TCTL-Preserving Translations from Timed-Arc Petri Nets to Networks of Timed Automata

Joakim Byg, Morten Jacobsen, Lasse Jacobsen, Kenneth Yrke Jørgensen[1], Mikael Harkjær Møller[1], Jiří Srba[1]

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

Abstract

We present a framework for TCTL-preserving translations between time-dependent modelling formalisms. The framework guarantees that once the original and the translated system are in one-by-many correspondence relation (a notion of behavioural equivalence between timed transition systems) then TCTL properties of the original system can be transformed too while preserving the verification answers. We demonstrate the usability of the technique on two reductions from bounded timed-arc Petri nets to networks for timed automata, providing unified proofs of the translations implemented in the verification tool TAPAAL. We evaluate the efficiency of the approach on a number of experiments: alternating bit protocol, Fischer's protocol, Lynch-Shavit protocol, MPEG-2 encoder, engine workshop and medical workflow. The results are encouraging and confirm the practical applicability of the approach.

Keywords: formal verification, TCTL, timed-arc Petri nets, timed automata

1. Introduction

Formal verification of embedded and hybrid systems is an active research area. Recently, a lot of attention has been devoted to the analysis of systems with quantitative attributes like timing, cost and probability. In particular, several different time-dependent models have been developed over the two last decades or so. These models are often introduced as time extensions of well-studied untimed formalisms and include, among others, networks of timed automata [5, 6] and different time extensions of the Petri net model [42]. These formalisms are nowadays supported by a number of

Email addresses: joakim@aptusoft.com (Joakim Byg), morten.jacobsen.2k@gmail.com (Morten Jacobsen), lassejacobsen@gmail.com (Lasse Jacobsen), kyrke@ist.aau.dk (Kenneth Yrke Jørgensen), m@mikael.hm (Mikael Harkjær Møller), srba@cs.aau.dk (Jiří Srba)

[1]The authors were supported by VKR Center of Excellence MT-LAB.

tools [8, 16, 17, 32, 23, 25, 11, 27] and applied in model-driven system design methodologies.

We shall focus on the Petri net model extended with continuous time. The timing aspects can be associated with different parts of the model in the various time-extended Petri net formalisms. For example, *timed transitions Petri nets* where transitions are annotated with their durations were proposed in [43]. A model in which time parameters are associated with places is called *timed places Petri nets* and it was introduced in [44]. *Time Petri nets* of Merlin and Faber [35, 36] were introduced in 1976 and associate time intervals to each transition. The intervals define the earliest and latest firing time of the transition since it became enabled. Yet another model of *timed-arc Petri nets* (TAPN) was first studied around 1990 by Bolognesi, Lucidi, Trigila and Hanisch [12, 26]. Here time information is attached to the tokens in the net representing their relative age while arcs from places to transitions contain time intervals that restrict the enabledness of the transitions (see [29] for a recent introduction to TAPNs). For an overview of the different extensions consult e.g. [15, 51, 40, 47]. The TAPN model is particularly suitable for modelling of manufacturing systems, workflow management and similar applications [4, 2, 38, 49, 50, 39] and a recently developed tool TAPAAL [19, 23] enables automatic verification of bounded TAPNs extended with transport/inhibitor arcs and age invariants.

One way to verify formal models is via a translation to another formalism with a well-established tool support and indeed, several such translations have already been developed (see e.g. [47] for an overview). Many of the translations utilize similar tricks that allow for the simulation of one system by another. Typically, a single step in one formalism is simulated by a sequence of steps in the other one. Inspired by the translations presented in the literature [21, 14, 24, 30, 9] we identify a general class of model transformations that preserve the satisfiability of *Timed Computation Tree Logic* (TCTL) [41], a logic suitable for a practical specification of many useful temporal properties. The main goal of this article is to provide a general proof framework directly applicable to the kinds of translations often implemented by the tool developers, and to demonstrate its applicability by presenting two concrete translations from timed-arc Petri nets to networks of timed automata that proved to be the most efficient and are implemented in the tool TAPAAL.

Unlike much work on TCTL where only infinite alternating runs are considered [41] or the details of the semantics are not fully discussed [21, 13], we consider also finite maximal runs that appear in the presence of stuck computations or time invariants (strict or nonstrict) and treat the semantics in its full generality as necessary for the correct semantics of the models used in many tools including e.g. Uppaal [8]. This is particularly important for liveness properties. While some translations in the literature (not necessarily only between timed-arc Petri nets and timed automata) preserve some variant of timed bisimilarity [21, 14, 24, 30], other translations preserve only reachability or trace equivalence [9, 18]. Our framework allows us to argue that several such translations preserve the full TCTL or at least its safety fragment.

Further we propose two novel TCTL-preserving translations from timed-arc

Petri nets extended with all the additional features implemented in TAPAAL to Uppaal networks of timed automata. Earlier translations either caused exponential blow-up in the size [45, 46, 14], or where not suitable for implementation in tools due to an inefficient use of clocks and communication primitives [46]. One of the translations from TAPN to Uppaal timed automata presented in this article is the first one to run in polynomial time while preserving the full TCTL. It uses the idea of a controller (like for example in [21] where it was employed for a translation from time Petri nets to timed automata) that is guiding the firing of transitions. We implemented both translations in the tool TAPAAL and the experiments that we shall present confirm their efficiency also in practice. One of the important contributions of this article is the fact that we treat the TAPN models in their full generality, including transport and inhibitor arcs and age invariants, exactly as implemented in the tool TAPAAL. This provides the evidence that the theory behind the translations in the tool is sound.

The work we present here is based on several conference papers [18, 28, 29, 46] where the initial translations and the theory were originally published. This journal article, as presented at ICTAC'11 in a half-day tutorial, unifies the different proofs into a common framework and considerably simplifies the correctness arguments. In Section 2 we present the proof framework for arguing about TCTL-preservation of the translations. We give a formal syntax and semantics of timed-arc Petri nets in Section 3 and of networks of timed automata in Section 4. In Section 5 and Section 6 we present two translations from TAPN to NTA, and prove their correctness using the proof framework from Section 2. Section 7 presents a selection of experiments where we compare the performance of the implemented translations. Finally, Section 8 gives a short conclusion.

2. Proof Framework

In this section, we shall present a general framework for arguing when a simulation of one time dependent system by another one preserves the satisfiability of TCTL formulae. We define the notion of one-by-many correspondence, a relation between two TTSs A and B. If A is in one-by-many correspondence with B then every transition in A can be simulated by a sequence of transitions in B. Further, every TCTL formula φ can be algorithmically translated into a formula $tr(\varphi)$ such that $A \models \varphi$ iff $B \models tr(\varphi)$. Before we can describe the framework, we need to introduce some preliminary definitions.

2.1. Preliminaries

We let $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote the sets of integers, natural numbers, non-negative integers, real numbers and non-negative real numbers, respectively.

Definition 1. A *timed transition system* (TTS) is a tuple $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ where S is a set of states (or processes), $\longrightarrow \subseteq (S \times S) \cup (S \times \mathbb{R}_{\geq 0} \times S)$ is a transition relation, $\mathcal{AP}$ is a set of atomic propositions, and $\mu : S \longrightarrow 2^{\mathcal{AP}}$ is a function assigning sets of true atomic propositions to states.

We write $s \longrightarrow s'$ whenever $(s, s') \in \longrightarrow$ and call them *discrete transitions*, and $s \xrightarrow{d} s'$ whenever $(s, d, s') \in \longrightarrow$ and call them *delay transitions*. We require that the TTSs we consider satisfy the following standard axioms for delay transitions (see e.g. [10]). For all $d, d' \in \mathbb{R}_{\geq 0}$ and $s, s', s'' \in S$:

1. **Time Additivity:** if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ then $s \xrightarrow{d+d'} s''$,
2. **Time Continuity:** if $s \xrightarrow{d+d'} s''$ then $s \xrightarrow{d} s' \xrightarrow{d'} s''$ for some s',
3. **Zero Delay:** $s \xrightarrow{0} s$ for each state s, and
4. **Time Determinism:** if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$.

By $s[d]$ we denote the state s' (if it exists) such that $s \xrightarrow{d} s'$. Note that time determinism ensures the uniqueness of $s[d]$. We write $s \longrightarrow$ if $s \longrightarrow s'$ for some $s' \in S$ and $s \not\longrightarrow$ otherwise. Similarly for $\xrightarrow{d}$. A *run* $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$ is a (finite or infinite) alternating sequence of time delays and discrete actions.

Definition 2. The set of all well formed *time intervals* $\mathcal{I}$ is defined by

$$\mathcal{I} = \{[a, a], [a, b], [a, b), (a, b], (a, b), [a, \infty), (a, \infty) \mid a \in \mathbb{N}_0, b \in \mathbb{N}, a < b\} \ .$$

2.2. Timed Computation Tree Logic

We shall now introduce the syntax and semantics of Timed Computation Tree Logic (TCTL) where temporal operators are indexed by intervals (see e.g. [41]).

Definition 3. Let $\mathcal{AP}$ be a set of *atomic propositions*. The set of TCTL formulae $\Phi(\mathcal{AP})$ over $\mathcal{AP}$ is given by

$$\varphi ::= \wp \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid E(\varphi_1 \, U_I \, \varphi_2) \mid A(\varphi_1 \, U_I \, \varphi_2) \mid E(\varphi_1 \, R_I \, \varphi_2) \mid A(\varphi_1 \, R_I \, \varphi_2)$$

where $\wp \in \mathcal{AP}$ and $I \in \mathcal{I}$. Formulae without any occurrence of the operators $A(\varphi_1 \, U_I \, \varphi_2)$ and $E(\varphi_1 \, R_I \, \varphi_2)$ form the *safety fragment* of TCTL.

In the syntax the symbol E stands for the existential and A for the universal path quantification and the intuition of the until (U) and release (R) operators (formalized later on) is as follows: $E(\varphi_1 \, U_I \, \varphi_2)$ is true if there exists a run such that φ_2 eventually holds within the interval I, and until it does, φ_1 continuously holds; $E(\varphi_1 \, R_I \, \varphi_2)$ is true if there exists a maximal run such that φ_2 holds within the interval I unless it was released by φ_1 satisfied earlier in the run.

As we aim to apply our framework to concrete case studies with possible tool support, we need to handle maximal runs in their full generality. There are two types of maximal runs. Either the infinite ones or the ones that are "stuck". In the second case, we annotate the last transition of such a run with one of the three special ending symbols (denoted by δ in the definition below).

Definition 4. A *maximal run* ρ is either

(i) an infinite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$, or

(ii) a finite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \ldots \longrightarrow s_n \xrightarrow{\delta}$ with $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$ where $d_n \in \mathbb{R}_{\geq 0}$ such that

- if $\delta = \infty$ then $s_n \xrightarrow{d}$ for all $d \in \mathbb{R}_{\geq 0}$,
- if $\delta = d_n^{\leq}$ then $s_n \xrightarrow{d} \not\longrightarrow$ for all $d > d_n$ and $s_n \xrightarrow{d_n} s_n[d_n]$ such that $s_n[d_n] \not\longrightarrow$, and
- if $\delta = d_n^{<}$ then $s_n \xrightarrow{d} \not\longrightarrow$ for all $d \geq d_n$, and there exists d_s, $0 \leq d_s < d_n$, such that for all d, $d_s \leq d < d_n$, we have $s_n \xrightarrow{d} s_n[d]$ and $s_n[d] \not\longrightarrow$.

By $\mathit{MaxRuns}(T, s)$ we denote the set of maximal runs in a TTS T starting at s.

Intuitively, the three conditions in case (ii) describe all possible ways in which a finite run can terminate. First, a run can end in a state where time diverges. The other two cases define a run which ends in a state from which no discrete transition is allowed after some time delay, but time cannot diverge either (typically caused by the presence of invariants in the model). These cases differ in whether the bound on the maximal time delay can be reached or not.

Let us now introduce some notation for a given maximal run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$. First, $r(i, d)$ denotes the total time elapsed from the beginning of the run up to some delay $d \in \mathbb{R}_{\geq 0}$ after the i'th discrete transition. Formally, $r(i, d) = \left(\sum_{j=0}^{i-1} d_j\right) + d$. Second, we define a predicate $\mathit{valid}_\rho : \mathbb{N} \times \mathbb{R}_{\geq 0} \times \mathcal{I} \to \{\mathit{true}, \mathit{false}\}$ such that $\mathit{valid}_\rho(i, d, I)$ checks whether the total time for reaching the state $s_i[d]$ in ρ belongs to the time interval I, formally

$$\mathit{valid}_\rho(i, d, I) = \begin{cases} d \leq d_i \wedge r(i, d) \in I & \text{if } d_i \in \mathbb{R}_{\geq 0} \\ r(i, d) \in I & \text{if } d_i = \infty \\ d \leq d_n \wedge r(i, d) \in I & \text{if } d_i = d_n^{\leq} \\ d < d_n \wedge r(i, d) \in I & \text{if } d_i = d_n^{<} \ . \end{cases}$$

Figure 1 illustrates a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$ and three points (marked with $\times$). Note that discrete transitions have zero durations and there is no special meaning as to whether the arrow for discrete transitions goes up or down, this is simply to keep the figure small. Even though time delays appear identical in the figure, they can be of different length (even zero). Let us now give some example of the application of the $\mathit{valid}_\rho(i, d, I)$ function. We see that $\mathit{valid}_\rho(1, d, I)$ is false because $s_1[d]$ lies outside the interval I. Similarly, $\mathit{valid}_\rho(2, d'', I)$ is false because $s_2[d'']$ is not a part of the run (since $d'' > d_2$). Finally, $\mathit{valid}_\rho(2, d', I)$ is true because $s_2[d']$ is a part of the run and within I.

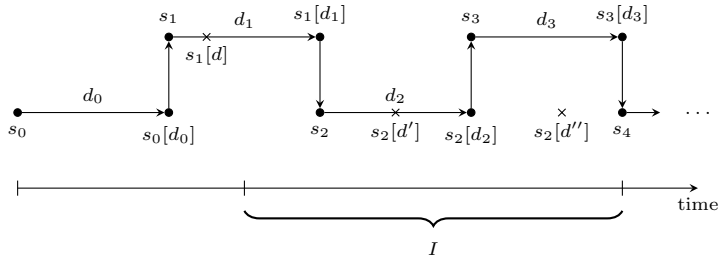

Figure 1: Illustration of a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$

Next, for a given maximal run ρ, we define a partial function $\mathit{history}_\rho : \mathbb{N} \times \mathbb{R}_{\geq 0} \hookrightarrow 2^{\mathbb{N} \times \mathbb{R}_{\geq 0}}$ such that $\mathit{history}_\rho(i, d)$ returns the set of pairs (j, d') that constitute all states $s_j[d']$ in ρ preceding $s_i[d]$. Formally,

$$\mathit{history}_\rho(i, d) = \{(j, d') \mid 0 \leq j < i \ \wedge \ 0 \leq d' \leq d_j\} \cup \{(i, d') \mid 0 \leq d' < d\}$$

for all values of i and d that are relevant to consider for the given run ρ.

Definition 5. We define the satisfaction relation $s \models \varphi$ for a state $s \in S$ in a TTS $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ and a TCTL formula φ as follows.

$$
\begin{array}{ll}
s \models \wp & \text{iff } \wp \in \mu(s) \\
s \models \neg\varphi & \text{iff } s \not\models \varphi \\
s \models \varphi_1 \wedge \varphi_2 & \text{iff } s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models E(\varphi_1 \, U_I \, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s) \,. \\
 & \quad \exists i \geq 0 \,. \exists d \in \mathbb{R}_{\geq 0} \,. \, [\mathit{valid}_\rho(i, d, I) \wedge s_i[d] \models \varphi_2 \wedge \\
 & \quad \forall (j, d') \in \mathit{history}_\rho(i, d) \,. \, s_j[d'] \models \varphi_1] \\
s \models E(\varphi_1 \, R_I \, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s) \,. \\
 & \quad \forall i \geq 0 \,. \forall d \in \mathbb{R}_{\geq 0} \,. \, \mathit{valid}_\rho(i, d, I) \Rightarrow \\
 & \quad [s_i[d] \models \varphi_2 \vee \exists (j, d') \in \mathit{history}_\rho(i, d) \,. \, s_j[d'] \models \varphi_1]
\end{array}
$$

The operators $A(\varphi_1 \, U_I \, \varphi_2)$ and $A(\varphi_1 \, R_I \, \varphi_2)$ are defined analogously by replacing the quantification $\exists \rho \in \mathit{MaxRuns}(T, s)$ with $\forall \rho \in \mathit{MaxRuns}(T, s)$.

Figure 2 illustrates the satisfaction of an until formula and Figure 3 illustrates the satisfaction of a release formula. In particular, notice that in Figure 3 there are four possible ways for a release formula to be satisfied. First, φ_1 may have occurred in the past (outside the interval), which releases φ_2, effectively ensuring that φ_2 need not hold in the interval I at all. Second, φ_2 may not be released, which means that it must hold continuously within the entire interval I. Third, φ_2 can hold continuously in the interval I, until some point in the

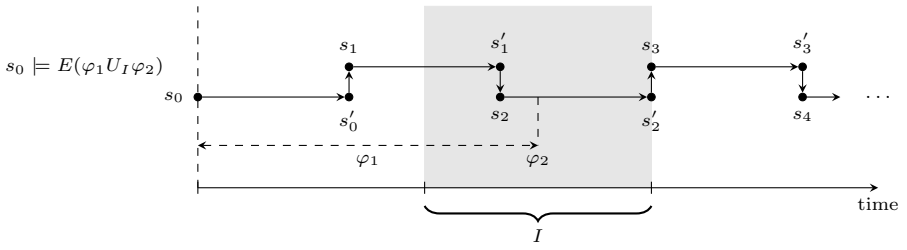

Figure 2: Illustration of a run satisfying an until formula

interval where $\varphi_1 \wedge \varphi_2$ holds, thereby releasing φ_2. Finally, φ_2 can hold in the interval I until the run stops and cannot be extended any further.

As expected, the until and release operators are dual.

Lemma 6. *Let $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ be a TTS and $s \in S$. Then $s \models A(\varphi_1 \, R_I \, \varphi_2)$ iff $s \models \neg E(\neg\varphi_1 \, U_I \, \neg\varphi_2)$, and $s \models A(\varphi_1 \, U_I \, \varphi_2)$ iff $s \models \neg E(\neg\varphi_1 \, R_I \, \neg\varphi_2)$.*

Proof. Straightforward by applying the duality of the logical operators and the existential and universal quantifiers. □

2.3. One-By-Many Correspondence

We are now ready to define the notion of one-by-many correspondence that relates two TTSs A and B whenever every transition in A can be simulated by a sequence of transitions in B. The relation is defined in a way that a given TCTL formula φ can be algorithmically translated into a formula $tr(\varphi)$ such that $A \models \varphi$ iff $B \models tr(\varphi)$. In the rest of this section, we shall use A and B to refer to the original and the translated system, respectively.

As the system B is simulating a single transition of A by a sequence of transitions, the systems A and B are comparable only in the states before and after this sequence was performed. This is a similar idea as in the notion of stuttering bisimulation [37]. We say that B is stable in such states and introduce a fresh atomic proposition called *stable* to explicitly identify this situation. States not satisfying the proposition *stable* are called intermediate or unstable. The point is that that once a sequence of transitions through intermediate states reaches a stable one, this corresponds to a single computation step in the system A.

We now define three conditions that B should possess in order to apply to our framework. The third condition is optional and necessary only for the preservation of liveness TCTL properties. A TTS $(S, \to, \mathcal{AP}, \mu)$ such that $\mathit{stable} \in \mathcal{AP}$ is called

- *delay-implies-stable* if, for any $s \in S$, whenever $s \xrightarrow{d}$ for some $d > 0$ then $s \models \mathit{stable}$,
- *delay-preserves-stable* if, for any $s \in S$ such that $s \models \mathit{stable}$, whenever $s \xrightarrow{d} s[d]$ for some $d \geq 0$ then $s[d] \models \mathit{stable}$, and

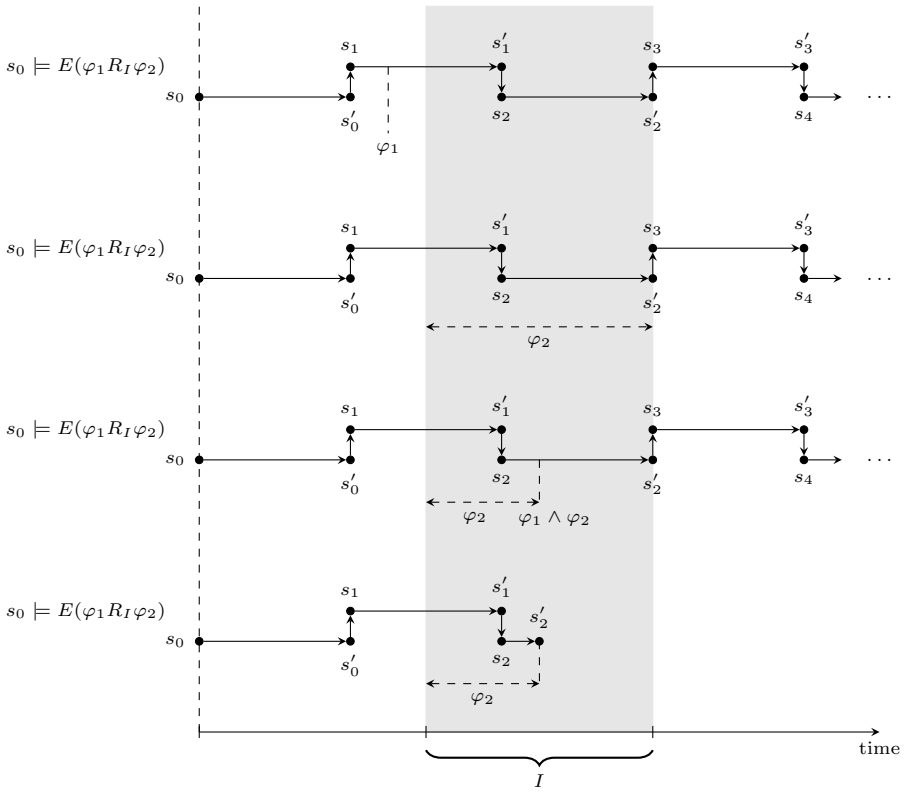

Figure 3: Illustration of runs satisfying a release formula

- *eventually-stable* if for any $s_0 \in S$ such that $s_0 \models \mathit{stable}$ and for any infinite sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow \ldots$ or any finite nonempty sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \not\longrightarrow$ there exists an index $i \geq 1$ such that $s_i \models \mathit{stable}$. We call such a sequence a *maximal discrete sequence*.

We write $s \rightsquigarrow s'$ if there is an alternating sequence $s = s_0 \longrightarrow s_1 \xrightarrow{0} s_1 \longrightarrow s_2 \xrightarrow{0} s_2 \longrightarrow \cdots \xrightarrow{0} s_{n-1} \longrightarrow s_n = s'$ such that $s \models \mathit{stable}$, $s' \models \mathit{stable}$, and $s_j \not\models \mathit{stable}$ for $1 \leq j \leq n-1$.

Remark 7. For technical convenience, we introduced zero delays in the definition of $\rightsquigarrow$ in order to preserve the alternating nature of the sequence. Note that this is not restrictive as for any $s \in S$ we always have $s \xrightarrow{0} s$.

Definition 8. Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs such that $\mathit{stable} \in \mathcal{AP}_B$ and B is *delay-implies-stable* and *delay-preserves-stable* TTS. A relation $\mathcal{R} \subseteq S \times T$ is a *one-by-many correspondence* if there exists a function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ such that whenever $(s,t) \in \mathcal{R}$ then

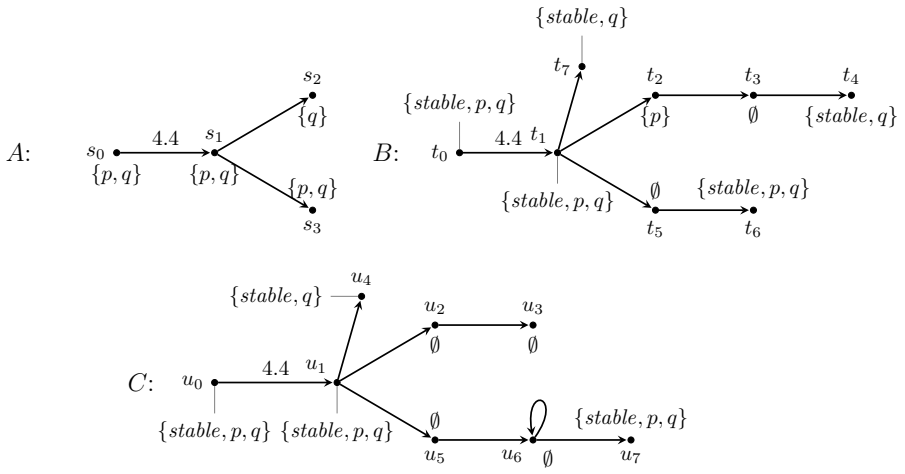

Figure 4: Three TTSs such that $s_0 \overset{\rightarrow}{=}_c t_0$ and $s_0 \overset{\rightarrow}{=} u_0$

1. $t \models \mathit{stable}$,
2. $s \models \wp$ iff $t \models tr_p(\wp)$ for all $\wp \in \mathcal{AP}_A$,
3. if $s \longrightarrow s'$ then $t \rightsquigarrow t'$ and $(s', t') \in \mathcal{R}$,
4. if $s \xrightarrow{d} s[d]$ then $t \xrightarrow{d} t[d]$ and $(s[d], t[d]) \in \mathcal{R}$,
5. if $t \rightsquigarrow t'$ then $s \longrightarrow s'$ and $(s', t') \in \mathcal{R}$, and
6. if $t \xrightarrow{d} t[d]$ then $s \xrightarrow{d} s[d]$ and $(s[d], t[d]) \in \mathcal{R}$.

If B is moreover an *eventually-stable* TTS, then we say that $\mathcal{R}$ is a *complete one-by-many correspondence.* We write $s \overset{\rightarrow}{=} t$ (resp. $s \overset{\rightarrow}{=}_c t$) if there exists a relation $\mathcal{R}$ which is a one-by-many correspondence (resp. a complete one-by-many correspondence) such that $(s, t) \in \mathcal{R}$.

Example 9. Consider the TTSs A, B and C in Figure 4 where the sets of propositions for A, B and C are $\mathcal{AP}_A = \{p, q\}$ and $\mathcal{AP}_B = \mathcal{AP}_C = \{p, q, \mathit{stable}\}$. Then $\{(s_0[d], t_0[d]) \mid 0 \leq d \leq 4.4\} \cup \{(s_1, t_1), (s_2, t_4), (s_3, t_6), (s_2, t_7)\}$ is a complete one-by-many correspondence which implies that $s_0 \overset{\rightarrow}{=}_c t_0$ and $\{(s_0[d], u_0[d]) \mid 0 \leq d \leq 4.4\} \cup \{(s_1, u_1), (s_2, u_4), (s_3, u_7)\}$ is a one-by-many correspondence which implies that $s_0 \overset{\rightarrow}{=} u_0$. Notice that the system C is not *eventually-stable* since the two maximal discrete sequences $u_1 \longrightarrow u_5 \longrightarrow u_6 \longrightarrow u_6 \longrightarrow u_6 \longrightarrow u_6 \longrightarrow \cdots$ and $u_1 \longrightarrow u_2 \longrightarrow u_3$ do not contain any stable states except for u_1.

We shall now describe how to translate TCTL formulae used for property specification in the system A into equivalent formulae for the system B. Referring again to Figure 4, we can see that the maximal run $\rho = s_0 \xrightarrow{4.4} s_1 \longrightarrow s_2 \xrightarrow{0^{\leq}}$ in A witnesses that $s_0 \models E(\neg p \, R_{[3,5]} \, q)$ as q holds in

the whole interval $[3,5]$. The formula can translated into an equivalent one $E((\neg p \wedge \mathit{stable})\, R_{[3,5]}\, (q \vee \neg \mathit{stable}))$ for the system B, requiring that q has to hold unless we are in an unstable state, and it can be released by $\neg p$ while being in a stable state. Indeed, the maximal run $\rho' = t_0 \xrightarrow{4.4} t_1 \rightsquigarrow t_4 \xrightarrow{0^{\leq}}$ witnesses that $t_0 \models E((\neg p \wedge \mathit{stable})\, R_{[3,5]}\, (q \vee \neg \mathit{stable}))$.

A general translation of TCTL formulae follows. Let $\mathcal{AP}_A$ and $\mathcal{AP}_B$ be sets of atomic propositions such that $\mathit{stable} \in \mathcal{AP}_B$ and let $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ be a function translating atomic propositions. We define $tr : \Phi(\mathcal{AP}_A) \to \Phi(\mathcal{AP}_B)$, depending on tr_p, as follows:

$$\begin{aligned}
tr(\wp) &= tr_p(\wp) \\
tr(\neg\varphi_1) &= \neg tr(\varphi_1) \\
tr(\varphi_1 \wedge \varphi_2) &= tr(\varphi_1) \wedge tr(\varphi_2) \\
tr(E(\varphi_1\, U_I\, \varphi_2)) &= E((tr(\varphi_1) \vee \neg \mathit{stable})\, U_I\, (tr(\varphi_2) \wedge \mathit{stable})) \\
tr(A(\varphi_1\, U_I\, \varphi_2)) &= A((tr(\varphi_1) \vee \neg \mathit{stable})\, U_I\, (tr(\varphi_2) \wedge \mathit{stable})) \\
tr(E(\varphi_1\, R_I\, \varphi_2)) &= E((tr(\varphi_1) \wedge \mathit{stable})\, R_I\, (tr(\varphi_2) \vee \neg \mathit{stable})) \\
tr(A(\varphi_1\, R_I\, \varphi_2)) &= A((tr(\varphi_1) \wedge \mathit{stable})\, R_I\, (tr(\varphi_2) \vee \neg \mathit{stable}))
\end{aligned}$$

We are now ready to state our main result. From now on we fix two TTS $A = (S, \to_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \to_B, \mathcal{AP}_B, \mu_B)$ such that $\mathit{stable} \in \mathcal{AP}_B$ and B has the properties *delay-implies-stable* and *delay-preserves-stable*.

Theorem 10. *Let $s_0 \in S$ and $t_0 \in T$.*

- *If $s_0 \overset{\rightarrow}{=} t_0$ then for any formula φ from the safety fragment of TCTL, $s_0 \models \varphi$ if and only if $t_0 \models tr(\varphi)$.*
- *If $s_0 \overset{\rightarrow}{=}_c t_0$ then for any TCTL formula φ, $s_0 \models \varphi$ if and only if $t_0 \models tr(\varphi)$.*

The rest of this section is devoted to the proof of this theorem. Before that we need to introduce some notation and preliminary observations.

Definition 11. Given two finite alternating runs ρ in A and ρ' in B of the form

$$\begin{aligned}
\rho &= s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \cdots \longrightarrow s_n \xrightarrow{d_n} s_n[d_n] \\
\rho' &= t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow \cdots \rightsquigarrow t_n \xrightarrow{d_n} t_n[d_n]
\end{aligned}$$

we write $\rho \overset{\rightarrow}{=} \rho'$ if $s_i[d] \overset{\rightarrow}{=} t_i[d]$ for all $i \leq n$ and all $d \leq d_i$.

Lemma 12. *Let $s_0 \in S$ and $t_0 \in T$ be such that $s_0 \overset{\rightarrow}{=} t_0$. There is a finite run*

$$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \cdots \longrightarrow s_n \xrightarrow{d_n} s_n[d_n]$$

in A if and only if there is a finite run

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow \cdots \rightsquigarrow t_n \xrightarrow{d_n} t_n[d_n]$$

in B such that $\rho \overset{\rightarrow}{=} \rho'$.

PROOF. Using Conditions 3 and 4 of Definition 8 we can for any finite run ρ in A construct a run ρ' in B such that $\rho \overset{\rightarrow}{=} \rho'$. Similarly, Conditions 5 and 6 allow us to construct a corresponding finite run ρ in A for every finite run ρ' in B. □

Lemma 12 is sufficient for proving the first part (safety fragment) of Theorem 10. For the second result of Theorem 10 we need some careful considerations about the maximal runs.

Definition 13. Given two maximal runs ρ in A and ρ' in B we write $\rho \overset{\rightarrow}{=} \rho'$ if

- ρ is an infinite maximal run

  $$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots,$$

  ρ' is an infinite maximal run

  $$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \ldots,$$

  and $s_i[d] \overset{\rightarrow}{=} t_i[d]$ for all $i \geq 0$ and all $d \leq d_i$, or

- ρ is a finite maximal run of the form

  $$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots \longrightarrow s_n \xrightarrow{\delta},$$

  ρ' is a finite maximal run of the form

  $$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \ldots \longrightarrow t_n \xrightarrow{\delta},$$

  for some $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$ such that,

  - $s_i[d] \overset{\rightarrow}{=} t_i[d]$ for all $i < n$ and all $d \leq d_i$,
  - $s_n[d] \overset{\rightarrow}{=} t_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$ if $\delta = \infty$,
  - $s_n[d] \overset{\rightarrow}{=} t_n[d]$ for all $d \leq d_n$ if $\delta = d_n^{\leq}$ and
  - $s_n[d] \overset{\rightarrow}{=} t_n[d]$ for all $d < d_n$ if $\delta = d_n^{<}$.

Lemma 14. *Let $s_0 \overset{\rightarrow}{=}_c t_0$. There is a maximal run $\rho \in MaxRuns(A, s_0)$ if and only if there is a maximal run $\rho' \in MaxRuns(B, t_0)$ such that $\rho \overset{\rightarrow}{=}_c \rho'$.*

PROOF. ($\Rightarrow$): Maximal runs can be either finite or infinite. For the infinite ones, we can use the same arguments as in the proof of Lemma 12. Hence we shall argue only for the finite maximal runs.

Let $s_0 \overset{\rightarrow}{=}_c t_0$ and let

$$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots \longrightarrow s_n \xrightarrow{\delta}$$

be a finite maximal run in A where $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$. We will show how to construct a finite maximal run in B of the form

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \ldots \rightsquigarrow t_n \xrightarrow{\delta}$$

such that $\rho \overset{\rightarrow}{=}_c \rho'$. Using Lemma 12 it follows that from the prefix of ρ up to and including s_n we can construct the prefix of ρ' up to and including t_n such that these two prefixes are related as desired and in particular $s_n \overset{\rightarrow}{=}_c t_n$. Hence we have to discuss only the last transition of the runs. Conditions 4 and 6 of Definition 8 imply, for all $d \in \mathbb{R}_{\geq 0}$, that $s_n \xrightarrow{d}$ iff $t_n \xrightarrow{d}$. So if $\delta = \infty$ then clearly $\rho \overset{\rightarrow}{=}_c \rho'$. If $\delta = d_n^{\leq}$ we still have that the same sets of delay transitions are possible from s_n and t_n. As B is *eventually-stable* we have that $t_n[d_n] \longrightarrow$ iff $t_n[d_n] \rightsquigarrow$. Thus $t_n[d_n] \not\rightsquigarrow$ since $s_n[d_n] \not\rightsquigarrow$. Hence $\rho \overset{\rightarrow}{=}_c \rho'$ also if $\delta = d_n^{<}$. The last case where $\delta = d_n^{<}$ is similar.

($\Leftarrow$): First, assume that there is an infinite maximal run in B of the form

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \longrightarrow t_1 \xrightarrow{d_1} t_1[d_1] \longrightarrow t_2 \xrightarrow{d_2} t_2[d_2] \longrightarrow \cdots$$

where intermediate states are also indexed. Let $j_1 < j_2 < j_3 < \ldots$ be all the indices such that $t_{j_i} \models \mathit{stable}$ for all $i > 0$. Since B is an *eventually-stable* TTS, there are infinitely many stable states in ρ'. Moreover, since B is a *delay-implies-stable* and *delay-preserves-stable* TTS, we can write ρ' in the form

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_{j_1} \xrightarrow{d_{j_1}} t_{j_1}[d_{j_1}] \rightsquigarrow t_{j_2} \xrightarrow{d_{j_2}} t_{j_2}[d_{j_2}] \rightsquigarrow \cdots \ .$$

Now, using Definition 8, we can easily construct an infinite maximal run $\rho \in \mathit{MaxRuns}(A, s_0)$ such that $\rho \overset{\rightarrow}{=}_c \rho'$.

Second, assume that there exists a finite maximal run in B of the form

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \longrightarrow t_1 \xrightarrow{d_1} t_1[d_1] \longrightarrow t_2 \xrightarrow{d_2} t_2[d_2] \longrightarrow \cdots \longrightarrow t_n \xrightarrow{\delta}$$

where intermediate states are also indexed and $\delta = \{\infty, d_n^{<}, d_n^{\leq}\}$. Let $j_1 < j_2 < j_3 < \ldots < j_k$ be, as before, all the indices such that $t_{j_i} \models \mathit{stable}$ for all $0 < i \leq k$. Since B is an *eventually-stable* TTS, we know that $j_k = n$. Further, because B is a *delay-implies-stable* and *delay-preserves-stable* TTS, we can write ρ' in the form

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_{j_1} \xrightarrow{d_{j_1}} t_{j_1}[d_{j_1}] \rightsquigarrow \cdots \rightsquigarrow t_{j_k} \xrightarrow{\delta} \ .$$

Following a similar strategy as in the other implication, we can construct a finite maximal run $\rho \in \mathit{MaxRuns}(A, s_0)$ that ends with $\xrightarrow{\delta}$ such that $\rho \overset{\rightarrow}{=}_c \rho'$. □

We are now ready to prove Theorem 10.

Proof (of Theorem 10). We shall prove this theorem by structural induction on φ. By Lemma 6 it is sufficient to handle the operators $\wp$, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $E(\varphi_1 \, U_I \, \varphi_2)$ and $E(\varphi_1 \, R_I \, \varphi_2)$. We start by arguing about the second part of the theorem assuming that $s_0 \overset{\rightarrow}{=}_c t_0$.

- Case $\varphi = \wp$ is trivial and cases $\varphi = \neg\varphi_1$ and $\varphi = \varphi_1 \wedge \varphi_2$ follow immediately from the induction hypothesis.

- Case $\varphi = E(\varphi_1 \, U_I \, \varphi_2)$:

  ($\Rightarrow$) : Assume that $s_0 \models_A E(\varphi_1 \, U_I \, \varphi_2)$. Thus there exists a maximal run $\rho \in \mathit{MaxRuns}(A, s_0)$ that witnesses φ, meaning that there exists a prefix of ρ of the form

  $$\rho_{\mathit{prefix}} = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{d_{n-1}} s_{n-1}[d_{n-1}] \longrightarrow s_n \xrightarrow{d} s_n[d]$$

  such that $\mathit{valid}_\rho(n, d, I)$, $s_n[d] \models_A \varphi_2$ and $s_k[d'] \models_A \varphi_1$ for all $(k, d') \in \mathit{history}_\rho(n, d)$. By Lemma 12 there is a run ρ'_{prefix} in B of the form

  $$\rho'_{\mathit{prefix}} = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow \cdots \rightsquigarrow t_{n-1} \xrightarrow{d_{n-1}} t_{n-1}[d_{n-1}] \rightsquigarrow t_n \xrightarrow{d} t_n[d]$$

  such that $\rho_{\mathit{prefix}} \overset{\rightarrow}{=} \rho'_{\mathit{prefix}}$ and because B is *eventually-stable* then also $\rho_{\mathit{prefix}} \overset{\rightarrow}{=}_c \rho'_{\mathit{prefix}}$.
  We want to show that $t_0 \models_B E((\mathit{tr}(\varphi_1) \vee \neg \mathit{stable}) \, U_I \, (\mathit{tr}(\varphi_2) \wedge \mathit{stable}))$. Because $s_n[d] \overset{\rightarrow}{=}_c t_n[d]$ and $s_n[d] \models_A \varphi_2$, we have by the induction hypothesis that $t_n[d] \models_B \mathit{tr}(\varphi_2)$ and from Condition 1 of Definition 8 we have that $t_n[d] \models_B \mathit{stable}$. Let j be the index corresponding to the occurrence of t_n in the alternating sequence which unfolds the $\rightsquigarrow$ steps. Because time delays are equivalent in the two runs, it follows that $\mathit{valid}_{\rho'}(j, d, I)$ and moreover, for any pair $(k, d') \in \mathit{history}_{\rho'}(j, d)$ either

  - $t_k[d']$ is an intermediate state and thus $t_k[d'] \models_B \neg \mathit{stable}$, or
  - $t_k[d']$ is a stable state. From the construction of ρ'_{prefix} it follows that there exists a pair $(k', d') \in \mathit{history}_\rho(n, d)$ such that $s_{k'}[d'] \overset{\rightarrow}{=}_c t_k[d']$. By the induction hypothesis and the fact that $s_{k'}[d'] \models_A \varphi_1$ we get $t_k[d'] \models_B \mathit{tr}(\varphi_1)$.

  This means that $t_k[d'] \models_B \mathit{tr}(\varphi_1) \vee \neg \mathit{stable}$.

  Thus any maximal run $\rho' \in \mathit{MaxRuns}(B, t_0)$ that extends ρ'_{prefix} witnesses $\mathit{tr}(\varphi)$, meaning that $t_0 \models_B E((\mathit{tr}(\varphi_1) \vee \neg \mathit{stable}) \, U_I \, (\mathit{tr}(\varphi_2) \wedge \mathit{stable}))$.

  ($\Leftarrow$) : Assume $t_0 \models_B E((\mathit{tr}(\varphi_1) \vee \neg \mathit{stable}) \, U_I \, (\mathit{tr}(\varphi_2) \wedge \mathit{stable}))$. Thus there exists a maximal run ρ' in B that witnesses $\mathit{tr}(\varphi)$. By the fact that B is a *delay-implies-stable* and *delay-preserves-stable* TTS there exists a prefix of ρ' of the form

  $$\rho'_{\mathit{prefix}} = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow \cdots \rightsquigarrow t_{n-1} \xrightarrow{d_{n-1}} t_{n-1}[d_{n-1}] \rightsquigarrow t_n \xrightarrow{d} t_n[d]$$

  such that $\mathit{valid}_{\rho'}(j, d, I)$, $t_n[d] \models_B \mathit{tr}(\varphi_2) \wedge \mathit{stable}$ and $t_k[d'] \models_B \mathit{tr}(\varphi_1) \vee \neg \mathit{stable}$ for all $(k, d') \in \mathit{history}_{\rho'}(j, d)$ where j is the index corresponding

to the occurrence of t_n in the alternating sequence which unfolds the $\rightsquigarrow$ steps as before.

By Lemma 12 there exists a run ρ_{prefix} in A of the form

$$\rho_{prefix} = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{d_{n-1}} s_{n-1}[d_{n-1}] \longrightarrow s_n \xrightarrow{d} s_n[d]$$

such that $\rho_{prefix} \overset{\rightarrow}{=} \rho'_{prefix}$ and hence also $\rho_{prefix} \overset{\rightarrow}{=}_c \rho'_{prefix}$.

We want to show that $s_0 \models_A E(\varphi_1 \, U_I \, \varphi_2)$. Because $s_n[d] \overset{\rightarrow}{=}_c t_n[d]$ and $t_n[d] \models_B tr(\varphi_2) \wedge \mathit{stable}$, we have by the induction hypothesis that $s_n[d] \models_A \varphi_2$. Since time delays are equivalent in the two runs, it follows that $\mathit{valid}_\rho(n, d, I)$. Further, for any pair $(k', d') \in \mathit{history}_\rho(n, d)$, it follows that there exists a $(k, d') \in \mathit{history}_{\rho'}(j, d)$ such that $s_{k'}[d'] \overset{\rightarrow}{=}_c t_k[d']$. By the induction hypothesis, it follows that $s_{k'}[d'] \models_A \varphi_1$ because $t_k[d'] \models_B tr(\varphi_1)$ due to the fact that $t_k[d'] \models_B \mathit{stable}$.

It then follows that any maximal run $\rho \in \mathit{MaxRuns}(A, s_0)$ starting with ρ_{prefix} witnesses φ, thus $s_0 \models_A E(\varphi_1 \, U_I \, \varphi_2)$.

- Case $\varphi = E(\varphi_1 \, R_I \, \varphi_2)$:

  ($\Rightarrow$) : Assume that $s_0 \models_A E(\varphi_1 \, R_I \, \varphi_2)$. By assumption, there exists a maximal run (infinite or finite)

  $$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \cdots$$

  in A that witnesses φ. This means that for all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$, if $\mathit{valid}_\rho(i, d, I)$ is true then either $s_i[d] \models_A \varphi_2$ or there exists a $(k, d') \in \mathit{history}_\rho(i, d)$ such that $s_k[d'] \models_A \varphi_1$.

  By Lemma 14 it follows that there exists a maximal run

  $$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \cdots \tag{1}$$

  in B such that $\rho \overset{\rightarrow}{=}_c \rho'$ (see Definition 13).

  We want to show that $t_0 \models_B E((tr(\varphi_1) \wedge \mathit{stable}) \, R_I \, (tr(\varphi_2) \vee \neg \mathit{stable}))$. For all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$, if $\mathit{valid}_{\rho'}(i, d, I)$ is true then either

  - $t_i[d]$ is an intermediate state and then $t_i[d] \models_B \neg \mathit{stable}$, or
  - $t_i[d]$ is a stable state. This means that $t_i[d]$ has a distinguished index in Equation 1. Let h be the index of $t_i[d]$ in Equation 1. It follows from the construction of ρ' that $s_h[d] \overset{\rightarrow}{=}_c t_h[d]$. There are two cases to consider:
    - either $s_h[d] \models_A \varphi_2$ and then by the induction hypothesis it follows that $t_h[d] \models_B tr(\varphi_2)$, or

* there exists $(\ell', d') \in \mathit{history}_{\rho}(h, d)$ such that $s_{\ell'}[d'] \models_A \varphi_1$. From the construction of ρ' it follows that there exists a pair $(\ell, d') \in \mathit{history}_{\rho'}(i, d)$ such that $s_{\ell'}[d'] \stackrel{\rightarrow}{=}_c t_{\ell}[d']$. By the induction hypothesis, it follows that $t_{\ell}[d'] \models_B tr(\varphi_1) \wedge \mathit{stable}$.

This in turn means that $t_0 \models_B E((tr(\varphi_1) \wedge \mathit{stable})\, R_I\, (tr(\varphi_2) \vee \neg \mathit{stable}))$.

($\Leftarrow$) : Assume that $t_0 \models_B E((tr(\varphi_1) \wedge \mathit{stable})\, R_I\, (tr(\varphi_2) \vee \neg \mathit{stable}))$. Hence there exists a maximal run (infinite or finite) in B that witnesses $tr(\varphi)$ and since B is a *delay-implies-stable*, *delay-preserves-stable* and *eventually-stable* TTS, the run has the following form:

$$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \cdots$$

This means that for all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$, if $\mathit{valid}_{\rho'}(i, d, I)$ is true then either $t_i[d] \models_B tr(\varphi_2) \vee \neg \mathit{stable}$ or there exists a $(k, d') \in \mathit{history}_{\rho'}(i, d)$ such that $t_k[d'] \models_B tr(\varphi_1) \wedge \mathit{stable}$.

By Lemma 14 it follows that there exists a maximal run

$$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \cdots$$

in A such that $\rho \stackrel{\rightarrow}{=}_c \rho'$.

We want to show that $s_0 \models_A E(\varphi_1\, R_I\, \varphi_2)$. For all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$, we have that $s_i[d] \stackrel{\rightarrow}{=}_c t_i[d]$ and whenever $\mathit{valid}_{\rho}(i, d, I)$ is true then

- either $t_i[d] \models_B tr(\varphi_2)$ and then by the induction hypothesis we have that $s_i[d] \models_A \varphi_2$, or
- there exists some $(\ell', d') \in \mathit{history}_{\rho'}(j, d)$, where j is the index corresponding to the occurrence of t_i in the alternating sequence which unfolds the $\rightsquigarrow$ steps, such that $t_{\ell'}[d'] \models_B tr(\varphi_1) \wedge \mathit{stable}$. From the construction of ρ, it follows that there exists a pair $(\ell, d') \in \mathit{history}_{\rho}(i, d)$ such that $s_{\ell}[d'] \stackrel{\rightarrow}{=}_c t_{\ell'}[d']$. By the induction hypothesis, it follows that $s_{\ell}[d'] \models_A \varphi_1$.

This in turn means that $s_0 \models_A E(\varphi_1\, R_I\, \varphi_2)$.

Observe that for the case of $E(\varphi_1\, U_I\, \varphi_2)$ we used Lemma 12 which requires only a one-by-many correspondence. On the other hand, to prove the case of $E(\varphi_1\, R_I\, \varphi_2)$ we used the *eventually-stable* property and Lemma 14, which requires a complete one-by-many correspondence. Hence, if the relation is only a one-by-many correspondence then the theorem still works for the safety fragment of TCTL as claimed in the first part of the theorem. □

Example 15. The reason we need a complete one-by-many correspondence to preserve the full TCTL can be illustrated by considering systems A and C in Figure 4 where $\{(s_0, u_0), (s_1, u_1), (s_2, u_4), (s_3, u_7)\}$ is a one-by-many correspondence between states in A and C. In this particular example, $s_0 \models_A A(p\, U_{[3,5]}\, q)$

but $u_0 \not\models_B tr(A(p\,U_{[3,5]}\,q)) = A((p \vee \neg stable)\,U_{[3,5]}\,(q \wedge stable))$. Both of the following maximal runs

$$\rho = u_0 \xrightarrow{4.4} u_1 \longrightarrow u_2 \xrightarrow{0} u_2 \longrightarrow u_3 \xrightarrow{0^{\leq}}$$
$$\rho' = u_0 \xrightarrow{4.4} u_1 \longrightarrow u_5 \xrightarrow{0} u_5 \longrightarrow u_6 \xrightarrow{0} u_6 \longrightarrow u_6 \xrightarrow{0} u_6 \longrightarrow u_6 \xrightarrow{0} \cdots$$

in C are counter examples to $tr(A(p\,U_{[3,5]}\,q))$.

2.4. Overall Methodology

We finish this section by recalling the steps needed in order to apply the framework to a particular translation between two time-dependent systems. Assume that we designed an algorithm that for a given system A constructs a system B together with the notion of stable states in the system B. In order to apply our framework, we need to perform the following steps.

1. Show that B is a *delay-implies-stable* and *delay-preserves-stable* TTS (and optionally an *eventually-stable* TTS).
2. Define a proposition translation function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$.
3. Define a relation $\mathcal{R}$ and show that it fulfills Conditions 1–6 of Definition 8.

Theorem 10 now allows us to conclude that the translation preserves the full TCTL (or its safety fragment if $\mathcal{R}$ is only a one-by-many correspondence).

The framework generalizes earlier translations presented in [18] and [21] that dealt with concrete cases. Apart from these, the framework covers some other concrete results as those in [14, 24, 30, 46].

In what follows, we shall demonstrate an application of this general method to proving the correctness of two translation from timed-arc Petri nets to networks of timed automata.

3. Timed-Arc Petri Net

We shall now define the model of Timed-Arc Petri nets (TAPN) where time features are associated to tokens in the net and arcs contain time intervals restricting the ages of tokens available for transition firing. We define the model in its full generality as implemented in TAPAAL, including age invariants and transport/inhibitor arcs. We start by an informal description of the model.

Figure 5 shows a graphical representation of a simple TAPN. The net contains five places drawn as circles, and one transition drawn as a rectangle. In the places $p0$ and $p1$ there is one token of age 0, and in the place $p3$ there is one token of age 2. The tokens' age and their placement in the net constitute a *marking*. The net contains three types of arcs: normal arcs drawn with an arrow tip, transport arcs drawn with a diamond tip, and inhibitor arcs drawn with a circle tip. Arcs pointing from a place to a transition are called *input arcs* and arcs from transitions to places are called *output arcs*. The places $p2$ and $p5$ moreover contains *age invariants* written below the places. Note that the transport arcs always come in pairs of an input and an output arc and the

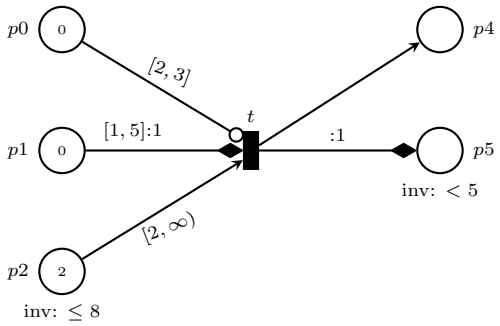

Figure 5: A timed arc Petri Net

annotation :1 after the interval and on the output arc denotes the pairing of the two arcs (if there was another pair of transport arcs associated with the transition t it would have the annotation :2 and so on).

The behaviour of the net is described through evolution of markings. A marking can evolve either through a *time-delay* where all tokens in the net simultaneously become older, or trough *transition firing* where tokens are moved between places. A time-delay increases the age of all tokens with the same value as long as no tokens break the age invariants in their locations. In the example of Figure 5 we can delay e.g. 1.7 units of time but we can not make a time-delay of 6.1 as this would break the age invariant in place $p2$. A transition can *fire* only if all its normal and transport input arcs have at least one token in their input places such that the ages of these tokens satisfy the time intervals on the corresponding arcs. On the contrary, an inhibitor arc inhibits the firing of a transition if there is a token within the time-interval on the inhibitor arc. When we fire a transition we remove a token from each input place and create a new token in each of the output places. For normal arcs we produce new tokens of age 0; for transport arcs we carry the age of the consumed token to the target place. Moreover, a transition can only be fired if the tokens transmitted by the transport arcs do not violate any age invariants.

3.1. Syntax

We define the set of invariant intervals $\mathcal{I}^{inv}$ as a subset of time intervals $\mathcal{I}$ that contain the value zero, i.e., $\mathcal{I}^{inv} = \{I \in \mathcal{I} \mid 0 \in I\}$. Let also **Type** be a set of arc types such that $\textbf{Type} = \{Normal, Inhib\} \cup \{Transport_i \mid i \in \mathbb{N}\}$ where the transport arc types have an index as we need to pair exactly one input and output arc in order to form a pair of transport arcs.

Definition 16 (Timed-Arc Petri Net). A timed-arc Petri net (TAPN) is a 7-tuple $N = (P, T, IA, OA, c, Type, Inv)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $P \cap T = \emptyset$,

- $IA \subseteq P \times T$ is a finite set of *input arcs*,
- $OA \subseteq T \times P$ is a finite set of *output arcs*,
- $c : IA \to \mathcal{I}$ is a *time constraint function* assigning intervals to all input arcs,
- $Type : IA \cup OA \to$ **Type** is a *type function* assigning a type to all arcs such that
  - if $Type(a) = Inhib$ then $a \in IA$,
  - if $Type((p,t)) = Transport_i$ for some $(p,t) \in IA$ then there is exactly one $(t,p') \in OA$ such that $Type((t,p')) = Transport_i$, and
  - if $Type((t,p')) = Transport_i$ for some $(t,p') \in OA$ then there is exactly one $(p,t) \in IA$ such that $Type((p,t)) = Transport_i$,
- $Inv : P \to \mathcal{I}^{inv}$ is a function assigning *age invariants* to places.

The preset of input places of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p,t) \in IA, Type((p,t)) \neq Inhib\}$. Similarly, the postset of output places of t is defined as $t^\bullet = \{p \in P \mid (t,p) \in OA\}$. Let the inhibitor preset be defined as ${}^\circ t = \{p \in P \mid (p,t) \in IA, Type((p,t)) = Inhib\}$. We also let $\max(t) = \max(|{}^\bullet t|, |t^\bullet|)$ denote the largest in or out degree of the transition t.

3.2. Semantics

We will now introduce the semantics of TAPN. Given a set X, we define the set of finite multisets on X, $\mathcal{B}(X)$, as the set of functions $B : X \to \mathbb{N}_0$ such that $|\{x \in X \mid B(x) > 0\}| < \infty$. We say that $x \in B$ iff $B(x) > 0$ and use the standard set notation and operations for manipulating with multisets.

Now we define the markings of a TAPN considering finite multisets of non-negative real numbers in $\mathcal{B}(\mathbb{R}_{\geq 0})$. These real numbers represent the ages of tokens in each place and of course they have to respect the corresponding age invariants.

Definition 17 (Marking). Let $N = (P, T, IA, OA, c, Type, Inv)$ be a TAPN. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}_{\geq 0})$ where for every place $p \in P$ and every (age of a) token $x \in M(p)$ we have $x \in Inv(p)$. The set of all markings over N is denoted by $\mathcal{M}(N)$.

We will use the notation (p,x) to refer to a token in the place p of age $x \in \mathbb{R}_{\geq 0}$. Likewise, we will write $M = \{(p_1,x_1),(p_2,x_2),\ldots,(p_n,x_n)\}$ for a multiset representing a marking M with n tokens. A *marked* TAPN is a pair (N, M_0) where N is a TAPN and M_0 is an initial marking on N where all tokens have the age 0.

Definition 18 (Enabledness). Let $N = (P, T, IA, OA, c, Type, Inv)$ be a TAPN. We say that a transition $t \in T$ is *enabled* in a marking M by the sets of tokens $In(t) = \{(p, x_p) \mid p \in {}^\bullet t\} \subseteq M$ if

- for all input arcs except the inhibitor arcs there is a token in the input place with an age satisfying the age guard of the arc, i.e.

$$\forall p \in {}^\bullet t.\ x_p \in c((p,t))$$

- for all inhibitor arcs there is no token in the input place of the arc with an age satisfying the age guard of the arcs respectively, i.e.

$$\forall p \in t^\circ.\ \forall x \in M(p).\ x \notin c((p,t))$$

- for any pair of transport arcs, the age of the transported token must satisfy the invariant of the target place, i.e.

$$\forall (p,t) \in IA.\ \forall (t,p') \in OA.$$

$$Type((p,t)) = Type((t,p')) = Transport_i \Rightarrow x_p \in Inv(p')\ .$$

Definition 19 (Behaviour). Let $N = (P, T, IA, OA, c, Type, Inv)$ be a TAPN, M a marking on N and $t \in T$ a transition.

- Let t be enabled in the marking M by the set $In(t) = \{(p, x_p) \mid p \in {}^\bullet t\} \subseteq M$ and let

$$Out(t) = \{(p', 0) \mid (t,p') \in OA,\ Type((t,p')) = Normal\} \cup$$

$$\{(p', x_p) \mid (t,p') \in OA,\ Type((t,p')) = Transport_i = Type((p,t))\}\ .$$

  Then t can *fire* and produce a marking M' defined as

$$M' = (M \setminus In(t)) \cup Out(t)$$

  where $\setminus$ and $\cup$ are operations on multisets.

- A *time delay* $d \in \mathbb{R}_{\geq 0}$ is allowed in M if $(x + d) \in Inv(p)$ for all $p \in P$ and all $x \in M(p)$, i.e., by delaying d time units no token violates any of the age invariants. By delaying d time units in M we reach a marking M' defined as

$$M'(p) = \{x + d \mid x \in M(p)\}$$

  for all $p \in P$.

A given TAPN N defines a timed transition system $(\mathcal{M}(N), \longrightarrow, \mathcal{AP}, \mu)$ where states are markings of N and for two markings M and M' we have $M \longrightarrow M'$ if by firing some transition in M we can reach the marking M' and $M \xrightarrow{d} M'$ if by delaying d time units in M we reach the marking M'. For notational convenience, we shall sometimes annotate discrete transitions with the names of the transitions that were fired. Observe that due to age invariants there may be markings from which no firing nor time delay transitions are possible.

The atomic propositions are defined as

$$\mathcal{AP} \stackrel{\text{def}}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\} .$$

The interpretation is that a proposition $(p \bowtie n)$ is true in a marking M iff the number of tokens in the place p satisfies the proposition in question with respect to n, formally $\mu(M) \stackrel{\text{def}}{=} \{(p \bowtie n) \mid |M(p)| \bowtie n\}$ where $\bowtie$ is one of the (standard mathematical) operators and $|M(p)|$ is the cardinality of the multiset $M(p)$.

We made the choice of observing only the number of tokens in the places of the net, not their ages. This will simplify our exposition but it is still possible to ask about the presence of a token of a given age d in a place p. This can be, for example, done by adding a new transition consuming a token in the interval $[d, d]$ from p while marking a newly added "observer" place. Then the reachability of a marking with a token in the added place corresponds to the presence of a token of the age d in the place p.

4. Networks of Timed Automata

Now we introduce the model of Networks of Timed Automata (NTA) with integer variables, handshake synchronization and broadcast synchronization.

4.1. Preliminaries

Let $C = \{c_1, c_2, \ldots\}$ be a finite set of real-valued *clocks*. A *clock guard* is a boolean expression defined by the abstract syntax:

$$g_1, g_2 ::= \mathit{true} \mid c_1 \bowtie n \mid c_1 - c_2 \bowtie n \mid g_1 \wedge g_2$$

where $c_1, c_2 \in C$, $n \in \mathbb{N}_0$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. The set of all clock guards over the set of clocks C is denoted by $\mathcal{G}(C)$. Invariants are defined as guards with the restriction that $\bowtie \in \{\leq, <\}$. We denote the set of all clock invariants over the set of clocks C as $\mathcal{G}_{Inv}(C)$.

A *clock valuation* of C is a function $v : C \to \mathbb{R}_{\geq 0}$. Let v be a valuation and $d \in \mathbb{R}_{\geq 0}$. We define the valuation $v + d$, where all clocks are delayed by d, by $(v + d)(c) \stackrel{\text{def}}{=} v(c) + d$ for every $c \in C$. For every set $R \subseteq C$ we define the valuation $v[R] : C \to \mathbb{R}_{\geq 0}$, where all clocks in R are reset to zero, by $v[R](c) \stackrel{\text{def}}{=} v(c)$ for $c \in C \setminus R$ and $v[R](c) = 0$ for $c \in R$.

The satisfaction relation $v \models g$ (i.e. when a valuation satisfies a guard or an invariant g) is defined in the natural way.

Let X be a finite set of integer variables. *Arithmetic variable expressions over X* are defined by the abstract syntax:

$$\mathit{expr}_1, \mathit{expr}_2 ::= m \mid x \mid \mathit{expr}_1 + \mathit{expr}_2 \mid \mathit{expr}_1 - \mathit{expr}_2$$

where $m \in \mathbb{Z}$ and $x \in X$. We denote the set of all arithmetic variable expressions over the set of variables X by $VE(X)$.

A *variable guard over* X is a boolean expression defined by the following abstract syntax

$$\phi_1, \phi_2 ::= \mathit{true} \mid \mathit{expr}_1 \bowtie \mathit{expr}_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

where $\mathit{expr}_1, \mathit{expr}_2 \in \mathit{VE}(X)$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. We denote the set of all variable guards over the set of variables X by $\mathit{VG}(X)$.

Variable assignments over X are expressions of the form $x := \mathit{expr}$ where $x \in X$ and $\mathit{expr} \in \mathit{VE}(X)$. We denote the set of all variable assignments over X by $\mathit{VA}(X)$. A set of *non-conflicting variable assignments* $\mathcal{A}$ is a finite subset of $\mathit{VA}(X)$ where for every $x \in X$ whenever $(x := \mathit{expr}_1) \in \mathcal{A}$ and $(x := \mathit{expr}_2) \in \mathcal{A}$ then $\mathit{expr}_1 = \mathit{expr}_2$.

A *variable valuation* is a function $z : X \longrightarrow \mathbb{Z}$ and it is in the natural way extended to arithmetic variable expressions. A variable valuation z satisfies a variable guard $\phi \in \mathit{VG}(X)$ (written as $z \models \phi$) if the variable guard evaluates to true under the valuation z.

Given a set of non-conflicting variable assignments $\mathcal{A}$ and a variable valuation z, by $z[\mathcal{A}]$ we denote the variable valuation such that

$$z[\mathcal{A}](x) = \begin{cases} z(\mathit{expr}) & \text{if } (x := \mathit{expr}) \in \mathcal{A} \\ z(x) & \text{otherwise.} \end{cases}$$

4.2. Syntax

We can now define the model of timed automata.

Definition 20 (Timed Automaton). A *Timed Automaton* (TA) is a tuple $(L, \mathit{Act}, C, X, \longrightarrow, I_C, I_X, \ell_0)$, where

- L is a finite set of locations,
- Act is a finite set of actions,
- C is a finite set of clocks,
- X is a finite set of integer variables,
- $\longrightarrow \subseteq L \times \mathcal{G}(C) \times \mathit{VG}(X) \times \mathit{Act} \times 2^C \times 2^{\mathit{VA}(X)} \times L$ is a finite set of edges where for any $(\ell, g, \phi, a, R, \mathcal{A}, \ell') \in \longrightarrow$ we require that $\mathcal{A}$ is a finite non-conflicting set of variable assignments,
- $I_C : L \to \mathcal{G}_{\mathit{Inv}}(C)$ is a function assigning clock invariants to locations,
- $I_X : L \to \mathit{VG}(X)$ is a function assigning variable invariants to locations,
- ℓ_0 is the initial location.

We write $\ell \xrightarrow{g,\phi,a,R,\mathcal{A}} \ell'$ instead of $(\ell, g, \phi, a, R, \mathcal{A}, \ell') \in \longrightarrow$ and say that ℓ is the source location, g is the tested clock guard and ϕ the variable guard, a is the executed action, R is the set of clocks to be reset, $\mathcal{A}$ is the set of non-conflicting

variable assignments to make and ℓ' is the target location. For notational convenience, we shall sometimes conjunct clock and variable invariants when drawing the automata.

A network of timed automata (NTA) is a parallel composition of a finite number of timed automata. We adopt the handshake synchronization and broadcast synchronization as supported by UPPAAL.

Let *Chan* be a finite set of channels for handshake synchronization and let *Broad* be a finite set of channels for broadcast synchronization such that $\mathit{Chan} \cap \mathit{Broad} = \emptyset$. Let τ be an internal action such that $\tau \notin \mathit{Chan} \cup \mathit{Broad}$. We then define

$$\mathit{Act} = \{c!, c? \mid c \in \mathit{Chan}\} \cup \{a!, a? \mid a \in \mathit{Broad}\} \cup \{\tau\} \ .$$

The intuition is that $c!$ makes a handshake synchronization request on the channel c and $c?$ is used to indicate that an automaton is prepared to receive the handshake signal. Furthermore, $a!$ indicates broadcasting on the channel a and $a?$ means that an automaton is prepared to receive a broadcast signal on the channel a.

Definition 21 (Network of Timed Automata). Let $n \in \mathbb{N}$ and let $A_i = (L_i, \mathit{Act}, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$, for all $1 \leq i \leq n$, be timed automata over a fixed set of actions *Act*, clocks C and integer variables X. A *Network of Timed Automata* (NTA) is a parallel composition of $A_1, A_2, \ldots, A_n$ denoted by $A = A_1 \parallel A_2 \parallel \ldots \parallel A_n$.

4.3. Semantics

The semantics of a network of timed automata is given in terms of a TTS. The technical details, in particular of the broadcast synchronization, may look complicated but this is a necessary step as our aim is to argue about (and implement) translations producing UPPAAL timed automata and then use the model checker UPPAAL. The semantics below reflects the semantical choices made in the tool.

A configuration of an NTA is a tuple $(\ell_1, \ell_2, \ldots, \ell_n, z, v)$ where $\ell_i \in L_i$ for all $1 \leq i \leq n$, z is a variable valuation over X and v is a clock valuation over C such that for every i, $1 \leq i \leq n$, we have $z \models I_X^i(\ell_i)$ and $v \models I_C^i(\ell_i)$. We will denote the set of all configurations of a given NTA A by $\mathit{Conf}(A)$.

An NTA $A = A_1 \parallel A_2 \parallel \ldots \parallel A_n$ where $A_i = (L_i, \mathit{Act}, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$ generates a TTS $T(A) = (S, \longrightarrow, \mathcal{AP}, \mu)$ such that

- $S = \mathit{Conf}(A)$,
- the transition relation $\longrightarrow$ consists of
  - *ordinary transitions:* $(\ell_1, \ldots, \ell_i, \ldots, \ell_n, z, v) \longrightarrow (\ell_1, \ldots, \ell_i', \ldots, \ell_n, z', v')$ if there exists an edge $\ell_i \xrightarrow{g,\phi,\tau,R,\mathcal{A}}_i \ell_i'$ in the i'th automaton such that $v \models g$, $z \models \phi$, $v' = v[R]$, $z' = z[\mathcal{A}]$, $v' \models I_C^i(\ell_i') \wedge \bigwedge_{j \neq i} I_C^j(\ell_j)$ and $z' \models I_X^i(\ell_i') \wedge \bigwedge_{j \neq i} I_X^j(\ell_j)$,

– *handshake synchronization transitions:*
$(\ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, z, v) \longrightarrow (\ell_1, \ldots, \ell'_i, \ldots, \ell'_j, \ldots, \ell_n, z', v')$ if $i \neq j$ and there are edges $\ell_i \xrightarrow{g_i, \phi_i, a!, R_i, \mathcal{A}_i}_i \ell'_i$ and $\ell_j \xrightarrow{g_j, \phi_j, a?, R_j, \mathcal{A}_j}_j \ell'_j$ such that $v \models g_i \wedge g_j$, $z \models \phi_i \wedge \phi_j$, $v' = v[R_i \cup R_j]$, $z' = (z[\mathcal{A}_i])[\mathcal{A}_j]$, $v' \models I^i_C(\ell'_i) \wedge I^j_C(\ell'_j) \wedge \bigwedge_{k \neq i,j} I^k_C(\ell_k)$ and $z' \models I^i_X(\ell'_i) \wedge I^j_X(\ell'_j) \wedge \bigwedge_{k \neq i,j} I^k_X(\ell_k)$,
– *broadcast synchronization transitions:*
$(\ell_1, \ldots, \ell_n, z, v) \longrightarrow (\ell'_1, \ldots, \ell'_n, z', v')$ if there exists an $a \in Broad$ such that
  * there exists an i, $1 \leq i \leq n$, such that there is an edge $\ell_i \xrightarrow{g_i, \phi_i, a!, R_i, \mathcal{A}_i}_i \ell'_i$ in the i'th automaton and $v \models g_i$ and $z \models \phi_i$,
  * let $\mathcal{J}$ be the set of all j, $1 \leq j \neq i \leq n$ where there is an edge $\ell_j \xrightarrow{g_j, \phi_j, a?, R_j, \mathcal{A}_j}_j \ell'_j$ in the j'th automaton such that $v \models g_j$ and $z \models \phi_j$,
  * for all $j \in \mathcal{J}$ we set ℓ'_j, $\mathcal{A}_j$ and R_j according to the edge $\ell_j \xrightarrow{g_j, \phi_j, a?, R_j, \mathcal{A}_j}_j \ell'_j$ (note that there may be more options),
  * for all $j \notin \mathcal{J}$, $1 \leq j \neq i \leq n$ we let $\ell'_j = \ell_j$, $\mathcal{A}_j = \emptyset$ and $R_j = \emptyset$,
  * $z' = (\ldots((\ldots(z[\mathcal{A}_i])[\mathcal{A}_1])[\mathcal{A}_2])\ldots[\mathcal{A}_{i-1}])[\mathcal{A}_{i+1}])\ldots[\mathcal{A}_n]$ and $z' \models \bigwedge_{k=1}^n I^k_X(\ell'_k)$, and
  * $v' = v[R]$ where $R = \bigcup_{k=1}^n R_k$ and $v' \models \bigwedge_{k=1}^n I^k_C(\ell'_k)$,
– *delay transitions:*
$(\ell_1, \ldots, \ell_n, z, v) \xrightarrow{d} (\ell_1, \ldots, \ell_n, z, v + d)$ for all $d \in \mathbb{R}_{\geq 0}$ such that $v + d \models \bigwedge_{i=1}^n I^i_C(\ell_i)$,

- $\mathcal{AP} \stackrel{\text{def}}{=} \{(\#\ell \bowtie m) \mid \ell \in \cup_{i=1}^n L_i, m \in \mathbb{N} \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$, and
- $\mu : S \longrightarrow 2^{\mathcal{AP}}$ is a function assigning sets of true atomic propositions to states. A proposition $(\#\ell \bowtie m)$ is true in a configuration $s \in S$ if and only if the number of parallel components that are currently in the location ℓ satisfies the proposition with respect to m.

The initial state is $(\ell^1_0, \ell^2_0, \ldots, \ell^n_0, z_0, v_0)$ where $z_0(x) = 0$ for all $x \in X$ and $v_0(c) = 0$ for all $c \in C$. We assume that z_0 and v_0 always satisfy the invariants, i.e. $z_0 \models \bigwedge_{i=1}^n I^i_X(\ell^i_0)$ and $v_0 \models \bigwedge_{i=1}^n I^i_C(\ell^i_0)$.

Remark 22. The set of atomic propositions used in the tool UPPAAL is more general; we defined only those that are needed for the purpose of our translations. Also note that for handshake and broadcast synchronizations, variable assignments are always evaluated in a specific order. First any assignments on the edge of the sender are evaluated and after that the assignments of any receivers are evaluated in the order from A_1 to A_n. This follows the semantics of UPPAAL timed automata and it will become important in the liveness-preserving translation. Finally, observe that transitions cannot be taken if the resulting configuration breaks some of the clock or variable invariants.

5. Safety-Preserving Translation

In this section we describe a safety-preserving reduction from timed-arc Petri nets to networks of timed automata so that we can use the model checker UPPAAL to verify TAPN properties. As the reachability problem for TAPN is in general undecidable [48], we restrict ourselves to bounded nets. A TAPN is k-bounded if the total number of tokens in any of its reachable markings is no more than k. A net is called *bounded* if it is k-bounded for some k.

Remark 23. In the actual implementation, we ask the user to provide a suggestion for k before we run the verification. If the net is k-bounded, and this is a decidable problem, then we provide conclusive answers. If not, then we still explore the state-space up to k tokens and report possibly discovered error traces but the answer can be also inconclusive. In this case the user is invited to increase k and try again.

In our first reduction we do not consider inhibitor arcs. This will allow us to develop an efficient safety-preserving translation for nets that do not use inhibitor arcs. In the section to follow, we will describe a different translation approach supporting inhibitor arcs and preserving also liveness properties.

In the presentation of the safety-preserving translation, we shall first describe a reduction from bounded TAPNs without inhibitor arcs to TAPNs of *degree* 2 where every transition has exactly two input and two output arcs. This reduction is followed by a translation from TAPNs of degree 2 to networks of timed automata using a handshake synchronization.

5.1. From k-Bounded TAPN to TAPN of Degree 2

To translate a given k-bounded TAPN with transitions that have more than two input or output places into a TAPN of degree 2 we have to simulate a single transition firing in the original net by a series of transitions in the net of degree 2. The problem is that when firing a given transition in a number of steps, other transition firings may interleave and thus some extra behaviour can be introduced. To prevent this from happening, we introduce a new mutex-like place called p_{stable} such that it contains a token that is consumed before the sequence of transition firings begins and the token is returned back after the simulation of the selected transition is ended.

The translation is demonstrated in Figure 6, where a simple 3-bounded TAPN is translated into a TAPN of degree 2. The idea is that the token in the place p_{stable} will travel through the intermediate places $p(t^1_{in})$, $p(t^2_{in})$, $p(t^2_{out})$, $p(t^1_{out})$ and finally return to p_{stable}. When the first transition $p(t^1_{in})$ is fired, a token of a suitable age from p_0 is consumed and it is placed into the holding place $p_h(t^1)$. Then a token from p_1 is consumed and placed into $p_h(t^2)$. Because $|{}^\bullet t| < |t^\bullet|$ a special place called $p_{capacity}$ (used as a repository of the presently unused tokens) is created. By firing the transition t^3 a new token of age 0 is produced into p_3. Finally, the tokens placed in $p_h(t^2)$ and $p_h(t^1)$ are moved to the appropriate output places by firing the transitions t^2_{out} and t^1_{out}. Note

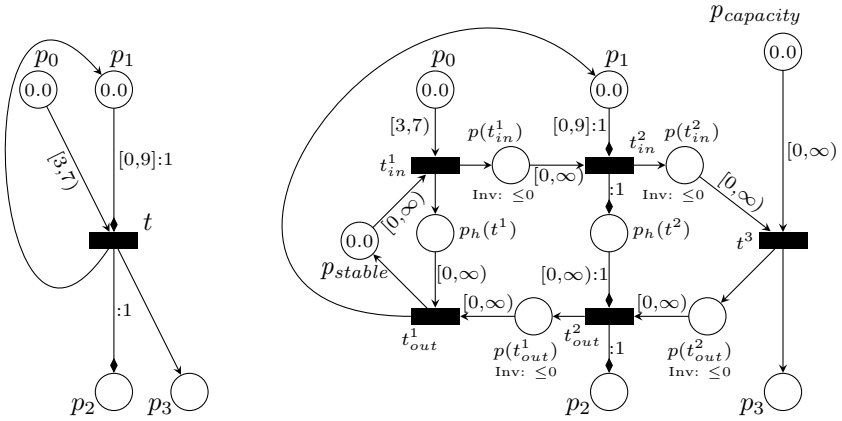

Figure 6: Example of a 3-bounded net and the corresponding net of degree 2

that because of the invariants on the intermediate places, time cannot elapse during such a series of transition firing and so the age of the token in p_1, that is moved via the two transport arcs into p_2, is preserved. Notice that the use of the holding places $p_h(t^1)$ and $p_h(t^2)$ is essential as without them a token from p_0 can be consumed by t^1_{in} while creating a new token in p_1. This may allow to fire t^2_{in} even if there were no tokens in p_1 in the original net. Also notice that by this construction we may introduce extra deadlocks (e.g. if there is no token present in p_1 after the transition t^1_{in} was fired). This relaxes the *eventually-stable* property but it is sufficient for preservation of safety properties.

Let us introduce some notation for each transition $t \in T$. The intuition is that the set $Pairing(t)$ defined below fixes the paths from input to output places on which the tokens travel when firing the transition t, and it also remembers the associated time intervals and the type of the path (*tarc* for transport arcs and *normal* for the standard arcs that reset the ages of produced tokens). There may be more than one pairing that satisfies the definition. It is important (for the use in the algorithms) to fix one such pairing.

$$
\begin{aligned}
\mathit{Pairing}(t) = \big\{ & (p, I, p', \mathit{tarc}) \mid (p,t) \in \mathit{IA}, c((p,t)) = I, (t,p') \in \mathit{OA}, \\
& \qquad \mathit{Type}((p,t)) = \mathit{Type}((t,p')) = \mathit{Transport}_i \big\} \cup \\
& \big\{ (p_1, I_1, p'_1, \mathit{normal}), \ldots, (p_m, I_m, p'_m, \mathit{normal}) \mid \\
& \qquad \{p_1, \ldots, p_\ell\} = \{p \mid (p,t) \in \mathit{IA}, \mathit{Type}((p,t)) = \mathit{Normal}\}, \\
& \qquad \{p'_1, \ldots, p'_{\ell'}\} = \{p \mid (t,p) \in \mathit{OA}, \mathit{Type}((t,p)) = \mathit{Normal}\}, \\
& \qquad m = \max(\ell, \ell'),\ I_i = c((p_i,t)) \text{ if } 1 \le i \le \ell \text{ else } I_i = [0,\infty), \\
& \qquad p_i = p_{\mathit{capacity}} \text{ if } \ell < i \le m,\ p'_i = p_{\mathit{capacity}} \text{ if } \ell' < i \le m \big\} \ .
\end{aligned}
$$

For the net in Figure 6 where $\max(t) = 3$ a possible pairing is $\mathit{Pairing}(t) = \{(p_0, [3,7), p_1, \mathit{normal}), (p_1, [0,9], p_2, \mathit{tarc}), (p_{\mathit{capacity}}, [0,\infty), p_3, \mathit{normal})\}$. The special capacity place simply ensures that the number of tokens in the net is constant.

Moreover, by $p \xrightarrow{I} t \longrightarrow p'$ we shall abbreviate the presence of an arc from p to t with the time interval I and an arc from t to p'; the type of the arcs (normal or transport) will be clear from the context. The complete translation is now given in Algorithm 1 and it clearly terminates as the body of the while-loop decreases the size of the set pairing. Notice that the algorithm causes only a modest growth in the size of the net. More precisely, the number of places P' in the constructed net is bounded by $|P| + 2 + 3|T|(D-1)$ and the number of transitions T' is bounded by $2|T|D$ where D is the maximum degree of any transition in the net, in other words the maximum of $\max(t)$ over all transitions $t \in T$.

We shall now argue about the correctness of the translation by using the methodology described in Section 2.4. Given a k-bounded TAPN (N, M_0), let (N', M'_0) be the corresponding net of degree 2 constructed by Algorithm 1. We define the proposition *stable* as $(p_{\mathit{stable}} = 1)$ meaning that stable markings contain a token in the place p_{stable}. The proposition translation function tr_p is defined as the identity function. We define the correspondence relation $\mathcal{R}$ between markings of $N = (P, T, \mathit{IA}, \mathit{OA}, c, \mathit{Type}, \mathit{Inv})$ and $N' = (P', T', \mathit{IA}', \mathit{OA}', c', \mathit{Type}', \mathit{Inv}')$ so that $(M, M') \in \mathcal{R}$ if and only if

$$
M'(p) = \begin{cases} M(p) & \text{if } p \in P \\ \{x\} & \text{if } p = p_{\mathit{stable}} \\ \{x_1, \ldots, x_{k-|M|}\} & \text{if } p = p_{\mathit{capacity}} \\ \emptyset & \text{otherwise .} \end{cases} \quad \text{for some } x, x_1, \ldots, x_{k-|M|} \in \mathbb{R}^{\ge 0}
$$

As mentioned before, the place p_{capacity} contains all unused tokens in the k-bounded TAPN N. Hence it contains $k - |M|$ tokens. The whole net can contain either $k+1$ or k tokens, depending on whether there is a token in the place p_{stable} or not.

Lemma 24. *The relation $\mathcal{R}$ is one-by-many correspondence.*

Algorithm 1: Translation from k-bounded TAPN to TAPN of degree 2

Input: A k-bounded TAPN $N = (P, T, IA, OA, c, Type, Inv)$ with marking M_0.
Output: A TAPN $N' = (P', T', IA', OA', c', Type', Inv')$ of degree 2 and marking M'_0.
begin

$$P' := P \quad \cup \{p_{stable}, p_{capacity}\} \cup \{p_h(t^i) \mid t \in T,\ 1 \le i < \max(t)\} \\ \cup \{p(t^i_{in}), p(t^i_{out}) \mid t \in T,\ 1 \le i < \max(t)\}$$

$$T' := \{t^i_{in}, t^i_{out} \mid t \in T,\ 1 \le i < \max(t)\} \cup \{t^{\max(t)} \mid t \in T\}$$

$$Inv'(p) := \begin{cases} Inv(p) & \text{if } p \in P \\ [0,0] & \text{if } p \in \{p(t^i_{in}), p(t^i_{out}) \mid t \in T,\ 1 \le i < \max(t)\} \\ [0,\infty) & \text{otherwise} \end{cases}$$

forall the $t \in T$ **do**

- $i := 1$
- **forall the** $(p, I, p', type) \in Pairing(t)$ **do**
  - **if** $i < |Pairing(t)|$ **then**
    - Add arcs $p \xrightarrow{I} t^i_{in} \longrightarrow p_h(t^i)$ and $p_h(t^i) \xrightarrow{[0,\infty)} t^i_{out} \longrightarrow p'$ of type *type*.
  - **else**
    - Add arcs $p \xrightarrow{I} t^i \longrightarrow p'$ of type *type*.
  - $i := i + 1$
- Add normal arcs $p_{stable} \xrightarrow{[0,\infty)} t^1_{in} \longrightarrow p(t^1_{in})$ and $p(t^1_{out}) \xrightarrow{[0,\infty)} t^1_{out} \longrightarrow p_{stable}$.
- Add normal arcs $p(t^i_{in}) \xrightarrow{[0,\infty)} t^{i+1}_{in} \longrightarrow p(t^{i+1}_{in})$ for $1 \le i < \max(t) - 1$.
- Add normal arcs $p(t^{\max(t)-1}_{in}) \xrightarrow{[0,\infty)} t^{\max(t)} \longrightarrow p(t^{\max(t)-1}_{out})$.
- Add normal arcs $p(t^{i+1}_{out}) \xrightarrow{[0,\infty)} t^{i+1}_{out} \longrightarrow p(t^i_{out})$ for $1 \le i < \max(t) - 1$.

$$M'_0(p) = \begin{cases} M_0(p) & \text{if } p \in P \\ \{0\} & \text{if } p = p_{stable} \\ \{\underbrace{0, \ldots, 0}_{k - |M_0|}\} & \text{if } p = p_{capacity} \\ \emptyset & \text{otherwise} \end{cases}$$

PROOF. We will first notice that the transition system generated by (N', M'_0) is a *delay-implies-stable* and *delay-preserves-stable* TTS. The first property follows from the construction by noticing that invariants on the intermediate places prevent time elapsing whenever the place p_{stable} is empty. The second property is immediate as time delays do not change the distribution of tokens. We shall now argue that the relation $\mathcal{R}$ satisfies the remaining six conditions of Definition 8. Let $(M, M') \in \mathcal{R}$.

1. $M' \models (p_{stable} = 1)$ follows from the definition of $\mathcal{R}$.
2. $M \models \wp$ iff $M' \models tr_p(\wp)$ again follows directly from the definition of $\mathcal{R}$.
3. Assume that $M \xrightarrow{t} M_1$ by firing some transition t. As $(M, M') \in \mathcal{R}$, the tokens consumed when t is fired in the marking M are present also in M' in identically named places and with the same age. From the construction of the net N' we can see that firing the series of transitions $t^1_{in}, t^2_{in}, \ldots, t^{\max(t)-1}_{in}, t^{\max(t)}, t^{\max(t)-1}_{out}, t^{\max(t)-2}_{out}, \ldots, t^1_{out}$ will bring

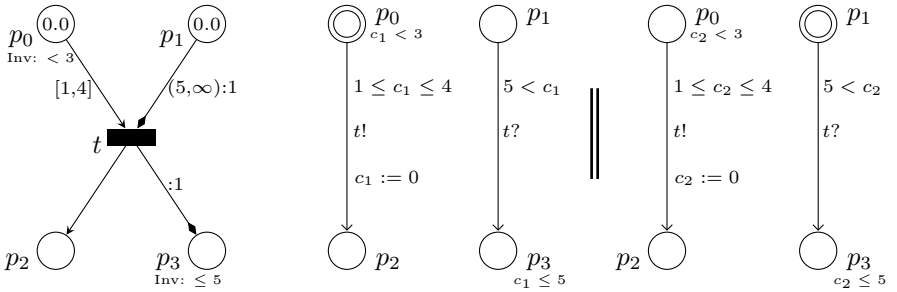

Figure 7: Example of the translation from TAPN to NTA

the net N' into a stable marking M_1' such that $(M_1, M_1') \in \mathcal{R}$.

4. Assume that $M \xrightarrow{d} M_1$. As M' is a stable marking, there are no tokens in any of the newly added places except for p_{stable}. Because p_{stable} has the invariant $[0, \infty)$, all time delays allowed in M are allowed also in M' and by $M' \xrightarrow{d} M_1'$ we clearly get $(M_1, M_1') \in \mathcal{R}$.
5. Assume that $M' \rightsquigarrow M_1'$. By the construction of N' we observe that the actual firing sequence must have the form: $t_{in}^1, t_{in}^2, \ldots, t_{in}^{\max(t)-1}, t^{\max(t)}$, $t_{out}^{\max(t)-1}, t_{out}^{\max(t)-2}, \ldots, t_{out}^1$ for some t. Since every transition in this sequence can be fired, we get that $M \xrightarrow{t} M_1$ such that $(M_1, M_1') \in \mathcal{R}$ by selecting the same ages of tokens in places from ${}^\bullet t$ as those consumed in the firing of the transitions $t_{in}^1, t_{in}^2, \ldots, t_{in}^{\max(t)-1}, t^{\max(t)}$.
6. Assume that $M' \xrightarrow{d} M_1'$. By the construction the same time delay is possible also from M such that $M \xrightarrow{d} M_1$ (all invariants in N are present also in N') and clearly $(M_1, M_1') \in \mathcal{R}$. □

5.2. From TAPN of Degree 2 to Networks of Timed Automata

We can now assume a given net of degree 2 (with no inhibitor arcs) and we will continue with a construction of a network of timed automata. The idea of the translation is to represent each token in the net by a single timed automaton with one local clock, and to simulate a transition firing by a handshake synchronization on a channel named after the transition.

The intuition is described on an example in Figure 7. We can see that every place in the net gives rise to an identically named location in the parallel component corresponding to a given token, while all invariants are carried over. Time intervals on arcs are naturally transformed into guards and the local clocks of each parallel component are reset if and only if the transitions correspond to

Algorithm 2: Algorithm for translation of TAPN of degree 2 to NTA

Input: A TAPN $N = (P, T, IA, OA, c, Type, Inv)$ of degree 2 and a marking M_0.
Output: An NTA $P_{TA} = A_1 \| A_2 \| \dots \| A_{|M_0|}$ where $A_i = (L, \mathcal{A}ct, C, \emptyset, \longrightarrow_i, I^i_C, I^i_X, \ell^i_0)$.
begin
  $L := P$; $\mathcal{A}ct := \{t!,\ t? \mid t \in T\}$; $C := \{c_1, c_2, \dots, c_{|M_0|}\}$
  forall the $t \in T$ **do**
    Let $\{(p_1, I_1, p'_1, type_1), (p_2, I_2, p'_2, type_2)\} := Pairing(t)$.
    for $i := 1$ *to* $|M_0|$ **do**
      Add $p_1 \xrightarrow{c_i \in I_1, t!, R}_i p'_1$ such that $R = \{c_i\}$ if $type_1 = normal$ else $R = \emptyset$.
      Add $p_2 \xrightarrow{c_i \in I_2, t?, R}_i p'_2$ such that $R = \{c_i\}$ if $type_2 = normal$ else $R = \emptyset$.
  $i := 1$; **forall the** $p \in P$, **forall** $Token \in M_0(p)$ **do** $\ell^i_0 := p$; $i := i + 1$;
  for $i := 1$ *to* $|M_0|$ **do**
    forall the $p \in P$ **do**
      $I^i_C(p) := (c_i \in Inv(p))$
      $I^i_X(p) := true$

normal arcs. In fact, the timed automata for all tokens in the net are identical, except for their initial locations that are determined by the placement of tokens in the initial marking and the names of their local clocks. For clarity reasons, we draw in the illustrations also clearly unreachable parts of the timed automata; in the actual implementation any unreachable states can be omitted.

In the general construction, as presented in Algorithm 2, we create for a k-bounded TAPN of degree 2 a network of k parallel components, each of them of a linear size w.r.t. the size of the input net (with the number of locations corresponding to the number of places and with twice as many edges as the Petri net transitions).

We are not using any integer variables in this translation, so the variable guards and assignments are left out; we also use expressions like $c \in I$ for lock guards/invariants on the transitions and implicitly assume that they are converted to the correct timed automata syntax (a straightforward conjunction of two guard constraints).

For the proof of correctness, we will again apply the methodology from Section 2.4. In this case every configuration of the NTA is stable, thus we set the proposition *stable* everywhere to true. The proposition translation function tr_p will translate atomic Petri net propositions of the form $(p \bowtie n)$ into NTA propositions $(\#p \bowtie n)$, meaning that the number of components in the location p determines the number of tokens in the place p.

Let M be a marking of a k-bounded TAPN of degree 2 and let $s = (\ell_1, \cdots, \ell_k, v)$ be a configuration of the constructed NTA (variable valuation is omitted as it is not used). Then we define $(M, s) \in \mathcal{R}$ if and only if we can enumerate the elements of M as $\{(p_1, r_1), (p_2, r_2), \cdots, (p_k, r_k)\}$ such that $p_i = \ell_i$ and $r_i = v(c_i)$ for all i, $1 \leq i \leq k$.

Lemma 25. *The relation $\mathcal{R}$ is a complete one-by-many correspondence.*

PROOF. We need to prove that $\mathcal{R}$ satisfies all requirements of Definition 8. As every configuration of the constructed NTA satisfies the proposition *stable*, we immediately get that the generated transition system is a *delay-implies-stable*, *delay-preserves-stable* and *eventually-stable* TTS. Let $M \in \mathcal{M}(N, M_0)$ and let s be a reachable configuration of P_{TA} constructed by the algorithm such that $(M, s) \in \mathcal{R}$. We shall prove that $\mathcal{R}$ satisfies Conditions 1-6 of Definition 8.

1.,2. These two conditions are immediate from the definition of $\mathcal{R}$ and the proposition translation function.

3. Assume that $M \xrightarrow{t} M'$. Because t has exactly two input places, say p_0 and p_1, that contain tokens with ages satisfying the invariants on arcs and $(M, s) \in \mathcal{R}$, we can find two parallel components A_j and A_k in P_{TA} such that their current locations in s are p_0 and p_1, and their local clocks c_j and c_k correspond to the ages of the two tokens in p_0 and p_1 consumed when firing t. Because these components share the same time constraints as the time intervals on the arcs in the Petri net, they can synchronize on the channel t and produce a state s' where clearly $(M', s') \in \mathcal{R}$.

4.,6. By construction of the timed automata network where all invariants are simply overtaken, any time delay allowed in M are also allowed in s and vice versa.

5. Let $s \xrightarrow{t} s'$ via a synchronization of two parallel components A_j and A_k on a channel t. Due to the construction we can see that this can happen only if the transition t can be fired in M. Now let $M \xrightarrow{t} M'$ such that the firing of t involves tokens in ${}^\bullet t$ that correspond to the parallel components A_j and A_k. Clearly, $(M', s') \in \mathcal{R}$. □

5.2.1. Summary

We have now proved that the first step of the translation preserves the safety fragment of TCTL and the second preserves the full TCTL. This means that the combined translation from k-bounded TAPN into NTA (via Algorithm 1 followed by Algorithm 2) will preserve the safety fragment of TCTL.

Theorem 26. *Any k-bounded TAPN without inhibitor arcs can be translated into a network of timed automata while preserving the safety fragment of TCTL.*

Clearly, if the input k-bounded net has at most two input and at most two output arcs for each transition, it can be easily turned into a degree 2 net by adding the $p_{capacity}$ place together with the missing input or output arcs. We can now apply directly the second translation, hence preserving also the liveness properties. Moreover, in the actual implementation in the tool TAPAAL, we exploit the possibility of a direct handshake synchronization for transitions of degree 2 presented in Figure 7 and use the full construction depicted in Figure 6 only for transitions with more than two input or output arcs.

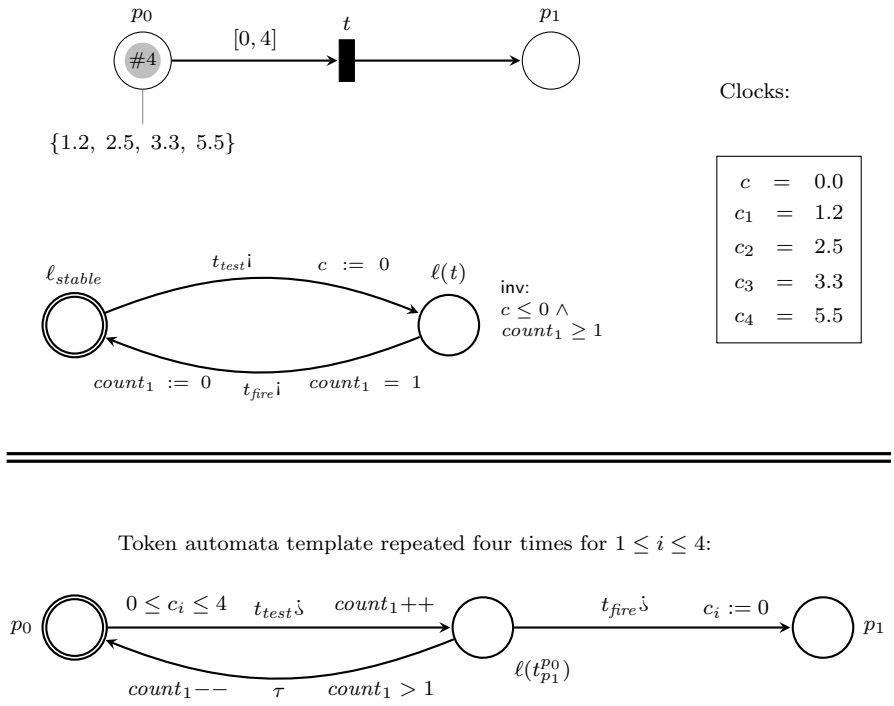

Figure 8: A simple TAPN and the constructed NTA

6. Liveness-Preserving Translation

We will now present another translation from k-bounded TAPN to NTA, this time considering also inhibitor arcs. As in the previous translation, there will be one TA with a local clock for each token in the net. Because there may not always be exactly k tokens present during the execution of the net, we add a new location $\ell_{capacity}$ in order to represent currently unused tokens. In addition to the automata modelling tokens, we create a single control automaton. Its purpose is to simulate the firing of transitions and to move tokens around via broadcast communication. The control automaton has a location ℓ_{stable} and it moves out of this location once the simulation of a transition begins and returns back once the simulation of the transition ends. Moreover, each time the automaton is in ℓ_{stable}, the token automata in the composed NTA correspond to a marking in the TAPN. The whole point of this construction is to ensure that we do not introduce any new deadlocks as it was the case in the previous translation. We will first show how the translation works on two examples.

Example 27. Figure 8 shows a simple TAPN with a single transition, no inhibitor arcs, and four tokens of different ages. The translated NTA consists of five automata, one control automaton (topmost automaton) and four token automata, one for each token. Notice that in this example we have refrained from drawing the $p_{capacity}$ location as it is not used.

The translated NTA first tests using the broadcast communication whether a transition can be fired at all and then executes the firing. Concretely, the control automaton first broadcasts on the channel t_{test}. Any token automaton with its clock in the interval $[0, 4]$ is forced to participate in the broadcast; in our case three token automata will participate. We use integer variables to count the number of token automata that took part in the broadcast. Because the preset of t has size one, we only need one counter variable $count_1$. Once the token automata synchronized in the broadcast, they move to the intermediate locations $\ell(t^{p_0}_{p_1})$ and during the update each increments $count_1$ by one[2]; in our case the value of $count_1$ will become three. This means that the invariant on $\ell(t)$ in the control automaton is satisfied. In other words, we know that there are enough tokens with appropriate ages in the input places for t to fire. Notice that if there were not enough tokens in some of the input places, then the invariant on $\ell(t)$ was not satisfied and the broadcast could not take place at all. This is one of the crucial aspects to realize in order to see why this translation preserves liveness properties.

Now the value of $count_1$ is three and the control automaton may not broadcast on the t_{fire} channel yet since the guard ensures that this is only possible when exactly one token automaton remains in its intermediate place. Therefore, we are forced to move two of the token automata back to p_0 via the τ-transitions. This is possible only as long as $count_1$ is strictly greater than one. Hence exactly one token automaton has to remain in its intermediate place before the control automaton can broadcast on the t_{fire} channel and finalize the simulation of firing t. Note that due to the invariant $c \leq 0$ in the control automaton, no time delay is possible during the simulation of the transition.

After demonstrating the basic idea of the broadcast translation, let us discuss a slightly more elaborate example using all of the features of the TAPN model.

Example 28. Consider the TAPN model in Figure 9 that uses transport arcs (the pair of arcs with diamond tips from p_1 to p_4) for moving tokens while preserving their ages, an inhibitor arc (the arc with the circle tip) and an invariant in place p_4. The NTA created by our algorithm is below the net. As before, the template is repeated three times, once for each token, the only difference being the initial location (p_1, p_2 and p_3, respectively) and the name of the clock (c_1, c_2 and c_3, respectively).

We see that the control automaton has a test-fire loop for every transition in the TAPN model. There are some special constructions worth mentioning. First of all, consider the inhibitor arc from p_3 to t. This arc is encoded using a self-loop participating in the t_{test} broadcast transition. We use a counter variable to count the number of automata that take this edge. We simply encode the requirement that there is no token in the interval $[0, 2]$ by adding the invariant $count_3 = 0$ on the location $\ell(t)$.

[2]The updates for each token automaton are executed in a fixed order but the ordering is not important as the resulting value will only depend on the number of participating automata.

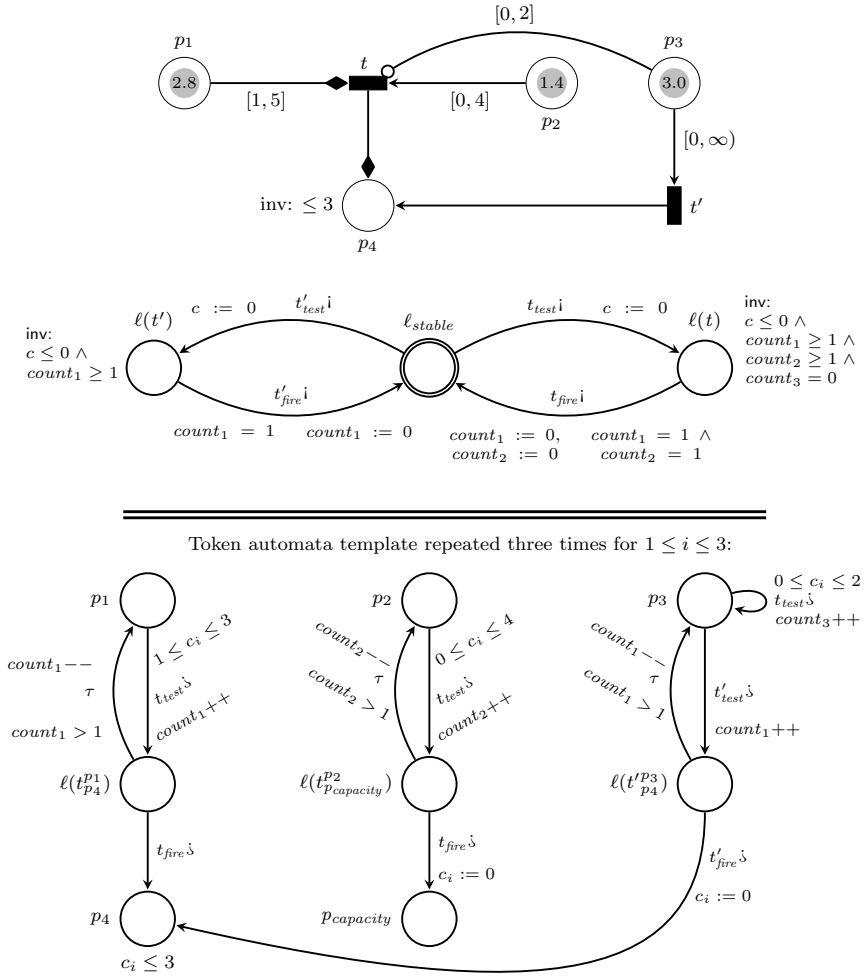

Figure 9: A TAPN and the constructed NTA

The second observation is related to the guard on the edge from p_1 to $\ell(t^{p_1}_{p_4})$. It is evident that this does not match the interval $[1, 5]$ located on the arc from p_1 to t in the TAPN model. The guard $1 \leq c_i \leq 3$ is in fact the intersection of the interval $[1, 5]$ and the invariant ≤ 3 representing the interval $[0, 3]$ on the place p_4. This is because the age of the token consumed in p_1 will be preserved once moved to p_4 and by intersecting the intervals we avoid possible deadlocks. One may think that it is enough to add the invariant $[0, 3]$ on the intermediate place, however, this may result in incorrect behavior. If there were two tokens in p_1 with ages 4 and 2, the broadcast on t_{test} would be blocked. This is because invariants block the entire broadcast transition even if only a single automaton with a satisfied guard cannot participate due to the violation of the invariant in its target location.

For our specific example, we need at least one token of age $[1,3]$ in p_1, at least one token of age $[0,4]$ in p_2 and zero tokens of age $[0,2]$ in p_3 in order for t to be enabled, which is precisely what is encoded in the invariant on $\ell(t)$. The reader may also notice that different transitions share counter variables. The variable $count_1$ is used in the simulation of both t and t' but they are used in a non-conflicting way, in the sense that we are never simulating t and t' at the same time. We also see that during the simulation of t' we do not take the invariant of the target location into account since the arc from t' to p_4 is a normal arc and produces a token of age zero which always satisfies any invariant.

The complete translation is given in Algorithm 3 where the function $Pairing$ is the one used also in the previous translation and where $NumVars(N) \stackrel{\text{def}}{=} \max_{t \in T}(|Pairing(t)| + |\{(p,t) \mid (p,t) \in IA, Type((p,t)) = Inhib\}|)$ denotes the maximum number of integer variables needed in the translation since we need a counter for each pair in $Pairing(t)$ plus a counter for each inhibitor place of t. As explained, these counters can be reused for different transitions. As before, we shall also use the expressions like $c \in I$ in guards and invariants of the timed automata, meaning the obvious conjunction of two constraints on the clock c.

Observe that the algorithm clearly terminates as the only while-loop always removes one element from $Pairing(t)$ and finishes once the set is empty. The algorithm creates k parallel timed automata for each token and one extra control component. The token automata contain at most $|P| + 1 + |T| \cdot D$ locations and $3 \cdot |T| \cdot D$ edges where D is the maximum degree of any transition. The control automaton contains $|T| + 1$ locations and $2|T|$ edges.

To prove the correctness of the translation, we will follow the methodology described in Section 2.4. We let (N, M_0) be a marked k-bounded TAPN and let P_{TA} be the NTA constructed by Algorithm 3 with the initial configuration s_0. We define the *stable* proposition as $(\#\ell_{stable} = 1)$. Observe that the proposition $(\#\ell_{stable} = 1)$ is true whenever the control TA is in the ℓ_{stable} location. The proposition translation function tr_p translates a TAPN proposition $(p \bowtie n)$ into $(\#p \bowtie n)$, hence the number of tokens in p corresponds to the number of token automata that are currently in their location p.

Now we define a correspondence relation $\mathcal{R}$ between markings of N and configurations of the constructed P_{TA}. Let $M = \{(p_1, r_1), (p_2, r_2), \ldots, (p_n, r_n)\}$ be a reachable marking of N and let $s = (\ell, \ell_1, \ell_2, \ldots, \ell_k, z, v)$ be a reachable configuration of P_{TA}. As (N, M_0) is k-bounded, clearly $n \leq k$. We define $(M, s) \in \mathcal{R}$ iff

- $\ell = \ell_{stable}$,
- $count_i = 0$ for all $1 \leq i \leq NumVars(N)$, and
- there is an injection $h : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, k\}$ such that
  - $\ell_{h(i)} = p_i$ and $v(c_{h(i)}) = r_i$ for all i where $1 \leq i \leq n$,
  - $\ell_j = p_{capacity}$ for all $j \in \{1, 2, \ldots, k\} \setminus range(h)$.

Algorithm 3: Translation from k-bounded TAPN to NTA.

Input: A k-bounded TAPN $N = (P, T, IA, OA, c, Type, Inv)$ with a marking M_0
Output: NTA $P_{TA} = A \| A_1 \| A_2 \| \dots \| A_k$ s.t. $A = (L, \mathcal{A}ct, C, X, \longrightarrow, I_C, I_X, \ell_0)$ and $A_i = (L_i, \mathcal{A}ct, C, X, \longrightarrow_i, I^i_C, I^i_X, \ell^i_0)$

begin

- **for** $i := 1$ *to* k **do** $L_i := P \cup \{p_{capacity}\}$
- $L := \{\ell_{stable}\}$; $\mathcal{A}ct := \{t_{test}?, t_{test}!, t_{fire}?, t_{fire}! \mid t \in T\} \cup \{\tau\}$
- $C := \{c, c_1, c_2, \dots, c_k\}$; $X := \{count_i \mid 1 \le i \le NumVars(N)\}$
- **forall the** $t \in T$ **do**
  - $j := 0$; $varInv_t := true$; $varGuard_t := true$
  - **forall the** $(p, I, p', type) \in Pairing(t)$ **do**
    - $j := j + 1$;
    - **for** $i := 1$ *to* k **do**
      - $L_i := L_i \cup \{\ell(t^p_{p'})\}$
      - Add $p \xrightarrow{g, true, t_{test}!, \emptyset, count_j{+}{+}}_i \ell(t^p_{p'})$ s.t. $g := (c_i \in I)$ if $type = normal$ else $g := (c_i \in I \cap Inv(p'))$
      - Add $\ell(t^p_{p'}) \xrightarrow{true, true, t_{fire}!, R, \emptyset}_i p'$ s.t. $R := \{c_i\}$ if $type = normal$ else $R := \emptyset$
      - Add $\ell(t^p_{p'}) \xrightarrow{true, count_j > 1, \tau, \emptyset, count_j{-}{-}}_i p$
    - $varInv_t := varInv_t \wedge count_j \ge 1$; $varGuard_t := varGuard_t \wedge count_j = 1$
  - **forall the** $p \in P$ *where* $(p,t) \in IA$ *and* $Type((p,t)) = Inhib$ **do**
    - $j := j + 1$;
    - **for** $i := 1$ *to* k **do** Add $p \xrightarrow{c_i \in c((p,t)), true, t_{test}!, \emptyset, count_j{+}{+}}_i p$
    - $varInv_t := varInv_t \wedge count_j = 0$; $varGuard_t := varGuard_t \wedge count_j = 0$
  - $L := L \cup \{\ell(t)\}$; Add $\ell_{stable} \xrightarrow{true, true, t_{test}?, \{c\}, \emptyset} \ell(t)$ and $\ell(t) \xrightarrow{true, varGuard_t, t_{fire}?, \emptyset, \{count_i := 0 \mid 1 \le i \le j\}} \ell_{stable}$
- **for** $i := 1$ *to* k **do**
  - $$I^i_C(p) := \begin{cases} c_i \le a & \text{if } p \in P \text{ and } Inv(p) = [0, a] \\ c_i < b & \text{if } p \in P \text{ and } Inv(p) = [0, b) \\ true & \text{if } p \in L_i \setminus P \end{cases} \qquad I^i_X(p) := true \text{ for } p \in L_i$$
- $$I_C(p) := \begin{cases} true & \text{if } p = \ell_{stable} \\ c \le 0 & \text{if } p \in L \setminus \{\ell_{stable}\} \end{cases}$$
- $$I_X(p) := \begin{cases} varInv_t & \text{if } p = \ell(t) \text{ for } t \in T \\ true & \text{if } L \setminus \{\ell(t) \mid t \in T\} \end{cases}$$
- $i := 1$; **forall the** $p \in P$ **do** **forall the** $Token \in M_0(p)$ **do** $\ell^i_0 := p; i := i + 1$
- **for** $i := |M_0| + 1$ *to* k **do** $\ell^i_0 := p_{capacity}$
- $\ell_0 := \ell_{stable}$

Intuitively, if $(M, s) \in \mathcal{R}$ then for every token in M there is a TA where its location and clock valuation matches the token data and vice versa.

We shall now prove that $\mathcal{R}$ is a complete one-by-many correspondence.

Lemma 29. *The relation $\mathcal{R}$ is a complete one-by-many correspondence.*

Proof. We will first show that P_{TA} possesses the three properties required by complete one-by-many correspondence. The transition system generated by P_{TA}

is clearly a *delay-implies-stable* TTS since all locations in the control automaton except for ℓ_{stable} have the invariant $c \leq 0$ and delay is hence possible only if the control automaton is in the location ℓ_{stable}. It is also easy to see that it is *delay-preserves-stable* TTS as time delays do not change the current locations of any automata, so if the control automaton is in the location ℓ_{stable} it will be there also after any time delay. Finally we should argue that the transition system of P_{TA} is also *eventually-stable* TTS. This follows from the construction. The control automaton moves out of the ℓ_{stable} location after broadcasting on some channel t_{test}. The automaton can broadcast only if for every input place of t there is at least one corresponding token automaton that will participate in the broadcast, otherwise the invariant in $\ell(t)$ would break. Now we are in an unstable situation and we can fire only the τ transitions returning some of the token automata into their initial locations, while decreasing the value of the corresponding variable *count*. This will happen only finitely many times as we eventually reach the situation when the guard in the control automaton broadcasting on the channel t_{fire} will be satisfied and the only remaining action is to return to a stable configuration.

We shall now prove that $\mathcal{R}$ satisfies Conditions 1-6 of Definition 8. Let $(M, s) \in \mathcal{R}$.

1.,2. These two conditions follow immediately from the definitions.

3. Assume that $M \xrightarrow{t} M'$. We must show that $s \rightsquigarrow s'$ such that $(M', s') \in \mathcal{R}$. We first notice that t_{test} is enabled in s. Since t_{test} is a broadcast channel, the first requirement is that the t_{test} sender has to be enabled. There are no guards in the control automaton, so only the invariant on $\ell(t)$ may block the broadcast. The second requirement is that every possible receiver (i.e. any receiver whose guard is satisfied) must participate in the broadcast synchronization. Due to the construction of P_{TA}, there are no invariants on the intermediate places $\ell(t^p_{p'})$. Thus, any receiver with a satisfied guard can participate in the broadcast synchronization. By the fact that $(M, s) \in \mathcal{R}$ it follows that enough TA will participate in the broadcast synchronization to satisfy the invariant on $\ell(t)$. By returning the token automata representing tokens that do not take part in firing t into their previous locations and issuing the broadcast on t_{fire} we get $s \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{t_{fire}} s_n = s'$ and hence $s \rightsquigarrow s'$ while clearly $(M', s') \in \mathcal{R}$.

4.,6. Time delays can only be restricted by invariants. By the fact that the invariant on each place p in N is carried over to the corresponding locations p in the network, we have that it is always possible to do the same time delays in M and s.

5. Assume that $s \rightsquigarrow s'$. By the construction of P_{TA}, we get that this sequence must be of the form (leaving out 0 delays) $s \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{t_{fire}} s_n$. Since $s \xrightarrow{t_{test}} s_1$ we know that the invariant on $\ell(t)$ is satisfied. By the construction, this in turn means that there are enough token automata that can synchronize on t_{test} in such a way that for each

input place $p \in {}^\bullet t$ there is at least one automaton whose current location is p and the clocks of these automata satisfy the guards on the t_{test} transitions. Further, for each place p' such that there is an inhibitor arc from p' to t, there is no automaton in location p' with a clock satisfying the respective guard. Finally, the token automata that moved to their intermediate locations and represent a token moving along a pair of transport arcs are guaranteed to satisfy the invariants in their target locations due to the fact that the guards were intersected with the invariants. This is exactly what is needed to fire t from M and we get $M \xrightarrow{t} M'$ such that $(M', s') \in \mathcal{R}$. □

6.1. Summary

We have now showed that any bounded TAPN with age invariants and transport/inhibitor arcs can be translated into a network of timed automata with (a very restricted use of) integer variables and with broadcast communication like used in the tool UPPAAL. As the original and the translated models are related using a complete one-by-many correspondence relation, we know that the whole logic TCTL is preserved.

Theorem 30. *Any k-bounded TAPN can be translated into a network of timed automata while preserving the full TCTL.*

As discussed in the previous translation, if a transition in a concrete k-bounded net has at most two input and two output arcs, the whole construction presented in this section can be replaced by a simple handshake synchronization (still preserving liveness properties). Such an optimization has been implemented in the tool TAPAAL.

7. Experiments

During the development of the tool TAPAAL we have tested several variants of possible translations and concluded that the two presented in this article are the most efficient ones. The translations are available in the current distribution of the tool and we shall document their efficiency by presenting a selection of experimental results and comparing the performance of our translations with native UPPAAL models. We present six selected experiments; all models are available from the download section at `www.tapaal.net`.

- *Alternating Bit Protocol* [34, 7] is a network protocol for communication between a sender and a receiver over a lossy communication channel. We verify a safety property that guarantees that the sender and the receiver never get out of synchrony; the property is satisfied.
- *Fischer's Protocol* [31] is a time based mutual exclusion protocol and it is a standard example for testing the performance of tools. The TAPN model was presented in [4] and it is available as a TAPAAL example net;

timed automata model is a part of the standard Uppaal distribution. We verify a safety property ensuring that at most one process can be in the critical section at any time; the property is satisfied.

- *Lynch and Shavit Protocol* [33] is another, more complex, timed-based mutual exclusion algorithm. Both TAPAAL and Uppaal models are taken from [3]. We verify a safety property as in the previous case study; it is also satisfied.

- *MPEG-2 Video Encoder* [1] is a model of the MPEG-2 encoding algorithm for analyzing its performance on a group of frames on a multiprocessor architecture; we scale the model by varying the number of B frames. The TAPN model was taken directly from [38]. We recreated the Uppaal model from the descriptions in [20] since their original Uppaal model was not available anymore. We verify a reachability query asking whether all frames can be produced within a given deadline and the deadline is set so that the query is unsatisfied and the whole state-space is searched.

- *Engine Workshop* is a time based model of a small workshop with several workers that maintain a number of running engines with different maintenance deadlines and requirements. We verify a liveness property asking, for a given number of engines, whether 15 workers can keep them running indefinitely; the query is satisfied and the reported times correspond to the effort needed to find a feasible maintenance schedule.

- *Medical Workflow* is a simplified model of a workflow with time requirements, inspired by [22], with three types of roles: patient, nurse and doctor. The case study is scaled by the number of patients. We verify a liveness query asking, for a given number of patients arriving in regular intervals, if in any execution scenario all patients are checked out before missing the deadlines. The query is satisfied and the whole state-space is searched.

The experiments where performed on a quad-core Intel Core I7 620 2.67 GHz with 4GB of RAM running Ubuntu 11.04, using the 64-bit Uppaal 4.1.7 verification engine and TAPAAL 2.0.0. Symmetry reduction was enabled in the Uppaal engine for all safety queries and disabled for the liveness ones as the Uppaal engine does not support trace generation in this case. We used standard engine settings with the BFS search strategy. Running times are given in seconds, experiments that did not finish within 5 minutes (300 seconds) are marked with ◷ and experiments that run out of memory are marked with OOM. For the translations, the reported time includes both the translation time to NTA (often negligible) plus the actual verification.

Table 1 provides a summary of the experiments where we tested safety properties. For the first three experiments, both the safety and liveness translations show similar execution times. The reason is that in these models most of the transitions have no more than two input and two output arcs and the handshake optimization technique creates a similarly performing Uppaal models.

	Messages	Safety Translation	Liveness Translation	UPPAAL
Alternating Bit	10	1.18	1.23	0.85
	11	1.98	1.90	2.00
	12	4.84	4.56	3.85
	13	7.94	7.67	7.95
	14	23.61	18.28	19.02
	15	61.61	48.13	51.44
	16	214.7	165.91	145.92
	17	◷	◷	◷
	Processes	**Safety Translation**	**Liveness Translation**	**UPPAAL**
Fischer	40	0.89	0.88	1.97
	50	2.14	2.16	21.20
	70	8.92	8.81	30.25
	90	27.25	26.51	42.10
	110	67.03	65.01	244.51
	130	143.65	142.88	◷
	150	282.27	277.71	◷
	Processes	**Safety Translation**	**Liveness Translation**	**UPPAAL**
Lynch-Shavit	20	0.21	0.19	0.95
	30	0.69	0.67	8.69
	40	2.25	2.00	49.95
	50	5.15	4.97	179.43
	60	10.91	10.81	◷
	80	39.64	39.23	◷
	100	105.76	105.74	◷
	120	249.44	253.534	◷
	B-frames	**Safety Translation**	**Liveness Translation**	**UPPAAL**
MPEG-2 Encoder	1	≤ 0.1	0.11	≤ 0.1
	2	≤ 0.1	9.37	≤ 0.1
	3	0.12	◷	≤ 0.1
	4	0.35	OOM	≤ 0.1
	5	1.53	OOM	1.16
	6	8.96	OOM	1.60
	7	74.26	OOM	41.62
	8	◷	OOM	◷

Table 1: Verification results for safety properties

It is also noticeable that in case of Fischer's and Lynch-Shavit's protocols, the translated timed automata models verify considerably faster than the native UPPAAL models do. The reason seems to be that even though these translated models are more complex than the native UPPAAL ones and the number of symbolic states is larger, the DBM operations use less CPU time. This performance seems to be most visible in case of a large number of similarly behaving processes.

In case of the MPEG-2 encoder where there are transitions with a large indegree, we can see a weakness of the liveness translation as during the verification the number of explored token combinations in which a transition can be fired explodes, causing the UPPAAL verification engine quickly run out of memory.

On the other hand, the liveness reduction can handle also inhibitor arcs and verify liveness properties, as documented on the engine maintenance workshop

	Engines	Safety Translation	Liveness Translation	UPPAAL
Engine Workshop	55	-	3.38	3.04
	60	-	4.49	4.08
	65	-	7.42	4.22
	70	-	7.56	5.04
	75	-	7.99	6.12
	80	-	12.68	6.06
	85	-	13.23	7.34
	90	-	12.59	8.49
	91	-	◷	◷
	Patients	Safety Translation	Liveness Translation	UPPAAL
Medical Workflow	9	-	1.15	0.32
	10	-	2.16	0.78
	11	-	6.16	1.86
	12	-	14.19	4.44
	13	–	32.85	10.28
	14	-	75.32	23.64
	15	-	169.953	54.16
	16	-	◷	122.84
	17	-	◷	◷

Table 2: Verification results for liveness properties

and the medical workflow case studies in Table 2. The performance is slower, though comparable with that of the UPPAAL engine verifying native timed automata models.

8. Conclusion

We have presented a general framework for translating one time-dependent model into another one while preserving the satisfaction of TCTL properties. We further demonstrated the usefulness of the framework on two concrete translations from bounded timed-arc Petri nets to networks of timed automata, giving us more direct and simpler ways to argue about their correctness. Even though similar ideas were used in other translations, though without an explicit generalization of the techniques, we see as the strong point of our approach the fact that we handle the full generality of timed transitions systems that are generated by many time-dependent models used in nowadays tools like UPPAAL. Hence our framework is close to the tool developers and can facilitate more direct and simpler arguments about the correctness of the approach by simply defining a one-by-many correspondence and proving a few properties that are required. All the tedious technical details about the preservation of TCTL formulae are now hidden in the proof of the framework correctness and do not have to be repeated for each single instance of our approach. Moreover, the technique is independent of the actual underlying formal model and many translations between other time-dependent models, including e.g. Time Petri Nets of Merlin and Faber [35, 36], share the same characteristics and for several of them our technique is directly applicable, including [21, 14, 24, 30, 46].

As another contribution of this work, we now have a complete and unified proof of the translations implemented in the tool TAPAAL, one of them preserving the whole TCTL for the fully featured TAPAAL models and one preserving only the safety fragment for models without inhibitor arcs, though in some cases providing a significantly better performance than the liveness preserving translation. It is remarkable that even though the translated models are verified using the UPPAAL engine and they are significantly larger than the native UPPAAL models, in several cases we noticed a speedup in the verification times. This, perhaps a surprising, fact should be better understood and exploited as a possible pre-processing phase in verification of timed automata models before they are passed to the verification engine.

Acknowledgments. We would like to thank all current and past TAPAAL developers and many of our colleagues for their useful feedback. We also thank the anonymous reviewers for their detailed comments and suggestions.

References

[1] ISO/IEC 13818-1:2000(E). Information technology—generic coding of moving pictures and associated audio information: Systems, 2000.

[2] P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Forward reachability analysis of timed Petri nets. In *Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, volume 3253 of *LNCS*, pages 343–362. Springer-Verlag, 2004.

[3] P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Using forward reachability analysis for verification of timed Petri nets. *Nordic J. of Computing*, 14:1–42, 2007.

[4] P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *LNCS*, pages 53–70. Springer-Verlag, 2001.

[5] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proceedings of the 17th International Colloquium on Algorithms, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.

[6] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[7] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM*, 12:260–261, May 1969.

[8] G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, number 3185 in LNCS, pages 200–236. Springer-Verlag, 2004.

[9] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O.H. Roux. Comparison of the expressiveness of timed automata and time Petri nets. In *Proc. of FORMATS'05*, volume 3829 of *LNCS*, pages 211–225. Springer, 2005.

[10] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *Proc. of FORMATS'06*, volume 4202 of *LNCS*, pages 82–97. Springer, 2006.

[11] B. Berthomieu, P-O. Ribet, and F. Vernadat. The tool TINA — construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[12] T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification (Ottawa 1990)*, pages 1–14. North-Holland, Amsterdam, 1990.

[13] H. Boucheneb, G. Gardey, and O.H. Roux. TCTL model checking of time Petri nets. *Journal of Logic and Computation*, 19(6):1509–1540, 2009.

[14] P. Bouyer, S. Haddad, and P.A. Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. *Information and Computation*, 206(1):73–107, 2008.

[15] F.D.J. Bowden. Modelling time in Petri nets. In *Proceedings of the Second Australia-Japan Workshop on Stochastic Models*, 1996.

[16] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 546–550. Springer-Verlag, 1998.

[17] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 237–267. Springer-Verlag, 2004.

[18] J. Byg, K.Y. Jørgensen, and J. Srba. An efficient translation of timed-arc Petri nets to networks of timed automata. In *Proc. of ICFEM'09*, volume 5885 of *LNCS*, pages 698–716. Springer, 2009.

[19] J. Byg, K.Y. Jørgensen, and J. Srba. TAPAAL: Editor, simulator and verifier of timed-arc Petri nets. In *Proc. of ATVA'09*, volume 5799 of *LNCS*, pages 84–89. Springer, 2009.

[20] M.E. Cambronero, A.P. Ravn, and V. Valero. Using UPPAAL to analyze an MPEG-2 algorithm. In *Proc. of VII Workshop Brasileiro de Tempo Real*, pages 73–82, 2005.

[21] F. Cassez and O.H. Roux. Structural translation from time Petri nets to timed automata. *Journal of Systems and Software*, 79(10):1456–1468, 2006.

[22] S.C. Christov, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, and E.A. Henneman. A benchmark for evaluating the applicability of software engineering techniques to the improvement of medical processes. In *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care (SEHC'10)*, pages 50–56. ACM Press, 2010.

[23] A. David, L. Jacobsen, M. Jacobsen, K.Y. Jørgensen, M.H. Møller, and J. Srba. TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 492–497. Springer-Verlag, 2012.

[24] J.S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi. Timed Automata Patterns. *IEEE Transactions on Software Engingeering*, 34(6):844–859, 2008.

[25] G. Gardey, D. Lime, M. Magnin, and O.H. Roux. Romeo: A tool for analyzing time Petri nets. In *Proc. of CAV'05*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.

[26] H.M. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets (ICATPN'93)*, volume 691 of *LNCS*, pages 282–299, 1993.

[27] F. Heitmann, D. Moldt, K.H. Mortensen, and H. Rölke. Petri nets tools database quick overview (from Petri net world). `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/`, Accessed: 29.4.2013.

[28] L. Jacobsen, M. Jacobsen, M.H. Møller, and J. Srba. A framework for relating timed transition systems and preserving TCTL model checking. In *Proceedings of the 7th European Performance Engineering Workshop (EPEW'10)*, volume 6342 of *LNCS*, pages 83–98. Springer-Verlag, 2010.

[29] L. Jacobsen, M. Jacobsen, M.H. Møller, and J. Srba. Verification of timed-arc Petri nets. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11)*, volume 6543 of *LNCS*, pages 46–72. Springer-Verlag, 2011.

[30] A. Janowska, P. Janowski, and D. Wróblewski. Translation of Intermediate Language to Timed Automata with Discrete Data. *Fundamenta Informaticae*, 85(1-4):235–248, 2008.

[31] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[32] F. Laroussinie and K.G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII)*, pages 439–456. Kluwer, B.V., 1998.

[33] N. Lynch and N. Shavit. Timing-based mutual exclusion. In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 2–11, 1992.

[34] W. C. Lynch. Computer systems: Reliable full-duplex file transmission over half-duplex telephone line. *Commun. ACM*, 11:407–410, June 1968.

[35] P.M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, CA, USA, 1974.

[36] P.M. Merlin and D.J. Faber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.

[37] K.S. Namjoshi. A simple characterization of stuttering bisimulation. In S. Ramesh and G Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *LNCS*, pages 284–296. Springer-Verlag, 1997.

[38] F.L. Pelayo, F. Cuartero, V. Valero, H. Macia, and M.L. Pelayo. Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. In *Proceedings of the 10th International Multimedia Modelling Conference (MMM'04)*, pages 49–56. IEEE Computer Society, 2004.

[39] F.L. Pelayo, F. Cuartero, V. Valero, M.L. Pelayo, and M.G. Merayo. How does the memory work? By timed-arc Petri nets. In *Proceedings of the 4th IEEE International Conference on Cognitive Informatics (ICCI'05)*, pages 128–135, 2005.

[40] W. Penczek and A. Pólrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*. Springer-Verlag, 2006.

[41] W. Penczek and A. Pólrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer, 2006.

[42] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Darmstadt, 1962.

[43] C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1973.

[44] J. Sifakis. Use of Petri nets for performance evaluation. In *Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, Modelling and Evaluating Computer Systems (Bonn-Bad Godesberg, 1977)*, pages 75–93. Elsevier Science Publishers, Amsterdam, 1977.

[45] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proc. of STACS'96*, volume 1046 of *LNCS*, pages 347–359. Springer, 1996.

[46] J. Srba. Timed-arc Petri nets vs. networks of timed automata. In *Proceedings of the 26th International Conference on Application and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *LNCS*, pages 385–402. Springer-Verlag, 2005.

[47] J. Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In *Proc. of FORMATS'08*, volume 5215 of *LNCS*, pages 15–32. Springer, 2008.

[48] V. Valero, F. Cuartero, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM'99)*, pages 188–196, 1999.

[49] V. Valero, J.J. Pardo, and F. Cuartero. Translating TPAL specifications into timed-arc Petri nets. In *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets (ICATPN'02)*, volume 2360 of *LNCS*, pages 414–433. Springer-Verlag, 2002.

[50] V. Valero, F.L. Pelayo, F. Cuartero, and D. Cazorla. Specification and analysis of the MPEG-2 video encoder with timed-arc Petri nets. *Electronic Notes Theoretial Computer Science*, 66(2), 2002.

[51] J. Wang. *Timed Petri Nets, Theory and Application*. Kluwer Academic Publishers, 1998.