places of a button driver. Three different types of button press events can occur in the system, illustrated in Figure 17. A single push, which is when the button is pressed for at most 500 ms; a long push, which is when the button is pressed for at least 501 ms; and a double push, which is when the button is pressed for at least 501 ms and at most 1000 ms, followed by a second push within 1000 ms after the start of the first push. The state of the physical button as well as the three events are modeled as interface places. The length of a button press is saved as a token in a status place. The full model of a button driver can be seen in Appendix C.

	Interface	Status
I	ButtonPressed	Pushing
O	ButtonSinglePush	SavedPush
	ButtonLongPush	DoublePushDetected
	ButtonDoublePush	

Table 1: Interface and status places for a button driver. The dashed line separates input and output places

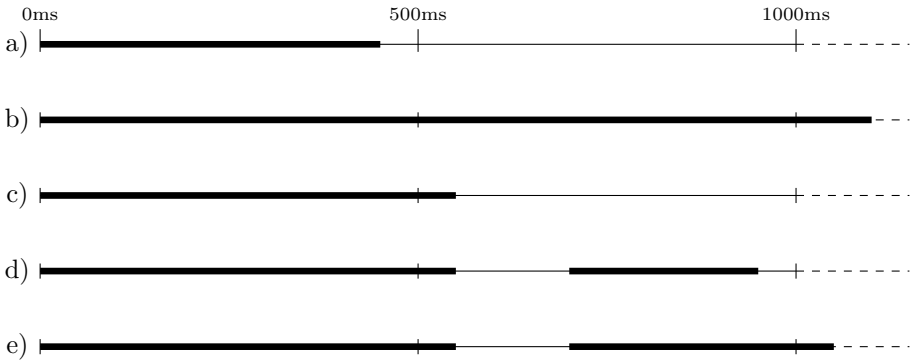


Figure 17: Durations of button presses corresponding to button events, a) single push, b) long push, c) long push, d) double push, e) double push

Actuator driver The house features a total of 16 actuator drivers, one for each of the lights. Each actuator driver handles a physical relay connected to a light in the house and determines whether or not an impulse should be sent to the relay toggling it on or off. Table 2 shows the status and interface places of a driver connected to a relay controlling a light in the house. The driver can receive an event to toggle the light, and send an impulse to the relay, which is modeled with interface places. As only four relays can receive impulses at a time in the house, each of the four relays are modeled as interface places as well, acting as resources. The relays need an impulse of a certain length in order to turn on, which is modeled with a status place containing the length of the impulse. The full model of a light relay driver can be seen in Appendix B.

Rules Rules in the smart house are defined using a syntax specified in [22], consists of a number of conditions followed by a set of actions performed in

	Interface	Status
I	LightDriverToggle	Timer
O	LightImpulse	ChangeState
	Relay1	
	Relay2	
	Relay3	
	Relay4	

Table 2: Interface and status places for a light driver. The dashed line separates input and output places

certain intervals. Our model of the system features a total of 37 rules, modeled as separate components:
Two rules for each of the buttons to turn a corresponding light on and off:

If `Button1.singlePush` when `Light1 == 0` then `Light1 := 1` in 0 for ∞;

If `Button1.singlePush` when `Light1 == 1` then `Light1 := 0` in 0 for ∞;

Two rules to turn all of the lights in house on or off:

If `Button15.longPush` when `true` then `Light1 := 1` in 0 for ∞; `Light2 := 1` in 0 for ∞; ... `Light16 := 1` in 0 for ∞;

If `Button16.longPush` when `true` then `Light1 := 0` in 0 for ∞; `Light2 := 0` in 0 for ∞; ... `Light16 := 0` in 0 for ∞;

Follow-me rule, for turning on hallway lights in sequence:

If `Button.singlePush` when `Day == 1` then `Light7 := 1` in 0 for 10; `Light8 := 1` in 10 for 20; `Light10 := 1` in 10 for 20;

Night motion rule, turning on corridor lights at night when motion is detected:

```
If MotionSensor.motion when Day == 0 then Light7 := 1 in 0 for 30;
Light8 := 1 in 0 for 30; Light10 := 1 in 0 for 30;
```

Buzzer rule, turning on the buzzer and outside lights when movement is detected and alarm mode is on:

```
If MotionSensor.motion when Alarm == 1 then Buzzer := 1 in 0 for 5;
Light1 := 1 in 0 for 5; Light2 := 1 in 0 for 5;
```

	Interface	**Status**
I	MovementDetected	Timer
	AlarmOn	
O	Light1On	
	Light1Off	
	LightDriver1Toggle	
	Light2On	
	Light2Off	
	LightDriver2Toggle	
	BuzzerOn	

Table 3: Interface and status places for the buzzer rule. The dashed line separates input and output places

Table 3 shows the status and interface places in the model of the buzzer rule. If the rule detects movement while the alarm is on, it starts a timer and turns on the lights and the buzzer in turn. When the timer exceeds five seconds, the lights and buzzer are turned off again. The full model of the buzzer rule can be seen in Appendix D. The remaining two special rules are similar in structure. The night motion rule, follow-me rule, and the two rules for turning every light

on and off have been further split into smaller workflows, in order to speed up verification due to the number of interface places in the workflow.

Relays The house features 16 physical relays connected to the lights. The relays receive an impulse event from the corresponding relay driver initiating a toggle of the relay. As mentioned earlier, the impulse has to be of a certain length in order for the relay to toggle with certainty. For an impulse shorter than the required minimum we cannot be sure whether the relay toggles or not, which leads to a failure state in the model. Additionally, the model includes a failure state for when the length of an impulse is too long, causing the relay to burn. The full model of a relay for a light can be seen in Appendix E.

External components The various external components of the system, e.g. the buttons, the state of the alarm, and the time of day, are modeled as simple using TAPNs. Figure 18 shows the modeling of a physical button in the house. The component runs as its own process and serves to produce an arbitrary sequence of button presses to the controller through the *ButtonPressed* place. The remaining external components are identical in structure.

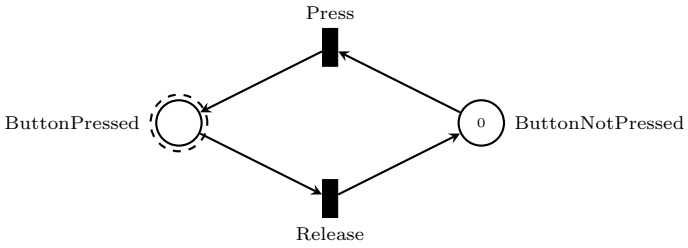


Figure 18: Model of a button

8.1 Analysis

Having modeled the smart house we would like to answer the following questions, as described in Section 3.

- Is it possible to reach a state where a relay burns out?
- Is it possible for a relay to receive too short an impulse, causing random behavior of the light switch?
- What is the responsiveness of a button push?

Statistics for the full model are shown in Table 4. Each of the components in the controller, as seen in Figure 16 is a TARWFN. As can be seen, the size of the state space renders explicit state space analysis infeasible, however using our compositional workflow analysis techniques we are able to determine

Components	124
- of them workflows	89
Places	727
Transitions	1445
Input arcs	2662
Output arcs	2419
Inhibitor arcs	1354
Pairs of transport arcs	181
Tokens	41
Shared places	271

Table 4: Statistics of the TAPN model of the smart house with 16 buttons, 16 lights, a motion sensor and a buzzer. Further 37 rules for the house has been modeled.

over-approximated minimum- and maximum execution times for the cycle in the controller. The analysis was run on a computer running 64-bit Linux, with a 2.60 GHz Intel i5 CPU and 8 GB of RAM.

8.1.1 Controller

The constants in the components are too large to verify with the directly with explicit state space exploration, so before running our compositional analysis we will therefore apply the over approximation techniques described in [9]. We scale every constant in a workflow to be between 0 and an approximated $\mathcal{C}$. For all checks, the approximated $\mathcal{C}$ can be very low and still provide conclusive answers, but with the disadvantage that the found minimum- and maximum running times are very imprecise. A selection of results for the first round of tests can be found in Table 5. All components were able to be verified with a very small approximated $\mathcal{C}$. In total running the checks with an approximated $\mathcal{C}$ of 10 took 29.097 seconds.

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$
ButtonDriver1	0	100001	0.24	N/A	10
MovementDriver	0	200001	0.038	N/A	10
RulesNightMotion	0	1000001	0.17	3	10
RulesFollowMe	0	500001	0.563	N/A	10
RulesBuzzer	0	500001	3.598	N/A	10
RulesSimpleOff1	44	176	0.03	N/A	10
RulesSimpleOn1	44	176	0.025	N/A	10
RulesLongOn	44	176	0.023	N/A	10
RulesLongOff	44	176	0.022	N/A	10
LightDriver1	0	60002	1.143	7	10

Table 5: Result on running all parts of the algorithm with a very low $\mathcal{C}$

Running our algorithm on the over-approximated model of the controller (89 components) in the case study, excluding a check to determine whether each component is 1-safe, 1-active and well-behaved, as these are already verified during the first run, took 3 hours and 19 minutes. An overview of the results can be found in Table 6. For all components see Appendix F. The approximated $\mathcal{C}$ for each component was set such that we could still verify the components. Many of the workflows include constants ranging from microseconds to seconds in size, resulting in an additional loss of precision of the smallest constants, when every constant in the net is scaled down, but we are still able to get reasonable results, and the analysis ends up with a minimum execution time of 1496 and maximum of 27273 for the cycle in the controller.

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$	$\mathcal{C}$
ButtonDriver1	0	251	300.884	5371	4000	1000001
MovementDriver	0	334	287.32	5642	6000	2000001
RulesSimpleOff1	44	176	0.03	N/A	176	176
RulesSimpleOn1	44	176	0.025	N/A	176	176
RulesLongOn	44	176	0.023	N/A	176	176
RulesLongOff	44	176	0.022	N/A	176	176
RulesNightMotion	0	2001	213.498	4288	5000	10000001
RulesFollowMe	0	1429	256.681	4778	3500	5000001
RulesBuzzer	0	2501	268.768	4034	2000	5000001
LightDriver1	0	688	382.581	5228	3500	300001

Table 6: A selection of results on running the strong soundness check.

8.1.2 Verification of Relay Safety

In order to determine whether or not it is possible for a relay to burn, or if the relay can receive too short an impulse, we construct the net N_{Relay} in Figure 19. The dashed "Light" box corresponds to the model seen in Appendix E. The remaining components of the controller have been abstracted away and replaced with a transition simulating the minimum- and maximum execution times of the components, obtained through the local strong soundness analysis. We simulate running the light driver, where between runs can manipulate all interface places except *LightImpulse*, as long as the reached passive marking is still 1-safe. Finally, we explore the state space to determine whether one of the failure places in the relay model can be reached, corresponding to either the relay burning or an impulse being too short.

N_{Relay} cannot reach a failure state and since it is an over-approximation of the original smart house, neither can the smart house. The questions were answered with an approximated $\mathcal{C} = 50$, and it took 2.26 seconds, led to 157991 discovered markings, where 64186 were explored, using 18 MB of memory.

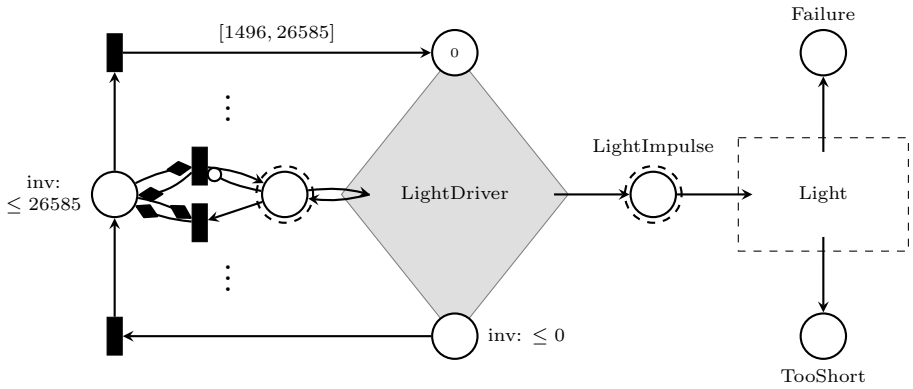


Figure 19: N_{Relay}

8.1.3 Responsiveness of a Button Push

To measure the responsiveness of a button push in the system, we construct the TAPN N_{Button} which can be seen in Figure 20. The net contains a button driver, one of the simple rules for turning a single light on, as well as a light driver, running in sequence abstracting away every other component of the program loop. We have an environmental assumption that no buttons have been pressed in the last 0.5 seconds. We assume this rather than simulating every possible starting condition for a specific reason: due to the structure of the controller, certain lights are treated more fairly than others under certain circumstances. In the program loop, the light drivers are evaluated in the same order each time, which means that in cases where the system receives a large amount of button presses from each button in the house, certain lights may never be turned on, seeing as only four relays can receive an impulse at a time. This results in situations where the first few light drivers constantly occupy the relays, turning on and off, making the later light drivers wait to use the relays forever. For these situations to arise, it would require four or more people to rapidly press buttons in the house, which is unrealistic in an everyday setting. We also assume that the push is a normal single push, which is longer than 35 ms[2] and shorter than 50 ms.

We say that the system is responsive given the environmental assumption for a button push if the light switches on within 200 ms of the start of the button push. The system satisfies this if the model in Figure 20 cannot reach the failure state. To reach the failure state is must have been in *wait* longer than 200 ms without the light being turned on in the meantime.

N_{Button} cannot reach *failure* and since it is an over-approximation of the original smart house, neither can the smart house. The question was answered

[2]The button push has to be longer than the found maximum execution time of the controller, otherwise there is a run where the push is too short to be recorded by the system.

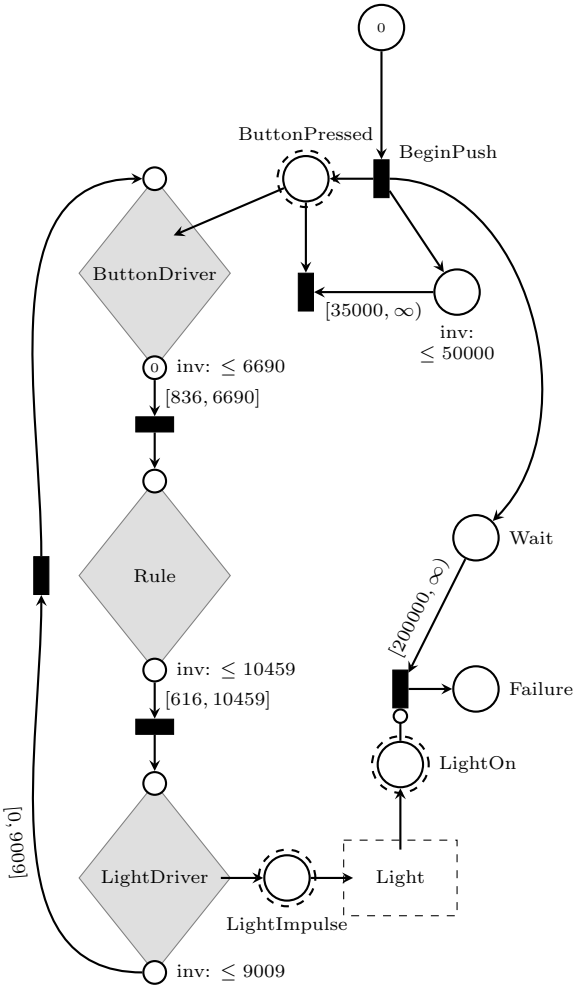


Figure 20: N_{Button}

with an approximated $\mathcal{C} = 1001$, and it took 11.37 seconds, lead to 504116 discovered markings, where 292817 were explored, and used 86 MB of memory. We have tested the system and it takes less than 110 ms for the light to turn on, so we can conclude that the system is responsive for a single button push.

8.2 Evaluation

We have looked into how the precision and time are affected by adjusting the approximated $\mathcal{C}$, and how the time is spent during the verification.

The trade-off between time and precision can be seen in Table 7. Here we have verified the button driver and the buzzer rule including 1-safe & 1-active

and well-behaved checks, with different approximated $\mathcal{C}$, ranging from 1 to 4000. It is clear to see that with small constants the explicit state space exploration is very fast, even with a high number of interface places (the buzzer rule has 9), but the computed min and maximum execution times are way to imprecise to be used to answers any of the questions asked for the case study. With a maximum execution time of 5 seconds for the buzzer rule it is definitely possible to burn a relay, and the system will not feel responsive at all. As we increase the approximated $\mathcal{C}$, the time and memory consumptions go up, but the bound for the execution times gets tighter. For both nets, the well-behaved check for these workflows consumes more memory than the soundness check, which is why the tests ran out of memory for lower values of $\mathcal{C}$ than before, see Table 6.

	ButtonDriver				RulesBuzzer			
Approx. C	Min	Max	Time (s)	Mem (MB)	Min	Max	Time (s)	Mem (MB)
1	0	1000002	0.088	N/A	0	5000002	1.205	4
10	0	100001	0.24	N/A	0	500001	3.617	N/A
100	0	10001	1.682	17	0	50001	28.704	150
500	0	2001	11.848	92	0	10001	155.512	886
750	0	1334	23.317	247	0	6667	234.004	1378
1000	0	1001	38.216	417	0	5001	325.091	1874
1500	0	667	79.38	859	0	3334	479.388	2916
2000	0	501	132.008	1476	-	-	-	>6000
3000	0	334	276.061	3144	-	-	-	>6000
4000	-	-	-	>6000	-	-	-	>6000

Table 7: Button driver and buzzer rule run with different approximated $\mathcal{C}$. As $\mathcal{C}$ increases, so does the precision, but also the time and memory consumption.

As can be seen from Table 6 and Table 7 neither the button driver nor the buzzer rule can be run with as high $\mathcal{C}$ when 1-safe, 1-active and well-behaved checks are run as well. In Table 8 we have run a button driver, a light driver and the three big rules to determine where the time is spent. As it can be seen, 1-safe, 1-active and well-behaved condition 1 and 2 are a bit faster than the soundness queries, and they also use a lot less memory. The latter because PTrie [27] can be used here. For the last three parts the time is almost evenly split and the differences is in how many interfaces the components versus the structure of the components. For the shown workflows the soundness and the well-behaved check are the most time consuming operations.

	ButtonDriver1	LightDriver1	RulesBuzzer	RulesFollowMe	RulesNightMotion
1-safe & 1-active	10.54%	15.41%	17.32%	12.61%	10.19%
Well-behaved cond. 1&2	7.81%	12.06%	17.41%	16.25%	10.86%
Well-behaved cond. 3	27.63%	19.67%	25.06%	29.48%	28.96%
Soundness	27.82%	26.86%	20.13%	21.20%	25.54%
Strong Soundness	26.19%	26.00%	20.07%	20.46%	24.45%
Verification time	35.497 s	111.317 s	298.795 s	66.821 s	19.266 s

Table 8: Distribution of time during verification of the five big nets with approximated $\mathcal{C} = 1000$.

9 Conclusion

We have proposed a formalism for workflows with interface and status places, which makes it possible to remember information in a workflow between executions as well as a simple form of communication with other TAPNs. For communication purposes we determined that binary information was sufficient and thus we do not allow timing information in interface places, which in turn also greatly reduces the state space during verification. For status places, we deemed the timing aspect much more critical, as it is often necessary to save specific timing information in the net. We have proposed a notion of local soundness and strong local soundness for the formalism, as well as an algorithm for checking (strong) local soundness by reducing the workflow to a workflow for which efficient soundness algorithms already exist.

The formalism shows promising results. Using our formalism we modeled a smart house including its controller and various hardware components, as well as the rules regarding the interactions between the various components. Using our soundness algorithm we managed to verify properties of the real-world model that would otherwise be impossible to verify. We discovered that the amount of interface places in the model has a significant impact on the verification time, as adding more interface places increases the amount of initial markings that are generated during verification. The size of the approximated C of the net similarly has an impact on the net, but we were able to successfully apply over-approximation techniques by Birch et al. [9] to scale the constants allowing the workflows to be verified at the cost of precision. Despite the loss of precision, we were successfully able to verify various properties on the model in reasonable time.

10 Future work

Our algorithm only works on 1-active workflows, because in practice it turned out to be difficult to construct workflows with multiple active status places that did not create situations where tokens are added in a way that made the workflow either not 1-safe or not sound for the full set of initial markings. For the subset of reachable markings from a single initial marking, the workflows were sound. So a direction for future work is to instead of explicitly considering every possible combination of status and interface places, only consider combinations which can be reached during normal execution. We could introduce a notion of cyclic soundness, where we only consider a single initial marking of a TARWFN, restarting the net whenever a final marking is reached while repopulating the interface places with every legal combination of tokens. Thus the combinations of tokens in status places are only those which can occur naturally during executions. For a net to be cyclically sound it would thus have to be locally sound for every possible cycle reached from a single initial marking.

On the same note, it could be interesting to be able to apply a specification with different conditions on the sets of status (and interface) places on the

workflow. Thus making it possible to say that a group of status places is 1-active, having multiple groups. That allows the workflow to be modeled in such a way that multiple status places can be active at a time. Another idea is to link interface and status places to simply reduce the set of initial markings needed to be searched. For example if certain places never have a token at the same time, or if certain places always have tokens in them at the same time as for instance is the case in the LightDriver model where *LightImpulse* and *Timer* are linked in such a way.

Another interesting aspect is automatic detecting of status places. Currently the set of status places is provided to the algorithm beforehand, but an attempt could be made at determining the set of status places automatically. Running a soundness check in a net with status places without specifying the set of status places beforehand would fail, as a token will most likely be present in the net when the *out* place is reached. Using this information, the offending place can be added to the set of status places in the model and the soundness check can be run again eventually reaching a fixed point with regards to the set of status places.

Overall, our workflow analysis was limited by memory constraints when verifying soundness. The verification engine in TAPAAL used for the experiments utilizes PTries [27] for verification, significantly reducing the memory consumption. PTries are however not implemented for workflow analysis. It could be interesting to work on implementing PTries or a similar memory efficient data structure to determine the effect of the analysis and the precision of the results.

Lastly, we could look into automatic optimization of the timing in status places. During the initial phase of the workflow transformation, every age of token is produced in status places. It could be interesting to analyze the timing constraints on the status places and thereby only generate the minimum amount of tokens required to cover every outcome, creating a LTS that is bisimilar with the one created when generating tokens of every age up to $C_{max} + 1$, thus also significantly reducing the state space.

References

[1] Wil MP Van Der Aalst. Three good reasons for using a Petri-net-based workflow management system. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC96)*, pages 179–201. Cambridge, Massachusetts, 1996.

[2] Wil MP Van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, pages 407–426. Springer, 1997.

[3] Wil MP Van der Aalst. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[4] Carl Adam Petri. Kommunikation mit automaten. 1962.

[5] Kees Van Hee, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. Soundness of resource-constrained workflow nets. In *Applications and Theory of Petri Nets 2005*, pages 250–267. Springer, 2005.

[6] Gabriel Juhás, Igor Kazlov, and Ana Juhásová. Instance deadlock: A mystery behind frozen programs. In *Applications and theory of Petri nets*, pages 1–17. Springer, 2010.

[7] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An operating guideline approach to the SOA. 2005.

[8] Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Yrke Jørgensen, Mikael H Møller, and Jiří Srba. TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 492–497. Springer, 2012.

[9] Sine Viesmose Birch, Thomas Stig Jacobsen, Jacob Jon Jensen, Christoffer Moesgaard, Niels Nørgaard Samuelsen, and Jiří Srba. Interval abstraction refinement for model checking of timed-arc Petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 237–251. Springer, 2014.

[10] Frank Puhlmann and Mathias Weske. *Investigations on soundness regarding lazy activities.* Springer, 2006.

[11] Frank Puhlmann and Mathias Weske. Interaction soundness for service orchestrations. In *Service-Oriented Computing–ICSOC 2006*, pages 302–313. Springer, 2006.

[12] Natalia Sidorova, Christian Stahl, and Nikola Trčka. Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In *Advanced Information Systems Engineering*, pages 530–544. Springer, 2010.

[13] Kees Van Hee, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. Soundness of resource-constrained workflow nets. In *Applications and Theory of Petri Nets 2005*, pages 250–267. Springer, 2005.

[14] Kamel Barkaoui and Laure Petrucci. Structural analysis of workflow nets with shared resources. 1998.

[15] Karsten Wolf. Does my service have partners? In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 152–171. Springer, 2009.

[16] Wil MP van der Aalst, Arjan J Mooij, Christian Stahl, and Karsten Wolf. Service interaction: Patterns, formalization, and analysis. In *Formal Methods for Web Services*, pages 42–88. Springer, 2009.

[17] Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *Business Process Management*, pages 235–253. Springer, 2000.

[18] Wil Van Der Aalst and Kees Max Van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004.

[19] HMW Verbeek, Moe Thandar Wynn, Wil MP van der Aalst, and Arthur HM ter Hofstede. Reduction rules for reset/inhibitor nets. *Journal of Computer and System Sciences*, 76(2):125–143, 2010.

[20] Wil MP van der Aalst, Kees M van Hee, Arthur HM ter Hofstede, Natalia Sidorova, HMW Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[21] Rajeev Alur, Alon Itai, Robert P Kurshan, and Mihalis Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[22] Søren B. Andersen, David Junker, Martin Z. Kristensen, Martin Lykke, Mads Mikkelsen, and Lasse E. Nielsen. Model driven development with time constraints for home automation systems, 2015. Student report at Aalborg University.

[23] Mathias Andersen, Heine Gatten Larsen, Jiří Srba, Mathias Grund Sørensen, and Jakob Haahr Taankvist. Verification of liveness properties on closed timed-arc Petri nets. In *Mathematical and Engineering Methods in Computer Science*, pages 69–81. Springer, 2013.

[24] José Antonio Mateo, Jiří Srba, and Mathias Grund Sørensen. Soundness of timed-arc workflow nets. In *Application and Theory of Petri Nets and Concurrency*, pages 51–70. Springer, 2014.

[25] V Valero Ruiz, David de Frutos Escrig, and F Cuartero Gomez. On non-decidability of reachability for timed-arc petri nets. In *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pages 188–196. IEEE, 1999.

[26] J. Byg, K.Y. Jørgensen, and J. Srba. TAPAAL: Editor, simulator and verifier of timed-arc Petri nets. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, volume 5799 of *LNCS*, pages 84–89. Springer-Verlag, 2009.

[27] Peter Gjøl Jensen, Kim Guldstrand Larsen, Jiří Srba, Mathias Grund Sørensen, and Jakob Haar Taankvist. Memory efficient data structures for explicit verification of timed systems. In *NASA Formal Methods*, pages 307–312. Springer, 2014.

A Status and Interface Places

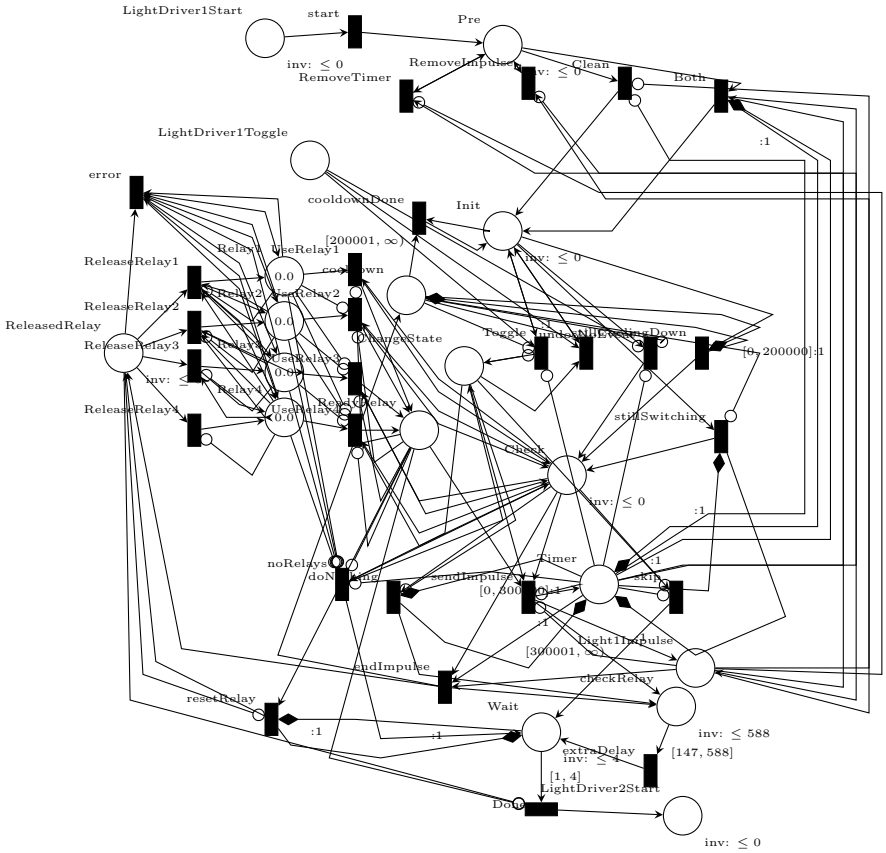
When running the strong local soundness algorithm, the user needs to specify, which places are interface and status places before the verification begins. In Table 9 all status places for the smart house modeled in TAPAAL are shown for the components. For components with multiple copies, only a single component is shown. The rest follow the same structure. The remaining components does not have any status places.

The interface places are all shared places which do not have a name ending with *start* or *end*.

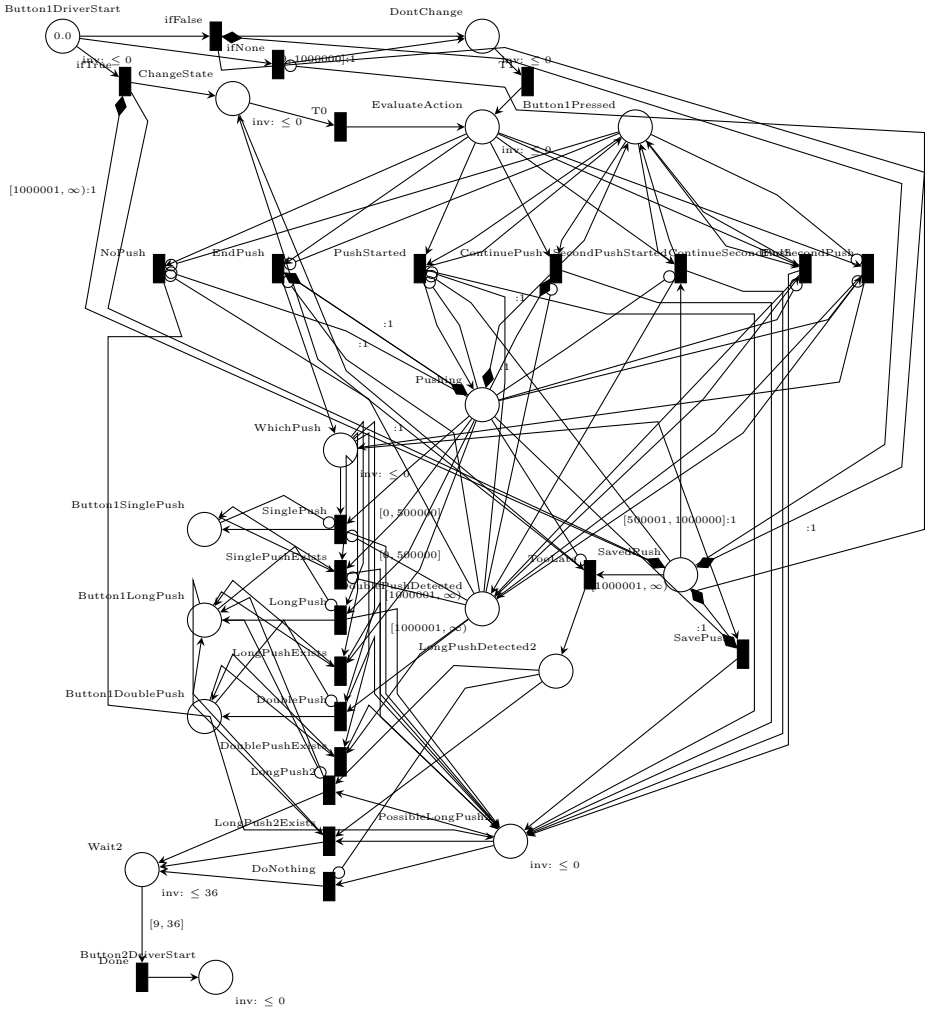
Component	Status places		
ButtonDriver1	Pushing	DoublePushDetected	SavedPush
LightDriver1	Cooldown	Timer	ChangeState
RulesNightMotion	Timer		
RulesBuzzer	Timer		
RulesFollowMe	TimerA	TimerB	
MovementDriver	timerSinceLastSent		

Table 9: Status places for the components of the smart house. For components that are not in this table nor have a copy presented here, there are no status places.

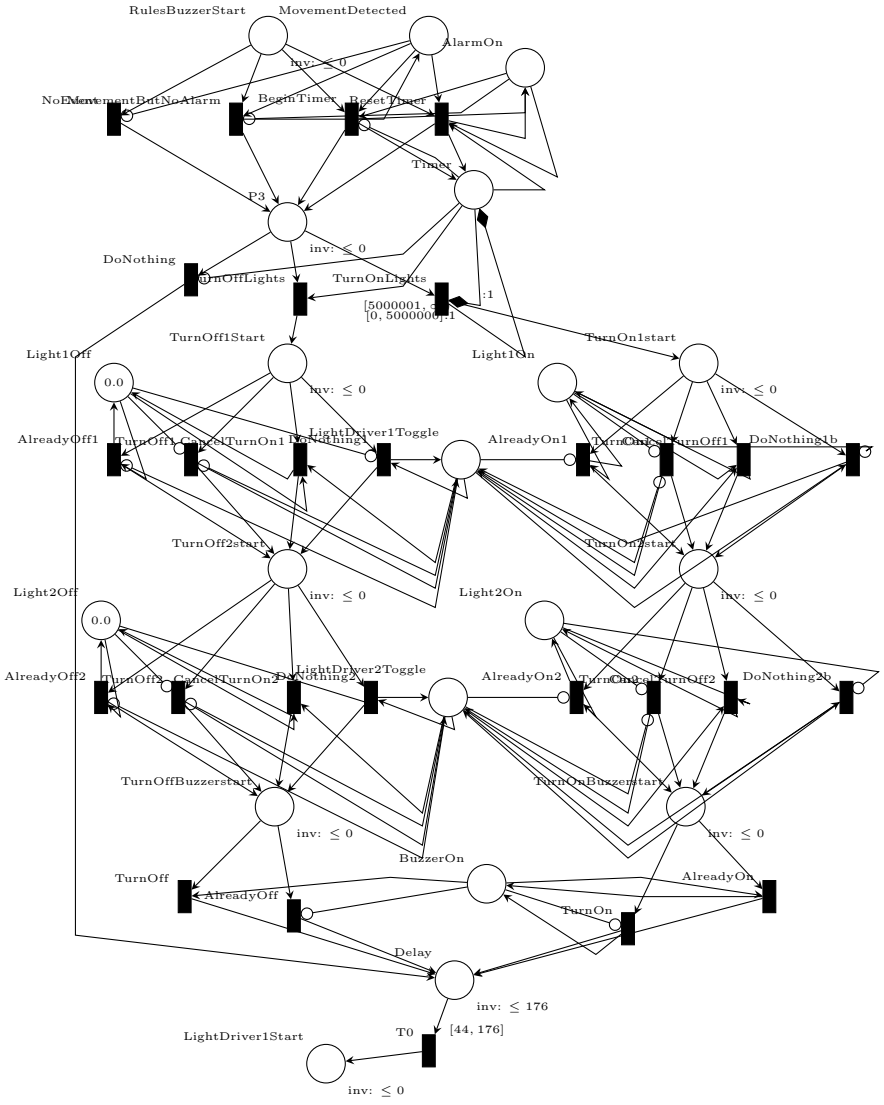
B Light Driver



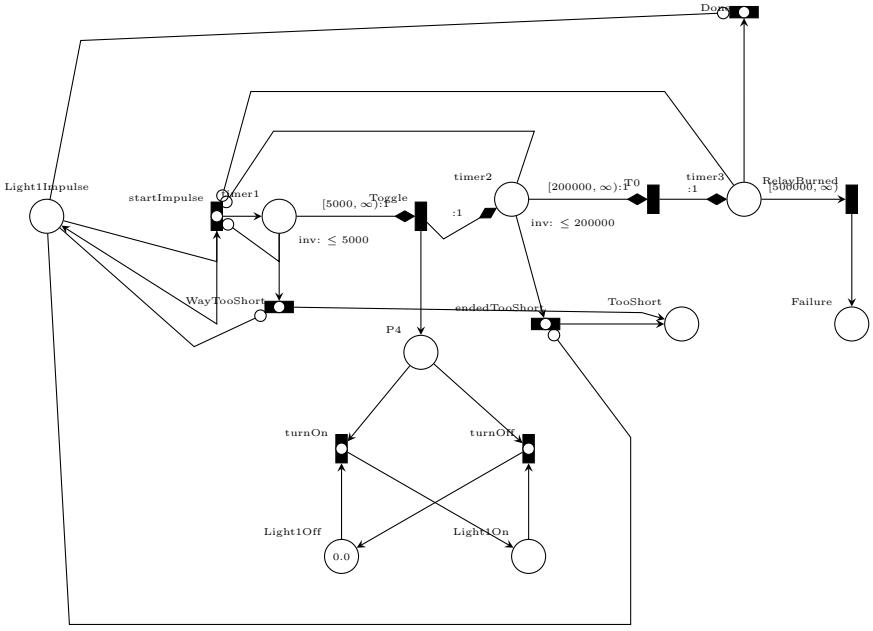
C Button Driver



D Buzzer Rule



E Light


Done
Light1Impulse
startImpulse
timer1
[5000, ∞):T5
Toggle
inv: ≤ 5000
timer2
:1
[200000, ∞):T0
inv: ≤ 200000
timer3
:1
RelayBurned
[500000, ∞)
Failure
WayTooShort
endedTooShort
TooShort
P4
turnOn
turnOff
Light1Off
Light1On
0.0

F Result from strong local soundness checks

Here are all tables with data from all 89 components in the model. Table 10 shows all 17 event drivers, Table 11 shows all 16 light drivers, Table 12 shows all rules and lastly Table 13 shows all the subparts for big rules we needed to verify the controller with a small enough cycle time.

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$	$\mathcal{C}$
ButtonDriver1	0	251	300.884	5371	4000	1000001
ButtonDriver2	0	251	300.701	5384	4000	1000001
ButtonDriver3	0	251	300.181	5409	4000	1000001
ButtonDriver4	0	251	301.373	5400	4000	1000001
ButtonDriver5	0	251	301.31	5392	4000	1000001
ButtonDriver6	0	251	300.681	5403	4000	1000001
ButtonDriver7	0	251	300.932	5381	4000	1000001
ButtonDriver8	0	251	301.024	5377	4000	1000001
ButtonDriver9	0	251	301.142	5400	4000	1000001
ButtonDriver10	0	251	300.114	5405	4000	1000001
ButtonDriver11	0	251	300.814	5390	4000	1000001
ButtonDriver12	0	251	300.02	5375	4000	1000001
ButtonDriver13	0	251	300.666	5389	4000	1000001
ButtonDriver14	0	251	300.38	5415	4000	1000001
ButtonDriver15	0	251	301.366	5383	4000	1000001
ButtonDriver16	0	251	300.76	5409	4000	1000001
MovementDriver	0	334	287.32	5642	6000	2000001

Table 10: Results from running the local strong soundness check on button and movement drivers

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$	$\mathcal{C}$
LightDriver1	0	688	382.581	5228	3500	300001
LightDriver2	0	688	380.646	5227	3500	300001
LightDriver3	0	688	383.308	5236	3500	300001
LightDriver4	0	688	382.553	5230	3500	300001
LightDriver5	0	688	383.126	5230	3500	300001
LightDriver6	0	688	382.938	5238	3500	300001
LightDriver7	0	688	382.925	5227	3500	300001
LightDriver8	0	688	382.277	5228	3500	300001
LightDriver9	0	688	381.171	5233	3500	300001
LightDriver10	0	688	381.415	5231	3500	300001
LightDriver11	0	688	381.054	5229	3500	300001
LightDriver12	0	688	382.213	5230	3500	300001
LightDriver13	0	688	380.55	5238	3500	300001
LightDriver14	0	688	382.882	5237	3500	300001
LightDriver15	0	688	382.912	5233	3500	300001
LightDriver16	0	688	381.752	5240	3500	300001

Table 11: Results from running the local strong soundness check on the light drivers

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$	$\mathcal{C}$
RulesSimpleOff1	44	176	0.03	N/A	176	176
RulesSimpleOff2	44	176	0.026	N/A	176	176
RulesSimpleOff3	44	176	0.025	N/A	176	176
RulesSimpleOff4	44	176	0.025	N/A	176	176
RulesSimpleOff5	44	176	0.026	N/A	176	176
RulesSimpleOff6	44	176	0.026	N/A	176	176
RulesSimpleOff7	44	176	0.027	N/A	176	176
RulesSimpleOff8	44	176	0.027	N/A	176	176
RulesSimpleOff9	44	176	0.025	N/A	176	176
RulesSimpleOff10	44	176	0.045	N/A	176	176
RulesSimpleOff11	44	176	0.047	N/A	176	176
RulesSimpleOff12	44	176	0.031	N/A	176	176
RulesSimpleOff13	44	176	0.032	N/A	176	176
RulesSimpleOff14	44	176	0.061	N/A	176	176
RulesSimpleOff15	44	176	0.043	N/A	176	176
RulesSimpleOff16	44	176	0.025	N/A	176	176
RulesSimpleOn1	44	176	0.025	N/A	176	176
RulesSimpleOn2	44	176	0.028	N/A	176	176
RulesSimpleOn3	44	176	0.026	N/A	176	176
RulesSimpleOn4	44	176	0.033	N/A	176	176
RulesSimpleOn5	44	176	0.028	N/A	176	176
RulesSimpleOn6	44	176	0.032	N/A	176	176
RulesSimpleOn7	44	176	0.028	N/A	176	176
RulesSimpleOn8	44	176	0.03	N/A	176	176
RulesSimpleOn9	44	176	0.041	3 MB	176	176
RulesSimpleOn10	44	176	0.028	N/A	176	176
RulesSimpleOn11	44	176	0.048	N/A	176	176
RulesSimpleOn12	44	176	0.046	N/A	176	176
RulesSimpleOn13	44	176	0.055	N/A	176	176
RulesSimpleOn14	44	176	0.04	3 MB	176	176
RulesSimpleOn15	44	176	0.035	N/A	176	176
RulesSimpleOn16	44	176	0.03	N/A	176	176
RulesLongOn	44	176	0.023	N/A	176	176
RulesLongOff	44	176	0.022	N/A	176	176
RulesNightMotion	0	2001	213.498	4288	5000	10000001
RulesFollowMe	0	1429	256.681	4778	3500	5000001
RulesBuzzer	0	2501	268.768	4034	2000	5000001

Table 12: Result from running local strong soundness checks on rules

Name	Min	Max	Time (s)	Mem (MB)	Approx. $\mathcal{C}$	$\mathcal{C}$
RulesNightMotionSub1	0	0	0.084	N/A	0	0
RulesNightMotionSub2	0	0	0.078	N/A	0	0
RulesFollowMeSub1	0	0	0.016	N/A	0	0
RulesFollowMeSub2	0	0	0.08	N/A	0	0
RulesFollowMeSub3	0	0	0.027	N/A	0	0
RulesFollowMeSub4	0	0	0.026	N/A	0	0
RulesFollowMeSub5	0	0	0.085	N/A	0	0
RulesLongOnSub1	0	0	0.085	N/A	0	0
RulesLongOnSub2	0	0	0.083	3	0	0
RulesLongOnSub3	0	0	0.09	N/A	0	0
RulesLongOnSub4	0	0	0.087	N/A	0	0
RulesLongOnSub5	0	0	0.096	N/A	0	0
RulesLongOnSub6	0	0	0.014	N/A	0	0
RulesLongOffSub1	0	0	0.095	N/A	0	0
RulesLongOffSub2	0	0	0.091	N/A	0	0
RulesLongOffSub3	0	0	0.104	N/A	0	0
RulesLongOffSub4	0	0	0.09	N/A	0	0
RulesLongOffSub5	0	0	0.081	N/A	0	0
RulesLongOffSub6	0	0	0.018	N/A	0	0

Table 13: Results from running the local strong soundness check on the subnets for the rules