Master Thesis

Modeling and Verification of Extended Timed-Arc Petri Nets

Lasse Jacobsen, Morten Jacobsen & Mikael Harkjær Møller
{*lassejac, mortenja, mikaelhm*}*@cs.aau.dk*
Department of Computer Science, Aalborg University, group d621a

June, 2010

Preface

This report documents a DAT6-project in the Department of Computer Science at Aalborg University. The project was carried out in the spring of 2010 within the Distributed and Embedded Systems research unit.

The report serves as a master thesis in computer science for Lasse Jacobsen, Morten Jacobsen and Mikael H. Møller and is a continuation of a DAT5-project, documented in a report entitled *Extending Timed-Arc Petri Nets* [27].

Chapter 1 (*Introduction*), Chapter 2 (*Preliminaries*), Chapter 3 (*Extended Timed-Arc Petri Nets*) and Section 4.1 (*Problems of Interest*) have for the most part been overtaken from the DAT5 report and reused in this thesis. The verification times for UPPAAL in Section 12.2 have also been overtaken from previous experiments. Further, parts of Chapter 11 (*TAPAAL*) have to a lesser extent been overtaken from the DAT5 report. Finally, Appendix A (except for the last section) have been included from the DAT5 report for comparison.

Furthermore, a paper entitled *Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants* has been published (see [28]). The main parts of this paper have been reprinted in Section 4.3.

We would like to thank our supervisor Jiří Srba for his feedback and suggestions, as well as discussions throughout the project. Furthermore, we would like to thank Kenneth Yrke Jørgensen for his help and ideas regarding TAPAAL and the translations. Finally, we thank Alexandre David for his help with UPPAAL.

Contents

Chapter 1

Introduction

Increasing demands on the reliability and safety of embedded software systems have fueled the need for extensive research into formal modeling and verification. Often, these embedded systems rely on timing-constraints (e.g. deadlines). To this end, discrete models have been extended to time-dependent models such as *Networks of Timed Automata* (NTA) [5, 6], Time Petri Nets (TPN) [33, 34] and Timed-Arc Petri Nets (TAPN) [13, 24]. These models are among the most studied time-dependent models. For a comparative overview of these models the reader is referred to [40].

Timed automata (TA) were introduced by Alur and Dill [5, 6]. A TA is a time-extension of a finite automaton where continuous and synchronous clocks can be used to model time constraints. An NTA is then a parallel composition of TA where different automata are allowed to communicate over channels.

In 1962, Carl Adam Petri introduced the Petri net model in his dissertation [37]. Since then, Petri nets has become a well-established and popular model of distributed systems, in part thanks to their intuitive graphical representation. TPN is a well-known time extension of Petri nets which was introduced by Merlin and Farber [33, 34]. In this model, time intervals are assigned to transitions which denote its earliest and latest firing time. This can be intuitively explained as each transition having its own clock which is ticking once the transition becomes enabled. A transition must fire before its latest firing time. However, it is possible that an enabled transition becomes disabled before it is fired (e.g. by firing another transition).

Another well-known time-extension of Petri nets is TAPN, introduced by Bolognesi et al. [13] and Hanisch [24]. In TAPN each token is assigned a real number indicating its age. Time intervals on arcs from places to transitions restrict which tokens can be used to fire transitions. Note that both discrete and continuous time versions of the timed-arc Petri net model have been introduced. In this report, we will focus on the continuous time version of the model. Recent work on the verification tool TAPAAL [1] by Byg et al. [17] extended the timed-arc Petri net model with invariants on places which gives an upper bound on the age of tokens in a place and so-called transport arcs which allow for transporting tokens between places while preserving their ages.

Recently, a substantial amount of research has focused on the correspondence be-

tween time-dependent formalisms. This has led researchers to develop several translations between these formalisms (see e.g. [12, 13, 15, 17, 18, 39]) in order to establish a correspondence between them up to some notion of equivalence (e.g. bisimulation [15, 18]). However, each translation requires a distinct proof of correctness, even though the translations often use the same ideas, e.g. simulating a single step in one model by a number of steps in another model. For a more complete overview of these translations the reader is referred to [36, 40].

In this thesis, we study TAPN extended with invariants, transport arcs and inhibitor arcs. We prove that invariants alone is enough to make boundedness and coverability undecidable, which was published in a recent paper [28] (the results of this paper are included in Section 4.3). In respect to formal verification of time-dependent models we present Timed Computation Tree Logic (TCTL) in its full generality, unlike much work on TCTL where maximal runs are not handled in full detail (see e.g. [18, 36]). Further, we identify a class of translations that preserve TCTL (or a specific subset of TCTL) by developing a general framework (motivated by [18]), which works at the level of timed transition systems, making it independent of the modeling formalism. We then develop two novel translations from TAPN to NTA and apply the framework to these in order to prove that they both preserve the full TCTL. Following this, we further extend the TAPN model such that each token carries both its age and an integer value and sketch how to extend the translations. Finally, we compare the performance of our translations to previous translations from TAPN to NTA.

1.1 Introductory Example

We shall now informally introduce the model we use in this report, called timed-arc Petri net with inhibitor arcs, transport arcs and invariants. To do so we shall use a model of a rollercoaster which is illustrated in Figure 1.1.

The model of the rollercoaster contains a number of *places* (drawn as circles), *transitions* (drawn as black squares) and *arcs* from places to transitions or transitions to places. Time intervals are associated with arcs from places to transitions. Further, the places in the model may contain *tokens*, each of which is associated with a real number signifying the age of the token. In the example, there are two tokens in the place called *station*, both with age 0.0. Invariants may be assigned to places which restrict the ages of the tokens in that place. In our example, the place *first_half* has an associated invariant of $[0, 4]$ (written inv: ≤ 4). There are different types of arcs: normal arcs (normal arrow tip), transport arcs (diamond arrow tip) and inhibitor arcs (circle arrow tip). The precise semantics of the model will be made clear later in this report. Intuitively, the normal arcs from places to transitions will consume tokens of appropriate ages and normal arcs from transitions to places will produce tokens of age zero. A transport arc works in a similar fashion except that any token produced will have the same age as the one consumed. Inhibitor arcs are used to test for the absence of tokens of certain ages in places.

The idea of the model is that we have two trains for the rollercoaster. The roller-

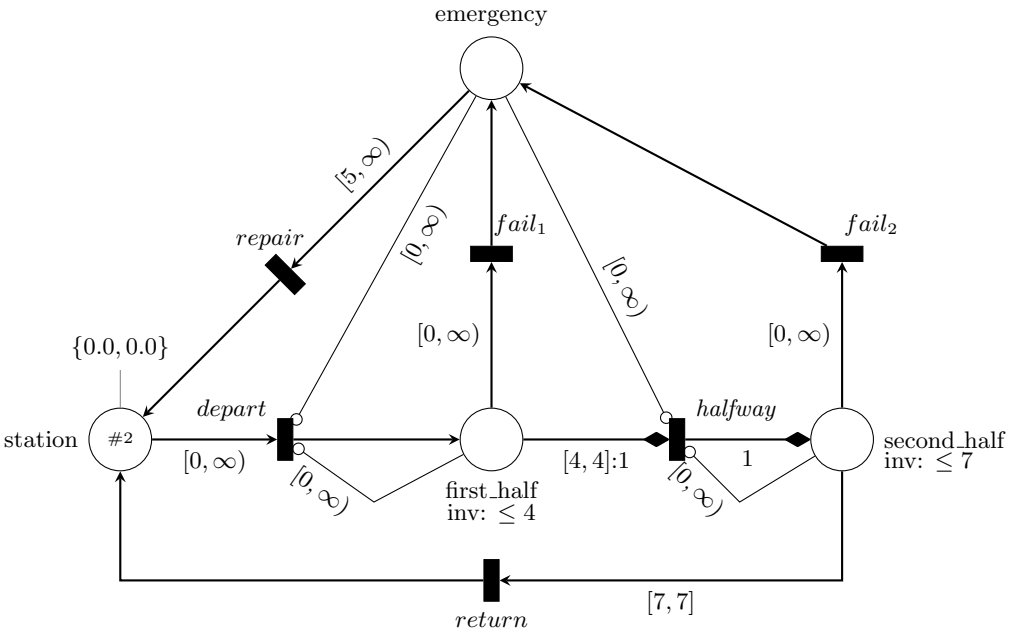


Figure 1.1: A rollercoaster modeled as a timed-arc Petri net with inhibitor arcs, transport arcs and invariants.

coaster must ensure the safety of the passengers which means that the two trains cannot be in the same part of the track at once. Further, since a train may experience some sort of failure at any given time during its trip around the rollercoaster, the system must ensure that the other train is stopped when there is an emergency. Thus, a train can only leave the station and enter the first half of the track if there is no emergency and no other train in the first half of the track (inhibitor arcs are used to check for the absence of tokens in these places) and similarly for entering the second half of the track. As a concrete example, say the first train fails in the second half of the track while the second train is on the first half. In this case, the second train will be stopped before it enters the second half of the track so there can be no crash between the two. It takes 4 time units for a train to run through the first half of the track and 3 time units to run through the second part. Thus, a full trip around the track takes 7 time units. Once the first train enters the second half of the track, the other train may leave the station. In the event that a train fails, it takes at least 5 time units to repair it and return it to the station.

Once we have a model of a system, we want to verify if certain properties are satisfied. For instance, we might want to verify if it is possible for the two trains to be in the same part of the track of the rollercoaster at once (except for the station). This property is a safety property because the proof of the property is a finite sequence of steps describing how to reach the situation where both trains are in the same part of the track. Naturally, if we can find such a sequence, we can conclude that there is an

error in the mode,l as it should not be possible for the two trains to be in the same part of the track at once. As another example of a property consider one that checks if it is possible for the two trains to run forever without any emergency. This is a liveness property because the proof is an infinite sequence of steps describing how the trains can run forever without an emergency.

Chapter 2

Preliminaries

In this chapter, we shall introduce some basic concepts and notation which we will use through this report.

We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the set of natural numbers and we let $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. In a similar way, we let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers (including 0). We shall use the notation 2^S, for some set S, to denote the power set of S, that is, the set of all subsets of S.

A multiset is a pair $S = (D, f)$ where D is a set and $f : D \longrightarrow \mathbb{N}_0$ is a multiplicity function that for a given element $d \in D$ returns the number of occurrences of d in the multiset. We will now define some of the standard operators on multisets. Let $S_1 = (D, f_1)$ and $S_2 = (D, f_2)$ be multisets over the same set D. Then count, union, subtraction, inclusion, size and membership are defined as

- $S_1(d) = f_1(d)$ for all $d \in D$,
- $(S_1 \cup S_2)(d) = f_1(d) + f_2(d)$ for all $d \in D$,
- $(S_1 \setminus S_2)(d) = max(0, f_1(d) - f_2(d))$ for all $d \in D$,
- $S_1 \subseteq S_2$ if $\forall d \in D \,.\, f_1(d) \leq f_2(d)$,
- $|S_1| = \sum_{d \in D} f_1(d)$, and
- $d \in S_1$ if $f_1(d) > 0$.

For notational convenience, we will use multisets as ordinary sets with the defined operations implicitly interpreted over multisets. An example of the union operator is $\{2, 2, 4.5\} \cup \{2, 3.1\} = \{2, 2, 2, 3.1, 4.5\}$. A multiset S is finite if $|S| < \infty$. We let the set of all finite multisets over a set S be denoted by $\mathcal{B}(S)$.

For a binary relation R on some set X, the reflexive closure of R is a binary relation R' such that,

- R' is reflexive, i.e. $x \, R' \, x$ for all $x \in X$,

- $R \subseteq R'$, and
- for any relation R'', if $R \subseteq R''$ and R'' is reflexive, then $R' \subseteq R''$, that is, R' is the smallest reflexive relation containing R.

Similarly, the transitive closure of R is a binary relation R' such that,

- R' is transitive, i.e. if $x\,R'\,y$ and $y\,R'\,z$ then $x\,R'\,z$ for all $x, y, z \in X$,
- $R \subseteq R'$, and
- for any relation R'', if $R \subseteq R''$ and R'' is transitive, then $R' \subseteq R''$, that is, R' is the smallest transitive relation containing R.

For a binary relation $\longrightarrow$ over some set X, we shall use the notation $\longrightarrow^*$ to mean the reflexive and transitive closure of $\longrightarrow$.

We shall now define the notion of a timed transition system.

Definition 1 (Timed Transition System) 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 relation on states called the transition relation consisting of discrete actions and time delays,
- $\mathcal{AP}$ is a set of atomic propositions, and
- $\mu : S \longrightarrow 2^{\mathcal{AP}}$ is a function assigning sets of true atomic propositions to states.$\diamondsuit$

We write $s \longrightarrow s'$ (resp. $s \xrightarrow{d} s'$ for some $d \in \mathbb{R}_{\geq 0}$) instead of $(s, s') \in \longrightarrow$ (resp. $(s, d, s') \in \longrightarrow$ for some $d \in \mathbb{R}_{\geq 0}$) for some $s, s' \in S$.

As evident by the definition, there are two types of transitions in a TTS. Transitions of the type $s \longrightarrow s'$ are ordinary transitions that result from performing actions. Transitions of the type $s \xrightarrow{d} s'$ for some $d \in \mathbb{R}_{\geq 0}$, are delay (or time-elapsing) transitions. For $s \xrightarrow{d} s'$ we shall sometimes refer to s' as $s[d]$. We write $s \longrightarrow$ (resp. $s \xrightarrow{d}$ for some $d \in \mathbb{R}_{\geq 0}$) whenever $s \longrightarrow s'$ (resp. $s \xrightarrow{d} s'$ for some $d \in \mathbb{R}_{\geq 0}$) for some $s' \in S$. Similarly, we write $s \not\longrightarrow$ (resp. $s \xrightarrow{d}\!\!\!\!\!\!/\;\;$ for some $d \in \mathbb{R}_{\geq 0}$) whenever there is no state s' such that $s \longrightarrow s'$ (resp. $s \xrightarrow{d} s'$ for some $d \in \mathbb{R}_{\geq 0}$).

We require that the TTS satisfy the following standard conditions for delay transitions (see e.g. [12]). For all $d, d' \in \mathbb{R}_{\geq 0}$ and $s, s', s'' \in S$ we have:

1. **Additivity:** if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ then $s \xrightarrow{d+d'} s''$,
2. **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. **Determinism:** if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$.

The first requirement states that if we can delay for d time units and thereby reach a state s', and subsequently delay d' to reach a state s'' then we can also delay $d + d'$ time units from s and reach s''. The second requirement states the inverse of the first, that is if we can delay for $d + d'$ time units in s to reach s'', then we can also do two subsequent delays of d and d' and reach the same state (through the intermediate state s'). The third requirement states that the only state we can reach from a state s by delaying 0 time units is s itself. Finally, the fourth requirement states that delay transitions are deterministic, i.e. we will always reach the same state from a state s when we delay d time units.

Chapter 3

Extended Timed-Arc Petri Nets

In this chapter, we shall explore our extension of the timed-arc Petri net model. Specifically, our model is an extension of the work by Byg et al. [17]. We extend their model with inhibitor arcs.

3.1 Basic Definitions

We shall now define our extended timed-arc Petri net model. We will first define the set of well-formed time intervals as the subset of time intervals satisfying the following abstract syntax where $a \in \mathbb{N}_0, b \in \mathbb{N}$ and $a < b$:

$$I ::= [a,a] \mid [a,b] \mid [a,b) \mid (a,b] \mid (a,b) \mid [a,\infty) \mid (a,\infty) \quad .$$

We denote the set of all well-formed time intervals by $\mathcal{I}$. Further, the set of all well-formed time intervals for invariants is denoted $\mathcal{I}_{\text{Inv}}$ and is defined according to the following abstract syntax:

$$I_{\text{Inv}} ::= [0,0] \mid [0,b] \mid [0,b) \mid [0,\infty) \quad .$$

The predicate $r \in I$ is defined for $r \in \mathbb{R}_{\geq 0}$ in the expected way.

Definition 2 (TAPN) A *Timed-Arc Petri Net with invariants, inhibitor arcs and transport arcs*, abbreviated TAPN, is a tuple $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ where

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of normal arcs called the flow relation,
- $c : F|_{P \times T} \longrightarrow \mathcal{I}$ is a function assigning time intervals to arcs from places to transitions,

- $F_{tarc} \subseteq (P \times T \times P)$ is the set of transport arcs that satisfy for all $(p,t,p') \in F_{tarc}$ and all $r \in P$:

$$\big((p,t,r) \in F_{tarc} \Rightarrow p' = r\big) \wedge \big((r,t,p') \in F_{tarc} \Rightarrow p = r\big) \wedge$$
$$(p,t) \notin F \wedge (t,p') \notin F,$$

- $c_{tarc} : F_{tarc} \longrightarrow \mathcal{I}$ is a function assigning time intervals to transport arcs,
- $F_{inhib} \subseteq P \times T$ is the set of inhibitor arcs satisfying for all $(p,t) \in F_{inhib}$ and all $p' \in P$

$$(p,t) \notin F \wedge (p,t,p') \notin F_{tarc},$$

- $c_{inhib} : F_{inhib} \longrightarrow \mathcal{I}$ is a function assigning time intervals to inhibitor arcs, and
- $\iota : P \longrightarrow \mathcal{I}_{inv}$ is a function assigning invariants to places. ♢

Note that we shall sometimes refer to less general models than the one above. For this we will use a special abbreviation: TAPN(inv) refers to a timed-arc Petri net model extended with invariants, TAPN(tarc) refers to one extended with transport arcs and TAPN(inhib) refers to one extended with inhibitor arcs.

For a given transition $t \in T$ we will define the *preset* of t as the set of all input places and the *postset* as the set of all output places. Formally, the preset is defined as ${}^\bullet t = \{p \in P \mid (p,t) \in F \vee \exists p' \in P \,.\, (p,t,p') \in F_{tarc}\}$. Note that the places from which there are inhibitor arcs to the transition t are not included in the preset. For our purposes we will only need the preset to refer to all places from which tokens will be consumed when firing t, hence the exclusion. Similarly, the postset is defined as $t^\bullet = \{p \in P \mid (t,p) \in F \vee \exists p' \in P \,.\, (p',t,p) \in F_{tarc}\}$.

We say that a transition t is of *degree* 2 if $|{}^\bullet t| = |t^\bullet| = 2$. We say that a TAPN N is of *degree* 2 if all transitions in the net are of degree 2.

3.2 Example

In order to provide some intuition for the TAPN model before introducing the formal semantics, we will now give an example which introduces the various features of the model in a step-by-step manner. We will start with a basic timed-arc Petri net and then gradually decorate that example with transport arcs, invariants and finally inhibitor arcs while explaining the intuition behind the change in behaviour of the net, induced by these additions.

The basic timed-arc Petri net is presented in Figure 3.1. It includes two transitions t_0 and t_1 and 5 places $p_i, 0 \leq i \leq 4$. There are tokens of age 0.0 in the places p_0, p_1 and p_2. Initially, only t_0 is enabled because its input place p_0 contains a token of an age which satisfies the constraint on the arc from p_0 to t_0. Transition t_1 requires a token of any age in p_1 but also a token of an age in the interval $[4,5]$ in p_2 which is

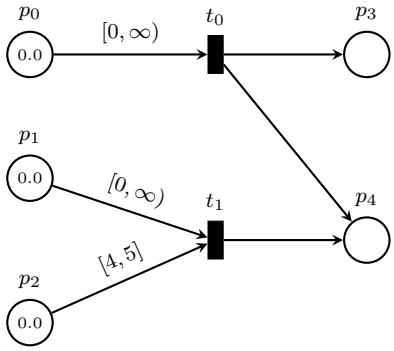


Figure 3.1: A basic timed-arc Petri net.

why t_1 is not enabled initially. Since t_0 is enabled, we can fire it whereby we would remove a token of an appropriate age from p_0 and produce a token of age 0.0 in each of the places p_3 and p_4. However, we could also choose to do a time delay of, say, 4.5 time units, whereby all tokens in the net would grow 4.5 time units older. Since all tokens are now of age 4.5 both t_0 and t_1 are enabled since both would have tokens of appropriate ages in all their input places.

Let us now introduce transport arcs into our example net. Specifically, we will replace the normal arcs from p_0 to t_0 and from t_0 to p_4 with a transport arc from p_0 through t_0 to p_4 as illustrated in Figure 3.2. Note that the : 1 on the transport arc is there to help distinguish where each token goes in case we have more than one transport arc through the transition.

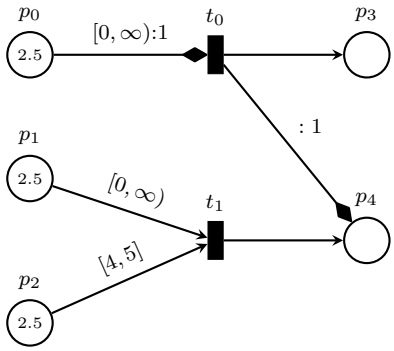


Figure 3.2: A timed-arc Petri net with transport arcs.

For the sake of illustration, assume that we have initially made a time delay of 2.5 time units such that all tokens are now of age 2.5. Transition t_0 is still the only enabled transition in net at this point, however there is a difference when we choose to fire t_0. By firing t_0 we will still remove a token of appropriate age from p_0 and produce a token of age 0.0 in the place p_3 as before. However, due to the transport arc we will produce a token at p_4 of the same age as the token we removed from p_0,

i.e. of age 2.5 in this case. In this way, the transport arcs allow us to preserve the age of tokens as we transport them around the net.

Let us now add invariants to our example net. Specifically, we will add invariants to the places p_2 and p_4 that disallow tokens older than 5, as illustrated in Figure 3.3.

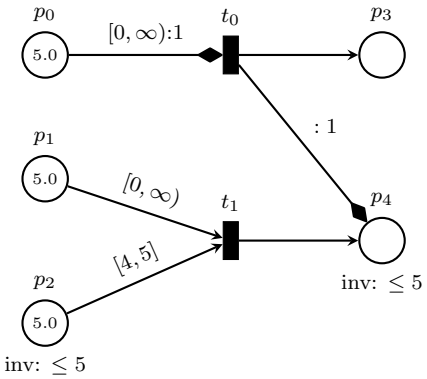


Figure 3.3: A timed-arc Petri net with transport arcs and invariants.

Note that by convention, if no invariant is given for a place then this implicitly means that the invariant for this place is $[0, \infty)$. Assume that we have done a time delay such that all tokens are of age 5.0. At this point we cannot do any further time delays because that would violate the invariant at p_2. We are forced to fire either t_0 or t_1, both of which are enabled. Thus, invariants facilitate urgency in the model. One of the particularities of the use of invariants and transport arcs is demonstrated by the invariant of the place p_4, which has implications for the transition t_0 because of the transport arc from p_0 through t_0 to p_4. Since transport arcs preserve the ages of tokens we can only fire t_0 when there is a token of age smaller than or equal to 5 in p_0, otherwise the token produced at p_4 would violate the invariant on that place.

Finally, let us introduce inhibitor arcs into our example. Specifically, we will replace the arc from p_2 to t_1 with an inhibitor arc as illustrated in Figure 3.4. Intuitively,

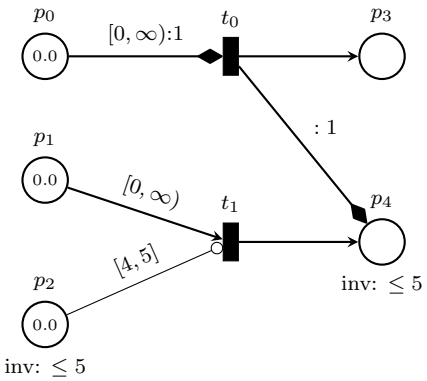


Figure 3.4: A timed-arc Petri net with transport arcs, invariants and inhibitor arcs.

an inhibitor arc is the opposite of a normal arc. For instance, t_1 will now be enabled whenever there is a token of any age in p_1 and no token with an age which belongs to the interval $[4, 5]$ in p_2. Thus, initially both t_0 and t_1 are enabled since there are tokens in p_0 and p_1 with appropriate ages, and there is no token with an age between 4 and 5 in p_2. In this way, inhibitor arcs allow us to test for the absence of tokens of a certain age in an input place.

This concludes our introductory example to the various features of the TAPN model.

3.3 Semantics

We will now define the semantics of the TAPN model. To do so, we shall first define the concept of a marking on a TAPN.

Definition 3 (Marking) Let $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ be a TAPN. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}_{\geq 0})$, such that for every place $p \in P$ and every token $x \in M(p)$ it holds that $x \in \iota(p)$. The set of all markings over N is denoted $\mathcal{M}(N)$. ♢

A *marked* TAPN is defined as a pair (N, M_0) where N is a TAPN and M_0 is the initial marking on N. Note that we only allow initial markings where all tokens have age 0.

Definition 4 (Enabledness) Let $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ be a TAPN. We say that a transition $t \in T$ is *enabled* in a marking M if

- in all places $p \in {}^\bullet t$ where $(p, t) \in F$ there is a token x with an age belonging to the time interval on the arc from p to t, i.e.

$$\forall p \in {}^\bullet t \text{ s.t. } (p, t) \in F \,.\, \exists x \in M(p) \,.\, x \in c(p, t) \,,$$

- in all places $p \in {}^\bullet t$ where $(p, t, p') \in F_{tarc}$ for some $p' \in P$ then moreover the age of the token x in p must satisfy the invariant at p', i.e.

$$\forall p \in {}^\bullet t \text{ s.t. } (p, t, p') \in F_{tarc} \,.\, \exists x \in M(p) \,.\, x \in c_{tarc}(p, t, p') \wedge x \in \iota(p') \,,$$

- in all places $p \in P$ where $(p, t) \in F_{inhib}$ there is no token with an age belonging to the time interval on the inhibitor arc from p to t, i.e.

$$\forall p \in P \text{ s.t. } (p, t) \in F_{inhib} \,.\, \neg\exists x \in M(p) \,.\, x \in c_{inhib}(p, t) \,. \qquad ♢$$

Definition 5 (Firing Rule) Let $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ be a TAPN, M some marking on N and $t \in T$ some transition of N. If t is enabled in the marking M, it can be *fired*, whereby we reach a marking M' defined as

$$\forall p \in P \,.\, M'(p) = \big(M(p) \setminus C_t^-(p)\big) \cup C_t^+(p)$$

where

- for every $p \in P$ such that $(p,t) \in F$
  $C_t^-(p) = \{x\}$ where $x \in M(p)$ and $x \in c(p,t)$,
- for every $p \in P$ such that $(t,p) \in F$
  $C_t^+(p) = \{0\}$,
- for every $p, p' \in P$ such that $(p,t,p') \in F_{tarc}$
  $C_t^-(p) = \{x\} = C_t^+(p')$ where $x \in M(p)$, $x \in c_{tarc}(p,t,p')$ and $x \in \iota(p')$, and
- in all other cases we set the above sets to $\emptyset$.

Note that there may be multiple choices for the sets $C_t^-(p)$ and $C_t^+(p)$ for each p. We simply fix these sets before firing t. ♢

Let us now define the notion of a time delay in our model. To do so, we shall introduce an addition operator on multisets. For a multiset $B = (\mathbb{R}_{\geq 0}, f)$ and a non-negative real $d \in \mathbb{R}_{\geq 0}$, we define the addition operator in the following manner,

- $(B+d)(x) = \begin{cases} f(x-d) & \text{if } x - d \geq 0 \\ 0 & \text{otherwise} \end{cases}$

where x ranges over the elements of $\mathbb{R}_{\geq 0}$.

Definition 6 (Time Delay) Let $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ be a TAPN and M some marking on N. A *time delay* $d \in \mathbb{R}_{\geq 0}$ is allowed if $(x + d) \in \iota(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying d time units no token violates the invariants. By delaying d time units we reach a marking M' defined as $M'(p) = M(p) + d$ for all $p \in P$. ♢

The semantics of a TAPN is given by a timed transition system. Specifically, a TAPN $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ defines a TTS $T(N) = (\mathcal{M}(N), \longrightarrow, \mathcal{AP}, \mu)$ where the set of states are the markings on N and the transition relation $\longrightarrow$ is defined such that $M \longrightarrow M'$ if by firing transition some transition t in marking M we get to marking M' and $M \xrightarrow{d} M'$ if by delaying d time units in marking M we get to marking M'. The set of atomic propositions $\mathcal{AP}$ and the labeling function μ are adopted from Byg et al. [17] and are defined as $\mathcal{AP} \stackrel{def}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$ and $\mu(M) \stackrel{def}{=} \{(p \bowtie n) \mid |M(p)| \bowtie n \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$. The intuition is that a proposition $(p \bowtie n)$ is true in a marking M if the number of tokens in the place p satisfies the proposition with respect to n.

Chapter 4

Undecidability Results for Timed-Arc Petri Nets

In this chapter, we will present an overview of undecidability results for various extensions of the standard TAPN model. We shall focus on three problems, namely reachability, coverability and boundedness.

4.1 Problems of Interest

Let us define the three problems for TAPN, starting with reachability. Reachability simply asks whether a given marking is reachable from the initial marking in some TAPN N.

Definition 7 (Reachability) Given a marked TAPN (N, M_0) and a marking $M \in \mathcal{M}(N)$. M is said to be *reachable* if $M_0 \longrightarrow^* M$. $\diamondsuit$

For some $M' \in \mathcal{M}(N)$ we let $\mathcal{M}(N, M')$ denote the set of reachable markings in N from the marking M'. The coverability problem is defined as follows.

Definition 8 (Coverability) Let (N, M_0) be a marked TAPN and let $M \in \mathcal{M}(N)$ be a marking. M is said to be *coverable* if there exists a reachable marking $M' \in \mathcal{M}(N, M_0)$, such that $M(p) \subseteq M'(p)$ for all places $p \in P$. $\diamondsuit$

Usually, coverability is used to prove safety properties. This is done by specifying a marking that characterizes bad behaviour and asking whether it is possible to cover that marking. For example, if we model some form of mutual exclusion protocol, we can ask whether we can cover a marking where there is more than one process in the critical section. If we can, the protocol is not correct.

For boundedness, we generally consider two variants, which are inter-related. The first is k-boundedness.

Definition 9 (k-boundedness) A marked TAPN (N, M_0) is said to be *k-bounded* if there exists a $k \in \mathbb{N}$ such that the total number of tokens in the net does not exceed k for any marking reachable from M_0. $\diamondsuit$

We define boundedness in terms of k-boundedness.

Definition 10 (Boundedness) A marked TAPN (N, M_0) is said to be *bounded* if it is k-bounded for some $k \in \mathbb{N}$. ♢

Note that if a net is not k-bounded for some given $k \in \mathbb{N}$, then we cannot conclude that it is unbounded. It may be k'-bounded for some $k' > k$. If a net is unbounded however, then we may conclude that it is not k-bounded for any $k \in \mathbb{N}$. The reason for this distinction, is that while boundedness is undecidable [28], k-boundedness is decidable. Boundedness is useful because reachability and coverability are decidable for bounded nets. This follows from the fact that there are only finitely many reachable markings in a bounded net since techniques such as regions [5, 6] or zones [9] can be used to represent continuous time in a finite way.

We shall later use a slightly different definition of boundedness, which we call place-boundedness to distinguish the two.

Definition 11 (Place-boundedness) A marked TAPN (N, M_0) is *place-bounded* if there exists some $k \in \mathbb{N}$ such that the total number of tokens in any place does not exceed k for any marking reachable from M_0. ♢

It is clear that boundedness and place-boundedness are related. If a marked TAPN is place-bounded for some k, then we know that it is k'-bounded where $k' = |P| \cdot k$. Similarly, if it is k-bounded then we know that it is place-bounded for k also.

4.2 Overview

We shall now give an overview of the known undecidability results for various extensions of TAPN. However, we will first briefly introduce the notion of a two-counter Minsky machine (2-CM).

Definition 12 A *Two-Counter Minsky Machine (2-CM)* with two non-negative registers r_1 and r_2 is a sequence of instructions $(I_1 : Ins_1;\ I_2 : Ins_2;\ \ldots\ I_{e-1} : Ins_{e-1};\ I_e : HALT)$ where for every $j,\ 1 \leq j < e,\ Ins_j$ is one of the two types:

- $r_i := r_i + 1;$ goto I_k; where $i \in \{1, 2\}$ and $k \in \{1, 2, \ldots, e\}$ (Increment).
- if $r_i > 0$ then $r_i := r_i - 1;$ goto $I_k;$ else goto $I_\ell;$ where $i \in \{1, 2\}$ and $k, \ell \in \{1, 2, \ldots, e\}$ (Test and decrement).

The last instruction is always the HALT instruction. A *configuration* of a 2-CM is a triple (j, v_1, v_2) where $j \in \{1, 2, \ldots, e\}$ is the index of instruction I_j to be executed and v_1 and v_2 are the values of the registers r_1 and r_2, respectively. ♢

The computational step relation of a 2-CM is defined as expected and we use the notation $(j, v_1, v_2) \rightarrow (j', v_1', v_2')$ to denote that we perform the current instruction I_j with values v_1 and v_2 in the registers, resulting in the configuration (j', v_1', v_2').

Definition 13 (The Halting Problem for 2-CM) Given a 2-CM, is it possible to reach the halt instruction from the initial configuration $(1, 0, 0)$, i.e. $(1, 0, 0) \rightarrow^* (e, v_1, v_2)$ for some $v_1, v_2 \in \mathbb{N}_0$? ♢

Theorem 14 (Minsky [35]) The halting problem for 2-CM is undecidable. ♢

Ruiz et al. [38] showed that for a slighty more general TAPN model (they allow real numbers in intervals on input arcs), the reachability problem is undecidable by reduction from the halting problem for 2-CM[1]. They essentially create a weak simulation of a 2-CM, in the sense that the behaviour of the 2-CM can be simulated in the net, though additional behaviour is possible. However, clever exploitation of time delays ensures that both registers can only be emptied completely if a faithful simulation of the 2-CM is performed in the net, thereby ensuring that a specific marking is reachable iff the 2-CM halts.

Abdulla and Nylén [3] showed that for Timed Petri Nets (TdPN) (essentially a TAPN where intervals are also present on output arcs, and the age of a produced token is non-deterministically chosen to be in this interval) the coverability problem is decidable. Specifically, they introduce the notion of existential zones, which represent (possibly infinite) upward-closed sets of markings (a set $\mathcal{S}$ is upward-closed if for all markings M, M' it holds that if $M \in \mathcal{S}$ and $M \subseteq M'$ then $M' \in \mathcal{S}$). The algorithm works by backward analysis. Specifically, they perform a fixpoint iteration, generating larger and larger upward-closed sets of markings, such that at iteration i, all markings from which it is possible to cover the target marking in i or fewer steps, are found. Termination of the algorithm is ensured by a well-quasi ordering among the existential zones. A reflexive and transitive binary relation (a quasi ordering) $\preceq$ on a set A is a well-quasi ordering if for all infinite sequences $a_0, a_1, a_2, \ldots \in A$ there exists $i, j \in \mathbb{N}_0$ such that $i < j$ and $a_i \preceq a_j$. This ordering ensures that the iteration will eventually reach a fixpoint and it allows them to represent the upwards-closed sets of markings in a finite way. Coverability from the initial marking then amounts to checking whether the initial marking is included in the fixpoint.

Abdulla et al. [4] proved that boundedness is decidable for TdPN. They use the notion of *regions* for TdPN. The notion of regions for TdPN is similar to the notion of regions for TA (see [5, 6]). They provide an algorithm similar to the coverability tree algorithm by Karp and Miller [32], where each node in the coverability tree is labelled by a region instead of a marking. Starting from the root node (labelled with a region satisfied by the intial marking) they successively build the tree by adding child nodes for each possible successor region. The algorithm by Karp and Miller [32] could also be used to decide boundedness by checking whenever we reach a new marking M if there exists a marking M' on the path from the root to M such that $M'(p) \subset M(p)$ for each place p (in this case the net is unbounded). The regions are used in a similar way to determine when the net is unbounded, i.e. whenever we reach a new region R we check if there exists another region R' on the path from the root to R such that

[1] Technically, they work with an extended two-counter machine that empties both registers before performing the halt instruction.

R is "smaller" than R. The algorithm is guaranteed to terminate due to a well-quasi ordering among the regions.

Hack [23] showed that the boundedness problem for untimed Petri nets with inhibitor arcs and untimed Petri nets with priorities on transitions is undecidable. They introduce the notion of a counter automaton which is essentially a generalized version of a two-counter machine that contains a finite set of counters and some additional instructions. They show that adding inhibitor arcs to an untimed Petri net allows them to simulate the counter automaton in the Petri net correctly, since an inhibitor arc allows them to test for zero (which is not possible in TAPN). Since a counter automaton is basically a generalization of a two-counter Minsky machine, it follows that the halting problem for counter automata is undecidable, which in turn implies undecidability of boundedness. They require only a small modification to obtain the same result using priorities (this is not surprising, given that the author also shows how to encode inhibitor arcs using priorities and vice versa). Though not explicitly mentioned by the author, this also implies the undecidability of coverability, because the simulation can easily be used to show that a marking is coverable iff the automaton halts.

Dufourd et al. [21] proved that boundedness is undecidable for untimed Petri nets with reset arcs. A reset arc is an arc from a place to a transition which upon firing the transition will remove all tokens in the place. Reset arcs do not influence the enabledness of transitions but only have an effect when firing transitions. They provide a reduction from Hilbert's Tenth Problem to boundedness for Petri nets with reset arcs.

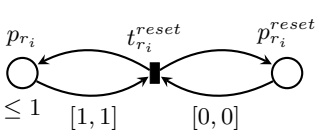
4.3 Timed-Arc Petri Nets with Invariants

In this section we will prove the undecidability of place-boundedness and coverability for the TAPN(inv) model by reduction from the halting problem for 2-CM. The contents of this section appeared in the paper [28].

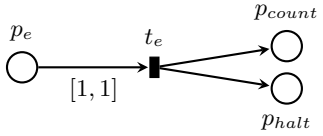
We will now describe the reduction from 2-CM to TAPN(inv). Given a 2-CM $(I_1 : Ins_1;\ I_2 : Ins_2;\ \ldots\ I_{e-1} : Ins_{e-1};\ I_e : HALT)$ we construct a TAPN(inv) (P, T, F, c, ι) where

- $P = \{p_j, q_j \mid 1 \leq j < e\} \cup \left\{p_{r_1}, p_{r_1}^{reset}, p_{r_2}, p_{r_2}^{reset}\right\} \cup \{p_{count}\} \cup \{p_e, p_{halt}\}$
- $T = \left\{t_{r_1}^{reset}, t_{r_2}^{reset}\right\} \cup \left\{t_j, t_j^{goto} \mid Ins_j \text{ is of type increment}\right\} \cup \left\{t_j^{else_1}, t_j^{else_2}, t_j^{then} \mid Ins_j \text{ is of type test and decrement}\right\} \cup \{t_e\}$

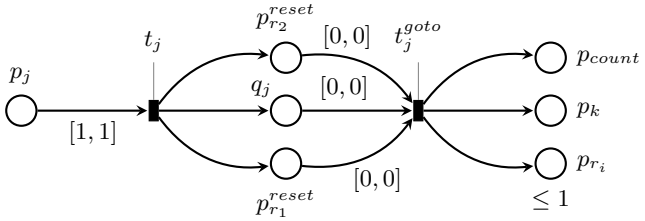
The number of tokens in p_{r_1} and p_{r_2} correspond to the values of r_1 and r_2, the number of tokens in p_{count} remembers the number of computation steps which have been simulated in the net and $p_1, \ldots, p_e$ corresponds to the instructions $Ins_1, \ldots, Ins_e$ such that the place p_j contains one token if and only if the current instruction is Ins_j. For the flow relation we will split it into 4 parts.



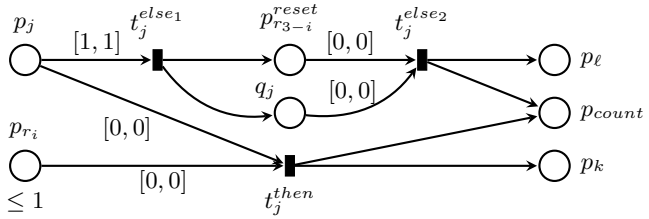
(a) Simulation of a register.



(b) Simulation of Halt instruction.



(c) Simulation of $I_j : r_i := r_i + 1;$ goto I_k.



(d) Simulation of I_j : if $r_i > 0$ then $r_i := r_i - 1;$ goto $I_k;$ else goto I_ℓ.

Figure 4.1: TAPN(inv) models for 2-CM simulation.

- F_1 contains the arcs for the registers. For each register $r_i,\ i \in \{1,2\}$, we add the following arcs to F_1

$$(p_{r_i}, t_{r_i}^{reset}),\ (t_{r_i}^{reset}, p_{r_i}),\ (p_{r_i}^{reset}, t_{r_i}^{reset}),\ (t_{r_i}^{reset}, p_{r_i}^{reset}) \text{ where}$$
$$c((p_{r_i}, t_{r_i}^{reset})) = [1,1],\ \ c((p_{r_i}^{reset}, t_{r_i}^{reset})) = [0,0] \text{ and } \iota(p_{r_i}) = [0,1]\ .$$

  This is illustrated in Figure 4.1a. The number of tokens on p_{r_i} indicates the value of the register. Notice the invariant on the register which disallows tokens with an age greater than 1. Placing a token on $p_{r_i}^{reset}$ allows us to reset the age of all tokens of age 1 in the register.

- F_2 contains the arcs for the increment instructions. For each increment instruction $I_j : r_i := r_i + 1;$ goto $I_k;$, we add the following arcs to F_2

$$(p_j, t_j),\ (t_j, p_{r_2}^{reset}),\ (t_j, q_j),\ (t_j, p_{r_1}^{reset}),\ (p_{r_2}^{reset}, t_j^{goto}),\ (q_j, t_j^{goto}),$$
$$(p_{r_1}^{reset}, t_j^{goto}),\ (t_j^{goto}, p_{count}),\ (t_j^{goto}, p_k),\ (t_j^{goto}, p_{r_i}) \text{ where } c((p_j, t_j)) = [1,1],$$
$$c((p_{r_2}^{reset}, t_j^{goto})) = [0,0],\ \ c((q_j, t_j^{goto})) = [0,0] \text{ and } c((p_{r_1}^{reset}, t_j^{goto})) = [0,0]\ .$$

This is illustrated in Figure 4.1c. Notice that we require a delay of one time unit before firing t_j. Because of this, we allow tokens in each register to be reset (by placing tokens on $p_{r_1}^{reset}$ and $p_{r_2}^{reset}$). Following this, by firing t_j^{goto} a token is added to p_{count}, register r_i is incremented by adding a token to p_{r_i} and control is given to the next instruction I_k by placing a token on p_k.

- F_3 contains the arcs for the test and decrement instructions. For each test and decrement instruction I_j : if $r_i > 0$ then $r_i := r_i - 1$; goto I_k; else goto I_ℓ;, we add the following arcs to F_3

  $(p_j, t_j^{else_1})$, (p_j, t_j^{then}), (p_{r_i}, t_j^{then}), $(t_j^{else_1}, q_j)$, $(t_j^{else_1}, p_{r_{3-i}}^{reset})$, $(q_j, t_j^{else_2})$, $(p_{r_{3-i}}^{reset}, t_j^{else_2})$, $(t_j^{else_2}, p_\ell)$, $(t_j^{else_2}, p_{count})$, (t_j^{then}, p_{count}), (t_j^{then}, p_k) where $c((p_j, t_j^{else_1})) = [1,1]$, $c((p_j, t_j^{then})) = [0,0]$, $c((p_{r_i}, t_j^{then})) = [0,0]$, $c((p_{r_{3-i}}^{reset}, t_j^{else_2})) = [0,0]$ and $c((q_j, t_j^{else_2})) = [0,0]$.

  This is illustrated in Figure 4.1d. Notice that when we follow the else branch (firing transition $t_j^{else_1}$), we can only reset the ages of tokens in the register on which we are not testing for emptiness.

- F_4 contains the arcs for the HALT instruction. Formally it is defined as

  $$F_4 = \{(p_e, t_e), (t_e, p_{count}), (t_e, p_{halt})\} \text{ where } c((p_e, t_e)) = [1,1] \ .$$

  This is illustrated in Figure 4.1b. Again we require a time delay of one time unit before t_e can be fired and a token placed at p_{halt}.

- The flow relation F can then be defined as the union of the four parts, i.e. $F = F_1 \cup F_2 \cup F_3 \cup F_4$ and we let $\iota(p) = [0, \infty)$ for all $p \in P \setminus \{p_{r_1}, p_{r_2}\}$.

We define the initial marking M_0 such that $M_0(p_1) = \{0\}$ and $M_0(p) = \emptyset$ for all $p \in P \setminus \{p_1\}$.

Let (N, M_0) be the marked TAPN(inv) simulating a given 2-CM. Notice that every place in the net except for $p_{r_1}, p_{r_2}, p_{count}$ is *1-safe* (i.e. contains at most one token). In a *correct* simulation of the 2-CM by our net, a configuration (j, v_1, v_2) of the 2-CM corresponds to any marking M where

$$M(p_j) = \{0\}, \quad M(p_{r_i}) = \underbrace{\{0, 0, \ldots, 0\}}_{v_i \text{ times}} \text{ for } i \in \{1, 2\}, \tag{4.1}$$

$|M(p_{count})| = n$ where $n \in \mathbb{N}_0$ and $M(p) = \emptyset$ for all $p \in P \setminus \{p_j, p_{r_1}, p_{r_2}, p_{count}\}$.

We will now describe how to simulate the three types of instructions of a 2-CM in a *correct* way. Assume there is a token of age 0 in p_j.

If I_j is an increment instruction, we need to delay for one time unit in order to enable t_j (see Figure 4.1c). Because we delayed one time unit, all tokens in the registers

are now of age 1. In a correct simulation, we fire repeatedly transitions $t_{r_1}^{reset}$ and $t_{r_2}^{reset}$ until all tokens in p_{r_1} and p_{r_2} are of age 0. Note that it is possible to *cheat* in the simulation, as it is possible to leave some tokens of age 1 in p_{r_1} or p_{r_2} when firing t_j^{goto}.

If I_j is a test and decrement instruction there are two possibilities (see Figure 4.1d). If there is a token of age 0 at p_{r_i}, we fire t_j^{then} in order to decrement the number of tokens in register r_i, and hand over the control to I_k by placing a token on p_k. Otherwise, in the correct simulation we delay one time unit before firing $t_j^{else_1}$. Then we reset the age of all the tokens in the other register, $p_{r_{3-i}}$ to 0. We then proceed by firing $t_j^{else_2}$. This will hand over control to instruction I_ℓ by placing a token on p_ℓ. Again note that it is possible to *cheat* in the simulation, either by leaving tokens of age 1 at $p_{r_{3-i}}$ when proceeding to the next instruction or by taking the *else*-branch even though there is a token at p_{r_i} (because the net does not force us to fire transition t_j^{then} when it is enabled).

If I_j is the halt instruction, we delay one time unit before we fire the last transition t_e and add a token to p_{halt}.

After every instruction one token is added to p_{count}. We will now prove a lemma detailing what happens if we cheat.

Lemma 15 Let (j, v_1, v_2) be the current configuration of a 2-CM CM, (N, M_0) the associated TAPN(inv) and M a marking corresponding to (j, v_1, v_2) (see Equation 4.1). If the net cheats then during the simulation of CM in the next computation step it is not possible to simulate an increment instruction, go to the halt state, nor to take the else-branch of a test and decrement instruction. Further, the net can do at most $v_1 + v_2$ decrements before getting stuck. ♢

Proof We can perform an incorrect simulation in two ways:

- If all tokens in p_{r_1} and p_{r_2} are not reset to age 0 in an increment or test and decrement instruction before going to the next instruction.

- In a test and decrement instruction, the net can fire the transition $t_j^{else_1}$ even if there is a token of age 0 in p_{r_i}. This is possible by delaying 1 time unit to enable the transition. However, this will result in the tokens in p_{r_i} having age 1 and these can not be reset before going to the next instruction.

In both cases we end up in a marking M' where there is at least one token of non-zero age in either p_{r_1} or p_{r_2}. Observe that the simulation of increment, halt and the else-branch of a test and decrement instruction all require a delay of 1 time unit (see Figure 4.1) which would violate the invariants $\iota(p_{r_1})$ or $\iota(p_{r_2})$. Thus, the only possibility is to take the then-branch of a test and decrement instruction. However, this is only possible as long as there are tokens of age 0 in p_{r_1} or p_{r_2}. There are v_1 and v_2 tokens in p_{r_1} and p_{r_2}, repectively. Thus, the net can do at most $v_1 + v_2$ decrements before getting stuck. ■

4.3.1 Undecidability Results

First we prove the undecidability of the place-boundedness problem.

Lemma 16 Given a 2-CM CM and the associated TAPN(inv) (N, M_0), CM halts if and only if N is place-bounded. ♢

PROOF We start by proving that if N is place-bounded then CM halts. Assume that N is place-bounded for some k. Further, assume by contradiction that CM does not halt. After simulating $k+1$ computational steps of CM correctly, the net will be in a marking M where $|M(p_{count})| = k+1$. This is a contradiction to the assumption that N is place-bounded for k.

Now we prove the implication in the other direction. Assume that CM halts in n steps. We will show that N is place-bounded for $2n$. If we simulate CM correctly, there will be at most n tokens at the registers, and exactly n tokens at p_{count}. Hence, the net must cheat in order to become unbounded. In the worst case, it cheats at the last step, when there are at most $n-1$ tokens in the registers and $n-1$ tokens at p_{count}. Then we have that the net is place-bounded for $2n$ since there will be at most $2(n-1)$ tokens at p_{count} by Lemma 15. ■

From Lemma 16 we conclude the following theorem.

Theorem 17 The place-boundedness problem is undecidable for TAPN(inv). ♢

We now prove the undecidability of the coverability problem.

Lemma 18 Let M be a marking such that $M(p_{halt}) = \{0\}$ and $M(p) = \emptyset$ for all $p \in P \setminus \{p_{halt}\}$. Given a 2-CM CM and the associated marked TAPN(inv) (N, M_0), as defined above, CM halts if and only if M is coverable from M_0. ♢

PROOF First we prove that if CM halts then M is coverable from M_0. Assume that the CM halts. By simulating CM correctly in N, we can easily see that we reach a marking M', with a token in p_{halt}, hence $M'(p) \supseteq M(p)$ for all $p \in P$.

Now we prove that if M is coverable from M_0 then CM halts. Assume that M is coverable from M_0. By assumption there exists a reachable marking M' such that $M'(p) \supseteq M(p)$ for all $p \in P$. By definition of coverability, it holds that $0 \in M'(p_{halt})$ and by Lemma 15 this is only possible if we simulate CM correctly in the net, hence CM halts. ■

From Lemma 18 we conclude the following theorem.

Theorem 19 The coverability problem is undecidable for TAPN(inv). ♢

4.4 Conjectures

There are still open problems for some TAPN extensions and in this section we will present conjectures for these problems. Let us start by considering coverability for TAPN(reset) (TAPN extended with reset arcs).

Dufourd et al. [21] showed that coverability is decidable for so-called *Reset Post G-nets*, which are basically untimed Petri nets with reset arcs and the ability to have polynomials as weights on output arcs. Thus, untimed Petri nets with reset arcs are a subclass of Reset Post G-nets where all output arcs have weight 1.

The addition of reset arcs does not break the monotonicity property of TAPN, i.e. for two markings M and M' where $M \subseteq M'$, if $M \xrightarrow{t}$ for some transition t then $M' \xrightarrow{t}$. In other words, it is not possible to disable transitions by adding additional tokens to a marking. This is easy to see since reset arcs does not influence the enabledness of transitions but only have an effect when firing a transition. This means that it should be possible to use the concept of existential zones to describe upward-closed sets of markings. Thus, it should be possible to prove the decidability of coverability for TAPN(reset) by extending the concepts and proofs in [3] to support reset arcs.

Conjecture 20 The coverability problem is decidable for TAPN(reset). ♢

Bouyer et al. [15] proved that coverability is decidable for TAPN extended with so-called *read arcs* which allow for testing for the presence of tokens without consuming any tokens when a transition is fired. The proof is an extension of the proof by Abdulla and Nylén [3]. Transport arcs are a generalization of read arcs and as for reset arcs they do not break the monotonicity property of TAPN. Thus, it should be possible to extended the proof in [3] to support transport arcs.

Conjecture 21 The coverability problem is decidable for TAPN(tarc). ♢

It should be possible to use the algorithm by Abdulla et al. [4] (which is an extension of the coverability tree algorithm by Karp and Miller [32]) to decide boundedness for TAPN(tarc).

Conjecture 22 The boundedness problem is decidable for TAPN(tarc). ♢

4.5 Summary

Table 4.1 shows a summary of the undecidability results mentioned in this chapter. Decidable problems are indicated by ✓ and undecidable problems are indicated by ✕. Further, our results are marked with * and conjectures are marked with ?.

	Reachability	**Coverability**	**Boundedness**
TAPN	✕[38]	✓[3]	✓[4]
TAPN(inv)	✕[38]	✕*	✕*
TAPN(tarc)	✕[38]	✓?	✓?
TAPN(inhib)	✕[38]	✕[23]	✕[23]
TAPN(prio)	✕[38]	✕[23]	✕[23]
TAPN(reset)	✕[38]	✓?	✕[21]

Table 4.1: Decidability Results

Chapter 5

Timed Computation Tree Logic

In this chapter, we shall explain the syntax and semantics of Timed Computation Tree Logic (TCTL) inspired by Penczek and Pólrola [36]. However, unlike [36] which only consider infinite maximal runs, we treat TCTL in its full generality, including finite maximal runs in which the computation gets stuck. We shall focus on TCTL formulae interpreted over continous time models.

We assume a set $\mathcal{AP}$ of atomic propositions. A TCTL formula is given by the abstract syntax:

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

where $\wp \in \mathcal{AP}$ is an atomic proposition and $I \in \mathcal{I}$ is a time interval as defined on page 9. We denote the set of all TCTL formulae over the set of atomic propositions $\mathcal{AP}$ as $\Phi(\mathcal{AP})$. Note that if we remove the operators $E(\varphi_1 R_I \varphi_2)$ and $A(\varphi_1 U_I \varphi_2)$ from the above syntax we get the so-called *safety fragment* of TCTL.

A run in a TTS $(S, \longrightarrow, \mathcal{AP}, \mu)$ is a (finite or infinite) alternating sequence of time delays and discrete transitions of the form $\rho = s_0 \xrightarrow{d_0} s_0' \longrightarrow s_1 \xrightarrow{d_1} s_1' \longrightarrow s_2 \xrightarrow{d_2} \ldots$ such that $s_i, s_i' \in S$ for all $i \geq 0$. For completeness, let us state the intuition behind the operators. The U stands for Until and the R stands for Release.

- $A(\varphi_1 U_I \varphi_2)$: On all runs φ_2 must eventually hold within the interval I, and until it does, φ_1 must hold continuously.
- $E(\varphi_1 U_I \varphi_2)$: There exists a run such that φ_2 eventually holds within the interval I, and until it does, φ_1 holds continuously.
- $A(\varphi_1 R_I \varphi_2)$: On all runs either φ_2 always holds within the interval I or φ_1 occurred previously.
- $E(\varphi_1 R_I \varphi_2)$: There exists a run such that either φ_2 always holds within the interval I or φ_1 occurred previously.

We shall now define the semantics of TCTL formulae. Let $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ be a TTS. A run ρ is said to be a maximal run if it cannot be extended any further by time delays or discrete transitions. Formally, there are three ways a run ρ can be maximal:

(i) ρ is an infinite alternating sequence of the form

$$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow s_3 \xrightarrow{d_3} \dots$$

(ii) ρ is a finite alternating sequence of the form

$$\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \dots \longrightarrow s_n \xrightarrow{\infty}$$

where $\xrightarrow{\infty}$ means that $s_n \xrightarrow{d} s_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$, or

(iii) ρ is a finite alternating sequence of the form:

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

Alternatively the run can be of the form

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

such that for any $d \in \mathbb{R}_{\geq 0}$ where $d > d_n$ (resp. $d \geq d_n$ for the alternate form), we have that $s_n \xrightarrow{d} \not\longrightarrow$ and for any $d' \in \mathbb{R}_{\geq 0}$ where $d' \leq d_n$ (resp. $d' < d_n$), whenever $s_n \xrightarrow{d'} s_n[d']$ then $s_n[d'] \not\longrightarrow$.

Intuitively, the three conditions above describe the possible ways in which a run can be maximal. The first case is an infinite alternating sequence of time delays and discrete transitions. In the second case the run ends in a state where time can diverge, that is a state where any time delay is possible. Finally, the third case describes a run which end in a state from which it is not possible to do a discrete transition after any time delay. Note that there are two different forms for such a run: the final time delay d_n can either be included or not ($d_n^{\leq}$ and $d_n^{<}$ respectively). This case differs from the second case in that time cannot diverge for these runs (any time delay greater than d_n is not possible), however, whenever a time delay is possible we are certain that no discrete transition is possible afterwards. Let the set of all maximal runs ρ in T starting from s be denoted by $Runs(T, s)$.

Figure 5.1 illustrates part of the maximal run $\rho = s_0 \xrightarrow{1} s_0[1] \longrightarrow s_1 \xrightarrow{2.5} s_1[2.5] \longrightarrow s_2 \xrightarrow{2} s_2[2] \longrightarrow s_3 \xrightarrow{1.3} s_3[1.3] \longrightarrow s_4 \xrightarrow{5.2} \dots$. The x-axis indicates elapsed time. Note that discrete transitions take zero time units (which is why discrete transitions are drawn as vertical lines) and that, although not shown in this

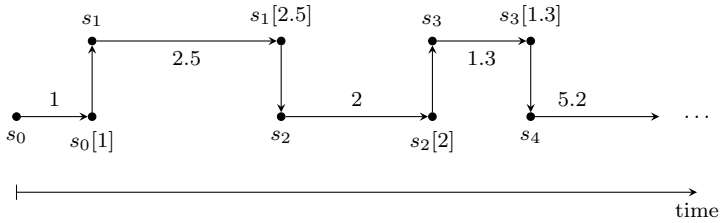


Figure 5.1: An illustration of the concrete run $\rho = s_0 \xrightarrow{1} s_0[1] \longrightarrow s_1 \xrightarrow{2.5} s_1[2.5] \longrightarrow s_2 \xrightarrow{2} s_2[2] \longrightarrow s_3 \xrightarrow{1.3} s_3[1.3] \longrightarrow s_4 \xrightarrow{5.2} \ldots$.

example, time delays can be zero. Hence, it is possible to do multiple discrete transitions in succession without any time progression in between. Further, there is no special meaning as to whether the arrow for a discrete transition goes up or down, this is simply to keep the figure small.

For a maximal run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$, an index $i \in \mathbb{N}$ and a non-negative real $d \in \mathbb{R}_{\geq 0}$ we let

$$r(i, d) = \left(\sum_{j=0}^{i-1} d_j \right) + d.$$

Intuitively, $r(i, d)$ denotes the total time elapsed from the beginning of the run up to some delay d after the i-th discrete transition.

Definition 23 For each maximal run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$, we define a predicate $valid_\rho : \mathbb{N} \times \mathbb{R}_{\geq 0} \times \mathcal{I} \to \{true, false\}$ as follows:

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

Intuitively, the arguments, i, d together describe a possible state $s_i[d]$ in the run ρ and this state lies within the interval I. Figure 5.2 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 although time delays appear identical in the figure they can be of different length (even zero). We see in Figure 5.2 that $valid_\rho(1, d, I)$ is false because $s_1[d]$ lies outside the interval I. Similarly, $valid_\rho(2, d'', I)$ is false because $s_2[d'']$ is not a part of the run (since $d'' > d_2$). Finally, $valid_\rho(2, d', I)$ is true because $s_2[d']$ is a part of the run and within I.

Finally, we define a function denoting the history of a run at some specified point.

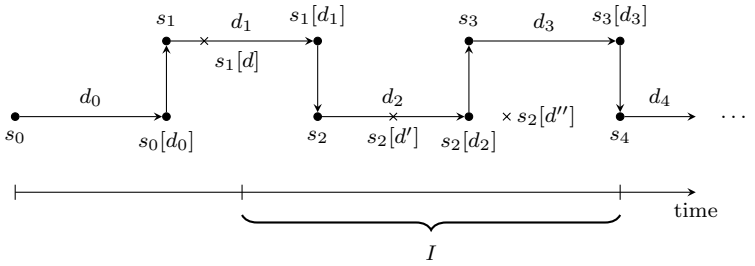


Figure 5.2: An illustration of a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow s_3 \xrightarrow{d_3} s_3[d_3] \longrightarrow s_4 \xrightarrow{d_4} \ldots$.

Definition 24 For a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$, we let $history_\rho : \mathbb{N} \times \mathbb{R}_{\geq 0} \to \mathbb{N} \times \mathbb{R}_{\geq 0}$ be a function which given an index i and a delay d, returns the pairs (j, d') that constitute all previous states $s_j[d']$ in the run:

$$history_\rho(i, d) = \{(j, d') \mid j < i \wedge d' \leq d_j\} \cup \{(i, d') \mid d' < d\} \qquad \diamondsuit$$

The satisfaction relation $s \models \varphi$ is defined inductively for a state $s \in S$ in a TTS T and a formula φ as follows:

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

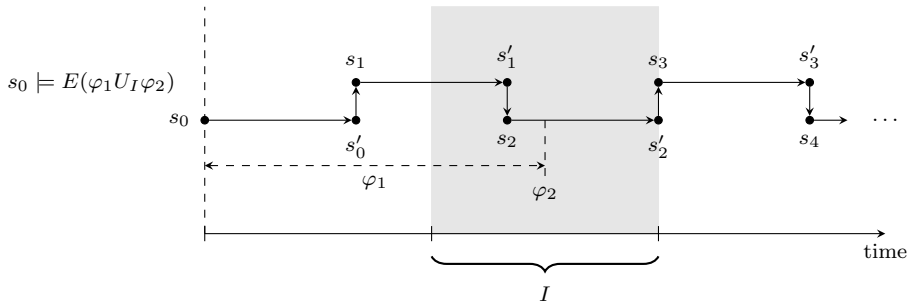


Figure 5.3: Illustration of a run satisfying an until formula.

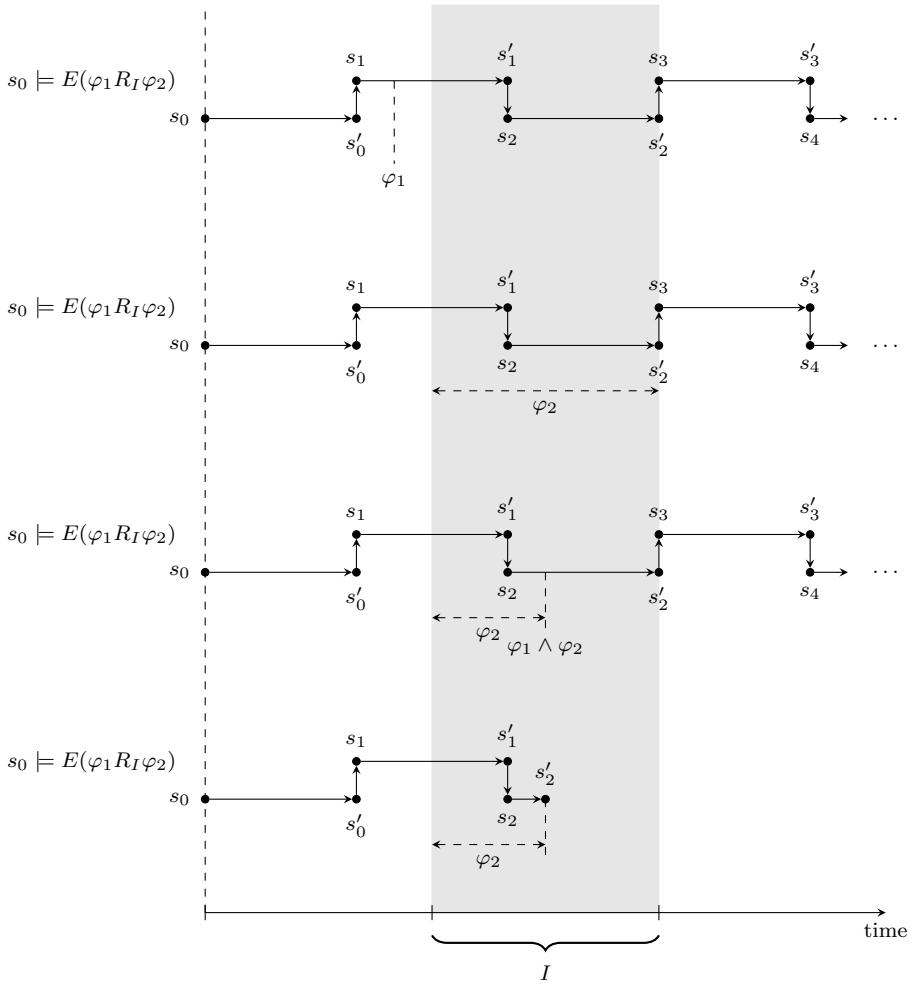


Figure 5.4: Illustration of runs satisfying an release formula. Notice that there are four ways a release formula can be satisfied.

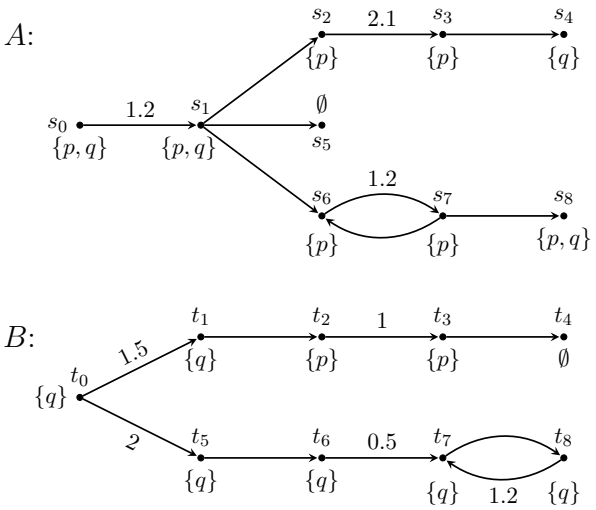


Figure 5.5: Two example TTSs.

Figure 5.3 illustrates the satisfaction of an until formula and Figure 5.4 illustrates the satisfaction of a release formula. The figure illustrates where the two subformulae φ_1 and φ_2 must hold.

In particular, notice that 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. Secondly, φ_2 may not be released, which means that it must hold continuously within the entire interval I. Thirdly, φ_2 can hold continuously in the interval I, until some point in the interval where $\varphi_1 \wedge \varphi_2$ holds, thereby releasing φ_2. Finally, φ_2 can hold continuously in the interval I until the run deadlocks.

As a concrete example consider the two TTSs in Figure 5.5. For both TTSs we have a set of atomic propositions $\mathcal{AP} = \{p, q\}$. Consider the maximal run $\rho = s_0 \xrightarrow{1.2} s_1 \longrightarrow s_2 \xrightarrow{2.1} s_3 \longrightarrow s_4 \xrightarrow{0^{\leq}}$ in A. This run witnesses the property $E(p\, U_{[2,4]}\, q)$ and thus we have that $s_0 \models E(p\, U_{[2,4]}\, q)$. For B, the two maximal runs

$$\rho = t_0 \xrightarrow{1.5} t_1 \longrightarrow t_2 \xrightarrow{1} t_3 \longrightarrow t_4 \xrightarrow{0^{\leq}}$$
$$\rho' = t_0 \xrightarrow{2} t_4 \longrightarrow t_5 \xrightarrow{0.5} t_6 \longrightarrow t_7 \xrightarrow{1.2} t_8 \longrightarrow t_7 \xrightarrow{1.2} \cdots$$

means we have that $t_0 \models A(p\, R_{[2,3]}\, q)$.

Penczek and Pólrola [36] mention that for CTL the operators R and U are dual. However, they do not say if this is the case for TCTL. Thus, we will now show that they are infact still dual for TCTL.

Lemma 25 Let $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ be a TTS and let $s \in S$. Then

$$s \models A(\varphi_1 R_I \varphi_2) \quad \text{if and only if} \quad s \models \neg E(\neg\varphi_1 U_I \neg\varphi_2) \qquad \diamondsuit$$

PROOF ($\Rightarrow$): Assume $s \models A(\varphi_1 R_I \varphi_2)$. We will show that $s \models \neg E(\neg\varphi_1 U_I \neg\varphi_2)$. Assume by contradiction that $s \models E(\neg\varphi_1 U_I \neg\varphi_2)$. This means that there exists a maximal run ρ starting from s such that there exists an $i \geq 0$ and a $d \in \mathbb{R}_{\geq 0}$ such that $valid_\rho(i, d, I)$ is true, $s_i[d] \models \neg\varphi_2$ and for all $(j, d') \in history_\rho(i, d)$ it holds that $s_j[d'] \models \neg\varphi_1$.

This is a contradiction to the assumption that $s \models A(\varphi_1 R_I \varphi_2)$ which by definition means that for all maximal runs ρ' starting from s, all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ it holds that $valid_{\rho'}(i, d, I)$ implies that either $s_i[d] \models \varphi_2$ or there exists a $(j, d') \in history_{\rho'}(i, d)$ such that $s_j[d'] \models \varphi_1$. Thus, it follows that $s \models A(\varphi_1 R_I \varphi_2)$ implies $s \models \neg E(\neg\varphi_1 U_I \neg\varphi_2)$.

($\Leftarrow$): Assume that $s \models \neg E(\neg\varphi_1 U_I \neg\varphi_2)$. This means that for all maximal runs ρ starting from s, all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ it holds that either $valid_\rho(i, d, I)$ is not true or $s_i[d] \not\models \neg\varphi_2$ or there exist a $(j, d') \in history_\rho(i, d)$ such that $s_j[d'] \not\models \neg\varphi_1$. By removing the double negations we see that this matches exactly the definition of $s \models A(\varphi_1 R_I \varphi_2)$ which says that for all maximal runs ρ starting from s, all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ it holds that if $valid_\rho(i, d, I)$ is true, then either $s_i[d] \models \varphi_2$ or there exists a $(j, d') \in history_\rho(i, d)$ such that $s_j[d'] \models \varphi_1$. Thus, we have $\neg E(\neg\varphi_1 U_I \neg\varphi_2)$ implies $s \models A(\varphi_1 R_I \varphi_2)$. ■

Lemma 26 Let $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ be a TTS and let $s \in S$. Then

$$s \models A(\varphi_1 U_I \varphi_2) \quad \text{if and only if} \quad s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2) \qquad \diamondsuit$$

PROOF ($\Rightarrow$): Assume $s \models A(\varphi_1 U_I \varphi_2)$. We will show that $s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2)$. Assume by contradiction that $s \models E(\neg\varphi_1 R_I \neg\varphi_2)$, this means that there exists a maximal run ρ starting from s s.t. for all $i \geq 0$ and for all $d \in \mathbb{R}_{\geq 0}$ if $valid_\rho(i, d, I)$ is true, then it holds that $s_i[d] \models \neg\varphi_2$ or there exist a $(j, d') \in history_\rho(i, d)$ s.t. $s_j[d'] \models \neg\varphi_1$.

This is a contradiction to the assumption that $s \models A(\varphi_1 U_I \varphi_2)$, which by definition means that for every maximal run ρ' starting from s there exists and $i \geq 0$ and a $d \in \mathbb{R}_{\geq 0}$ such that $valid_{\rho'}(i, d, I)$ is true, $s_i[d] \models \varphi_2$ and for all $(j, d') \in history_{\rho'}(i, d)$ it holds that $s_j[d'] \models \varphi_1$. Then it follows that $s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2)$ and thus $s \models A(\varphi_1 U_I \varphi_2)$ implies $s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2)$

($\Leftarrow$): Assume by contraposition that $s \not\models A(\varphi_1 U_I \varphi_2)$. By definition this means that there exists a maximal run ρ starting from s s.t. for all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ either $valid_\rho(i, d, I)$ is not true or $s_i[d] \models \neg\varphi_2$ or there exist a $(j, d') \in history_\rho(i, d)$ s.t. $s_j[d'] \models \neg\varphi_1$. This matches exactly the definition of $s \models E(\neg\varphi_1 R_I \neg\varphi_2)$ which says that there exists a maximal run ρ starting from s s.t. for all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ if $valid_\rho(i, d, I)$ is true then either $s_i[d] \models \neg\varphi_2$ or there exists $(j, d') \in history_\rho(i, d)$ s.t. $s_j[d'] \models \neg\varphi_1$. Thus we have that $s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2)$ implies $s \models A(\varphi_1 U_I \varphi_2)$. ■

Chapter 6

TCTL-Preserving Framework

There are several examples in the literature of translations from one time-dependent framework to another, see e.g. [15, 17, 18, 20, 29]. Usually, a time-dependent system A (e.g. a TAPN) and a formula φ are translated into another time-dependent system B (e.g. an NTA) and a translated formula φ' such that $A \models \varphi$ iff $B \models \varphi'$ or the two systems are shown equivalent up to some equivalence relation (e.g. bisimulation). However, the correctness of each of these translations requires a distinct proof.

Our goal is to generalize this by introducing a framework, in the form of a one-by-many correspondence, which is a relation between the states of TTSs A and B. Intuitively, when the states of A and B are related by a one-by-many correspondence we can simulate one step in A by a number of steps in B. If we can establish such a relation between states of A and B, then that will allow us to conclude that the translation from A to B preserves the full TCTL (or only the safety fragment depending on the requirements fulfilled by the relation).

Our framework can be seen as a generalization of the ideas by Cassez and Roux [18]. While their idea solely concerns a translation from time Petri nets to networks of timed automata, our framework works at the level of timed transition systems, making it independent of the modeling formalisms used. Moreover, the details of TCTL and runs are incomplete in [18]. Our framework handles TCTL in its full generality, as used in state-of-the-art verification tools like UPPAAL [2], making it directly applicable to tool developers.

In the rest of this chapter, we shall use A and B to refer to the original and translated system, respectively.

6.1 Stable Proposition

As mentioned, B is simulating a single step of A by a sequence of steps. Therefore, A and B are only comparable in the states before and after this sequence has been performed. We say that B is stable in these states.

Formally, we introduce an atomic proposition *stable* in the system B as a distinguished atomic proposition. We will refer to states in B which satisfy the proposition

stable as *stable* states and the other states as *intermediate* states.

We will now define three properties that B should possess in order to fit into our framework.

Definition 27 (TTS Properties) A TTS $(S, \to, \mathcal{AP}, \mu)$ where $\mathit{stable} \in \mathcal{AP}$ is said to be

- a *no-delay-in-intermediate-state* TTS if $s \xrightarrow{d}$ for some $d > 0$ implies $s \models \mathit{stable}$ for all $s \in S$,
- a *delay-preserves-stable* TTS if for any state $s \in S$ such that $s \models \mathit{stable}$, if $s \xrightarrow{d} s[d]$ then $s[d] \models \mathit{stable}$ for all $d \in \mathbb{R}_{\geq 0}$, and
- an *eventually-stable* TTS if for any infinite sequence of discrete transitions of length at least one

$$\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow \dots$$

where $s_0 \models \mathit{stable}$, there exists an $i \geq 1$ such that $s_i \models \mathit{stable}$, and for any finite sequence

$$\rho = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \not\to$$

where $s_0 \models \mathit{stable}$, there exists an $i \leq n$ such that $s_i \models \mathit{stable}$. We refer to such sequences as maximal discrete sequences. ♢

Observe that in a *no-delay-in-intermediate-state* TTS, it is only possible to do time delays of 0 in any intermediate state. Further, notice that in an *eventually-stable* TTS, whenever an intermediate state is reached from a stable state, we will eventually return to a stable state.

We will now define a shorthand for the notion of a sequence of intermediate states between two stable states.

Definition 28 Let $(S, \to, \mathcal{AP}, \mu)$ be a TTS such that $\mathit{stable} \in \mathcal{AP}$. For states $s, s' \in S$ we write $s \rightsquigarrow s'$ if

- $s \models \mathit{stable}$ and $s' \models \mathit{stable}$, and
- $s = s_0 \longrightarrow s_1 \xrightarrow{0} s_1 \longrightarrow s_2 \xrightarrow{0} \cdots \xrightarrow{0} s_{n-1} \longrightarrow s_n = s'$ such that $s_j \not\models \mathit{stable}$ for $1 \leq j \leq n-1$. ♢

We have included zero delays in the definition of $\rightsquigarrow$ in order to preserve the alternating nature of the sequence. This allows us to construct alternating runs using $\rightsquigarrow$ which is needed when dealing with maximal runs.

6.2 One-By-Many Correspondence

We will now define the notion of one-by-many correspondence, which is a relation between states in two TTSs.

Definition 29 Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs, where $\mathit{stable} \in \mathcal{AP}_B$. We say that a relation $\mathcal{R} \subseteq S \times T$ is a *one-by-many correspondence* if B is a *no-delay-in-intermediate-state* and *delay-preserves-stable* TTS and there exists a function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ such that whenever $s \,\mathcal{R}\, t$ then

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 $s' \,\mathcal{R}\, t'$,
4. if $s \xrightarrow{d} s[d]$ then $t \xrightarrow{d} t[d]$ and $s[d] \,\mathcal{R}\, t[d]$ for all $d \in \mathbb{R}_{\geq 0}$,
5. if $t \rightsquigarrow t'$ then $s \longrightarrow s'$ and $s' \,\mathcal{R}\, t'$, and
6. if $t \xrightarrow{d} t[d]$ then $s \xrightarrow{d} s[d]$ and $s[d] \,\mathcal{R}\, t[d]$ for all $d \in \mathbb{R}_{\geq 0}$.

If B is moreover an *eventually-stable* TTS, then we say that $\mathcal{R}$ is a *complete one-by-many correspondence* .

We write $s \rrarr t$ (resp. $s \rrarr_c t$) if there exists a relation $\mathcal{R}$ such that $s \,\mathcal{R}\, t$ and $\mathcal{R}$ is a one-by-many correspondence (resp. complete one-by-many correspondence). ♢

Remark 30 Notice that if all states in B are stable, Definition 29 defines a strong timed bisimulation between states in A and B, since all $\rightsquigarrow$ transitions are of length one. ♢

Consider Figure 6.1. The set of propositions for the TTS A is $\mathcal{AP}_A = \{p, q\}$ and the sets of propositions for the TTSs B and C are $\mathcal{AP}_B = \mathcal{AP}_C = \{p, q, \mathit{stable}\}$. We assume that consecutive discrete actions are separated by a 0 time delay. Then $\{(s_0, t_0), (s_1, t_1), (s_2, t_4), (s_3, t_6), (s_2, t_7)\}$ is a complete one-by-many correspondence relating the states of A and B. The relation $\{(s_0, u_0),\ (s_1, u_1), (s_2, u_4), (s_3, u_7)\}$ is a one-by-many correspondence for the systems A and C. Notice that system C is not an *eventually-stable* TTS since both of the maximal sequences

$$\rho = u_1 \longrightarrow u_2 \longrightarrow u_3$$
$$\rho' = u_1 \longrightarrow u_5 \longrightarrow u_6 \longrightarrow u_6 \longrightarrow u_6 \longrightarrow \cdots$$

break this property.

Let us now introduce some notation for relating runs in the two systems.

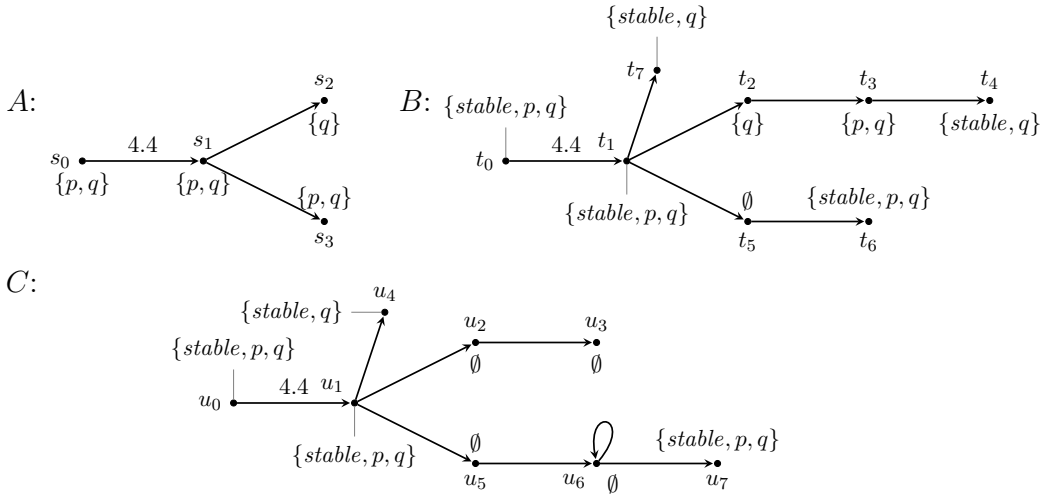


Figure 6.1: Three example TTSs.

Definition 31 Let $A = (S, \to_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \to_B, \mathcal{AP}_B, \mu_B)$ be two TTSs, where $\mathit{stable} \in \mathcal{AP}_B$.

For two finite alternating runs ρ in A and ρ' in B of the form

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

we write $\rho \cong \rho'$ if $s_i[d] \cong t_i[d]$ for all $i \leq n$ and all $d \leq d_i$. ♢

We can generalize this notion of related runs to maximal runs.

Definition 32 Let $A = (S, \to_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \to_B, \mathcal{AP}_B, \mu_B)$ be two TTSs, where $\mathit{stable} \in \mathcal{AP}_B$. For two maximal runs ρ in A and ρ' in B we write $\rho \cong \rho'$ 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 $s_i[d] \cong t_i[d]$ 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,

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

Let us now establish a relationship between finite non-maximal runs in systems A and B.

Lemma 33 Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs, where *stable* $\in \mathcal{AP}_B$. Further let $s_0 \in S$ and $t_0 \in T$ be such that $s_0 \cong t_0$. Then there exists 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 exists 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 $\rho \cong \rho'$. ♢

Proof ($\Rightarrow$): Assume that there exists a 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. By induction on i and using condition 3 and 4 of Definition 29 we can construct a 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.

($\Leftarrow$): Assume that there exists a 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. By induction on i and using condition 5 and 6 of Definition 29 we can construct a 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. ■

Notice that Lemma 33 requires the run in B to have a specific form. Thus, there may exist runs in B for which no related run in A exists.

Let us now introduce a lemma that is a direct consequence of the definition of a complete one-by-many correspondence.

Lemma 34 Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs, where *stable* $\in \mathcal{AP}_B$. Further let $s_0 \in S$ and $t_0 \in T$ be such that $s_0 \cong_c t_0$. Then there exists a maximal run $\rho \in \mathit{Runs}(A, s_0)$ if and only if there exists a maximal run $\rho' \in \mathit{Runs}(B, t_0)$ such that $\rho \cong_c \rho'$. ♢

PROOF ($\Rightarrow$): We shall prove this lemma in two steps, first for infinite maximal runs and then for finite maximal runs. Let

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

be any infinite maximal run in A. Since $s_0 \cong_c t_0$, we have by induction on the index i of states and using condition 3 and 4 of Definition 29 that there exists an infinite maximal run ρ' in B of the form

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

where $s_i[d] \cong_c t_i[d]$ for all $i \geq 0$ and all $d \leq d_i$. Thus, $\rho \cong_c \rho'$.

Now let

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

be any finite maximal run in A where $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$.

Since $s_0 \cong_c t_0$, we shall construct a finite maximal run in B of the form

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

Using Lemma 33 and noticing that the lemma also works with runs ending with a discrete action, it follows that from the prefix of ρ up to and including s_n (referred to as ρ_{prefix}), we can construct the prefix of ρ' up to and including t_n (referred to as ρ'_{prefix}) such that $\rho_{prefix} \cong_c \rho'_{prefix}$.

We shall now handle the final part of ρ' according to the value of δ in ρ.

$\delta = \infty$ In this case, $s_n \xrightarrow{d} s_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$. It follows from condition 4 of Definition 29 that $t_n \xrightarrow{d} t_n[d]$ such that $s_n[d] \cong_c t_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$. Hence $\rho \cong_c \rho'$.

$\delta = d_n^{\leq}$ In this case, $s_n \xrightarrow{d} s_n[d]$ and $s_n[d] \not\longrightarrow$ for all $d \leq d_n$. It follows from condition 4 of Definition 29 that $t_n \xrightarrow{d} t_n[d]$ such that $s_n[d] \cong_c t_n[d]$ and from condition 5 of Definition 29 that $t_n[d] \not\rightsquigarrow$ for all $d \leq d_n$. Moreover, by the *no-delay-in-intermediate-state* and *eventually-stable* properties of B, it follows that $t_n[d] \longrightarrow$ iff $t_n[d] \rightsquigarrow$, hence $t_n[d] \not\longrightarrow$ for all $d \leq d_n$. Finally, since $s_n \xrightarrow{d}$ for $d > d_n$, it follows from condition 6 of Definition 29 that $t_n \xrightarrow{d}$ for $d > d_n$. Hence $\rho \cong_c \rho'$.

$\delta = d_n^{<}$ In this case, $s_n \xrightarrow{d} s_n[d]$ and $s_n[d] \not\longrightarrow$ for all $d < d_n$. It follows from condition 4 of Definition 29 that $t_n \xrightarrow{d} t_n[d]$ such that $s_n[d] \cong_c t_n[d]$ and from condition 5 of Definition 29 that $t_n[d] \not\rightsquigarrow$ for all $d < d_n$. Moreover, by the *no-delay-in-intermediate-state* and *eventually-stable* properties of B, it follows that $t_n[d] \longrightarrow$ iff $t_n[d] \rightsquigarrow$, hence $t_n[d] \not\longrightarrow$ for all $d < d_n$. Finally, since $s_n \xrightarrow{d}$ for $d \geq d_n$, it follows from condition 6 of Definition 29 that $t_n \xrightarrow{d}$ for $d \geq d_n$. Hence $\rho \cong_c \rho'$.

($\Leftarrow$): Assume that there exists a (finite or 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 $i > 0$ (follows from the *eventually-stable* property of B). Moreover, since B is a *no-delay-in-intermediate-state* TTS, it follows that if $t_i \not\models \mathit{stable}$ then $d_i = 0$ for all $i > 0$. Finally, because B is also a *delay-preserves-stable* TTS, it follows that whenever $t_i \models \mathit{stable}$ then $t_i[d_i] \models \mathit{stable}$ for all $i \geq 0$.

These properties in turn imply that 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, following the same strategy as ($\Rightarrow$) case, we can conclude that for any ρ', there exist a maximal run $\rho \in \mathit{Runs}(A, s_0)$ such that $\rho \equiv_c \rho'$. ∎

6.3 Main Theorem

We will now describe how to translate TCTL formulae for A to formulae for B.

Definition 35 Let $\mathcal{AP}_A$ and $\mathcal{AP}_B$ be sets of atomic propositions with $\mathit{stable} \in \mathcal{AP}_B$ and let $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ be a function translating atomic propositions in $\mathcal{AP}_A$ to atomic propositions in $\mathcal{AP}_B$. We define a TCTL translation function $tr : \Phi(\mathcal{AP}_A) \to \Phi(\mathcal{AP}_B)$ in the following manner:

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

As an example consider Figure 6.1. The maximal run $\rho = s_0 \xrightarrow{4.4} s_1 \longrightarrow s_2 \xrightarrow{0^{\leq}}$ in system A witnesses the property $E(pU_{[3,5]}q)$. Similarly, one possible witness in system B for the formula $E((p \vee \neg \mathit{stable})U_{[3,5]}(q \wedge \mathit{stable}))$ is $\rho' = t_0 \xrightarrow{4.4} t_1 \rightsquigarrow t_4 \xrightarrow{0^{\leq}}$.

We are now ready to prove the main theorem of this chapter.

Theorem 36 Let $A = (S, \to_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \to_B, \mathcal{AP}_B, \mu_B)$ be two TTSs such that $\mathit{stable} \in \mathcal{AP}_B$. Further, let $s_0 \in S$ and $t_0 \in T$ be such that $s_0 \equiv_c t_0$. Then for any TCTL formula φ,

$$s_0 \models \varphi \quad \text{if and only if} \quad t_0 \models tr(\varphi) \quad . \qquad \diamondsuit$$

PROOF We shall prove this theorem by structural induction on φ. By Lemma 25 and Lemma 26 it is sufficient to handle the operators $\wp$, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $E(\varphi_1 U_I \varphi_2)$ and $E(\varphi_1 R_I \varphi_2)$.

- $\varphi = \wp$, $\varphi = \neg\varphi_1$, $\varphi = \varphi_1 \wedge \varphi_2$ and $\varphi = \varphi_1 \vee \varphi_2$: Follows trivially from the induction hypothesis and the definition of $\cong_c$.
- $\varphi = E(\varphi_1 U_I \varphi_2)$:

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

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

  where $valid_\rho(n, d, I)$, $s_n[d] \models_A \varphi_2$ and $s_k[d'] \models_A \varphi_1$ for all $(k, d') \in history_\rho(n, d)$. By Lemma 33 there exists a run ρ'_{prefix} in B of the form

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

  such that $\rho_{prefix} \cong_c \rho'_{prefix}$.

  We want to show that $t_0 \models_B E((tr(\varphi_1) \vee \neg stable) U_I (tr(\varphi_2) \wedge stable))$. Because $s_n[d] \cong_c t_n[d]$ and $s_n[d] \models_A \varphi_2$, we have by the induction hypothesis that $t_n[d] \models_B tr(\varphi_2)$ and from condition 1 of Definition 29 we have that $t_n[d] \models_B stable$. 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 $valid_{\rho'}(j, d, I)$ and moreover, for any pair $(k, d') \in history_{\rho'}(j, d)$ either,

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

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

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

  ($\Leftarrow$) : Assume $t_0 \models_B E((tr(\varphi_1) \vee \neg stable) U_I (tr(\varphi_2) \wedge stable))$. Thus, there exists a maximal run ρ' in B that witnesses $tr(\varphi)$. By the fact that B is a *no-delay-in-intermediate-state* and *delay-preserves-stable* TTS there exists a prefix of ρ'

of the form

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

such that $valid_{\rho'}(j, d, I)$, $t_n[d] \models_B tr(\varphi_2) \wedge stable$ and $t_k[d'] \models_B tr(\varphi_1) \vee \neg stable$ for all $(k, d') \in history_\rho(j, d)$ 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 33 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 $\rho_{prefix} \cong_c \rho'_{prefix}$.

We want to show that $s_0 \models_A E(\varphi_1 U_I \varphi_2)$. Because $s_n[d] \cong_c t_n[d]$ and $t_n[d] \models_B tr(\varphi_2) \wedge stable$, 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 $valid_\rho(n, d, I)$. Further, for any pair $(k', d') \in history_\rho(n, d)$, it follows that there exists a $(k, d') \in history_{\rho'}(j, d)$ such that $s_{k'}[d'] \cong_c t_k[d']$. By the induction hypothesis, it follows that $s_{k'}[d'] \models_A \varphi_1$ because $t_k[d'] \models_B tr(\varphi_1)$ and $t_k[d'] \models_B stable$.

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

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

  ($\Rightarrow$) : 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 $valid_\rho(i, d, I)$ is true then either $s_i[d] \models_A \varphi_2$ or there exist a $(k, d') \in history_\rho(i, d)$ s.t. $s_k[d'] \models_A \varphi_1$.

  By Lemma 34 it follows that there exists a maximal run

  $$\rho' = t_0 \xrightarrow{d_0} t_0[d_0] \rightsquigarrow t_1 \xrightarrow{d_1} t_1[d_1] \rightsquigarrow t_2 \xrightarrow{d_2} t_2[d_2] \rightsquigarrow \cdots \tag{6.1}$$

  in B such that $\rho \cong_c \rho'$ (see Definition 32).

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

  - $t_k[d]$ is an intermediate state and then $t_k[d] \models_B \neg stable$, or

– $t_k[d]$ is a stable state. This means that $t_k[d]$ has a distinguished index in Equation (6.1). Let h be the index of $t_k[d]$ in Equation (6.1). It follows from the construction of the ρ' that $s_h[d] \cong_c t_h[d]$. 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 $(\ell', d') \in history_\rho(h, d)$ such that $s_{\ell'}[d'] \models_A \varphi_1$. From the construction of ρ' it follows that there exists a $(\ell, d') \in history_{\rho'}(k, d)$ such that $s_{\ell'}[d'] \cong_c t_\ell[d']$. By the induction hypothesis, it follows that $t_\ell[d'] \models_B tr(\varphi_1) \wedge stable$.

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

($\Leftarrow$) : Assume that $t_0 \models_B E((tr(\varphi_1) \wedge stable) R_I (tr(\varphi_2) \vee \neg stable))$. Since B is a *no-delay-in-intermediate-state*, *delay-preserves-stable* and *eventually-stable* TTS, there exists a maximal run (infinite or finite)

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

in B that witnesses $tr(\varphi)$. This means that for all $i \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$ if $valid_{\rho'}(i, d, I)$ is true then either $t_i[d] \models_B tr(\varphi_2) \vee \neg stable$ or there exists a $(k, d') \in history_{\rho'}(i, d)$ s.t. $t_k[d'] \models_B tr(\varphi_1) \wedge stable$.

By Lemma 34 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 $\rho \cong_c \rho'$.

We want to show that $s_0 \models_A E(\varphi_1 R_I \varphi_2)$. For all $k \geq 0$ and all $d \in \mathbb{R}_{\geq 0}$, we have that $s_k[d] \cong_c t_k[d]$ and whenever $valid_\rho(k, d, I)$ is true then

– Either $t_k[d] \models_B tr(\varphi_2)$ and then by the induction hypothesis we have that $s_k[d] \models_A \varphi_2$, or
– there exists some $(\ell', d') \in history_{\rho'}(j, d)$ where j is the index corresponding to the occurrence of t_k in the alternating sequence which unfolds the $\rightsquigarrow$ steps, such that $t_{\ell'}[d'] \models_B tr(\varphi_1) \wedge stable$. From the construction of ρ, it follows that there exists a pair $(\ell, d') \in history_\rho(k, d)$ such that $s_\ell[d'] \cong_c t_{\ell'}[d']$. 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 in the proof above that for the case of $E(\varphi_1 U_I \varphi_2)$ (dually for the case $A(\varphi_1 R_I \varphi_2)$), we only used Lemma 33 which requires only a one-by-many correspondence. On the other hand, to prove the case of $E(\varphi_1 R_I \varphi_2)$ (dually, $A(\varphi_1 U_I \varphi_2)$) we used the *eventually-stable* property and Lemma 34, which requires a complete one-by-many correspondence. Hence, we may conclude the following corollary.

Corollary 37 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 *stable* $\in \mathcal{AP}_B$. Further, let $s_0 \in S$ and $t_0 \in T$ be such that $s_0 \cong t_0$. Then for any formula φ from the *safety fragment* of TCTL

$$s_0 \models_A \varphi \quad \text{if and only if} \quad t_0 \models_B tr(\varphi) \qquad \diamondsuit$$

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 6.1 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, $s_0 \models_A A(pU_{[3,5]}q)$ but $u_0 \not\models_B tr(A(pU_{[3,5]}q)) = A((p \vee \neg stable)U_{[3,5]}(q \wedge stable))$. 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 $tr(A(pU_{[3,5]}q))$. Moreover, there exists no run ρ'' in A such that $\rho'' \cong \rho$ or $\rho'' \cong \rho'$.

6.4 Overall Methodology

This section will describe what is needed to apply our framework to actual translations. The following list outlines the steps needed.

1. Give an algorithm that for a given system A constructs a system B with the notion of stable states in B.
   (a) Show that B is a *no-delay-in-intermediate-state* and *delay-preserves-stable* TTS (and optionally an *eventually-stable* TTS).
2. Define the proposition translation function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$.
3. Define the relation $\mathcal{R}$ and show that it fulfills condition 1-6 of Definition 29.
4. Conclude that the translation preserves TCTL (or only the safety fragment if $\mathcal{R}$ is only a one-by-many correspondence).

6.5 Applications

In Chapter 8 and Chapter 9, we shall present two novel translations from arbitrary bounded TAPN to NTA and apply our framework to show that they preserve the full TCTL. To show the applicability of our framework, we shall apply our methodology to translations from the literature.

Cassez and Roux [18] give a translation from Timed Petri Nets (TPN) to networks of timed automata with integer variables which preserves TCTL. They create a timed

automaton for every transition, as well as a supervisor automaton. A transition automaton can be in one of three states: enabled, disabled or firing. A global integer array represents the current marking: the integer at index i gives the number of tokens in place p_i. We shall refer to this array as the marking array. Let us briefly describe how the constructed NTA works. Assume that we want to simulate the firing of transition t_i and it is enabled. First the supervisor synchronizes with transition automaton A_i on the channel *pre* (naturally, the interval associated with the transition is checked here). This removes any consumed tokens from the marking array. Next, the supervisor broadcasts on the channel *update*. By construction, all automata will participate in this synchronization. This changes the current location of all transition automata to match the enabledness/disabledness in the TPN (remember the transition automata has a location representing enabled and one representing disabled). Next, automaton A_i can synchronize with the supervisor automaton on the channel *post*. This adds produced tokens to the marking array. Again, the supervisor broadcasts on the *update* channel. As before, this involves all automata and the locations of all transition automata are updated to reflect the enabledness/disabledness of the corresponding transition. Moreover, due to the use of committed locations, no time can elapse during this sequence.

Our framework can be applied to this translation as follows. First, a state is stable whenever the supervisor can synchronize on a *pre* channel. Further, the *no-delay-in-intermediate-state* property is satisfied, since time delays (except 0) are not possible in intermediate locations. Time delays do not change the current location, hence the *delay-preserves-stable* property is satisfied. The authors show that any finite run ends in a stable state. This generalizes to maximal runs (for infinite maximal runs, a stable state appears infinitely often) which means that the *eventually-stable* property is satisfied. Finally, they present a theorem that shows condition $3 - 6$ of our definition, thus it follows that our framework can be applied to show that their translation preserves the full TCTL.

Byg et al. [17] defines a translation from k-bounded TAPN to NTA that preserves the safety fragment of a subset of Computation Tree Logic (CTL). First they translate the k-bounded TAPN to a TAPN of degree 2 (a net where all transitions have two incoming and two outgoing arcs). The idea is that a transition in the original net is simulated by a number of degree 2 transitions in the degree 2 net, such that input tokens are first removed one-by-one after which the tokens are put into the respective output places one-by-one. During the simulation of a transition, delays cannot occur due to the use of invariants. A special place called p_{lock} acts as a mutex, ensuring that the simulations of two transitions cannot interleave. However, extra deadlocks are introduced because it is possible to start the simulation of a transition and get stuck midways (e.g. if a token of an appropriate age is not present in one of the input places). Thus, this translation preserves only safety. A TAPN of degree 2 is simulated in an NTA via handshake synchronizations between two automata (each representing a token). The TAPN of degree 2 is shown strongly timed bisimilar to the NTA.

Our framework also applies to these translations. First, a marking in the de-

gree 2 net is stable whenever a token is in p_{lock}. Because time cannot elapse during the simulation of a transition, it follows that the underlying TTS is *no-delay-in-intermediate-state*. Similarly, the *delay-preserves-stable* property is fulfilled since delays do not change the placement of tokens. We cannot show that the translated system is *eventually-stable*, since extra deadlocks are introduced. The authors prove condition 3-6 of Definition 29 in the paper. Finally. since atomic propositions follow the same style as ours, it follows that our framework can be applied to show that the safety fragment of TCTL is preserved, improving on the result of the paper. Since their second translation preserves strong timed bisimulation, all states are stable and hence, as noted in Remark 30, it follows that we can apply our framework to show that it preserves the full TCTL. In Chapter 9 we extend the translation from k-bounded TAPN to NTA such that it preserves the full TCTL.

In theory, our framework should apply to any translation that has been shown to preserve strong timed bisimulation, as noted in Remark 30 (see e.g. [15, 20, 29] for examples of strong bismulation preserving translations where our framework applies). However, some papers (e.g. [22]) use a non-standard definition of strong timed bisimulation, which means our framework may not apply to their translations.

Cortés et al. [19] presents a translation from 1-safe PRES+ models (essentially, a generalization of time Petri nets) to timed automata. While they do not show any relationship between the two models (though they implicitly claim that formal TCTL model checking is possible on 1-safe PRES+ models using the translation), their translation actually preserves strong timed bisimulation, and hence our framework could be applied to prove the correctness of the translation.

Chapter 7

Networks of Timed Automata with Integer Variables

Timed automata were first introduced by Alur and Dill [5, 6] and have become one of the most well-studied formalisms for the modeling and verification of real-time systems. While there exists a number of tools for real-time model checking and verification of timed automata, this report will focus on the tool UPPAAL [2], as the UPPAAL is a state-of-the-art verification tool and it is used behind-the-scenes in the verification tool TAPAAL [1].

7.1 Clocks

Before we can introduce the syntax and semantics of a timed automaton, we need to define some of the main concepts in the formalism. Because timed automata, as the name implies, allow for modelling of timing constraints, we need a way to model time. This is done via a finite set of real-valued clocks $C = \{c_1, c_2, \ldots\}$ whose elements represent names of clocks.

A clock constraint (or 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 $CG(C)$. For our purposes, invariants follow the same syntax as guards with the one exception that $\bowtie \in \{\leq, <\}$. We denote the set of all clock invariants over a set of clocks C as $CI(C)$.

A (clock) valuation of C is a function $v : C \to \mathbb{R}_{\geq 0}$, which for every clock $c \in C$ returns the value of c. In order to allow time to pass, we define a delay operation as follows. Let v be a valuation and d a non-negative real. We let $v + d$ be the valuation such that $(v + d)(c) = v(c) + d$ for every $c \in C$. Similarly, we need to be able to reset

clocks. For a subset of clocks $r \subseteq C$, we let $v[r]$ be the valuation such that

$$v[r](c) = \begin{cases} 0 & \text{if } c \in r \\ v(c) & \text{otherwise.} \end{cases}$$

A clock valuation v over C satisfies a clock guard $g \in CG(C)$ or a clock invariant $g \in CI_{inv}(C)$ (written $v \models g$) if the clock constraint evaluates to true under the clock assignments of v. Formally, $v \models g$ is defined inductively on the structure of g as follows.

$$\begin{aligned} &v \models \mathit{true} \\ &v \models c_1 \bowtie n \text{ iff } v(c_1) \bowtie n \\ &v \models c_1 - c_2 \bowtie n \text{ iff } v(c_1) - v(c_2) \bowtie n \\ &v \models g_1 \wedge g_2 \text{ iff } v \models g_1 \text{ and } v \models g_2 \end{aligned}$$

where $c_1, c_2 \in C$, $n \in \mathbb{N}_0$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. If v does not satisfy g we write $v \not\models g$ as expected.

7.2 Integer Variables

We will now define the concept of integer variables in timed automata. Let X be a finite set of integer variables. The set of arithmetic variable expressions over X (denoted $VE(X)$) is defined according to the following abstract syntax

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

where $m \in \mathbb{Z}$, $x \in X$ and $\oplus \in \{+, -\}$. A variable guard is a boolean expression defined by the following abstract syntax

$$\varphi_1, \varphi_2 ::= \mathit{true} \mid \mathit{expr}_1 \bowtie \mathit{expr}_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

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

$$x := \mathit{expr}$$

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

Finally, we will define a variable valuation as a total mapping $z : X \longrightarrow \mathbb{Z}$ which given some variable $x \in X$ returns the current value of x. We will extend this mapping to support expressions in $VE(X)$ as follows

$$\begin{aligned} z(m) &= m \quad \text{if } m \in \mathbb{Z} \\ z(e_1 \oplus e_2) &= z(e_1) \oplus z(e_2) \quad \text{if } e_1, e_2 \in VE(X) \end{aligned}$$

where $\oplus \in \{+, -\}$.

A variable valuation z satisfies a variable guard $\varphi \in VG(X)$ (written $z \models \varphi$) if the variable guard evaluates to true under the valuation z. Formally, $z \models \varphi$ is defined inductively on the structure of φ as follows

$$
\begin{aligned}
&z \models \mathit{true} \\
&z \models e_1 \bowtie e_2 \text{ iff } z(e_1) \bowtie z(e_2) \\
&z \models \varphi_1 \wedge \varphi_2 \text{ iff } z \models \varphi_1 \text{ and } z \models \varphi_2 \\
&z \models \varphi_1 \vee \varphi_2 \text{ iff } z \models \varphi_1 \text{ or } z \models \varphi_2
\end{aligned}
$$

where $e_1, e_2 \in VE(X)$ and $\bowtie \in \{<, \leq, =, \geq, >\}$.

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

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

7.3 Timed Automata with Integer Variables

We can now define the notion of a timed automaton.

Definition 38 (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,
- $\longrightarrow \subseteq L \times CG(C) \times VG(X) \times \mathit{Act} \times 2^C \times 2^{VA(X)} \times L$ is a finite set of edges such that whenever $(\ell, g, \varphi, a, r, \mathcal{A}, \ell') \in \longrightarrow$ then $\mathcal{A}$ is a finite non-conflicting set,
- $I_C : L \to CI(C)$ is a function assigning clock invariants to locations,
- $I_X : L \to VG(X)$ is a function assigning variable invariants to locations, and
- ℓ_0 is the initial location.

We write $\ell \xrightarrow{g,\varphi,a,r,\mathcal{A}} \ell'$ instead of $(\ell, g, \varphi, a, r, \mathcal{A}, \ell') \in \longrightarrow$, where ℓ is the source location, g is the clock guard, φ is the variable guard, a is the action, r is the set of clocks to be reset, $\mathcal{A}$ is the set of non-conflicting variable assignments and ℓ' is the target location.$\diamondsuit$

Note for notational convenience, we shall sometimes conjunct the clock invariant and the variable invariant when drawing TA.

Let us now turn our attention to networks of timed automata. A network of timed automata is basically a parallel composition of timed automata. Besides handshake synchronization, UPPAAL also supports broadcast channels. Thus, our network of timed automata will support both handshake synchronization in which two timed automata can synchronize, as well as a notion of broadcast synchronization which allows for more than two timed automata to participate in a synchronization. In order to handle synchronizations, we assume the existence of a finite set of channel names *Chan* (for handshake synchronizations), a finite set of channel names *Broad* (for broadcast synchronizations) and a finite set of ordinary actions, denoted by N, which includes the internal action τ, such that $\mathit{Chan} \cap \mathit{Broad} \cap N = \emptyset$. We then define

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

The intuition is that $c!$ indicates output, i.e. that an automation sends a handshake synchronization request on the channel c, and $c?$ is used for input, i.e. that an automaton is prepared to receive a handshake synchronization request on the channel c. Further, $a!$ indicates broadcasting on channel a and $a?$ indicates that an automaton is prepared to receive a broadcast synchronization request on channel a.

We can now define a network of timed automata.

Definition 39 (Network of Timed Automata) Let $n \in \mathbb{N}$ and let $A_i = (L_i, \mathit{Act}, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$ be timed automata for all $1 \leq i \leq n$, 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 $A = A_1 \parallel A_2 \parallel \ldots \parallel A_n$. ♢

The semantics of a network of timed automata is given in terms of a TTS. 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 $1 \leq i \leq n$ it holds that $z \models I_X^i(\ell_i)$ and $v \models I_C^i(\ell_i)$. In other words, the configuration specifies which location each automaton in the network is in, and the value of every variable and clock in the network (with the requirement, that the invariants of all locations are satisfied by the variable and clock valuations). We will denote the set of all configurations of a given NTA A by $\mathit{Conf}(A)$.

We can now define the precise semantics of networks of timed automata.

Definition 40 Let $A = A_1 \parallel A_2 \parallel \ldots \parallel A_n$ where $A_i = (L_i, \mathit{Act}, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$ for all $1 \leq i \leq n$, be an NTA over a fixed set of actions Act, clocks C and integer variables X. The TTS generated by A, is defined as $T(A) = (S, \longrightarrow, \mathcal{AP}, \mu)$, where

- $S = \mathit{Conf}(A)$,
- 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 $a \in N$ such that there is an edge $\ell_i \xrightarrow{g,\varphi,a,r,\mathcal{A}}_i \ell'_i$ in the i'th automaton such that $v \models g$, $z \models \varphi$, $v' = v[r]$, $z' = z[\mathcal{A}]$, $v' \models I^i_C(\ell'_i) \wedge \bigwedge_{j \neq i} I^j_C(\ell_j)$ and $z' \models I^i_X(\ell'_i) \wedge \bigwedge_{j \neq i} I^j_X(\ell_j)$,
- *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 $\ell_i \xrightarrow{g_i,\varphi_i,a!,r_i,\mathcal{A}_i}_i \ell'_i$ and $\ell_j \xrightarrow{g_j,\varphi_j,a?,r_j,\mathcal{A}_j}_j \ell'_j$ such that $a \in \mathit{Chan}$, $v \models g_i \wedge g_j$, $z \models \varphi_i \wedge \varphi_j$, $v' = v[r_i \cup r_j]$, $z' = (z[\mathcal{A}_i])[\mathcal{A}_j]$, $v' \models I^i_C(\ell'_i) \wedge I^j_C(\ell'_j) \wedge \bigwedge_{k \neq i,j} I^k_C(\ell_k)$ 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 $a \in \mathit{Broad}$ such that
  - there exists an i, $1 \leq i \leq n$ such that there is an edge $\ell_i \xrightarrow{g_i,\varphi_i,a!,r_i,\mathcal{A}_i}_i \ell'_i$ in the i'th automaton where $v \models g_i$ and $z \models \varphi_i$,
  - let $\mathcal{J}$ be the set of all j, $1 \leq j \neq i \leq n$ where there is an edge $\ell_j \xrightarrow{g_j,\varphi_j,a?,r_j,\mathcal{A}_j}_j \ell'_j$ in the j'th automaton such that $v \models g_j$ and $z \models \varphi_j$,
  - for all $j \in \mathcal{J}$ we set ℓ'_j, $\mathcal{A}_j$ and r_j according to the edge $\ell_j \xrightarrow{g_j,\varphi_j,a?,r_j,\mathcal{A}_j}_j \ell'_j$ (note that there may be multiple edges to choose from),
  - for all $j \notin \mathcal{J}$, $1 \leq j \neq i \leq n$ we let $\ell'_j = \ell_j$, $\mathcal{A}_j = \emptyset$ and $r_j = \emptyset$,
  - $z' = (\ldots((\ldots(s[\mathcal{A}_i])[\mathcal{A}_1])[\mathcal{A}_2])\ldots[\mathcal{A}_{i-1}])[\mathcal{A}_{i+1}])\ldots[\mathcal{A}_n]$ 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 $v + d \models \bigwedge_{i=1}^n I_i(\ell_i)$,

- $\mathcal{AP} \stackrel{\text{def}}{=} \{(\#\ell \bowtie m) \mid \ell \in \cup_{i=1}^n L_i, m \in \mathbb{N} \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$, 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 configuration 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 z_0 and v_0 always satisfy the invariants of ℓ^i_0 for all $1 \leq i \leq n$, i.e. $z_0 \models \bigwedge_{i=1}^n I^i_X(\ell^i_0)$ and $v_0 \models \bigwedge_{i=1}^n I^i_C(\ell^i_0)$. $\diamondsuit$

Remark 41 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 following this the assignments of any receivers are evaluated in the order from A_1 to A_n. ♢

7.4 Complexity and Computability

In the following, we will briefly discuss the decidability and complexity of reachability and TCTL model checking of timed automata and timed automata with integer variables. We shall focus on the problems for TA, as an NTA can be viewed as a single product automaton.

Even though the state space is infinite, the reachability problem for a TA was shown decidable and PSPACE-complete in [5, 6]. The basic idea is to construct a region graph, which represents the behaviour of a TA in a finite way. Intuitively, a region is a collection of states with the same behaviour. From this abstraction the reachability problem is reduced to checking reachability in a finite graph.

The construction of the finite region graph also has other applications. In [7], this construction is used to prove the decidability of TCTL model checking for TA. The authors also prove that this problem is PSPACE-complete.

In Definition 38, the TA is extended with integers variables. If we restrict TA to use only bounded integer variables, then it is still possible to construct a finite region graph. This is because the bounded range of integer variables only result in finitely many extra states. However, if use unbounded integers, the region abstraction is no longer capable of representing the state space in a finite way. Thus, both reachability and TCTL model checking become undecidable in this case.

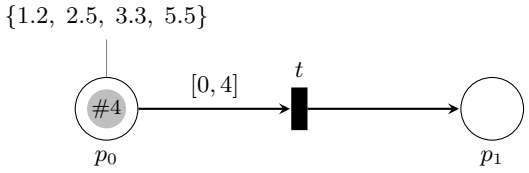
Chapter 8

Broadcast Translation

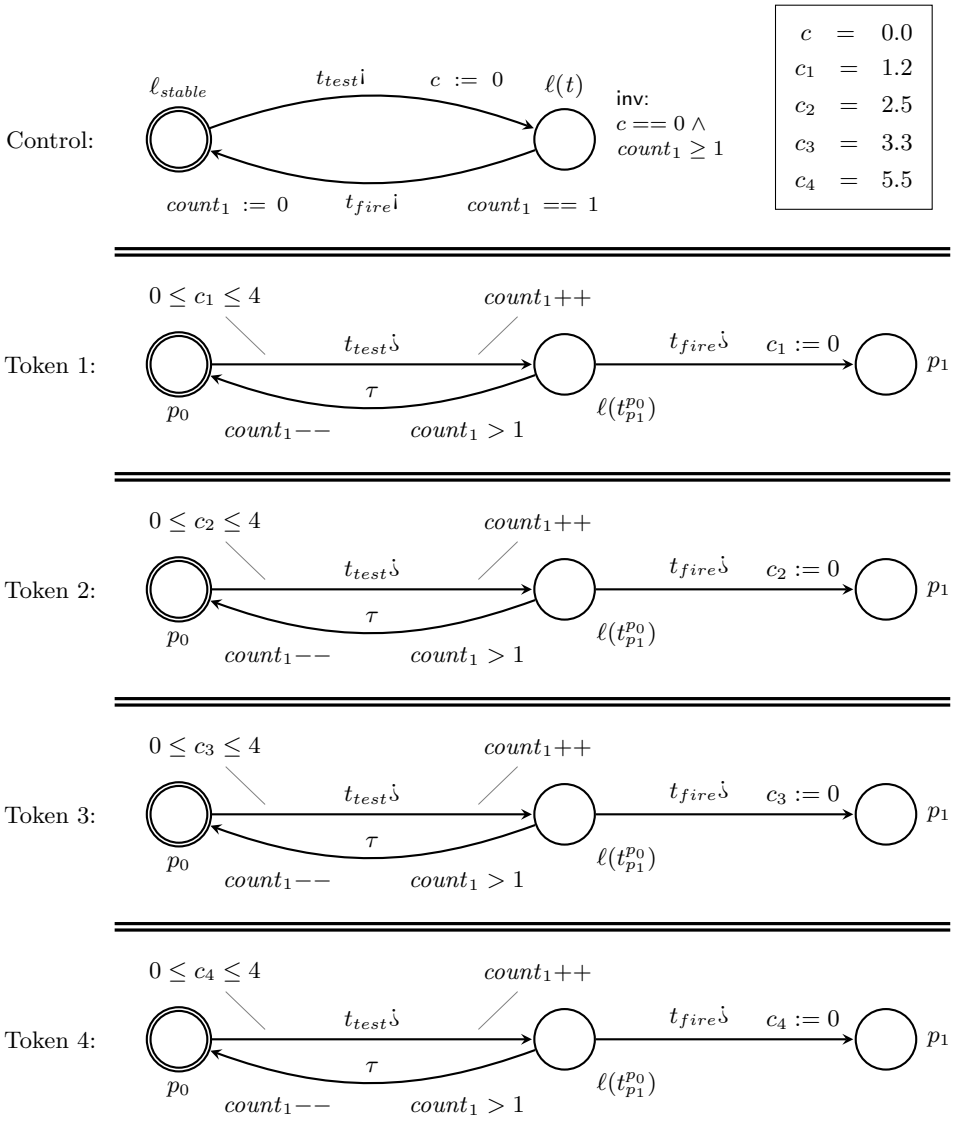
In this section, we will present a translation from k-bounded TAPN to NTA. The basic idea in the translation is to create a TA for each token in the TAPN. Since we cannot dynamically instantiate new TA during the verification process, we need to have a constant number of tokens in the net at all times. Since the translation works for bounded TAPN, we can assume that we know the bound of the TAPN. Thus, if we know the TAPN is k-bounded we construct k TA to simulate the tokens. In each of these TA there is a location for each place in the TAPN. A TA in one of these locations simulates a token in the corresponding place in the TAPN. We refer to these TA as token automata. Further, each TA has a clock which simulates the age of the token. All the token automata have the exact same structure, the only difference being their initial location, which corresponds to the initial marking of the TAPN, and the name of their clock. As there may not be k tokens in the initial marking, an additional place $\ell_{capacity}$ is introduced where automata representing currently unused tokens are waiting. Initially, there are $k - |M_0|$ token automata in $\ell_{capacity}$. When simulating a transition t where $|^\bullet t| > |t^\bullet|$, we can move $|^\bullet t| - |t^\bullet|$ token automata to $\ell_{capacity}$. Similarly, when simulating a transition t where $|^\bullet t| < |t^\bullet|$, we can move $|t^\bullet| - |^\bullet t|$ token automata out of $\ell_{capacity}$. Note that there is only one $\ell_{capacity}$ location in each automaton.

In addition to these token automata, we create a single control automaton. The purpose of this TA is to simulate the firing of transitions and move tokens around by synchronizing with some of the token automata. This TA has a location ℓ_{stable} which acts as a mutex, in the sense that the control automaton moves out of this location once the simulation of a transition begins and only returns once the simulation of the transition ends. Further, each time the control automaton is in this location, the locations and the clock values of the token automata correspond to a marking in the TAPN.

We will begin by demonstrating the translation on an example.



(a) A simple TAPN model.



(b) The NTA resulting from translation of the TAPN in Figure 8.1a. The box in the rightmost side of the picture shows the clock valuation of the NTA. Notice that the values of c_i, $0 \leq i \leq 4$, correspond exactly to the ages of the tokens in p_0.

Figure 8.1: Broadcast translation example.

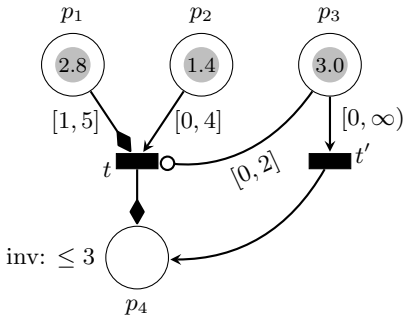
8.1 Example

Figure 8.1a shows a very simple TAPN with a single transition and four tokens, and the translated NTA is shown in Figure 8.1b. The NTA consists of five automata: one control automaton with a distinguished clock c and four token automata, one for each token in the TAPN. Intuitively, the value of c can be understood as the time that has elapsed since the last transition was executed. Notice that for the sake of this example we have refrained from adding the $\ell_{capacity}$ location because it is not needed.

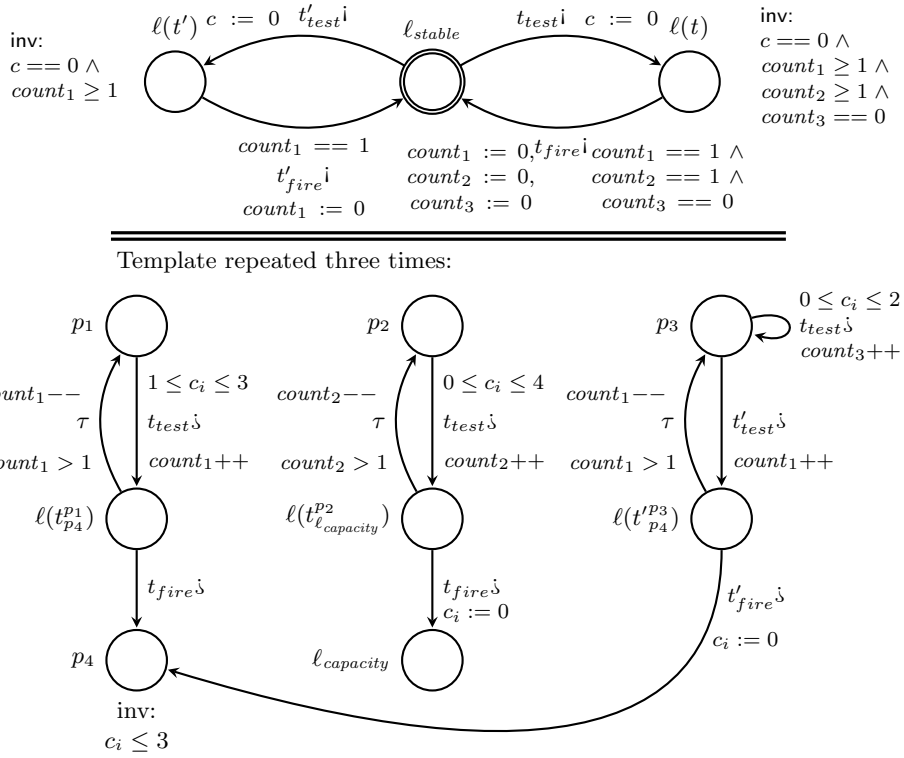
We shall now explain how the translated NTA works. When we want to simulate the firing of a transition t in the NTA, the control automaton will broadcast on the t_{test} channel. Any token automaton whose clock has a value within the interval $[0, 4]$ will be forced to participate in the broadcast. In our case, this means that token automata 1-3 will participate in the broadcast synchronization. Token automaton 4 will not participate since its clock has the value 5.5 and hence the guard prohibits it from participating. We will use integer variables to count the number of token automata which participate in the transition when broadcasting on the t_{test} channel. Because the preset and postset of t has size 1, we only need one counter variable. Thus, when these automata participate in the synchronization, they will move to the intermediate $\ell(t_{p_1}^{p_0})$ location thereby incrementing the counter variable $count_1$, resulting in the value 3. This means that the invariant on $\ell(t)$ is satisfied in the control automaton. In other words, we know that there are enough tokens with appropriate ages in the input places of t to fire it (in this case there are more than needed). Notice that if there were not enough tokens in some of the input places, then the invariant on $\ell(t)$ would not be satisfied when broadcasting on the t_{test} channel and thus this broadcast would not be possible. This is one of the crucial aspects to realize in order to see why this translation preserves liveness properties, as we will show later.

However, because the value of $count_1$ is 3, the control automaton cannot broadcast on the t_{fire} channel, since the guard ensures that this is only possible when exactly one token automaton remains in its intermediate location for each input place. Therefore, we are forced to move two of the token automata back to p_0 via the τ-transitions. Notice that this is only possible as long as $count_1 > 1$, which means one token automaton has to remain in its intermediate location. When all the extra token automata have been moved back to the original location, the control automaton can broadcast on the t_{fire} channel. The token automata remaining in the intermediate locations (in our case, a single automaton) must participate in this synchronization (because they have no guards), and thus the token automata move to the output places, which ends the simulation of firing transition t.

We shall now show how the translation works on a slightly more elaborate example using all of the features of the TAPN model. Figure 8.2a shows a TAPN model that uses transport arcs, invariants and inhibitor arcs. The translated NTA created by our algorithm is shown in Figure 8.2b. Note that Figure 8.2b only shows a template for one token automaton. This template is repeated three times, once for each token. The only difference between the templates is the initial location (p_1, p_2 and p_3 respectively)



(a) A TAPN model.



(b) The NTA resulting from translation of the TAPN in Figure 8.2a. c is a distinguished clock for the control automaton. Note that the lower TA is a template which is repeated three times with different initial locations (p_1, p_2 and p_3 resp.) and with different clock names (c_1, c_2 and c_3 resp.)

Figure 8.2: A more elaborate broadcast translation example.

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, in this case there are two of them. There are some special cases that are worth mentioning in NTA. First of all, consider the inhibitor arc from p_3 to t. We see that this arc is encoded using a self-loop participating in the t_{test} broadcast synchronization. Again, we use a counter variable to count the number of automata that take this transition. We simply encode the requirement of the inhibitor arc (that is, that no token satisfies the interval) in the invariant of $\ell(t)$, namely that $count_3 == 0$. In other words, the control automaton can only broadcast on the t_{test} channel if there are no token automata in the location p_3 that can synchronize on the t_{test} channel, since this is the only way for $count_3$ to maintain the value 0.

A second observation is the guard on the transition from p_1 to $\ell(t_{p_4}^{p_1})$. Specifically, 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. This guard has been changed according to the invariant on p_4 to avoid deadlocks. Since the arc from p_1 to p_4 is a transport arc, we need to test the invariant of the target location when we fire t_{test} also to avoid deadlocks. Thus, we make an intersection of the guard on the transport arc and the invariant on the target place which is then used as the guard on the t_{test} edge.

Remark 42 One would think that it was enough to simply add the invariant from p_4 to the intermediate location $\ell(t_{p_4}^{p_1})$, however, this will result in incorrect behavior, in the sense that it will block t_{test} if e.g. there are two tokens in p_1 with ages 4 and 2, because invariants block the entire broadcast transition if a single automaton with a satisfied guard cannot participate due to its target location invariant. ♢

For this 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 the invariant on $\ell(t)$. Notice that simulating transition t will move the token automaton in p_2 to $\ell_{capacity}$, since $|{}^\bullet t| > |t^\bullet|$. We also see that transitions share counter variables. The variable $count_1$ is used in the simulation of both transitions. This is perfectly acceptable, because it is used in a non-conflicting way, in the sense that we are never simulating t and t' at the same time.

Finally, it is clear that simulating transition t' will move a token automaton from p_3 to p_4. Notice that in this case, there is no guard on the t'_{test} transition in the token automata, because the interval does not restrict the ages in the TAPN model. Further, we do not take the invariant of the target location into account since the arc t' to p_4 is a normal arc and produces a token of age zero which always satisfies any invariant. This concludes our introduction to the translation.

8.2 Translation Algorithm

We will now describe the translation formally. We will adopt the set $Pairing(t)$, introduced by Byg et al. [17], which for a given transition t pairs input and output

places,

$$\begin{aligned} Pairing(t) = \{&(p, I, p', tarc) \mid (p, t, p') \in F_{tarc} \wedge I = c_{tarc}(p, t, p')\} \cup \\ &\{(p_1, I_1, p'_1, normal), \ldots, (p_m, I_m, p'_m, normal) \mid \\ &\{p_1, \ldots, p_\ell\} = \{p \mid (p, t) \in F\}, \{p'_1, \ldots, p'_{\ell'}\} = \{p \mid (t, p) \in F\}, \\ &m = \max(\ell, \ell'), I_i = c(p_i, t) \text{ if } 1 \leq i \leq \ell, I_i = [0, \infty) \text{ if } \ell < i \leq m, \\ &p_i = \ell_{capacity} \text{ if } \ell < i \leq m, p'_i = \ell_{capacity} \text{ if } \ell' < i \leq m\} \end{aligned}$$

The intuition behind $Pairing(t)$ is that it pairs input and output places in order to fix the paths on which tokens will travel when firing t as well as remember the time interval on the input arc and the type of the arc (*normal* for normal arcs and *tarc* for transport arcs). Note that $Pairing(t)$ is chosen a priory (there may be multiple choices for how to pair input and output places of a transition) and pairing a transition is always possible. Because inhibitor arcs do not consume tokens when firing a transition, they are not included in $Pairing(t)$. As a concrete example of this set consider the TAPN in Figure 8.2b. The pairing used in this example for transition t is $Pairing(t) = \{(p_1, [1, 5], p_4, tarc), (p2, [0, 4], \ell_{capacity}, normal)\}$. Note that this is the only possible pairing for this transition.

Let us define the number integer variables needed by the translation for a given TAPN.

Definition 43 Given a TAPN $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$, the number of integer variables needed by the translation is

$$NumVars(N) \stackrel{\text{def}}{=} \max(\{|Pairing(t)| + |\{(p, t) \mid (p, t) \in F_{inhib}\}| \mid t \in T\}) \qquad \diamondsuit$$

To present the algorithm, we shall first introduce some notation. Given an interval I and a clock c, we let $c \in I$ denote a guard that is satisfied if the value of c belongs to the interval I. For example, if $I = [3, 5)$ then $c \in I$ would correspond to the guard $c \geq 3 \wedge c < 5$. We let $c \in \emptyset$ denote the guard *false*, i.e. a guard that is never satisfied.

Algorithm 1 shows how to convert intervals to guards. As mentioned previously, we need to make sure the guard also encodes the invariant of the target location in case the arc is a transport arc to avoid deadlocks. Thus, if the arc in question is a transport arc, the clock value must lie in the intersection of the arc interval and the invariant interval. Otherwise, it must simply satisfy the arc interval. Notice that it is incorrect to create the guard $c \in I \cap \iota(p')$ in the case where *type* indicates a normal arc, since ages of tokens are reset when firing the transition. Restricting the interval to include the invariant in this case could disallow certain tokens to be used to fire the transition in the NTA even though they can be used in the TAPN model.

Algorithm 2 shows how to convert invariants for TAPN to invariants for NTA.

Algorithm 3 gives the translation. Note that for a k-bounded TAPN we will produce an NTA consisting of $k + 1$ components in parallel (one automaton for each of the k tokens and a control automaton). Further, we will assume that the control automaton is always the first automaton in the network.

Algorithm 1: Translation of intervals to guards.

Name: **CreateGuard**$(c, I, p', type)$
Input: A clock c, an arc interval I, a target location p' and an arc type *type*.
Output: A guard g for a timed automaton.
begin
  if $type \neq tarc$ **then**
    $g := c \in I$
  else
    $g := c \in I \cap \iota(p')$
end

Algorithm 2: Translation of invariants for TAPN to invariants for NTA.

Name: **CreateInvariant**(c, I)
Input: A clock c and an interval I of the form $I = [0, a]$ or $I = [0, b)$ for $a \in \mathbb{N}_0, b \in \mathbb{N} \cup \{\infty\}$.
Output: A invariant *inv* for a timed automaton.
begin
  if $I = [0, a]$ **then**
    $inv := c \leq a$
  else
    $inv := c < b$
end

Observe that because at most k token automata can participate in a broadcast synchronization, we have that the value of any counter variable will be less than or equal to k.

8.3 Correctness

For notational convenience, we will in this section sometimes write $\xrightarrow{t}$ for a discrete transition t. To prove the correctness of this translation, we will follow the methodology described in Section 6.4. We let (N, M_0) be a marked k-bounded TAPN and let P_{TA} be the NTA constructed by Algorithm 3 with initial configuration s_0.

We define the *stable* proposition as $(\#\ell_{stable} = 1)$. Recall that $(\#\ell_{stable} = 1)$ is true whenever there is one TA in the ℓ_{stable} location. Thus $(\ell, \ell_1, \ell_2, \ldots, \ell_k, z, v) \models (\#\ell_{stable} = 1)$ if and only if $\ell = \ell_{stable}$.

We will first show that P_{TA} possesses the three properties required by a complete one-by-many correspondence. Recall that $T(P_{TA})$ is the TTS generated by P_{TA}.

Lemma 44 Let P_{TA} be an NTA constructed by Algorithm 3. Then $T(P_{TA})$ is a *no-delay-in-intermediate-state* TTS. ♢

Proof Since all locations in the control automaton except ℓ_{stable} have the invariant $c \leq 0$, it follows that only 0 delays are possible in intermediate states. ■

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

Input: A k-bounded TAPN $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ with initial marking M_0.
Output: An NTA $P_{TA} = A \| A_1 \| A_2 \| \dots \| A_k$ where $A = (L, \mathcal{A}ct, C, X, \longrightarrow, I_C, I_X, \ell_0)$ and $A_i = (L_i, \mathcal{A}ct, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$.

begin
- **for** $i := 1$ *to* k **do** $L_i := P \cup \{\ell_{capacity}\}$
- $L := \{\ell_{stable}\}$
- $\mathcal{A}ct := \{t_{test}\text{¡}, t_{test}\text{¿}, t_{fire}\text{¡}, t_{fire}\text{¿} \mid t \in T\} \cup \{\tau\}$
- $C := \{c, c_1, c_2, \dots, c_k\}$
- $X := \{count_i \mid 1 \le i \le NumVars(N)\}$
- **forall** $t \in T$ **do**
  - $j := 0$
  - $varInv_t := true$; $varGuard_t := true$
  - **while** $|Pairing(t)| > 0$ **do**
    - $j := j + 1$
    - Remove some $(p, I, p', type)$ from $Pairing(t)$
    - **for** $i := 1$ *to* k **do**
      - $L_i := L_i \cup \{\ell(t^p_{p'})\}$
      - $g := CreateGuard(c_i, I, p', type)$
      - Add $p \xrightarrow{g, true, t_{test}\text{¿}, \emptyset, count_j{+}{+}}_i \ell(t^p_{p'})$
      - Add $\ell(t^p_{p'}) \xrightarrow{true, true, t_{fire}\text{¿}, R, \emptyset}_i p'$ 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** $p \in P$ *where* $(p, t) \in F_{inhib}$ **do**
    - $j := j + 1$
    - **for** $i := 1$ *to* k **do** Add $p \xrightarrow{c_i \in c_{inhib}(p,t), true, t_{test}\text{¿}, \emptyset, count_j{+}{+}}_i p$
    - $varInv_t := varInv_t \wedge count_j == 0$
    - $varGuard_t := varGuard_t \wedge count_j == 0$
  - $L := L \cup \{\ell(t)\}$
  - Add $\ell_{stable} \xrightarrow{true, true, t_{test}\text{¡}, \{c\}, \emptyset} \ell(t)$ and $\ell(t) \xrightarrow{true, varGuard_t, t_{fire}\text{¡}, \emptyset, \{count_i := 0 \mid 1 \le i \le j\}} \ell_{stable}$
- **for** $i := 1$ *to* k **do**
  - $I_C^i(p) := \begin{cases} CreateInvariant(c_i, \iota(p)) & \text{if } p \in P \\ true & \text{if } p \in L_i \setminus P \end{cases}$
  - $I_X^i(p) := true$ for $p \in L_i$
- $I_C(p) := \begin{cases} true & \text{if } p = \ell_{stable} \\ c \le 0 & \text{if } p \in L \setminus \{\ell_{stable}\} \end{cases}$
- $I_X(p) := \begin{cases} varInv_t & \text{if } p = \ell(t) \text{ for } t \in T \\ true & \text{if } L \setminus \{\ell(t) \mid t \in T\} \end{cases}$
- $i := 0$; **forall** $p \in P$ **do** **forall** $Token \in M_0(p)$ **do** $\ell_0^i := p; i := i + 1$
- **for** $i := |M_0| + 1$ *to* k **do** $\ell_0^i := \ell_{capacity}$
- $\ell_0 := \ell_{stable}$

end

Lemma 45 Let P_{TA} be an NTA constructed by Algorithm 3. Then $T(P_{TA})$ is a *delay-preserves-stable* TTS. ♢

PROOF Since time delays do not change the current location of any automaton, it follows that any time delay from a stable configuration will result in another stable configuration. Thus, $T(P_{TA})$ is a *delay-preserves-stable* TTS. ■

Lemma 46 Let P_{TA} be an NTA constructed by Algorithm 3. Then $T(P_{TA})$ is an *eventually-stable* TTS. ♢

PROOF We must show that for any (finite or infinite) maximal discrete sequence of length at least 1

$$\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \cdots$$

where $s_0 \models (\#\ell_{stable} = 1)$, there exists an $i \geq 1$ such that $s_i \models (\#\ell_{stable} = 1)$.

Because s_0 is stable, it follows that the control automaton is in location ℓ_{stable} in s_0. We will show that by construction any maximal discrete sequence starting in s_0 contains a prefix of the form

$$s_0 \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{t_{fire}} s_n$$

such that $n \leq k+1$ and $s_n \models (\#\ell_{stable} = 1)$.

If any discrete transition is enabled in s_0 then by construction, it is a broadcast transition on some channel t_{test}. Assume that $s_0 \xrightarrow{t_{test}} s_1$. In s_1 there are only two possibilities for discrete transitions, which are mutually exclusive and by construction one of them is always possible (due to the use of integer guards and invariants). Either there is an τ-transition enabled in some token automaton in case there are more than one token automata currently in the same intermediate location $\ell(t^p_{p'})$ or a broadcast on the t_{fire} channel is enabled when there is exactly one token automaton in each of the required $\ell(t^p_{p'})$ locations.

In the first case, the only possibility is to keep taking τ-transitions, which move the extra token automata back to their original locations. This must continue until some configuration s_{n-1} is reached where there are no more τ-transitions enabled. By construction, this will always happen. It follows by construction that t_{fire} will be the only enabled discrete transition in s_{n-1}. In the worst case, all k TA participated in the t_{test} broadcast synchronization and only a single TA is required to synchronize on the t_{fire} broadcast channel. Thus, we must move $k-1$ TA back to their original location.

In the second case, only the t_{fire} broadcast synchronization is enabled and we have by construction of P_{TA} that the invariants on the target location of all t_{fire} broadcast receivers will be satisfied. Since synchronizing on the t_{fire} channel brings the control automaton back to ℓ_{stable}, it follows that $s_n \models (\#\ell_{stable} = 1)$. Hence, $T(P_{TA})$ is an *eventually-stable* TTS. ■

Remark 47 By Lemma 44 and Lemma 46 we also have that for any *stable* configuration s with an enabled discrete transition, it holds that $s \rightsquigarrow s'$ for some configuration s'. ♢

We now define the proposition translation function tr_p. An atomic proposition for the TAPN always have the form $(p \bowtie n)$ where p is a place, $n \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Such a proposition is translated into $(\#p \bowtie n)$.

Let us now define a correspondence relation $\mathcal{R}$ between markings and configurations. The goal is to show that this relation is a complete one-by-many correspondence. Let $M = \{(p_1, r_1), (p_2, r_2), \ldots, (p_n, r_n)\}$ be a marking of N where $n \leq k$ and (p_i, r_i) is a token located in the place p_i with age $r_i \in \mathbb{R}_{\geq 0}$. Further, let $s = (\ell, \ell_1, \ell_2, \ldots, \ell_k, z, v)$ be a configuration of P_{TA}, where v is a clock valuation over the clocks $\{c, c_1, c_2, \ldots, c_k\}$ and z is a variable valuation over the variables $\{count_i \mid 1 \leq i \leq NumVars(N)\}$. We write $M \mathcal{R} s$ if there exists an injection $h : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, k\}$ such that $\ell = \ell_{stable}$, $\ell_{h(i)} = p_i$ and $v(c_{h(i)}) = r_i$ for $1 \leq i \leq n$, and $\ell_j = \ell_{capacity}$ for all $j \in \{1, 2, \ldots, k\} \setminus range(h)$. Intuitively, whenever $M \mathcal{R} s$, it means that for every token in M, there is a distinct TA location and clock valuation in the configuration s matches exactly that token and the remaining $k-n$ token automata are in the $\ell_{capacity}$ location.

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

Theorem 48 $\mathcal{R}$ is a complete one-by-many correspondence. ♢

Proof Let (N, M_0) be a marked k-bounded TAPN and P_{TA} be the NTA generated by Algorithm 3. We will show that $\mathcal{R}$ satisfies all requirements of Definition 29 and thus is a complete one-by-many correspondence. From Lemma 44, Lemma 45 and Lemma 46 it follows that $T(P_{TA})$ is a *no-delay-in-intermediate-state*, *delay-preserves-stable* and *eventually-stable* TTS.

Let M be a marking and let s be a configuration, such that $M \mathcal{R} s$. We shall now prove that $\mathcal{R}$ satisfies conditions 1-6 of Definition 29 on page 35.

1. $s \models (\#\ell_{stable} = 1)$ follows from the definition of $\mathcal{R}$.
2. From the definition of $\mathcal{R}$ it follows that $M \models \wp$ iff $s \models tr_p(\wp)$.
3. Assume $M \longrightarrow M'$. This means that there exists a transition t such that $M \xrightarrow{t} M'$. We must show that $s \rightsquigarrow s'$ such that $M' \mathcal{R} s'$. We will start by showing 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 on the sender transition, so only the invariant on $\ell(t)$ may block the t_{test} sender. 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 locations $\ell(t^p_{p'})$. Thus, any receiver with a satisfied guard can participate in the broadcast synchronization. By the fact that $M \mathcal{R} s$, it follows that enough TA will participate in the broadcast synchronization to satisfy the invariant on $\ell(t)$. Thus, $s \xrightarrow{t_{test}}$.

   Since $s \models (\#\ell_{stable} = 1)$ and $s \xrightarrow{t_{test}}$ we have by Lemma 46 that

$$s \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_n = s'$$

such that $s_i \not\models (\#\ell_{stable} = 1)$ for $1 \leq i < n$ and $s_n \models (\#\ell_{stable} = 1)$. By construction, this sequence has the form

$$s \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{t_{fire}} s_n$$

where $n \leq k+1$. Thus, since $M \xrightarrow{t} M'$ we can use the sequence above and Lemma 44 to get the sequence $s \rightsquigarrow s'$. Observe that if $|{}^\bullet t| > |t^\bullet|$ then $|{}^\bullet t| - |t^\bullet|$ token automata will have moved to $\ell_{capacity}$ after simulating t from s and if $|{}^\bullet t| < |t^\bullet|$ then $|t^\bullet| - |{}^\bullet t|$ token automata will have moved out of $\ell_{capacity}$ after simulating t from s. Since $M \mathcal{R} s$, we get $M' \mathcal{R} s'$ by matching the changes in M when firing t to the changes in s when executing the sequence above.

4.,6. Time delays can only be restricted by invariants. By Lemma 44 and the fact that the invariant on place p in N is carried over to location p in P_{TA}, we have that it is always possible to do the same time delays in M and s. By Lemma 45, it follows clearly that if $M \xrightarrow{d} M'$ then $s \xrightarrow{d} s'$ such that $M' \mathcal{R} s'$ and vice versa.

5. Assume $s \rightsquigarrow s'$. This implies that

$$s \longrightarrow s_1 \xrightarrow{0} s_1 \longrightarrow s_2 \xrightarrow{0} s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{0} s_{n-1} \longrightarrow s_n = s'$$

such that $s_i \not\models (\#\ell_{stable} = 1)$ for $1 \leq i < n$ and $s_n \models (\#\ell_{stable} = 1)$. By construction of P_{TA}, we have that $n \leq k$ and 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 satisfiable. By the construction of P_{TA}, 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 guard on t_{test}. This is exactly what is needed to fire t from M, and because $M \mathcal{R} s$, then $M \xrightarrow{t} M'$.

If any token automaton moves out of $\ell_{capacity}$ when $s \rightsquigarrow s'$ then additional tokens will be produced when firing transition t from M and if any token automaton moves into $\ell_{capacity}$ when $s \rightsquigarrow s'$ then t will consume more tokens than it produces when fired from M. Since $M \mathcal{R} s$, we clearly get $M' \mathcal{R} s'$ by matching the changes in s when simulating t to the changes in M when firing t. ∎

Since $\mathcal{R}$ is a complete one-by-many correspondence, Theorem 36 allows us to conclude the following corollary.

Corollary 49 Let (N, M_0) be a k-bounded TAPN and let P_{TA} be the NTA constructed by Algorithm 3. For any TCTL formula φ we have that

$$N \models \varphi \quad \text{if and only if} \quad P_{TA} \models tr(\varphi) \qquad \diamondsuit$$

As an example of this corollary, consider the TAPN N and translated NTA P_{TA} in Figure 8.2b on page 56. The TAPN model satisfies the query $\varphi = E((p_4 = 0)\, U_{[0,\infty)}\, (p_4 = 1))$ which asks whether it is possible to put a token in p_4. Let $M = \{(p_1, 2.8), (p_2, 1.4), (p_3, 3.0)\}$ denote the marking illustrated in the figure. The maximal run

$$\rho = M \xrightarrow{0.5} M_1 \xrightarrow{t} M_2 \xrightarrow{1} M_3 \xrightarrow{t'} M_4 \xrightarrow{2^{\leq}}$$

in $T(N)$ witnesses the property φ. Thus, we have $M \models E((p_4 = 0)\, U_{[0,\infty)}\, (p_4 = 1))$. In P_{TA}, we can construct a witness for the translated query $tr(\varphi) = E(((\#p_4 = 0) \vee \neg(\#\ell_{stable} = 1))U_{[0,\infty)}((\#p_4 = 1) \wedge (\#\ell_{stable} = 1)))$. Let s be a configuration such that $M \mathcal{R}\, s$. The maximal run

$$\rho' = s \xrightarrow{0.5} s_1 \xrightarrow{t_{test}} s_2 \xrightarrow{0} s_3 \xrightarrow{t_{fire}} s_4 \xrightarrow{1} s_5 \xrightarrow{t'_{test}} s_6 \xrightarrow{0} s_7 \xrightarrow{t'_{fire}} s_8 \xrightarrow{2^{\leq}}$$

in $T(P_{TA})$ witnesses $tr(\varphi)$. Thus, we have

$$s \models E(((\#p_4 = 0) \vee \neg(\#\ell_{stable} = 1))U_{[0,\infty)}((\#p_4 = 1) \wedge (\#\ell_{stable} = 1))).$$

Chapter 9

Degree 2 Broadcast Translation

We will now present an alternative translation using broadcasts. Byg et al. [17] presented a translation from TAPN with invariants and transport arcs to NTA. However, their translation preserves only the safety fragment of CTL. We incorporate the idea of using broadcast synchronizations into their algorithm to achieve a translation that preserves the full TCTL and also handles inhibitor arcs. It turns out that on some models, this translation is more efficient than the translation in the previous section, as will be demonstrated in Chapter 12.

We will again have a timed automaton for each token in the TAPN and one control automaton. Thus, a k-bounded TAPN will be translated to an NTA with $k+1$ parallel components. Token placement and age will be simulated in the same manner as in the translation in Chapter 8 and since the TAPN may not always contain k tokens in its current marking, we will again use the location $\ell_{capacity}$ where automata representing currently unused tokens are waiting.

The basic idea is still to test if a transition t is enabled by broadcasting on the channel t_{test}!, in order to ensure that we cannot get stuck during the simulation of t. However, this time the token automata will stay in the same location instead of moving to intermediate locations. Because of this, we do not need to move any "extra" tokens back via τ-transitions after executing t_{test}. Thus, we only use t_{test} as a mechanism to count the number of tokens (using integer variables, as before) of appropriate age from each input place of the transition . To simulate the production of tokens at the output places, we replace the t_{fire} broadcast with a sequence of handshake synchronizations. We still use the intermediate locations while executing the handshake synchronizations, however, the intermediate locations will now function more as temporary holding locations while tokens are moved out of the input places. Because we remove input tokens one-by-one in a sequence, these holding locations are needed to ensure that a produced token is not consumed by the next step in the sequence.

9.1 Example

Let us illustrate the translation on an example. We will demonstrate how the translation works on the TAPN from the second example of Chapter 8. The TAPN is reproduced in Figure 9.1a and the NTA resulting from the translation is presented in Figure 9.1b.

Note that the token automata are presented as a template which should be repeated three times with different initial locations (p_1, p_2 and p_3, respectively) and different clock names (c_1, c_2 and c_3, respectively). We will now explain how the NTA simulates the TAPN. Consider the transition t which has an inhibitor arc, a transport arc and a normal arc as input. We start simulating t when the control automaton broadcasts on the t_{test}i channel. As in the previous translation, this is only possible when there are tokens of appropriate ages in all input places due to the invariant of $\ell(t)$ (i.e. zero tokens of appropriate age in any input place with an inhibitor arc to t and at least one token of appropriate age otherwise). However, as mentioned above, we do not move the token automata to intermediate locations when executing the t_{test} broadcast synchronization. Instead, we use self-loops that increment a counter as illustrated in Figure 9.1b. As before, we intersect the guard and the target location invariant for transport arcs to avoid additional deadlocks.

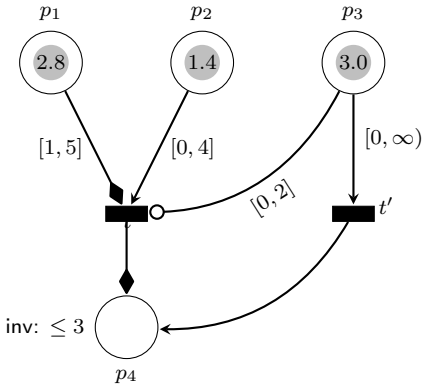
To complete the simulation of t we need to move one token of appropriate age from each input place in ${}^\bullet t$ to an output place. In this case, t consumes two tokens and produces one so we will move the token automaton in p_2 to $\ell_{capacity}$. Moving tokens to output places is done by executing a sequence of handshake synchronizations on the channels t^1_{in}, t^2, t^1_{out}. We start by moving a token from p_1 to the intermediate location $\ell(t^{p_1}_{p_3})$ by synchronizing on the t^1_{in} channel. Following this we will move a token from p_2 directly to the output location $\ell_{capacity}$ (because this is the last input token to be moved) by synchronizing on t^2 and finally we move the token in the intermediate location $\ell(t^{p_1}_{p_3})$ to the output place p_3 by synchronizing on t^1_{out}, whereby the control automaton also returns to ℓ_{stable}.

9.2 Translation Algorithm

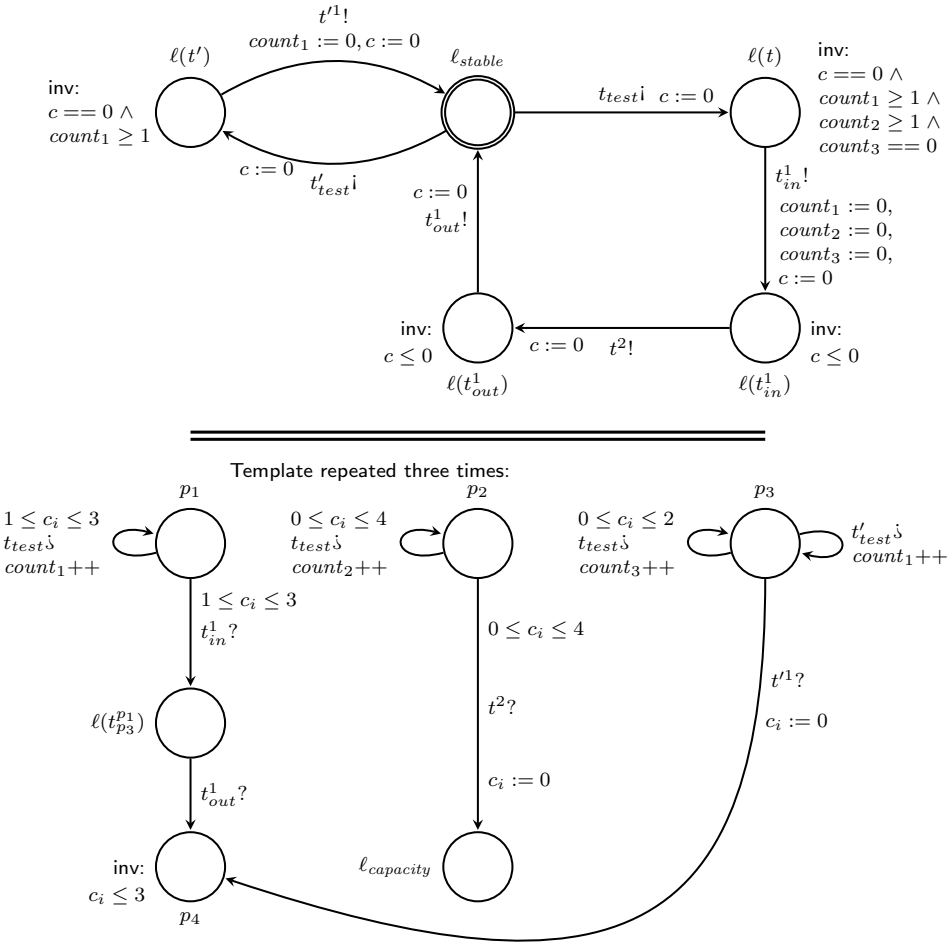
We will now describe the translation formally. Recall the definition of the set $Pairing(t)$ from Section 8.2 which we will use in the algorithm here as well.

$$\begin{aligned} Pairing(t) = \{&(p, I, p', tarc) \mid (p, t, p') \in F_{tarc} \wedge I = c_{tarc}(p, t, p')\} \cup \\ &\{(p_1, I_1, p'_1, normal), \ldots, (p_m, I_m, p'_m, normal) \mid \\ &\{p_1, \ldots, p_\ell\} = \{p \mid (p, t) \in F\}, \{p'_1, \ldots, p'_{\ell'}\} = \{p \mid (t, p) \in F\} \\ &m = \max(\ell, \ell'), I_i = c(p_i, t) \text{ if } 1 \leq i \leq \ell \text{ else } I_i = [0, \infty), \\ &p_i = \ell_{capacity} \text{ if } \ell < i \leq m, p'_i = \ell_{capacity} \text{ if } \ell' < i \leq m\} \end{aligned}$$

Further, we will also use the $CreateGuard(c_i, I, p', type)$ function from Algorithm 1 and the $CreateInvariant(c_i, \iota(p))$ function from Algorithm 2. We shall also reuse



(a) A simple TAPN model.



(b) The NTA resulting from translation of the TAPN in Figure 9.1a. Note that the token automata are illustrated by a template which is repeated three times with different initial locations (p_1, p_2 and p_3 resp.) and different clock names (c_1, c_2 and c_3 resp.).

Figure 9.1: Degree 2 broadcast translation example.

$NumVars(N)$ from Definition 43 of the previous chapter. Finally, for a transition t, we define $max(t) = max(|{}^\bullet t|, |t^\bullet|)$. This will allow us to calculate how many t^i_{in} (resp. t^i_{out}) channels we need.

The degree 2 translation is given formally in Algorithm 4.

9.3 Correctness

We shall now prove the correctness of the translation. For notational convenience, we will sometimes write $\xrightarrow{t}$ for a discrete transition t as before. We will also follow the methodology outlined in Section 6.4. Again, we define the *stable* proposition as $(\#\ell_{stable} = 1)$. Recall that $(\#\ell_{stable} = 1)$ is true whenever there is one TA in the ℓ_{stable} location.

We will first show that P_{TA} possesses the three properties required by a complete one-by-many correspondence.

Lemma 50 Let P_{TA} be an NTA constructed by Algorithm 4. Then $T(P_{TA})$ is a *no-delay-in-intermediate-state* TTS. ♢

Proof Similar to the proof for Lemma 44. ■

Lemma 51 Let P_{TA} be an NTA constructed by Algorithm 4. Then $T(P_{TA})$ is a *delay-preserves-stable* TTS. ♢

Proof Similar to the proof for Lemma 45. ■

Lemma 52 Let P_{TA} be an NTA constructed by Algorithm 4. Then $T(P_{TA})$ is an *eventually-stable* TTS. ♢

Proof We must show that for any (finite or infinite) maximal discrete sequence of length at least one

$$\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \cdots$$

where $s_0 \models (\#\ell_{stable} = 1)$, there exists an $i \geq 1$ such that $s_i \models (\#\ell_{stable} = 1)$.

Because s_0 is stable, it follows that the control automaton is in location ℓ_{stable} in s_0. We will show that by construction any maximal discrete sequence starting in s_0 contains a prefix of the form (we omit naming the intermediate states)

$$s_0 \xrightarrow{t_{test}} \xrightarrow{t^1_{in}} \xrightarrow{t^2_{in}} \cdots \xrightarrow{t^{\max(t)-1}_{in}} \xrightarrow{t^{\max(t)}} \xrightarrow{t^{\max(t)-1}_{out}} \xrightarrow{t^{\max(t)-2}_{out}} \cdots \xrightarrow{t^1_{out}} s'$$

such that $s' \models (\#\ell_{stable} = 1)$.

If any discrete transition is enabled in s_0 then by construction, it is a broadcast transition on some channel t_{test}. Assume that $s_0 \xrightarrow{t_{test}}$. This implies that the invariant in location $\ell(t)$ is satisfied. This invariant uses the counter variables to encode exactly

Algorithm 4: Translation from k-bounded TAPN to NTA.

Input: A k-bounded TAPN $N = (P, T, F, c, F_{tarc}, c_{tarc}, F_{inhib}, c_{inhib}, \iota)$ with initial marking M_0.
Output: An NTA $P_{TA} = A \| A_1 \| A_2 \| \dots \| A_k$ where $A = (L, \mathcal{A}ct, C, X, \longrightarrow, I_C, I_X, \ell_0)$ and $A_i = (L_i, \mathcal{A}ct, C, X, \longrightarrow_i, I_C^i, I_X^i, \ell_0^i)$.

begin
- **for** $i := 1$ *to* k **do** $L_i := P \cup \{\ell_{capacity}\}$
- $L := \{\ell_{stable}\} \cup \{\ell(t) \mid t \in T\} \cup \{\ell(t_{in}^i), \ell(t_{out}^i) \mid t \in T, 1 \le i < \max(t)\}$
- $\mathcal{A}ct := \{t_{test}\dot{\imath}, t_{test}\dot{\jmath}, t^{\max(t)} \mid t \in T\} \cup \{t_{in}^i, t_{out}^i \mid t \in T,\ 1 \le i < \max(t)\} \cup \{\tau\}$
- $C := \{c, c_1, c_2, \dots, c_k\}$; $X := \{count_i \mid 1 \le i \le NumVars(N)\}$
- **forall** $t \in T$ **do**
  - $j := 0$; $m := 0$; $varInv_t := true$
  - **while** $|Pairing(t)| > 1$ **do**
    - $j := j + 1$; $m := m + 1$; $varInv_t := varInv_t \wedge count_j \ge 1$
    - Remove some $(p, I, p', type)$ from $Pairing(t)$
    - **for** $i := 1$ *to* k **do**
      - $L_i := L_i \cup \{\ell(t_{p'}^p)\}$; $g := CreateGuard(c_i, I, p', type)$
      - Add $p \xrightarrow{g, true, t_{test}\dot{\jmath}, \emptyset, count_j{+}{+}}_i p$
      - Add $p \xrightarrow{g, true, t_{in}^m ?, R, \emptyset}_i \ell(t_{p'}^p)$ s.t. $R = \emptyset$ if $type = tarc$ else $R = \{c_i\}$
      - Add $\ell(t_{p'}^p) \xrightarrow{true, true, t_{out}^m ?, R, \emptyset}_i p'$ s.t. $R = \emptyset$ if $type = tarc$ else $R = \{c_i\}$
  - $j := j + 1$; $varInv_t := varInv_t \wedge count_j \ge 1$
  - Let $\{(p, I, p', type)\} := Pairing(t)$; $g := CreateGuard(c_i, I, p', type)$
  - **for** $i := 1$ *to* k **do**
    - Add $p \xrightarrow{g, true, t_{test}\dot{\jmath}, \emptyset, count_j{+}{+}}_i p$
    - Add $p \xrightarrow{g, true, t^{\max(t)}, R, \emptyset}_i p'$ s.t. $R = \emptyset$ if $type = tarc$ else $R = \{c_i\}$
  - **forall** $p \in P$ *where* $(p, t) \in F_{inhib}$ **do**
    - $j := j + 1$; $varInv_t := varInv_t \wedge count_j == 0$
    - **for** $i := 1$ *to* k **do** Add $p \xrightarrow{c_i \in c_{inhib}(p,t), true, t_{test}\dot{\jmath}, \emptyset, count_j{+}{+}}_i p$
  - Add $\ell_{stable} \xrightarrow{true, true, t_{test}\dot{\imath}, \{c\}, \emptyset} \ell(t)$
  - **if** $\max(t) = 1$ **then**
    - Add $\ell(t) \xrightarrow{true, true, t^1 !, \{c\}, \mathcal{A}} \ell_{stable}$ s.t. $\mathcal{A} = \{count_i := 0 \mid 1 \le i \le j\}$
  - **else**
    - Add $\ell(t) \xrightarrow{true, true, t_{in}^1 !, \{c\}, \mathcal{A}} \ell(t_{in}^1)$ s.t. $\mathcal{A} = \{count_i := 0 \mid 1 \le i \le j\}$
    - Add $\ell(t_{in}^{i-1}) \xrightarrow{true, true, t_{in}^{i-1} !, \{c\}, \emptyset} \ell(t_{in}^i)$ for all $2 \le i < \max(t)$
    - Add $\ell(t_{in}^{\max(t)-1}) \xrightarrow{true, true, t^{\max(t)} !, \{c\}, \emptyset} \ell(t_{out}^{\max(t)-1})$
    - Add $\ell(t_{out}^i) \xrightarrow{true, true, t_{out}^i !, \{c\}, \emptyset} \ell(t_{out}^{i-1})$ for all $2 \le i < \max(t)$
    - Add $\ell(t_{out}^1) \xrightarrow{true, true, t_{out}^1 !, \{c\}, \emptyset} \ell_{stable}$
- **for** $i := 1$ *to* k **do**
  - $I_C^i(p) := \begin{cases} CreateInvariant(c_i, \iota(p)) & \text{if } p \in P \\ true & \text{if } p \in L_i \setminus P \end{cases}$; $\quad I_X^i(p) := true$ for $p \in L_i$
- $I_C(p) := \begin{cases} true & \text{if } p = \ell_{stable} \\ c \le 0 & \text{if } p \in L \setminus \{\ell_{stable}\} \end{cases}$; $\quad I_X(p) := \begin{cases} varInv_t & \text{if } p = \ell(t) \text{ for } t \in T \\ true & \text{if } p \in L \setminus \{\ell(t) \mid t \in T\} \end{cases}$
- $i := 0$; **forall** $p \in P$ **do** **forall** $Token \in M_0(p)$ **do** $\ell_0^i := p; i := i + 1$
- $\ell_0 := \ell_{stable}$; **for** $i := |M_0| + 1$ *to* k **do** $\ell_0^i := \ell_{capacity}$

end

the requirements needed for the control automaton to return to ℓ_{stable}. In other words, the invariant being satisfied implies that for every channel

$$t^1_{in}, t^2_{in}, \ldots, t^{\max(t)-1}_{in}, t^{\max(t)}$$

there exists at least one distinct TA that can synchronize on the channel with the control automaton. Moreover, if t_{test} is enabled then we cannot reach a deadlock by firing this sequence of transitions. In fact, if a deadlock could be reached during this sequence, then t_{test} would not have been enabled (due to the invariant).

By construction, it follows that if we can synchronize on the channels $t^1_{in}, t^2_{in}, \ldots, t^{\max(t)-1}_{in}, t^{\max(t)}$, then it is possible to synchronize on the channels $t^{max(t)-1}_{out}$, $t^{max(t)-2}_{out}, \ldots, t^1_{out}$, and no deadlock can be reached during this sequence. This follows from the fact that the automaton synchronizing on channel t^i_{in} will also be the one to synchronize on channel t^i_{out} for all $1 \leq i \leq max(t) - 1$.

In turn, this means that any maximal discrete sequence starting in s_0 contains a prefix of the form (again, omitting the intermediate states)

$$s_0 \xrightarrow{t_{test}} \xrightarrow{t^1_{in}} \xrightarrow{t^2_{in}} \cdots \xrightarrow{t^{\max(t)-1}_{in}} \xrightarrow{t^{\max(t)}} \xrightarrow{t^{\max(t)-1}_{out}} \xrightarrow{t^{\max(t)-2}_{out}} \cdots \xrightarrow{t^1_{out}} s'.$$

It follows from the construction that synchronizing on the t^1_{out} channel brings the control automaton back to ℓ_{stable} which implies that $s' \models (\#\ell_{stable} = 1)$.

Thus, we will always return to a stable state and hence $T(P_{TA})$ is an *eventually-stable* TTS. ∎

Remark 53 By Lemma 52 we get that for any *stable* configuration s with an enabled discrete transition, it holds that $s \rightsquigarrow s'$ for some configuration s'. ♢

As for the previous translation, the proposition translation function tr_p translates propositions of the form $(p \bowtie n)$ to propositions of the form $(\#p \bowtie n)$ for some $\bowtie \in \{<, \leq, =, \geq, >\}$.

We define the correspondence relation $\mathcal{R}$ as before. That is, let $M = \{(p_1, r_1), (p_2, r_2), \ldots (p_n, r_n)\}$ be a marking of a k-bounded TAPN N where $n \leq k$ such that (p_i, r_i) denotes a token located in the place p_i with age $r_i \in \mathbb{R}_{\geq 0}$. Further, let $s = (\ell, \ell_1, \ell_2, \ldots, \ell_k, z, v)$ be a configuration of the NTA resulting from translating N by Algorithm 4, where z is a variable valuation over the set of variables $\{count_i \mid 1 \leq i \leq NumVars(N)\}$ and v is a clock valuation over the set of clocks $\{c, c_1, c_2, \ldots, c_k\}$. We write $M \mathcal{R} s$, if there exists an injection $h : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, k\}$ such that $\ell = \ell_{stable}$, $\ell_{h(i)} = p_i$ and $v(c_{h(i)}) = r_i$ for all i where $1 \leq i \leq n$, and $\ell_j = \ell_{capacity}$ for all $j \in \{1, 2, \ldots, k\} \setminus range(h)$.

We shall now prove that conditions 1-6 of Definition 29 are satisfied by $\mathcal{R}$.

Theorem 54 The relation $\mathcal{R}$ is a complete one-by-many correspondence. ♢

PROOF Let (N, M_0) be a marked k-bounded TAPN and P_{TA} be the NTA generated by Algorithm 4. We will show that $\mathcal{R}$ satisfies all requirements of Definition 29 and thus is a complete one-by-many correspondence. From Lemma 50, Lemma 51 and Lemma 52 it follows that $T(P_{TA})$ is a *no-delay-in-intermediate-state*, *delay-preserves-stable* and *eventually-stable* TTS.

Let M be a marking and let s be a configuration, such that $M \mathcal{R} s$. We shall now prove that $\mathcal{R}$ satisfies conditions 1-6 of Definition 29 on page 35.

1. $s \models (\#\ell_{stable} = 1)$ follows from the definition of $\mathcal{R}$.

2. From the definition of $\mathcal{R}$ it follows that $M \models \wp$ iff $s \models tr_p(\wp)$.

3. Assume that $M \longrightarrow M'$. This means that there exists a transition t such that $M \xrightarrow{t} M'$. We must show that $s \rightsquigarrow s'$ such that $M' \mathcal{R} s'$. We will start by showing that the t_{test} broadcast synchronization 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 on the sender transition, thus only the invariant on $\ell(t)$ may block the t_{test} sender. The second requirement is that every possible receiver (i.e. any receiver whose guard is satisfied) must participate in the broadcast synchronization. Since a self-loop is used for all receivers (with counter updates only), the target invariants will be satisfied for all of them. Thus, any receiver with a satisfied guard can participate in the broadcast synchronization. Further by the fact that $M \mathcal{R} s$, it follows that enough TA will participate in the broadcast synchronization to satisfy the invariant on $\ell(t)$. Thus, $s \xrightarrow{t_{test}}$.

   By Lemma 52 and the fact that $s \xrightarrow{t_{test}}$, we can conclude that

   $$s \xrightarrow{t_{test}} s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \longrightarrow s_n = s'$$

   such that $s_n \models (\#\ell_{stable} = 1)$ and $s_i \not\models (\#\ell_{stable} = 1)$ for $1 \leq i < n$. By construction, this sequence of transitions must be of the form

   $$t_{test}, t_{in}^1, t_{in}^2, \ldots, t_{in}^{max(t)-1}, t^{max(t)}, t_{out}^{max(t)-1}, t_{out}^{max(t)-2}, \ldots, t_{out}^1$$

   as this is the only sequence available in the control automaton involving t_{test}. This implies that $n = 2 \cdot max(t)$. Hence, since $M \xrightarrow{t} M'$, we can use the sequence above and Lemma 50 to get the sequence $s \rightsquigarrow s'$. Observe that by construction of P_{TA}, we know that if $|{}^\bullet t| > |t^\bullet|$ then $|{}^\bullet t| - |t^\bullet|$ token automata will be moved to $\ell_{capacity}$ when simulating the firing of transition t and if $|{}^\bullet t| < |t^\bullet|$ then $|t^\bullet| - |{}^\bullet t|$ token automata will be moved out of $\ell_{capacity}$ when simulating the firing of transition t. By matching the changes in M when firing t to the changes in s when executing the sequence above, we get that $M' \mathcal{R} s'$.

4., 6. Time delays can only be restricted by invariants. By Lemma 50 and the fact that the invariant on place p in N is carried over to location p in P_{TA}, we have that it is always possible to do the same time delays in M and s. By Lemma 51,

it follows clearly that if $M \xrightarrow{d} M'$ then $s \xrightarrow{d} s'$ such that $M' \mathcal{R} s'$ and vice versa.

5. Assume $s \rightsquigarrow s'$. This implies

$$s \longrightarrow s_1 \xrightarrow{0} s_1 \longrightarrow s_2 \xrightarrow{0} s_2 \longrightarrow \cdots \longrightarrow s_{n-1} \xrightarrow{0} s_{n-1} \longrightarrow s_n = s'$$

such that $s' \models (\#\ell_{stable} = 1)$ and $s_i \not\models (\#\ell_{stable} = 1)$ for all $1 \leq i < n$. By construction of P_{TA}, we have that this run must be of the form (leaving out 0 delays)

$$s \xrightarrow{t_{test}} s_1 \xrightarrow{t_{in}^1} \cdots \xrightarrow{t_{in}^{\max(t)-1}} s_{\max(t)} \xrightarrow{t^{\max(t)}} s_{\max(t)+1} \xrightarrow{t_{out}^{\max(t)-1}} \cdots \xrightarrow{t_{out}^1} s_n = s'$$

which is a simulation of some transition t and thus $n = 2 \cdot max(t)$.

Since $s \xrightarrow{t_{test}} s_1$ we know that the invariant on $\ell(t)$ is satisfiable. By the construction of P_{TA}, this in turn means that there are enough automata that can synchronize on the t_{test} channel 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 all these automata satisfy the guards on the t_{test} transitions. Further, for each place p' where there is an inhibitor arc from p' to t, there is no automaton in location p' with a clock satisfying the guard on the t_{test} transition. This is exactly what is needed to fire t, and because $M \mathcal{R} s$, then $M \xrightarrow{t} M'$.

If any token automaton moves out of $\ell_{capacity}$ when $s \rightsquigarrow s'$ then additional tokens will be produced when firing transition t from M and if any token automaton moves into $\ell_{capacity}$ when $s \rightsquigarrow s'$ then t will consume more tokens than it produces when fired from M. Since $M \mathcal{R} s$, we clearly get $M' \mathcal{R} s'$ by matching the changes in s when simulating t via the sequence above to the changes in M when firing t. ■

Since $\mathcal{R}$ is a complete one-by-many correspondence, Theorem 36 allows us to conclude the following corollary.

Corollary 55 Let (N, M_0) be a k-bounded TAPN and let P_{TA} be the NTA constructed by Algorithm 4. For any TCTL formula φ we have that

$$N \models \varphi \quad \text{if and only if} \quad P_{TA} \models tr(\varphi) \qquad \diamondsuit$$

As a concrete example, consider the TAPN N and translated NTA P_{TA} in Figure 9.1on page 67. Let $M = \{(p_1, 2.8), (p_2, 1.4), (p_3, 3.0)\}$ be the marking illustrated in the figure. The maximal run

$$\rho = M \xrightarrow{0.5} M_1 \xrightarrow{t} M_2 \xrightarrow{1} M_3 \xrightarrow{t'} M_4 \xrightarrow{2^\leq}$$

in $T(N)$ witnesses the property $\varphi = E((p_4 = 0)\, U_{[0,\infty)}\, (p_4 = 1))$. Thus, we have $M \models E((p_4 = 0)\, U_{[0,\infty)}\, (p_4 = 1))$. In P_{TA} we can construct a witness for the translated

query $tr(\varphi) = E(((\#p_4 = 0) \vee \neg(\#\ell_{stable} = 1))U_{[0,\infty)}((\#p_4 = 1) \wedge (\#\ell_{stable} = 1)))$. Let s be a configuration such that $M \mathcal{R} s$. The maximal run

$$\rho' = s \xrightarrow{0.5} s_1 \xrightarrow{t_{test}} s_2 \xrightarrow{0} s_3 \xrightarrow{t^1_{in}} s_4 \xrightarrow{0} s_5 \xrightarrow{t^2} s_6 \xrightarrow{0} s_7 \xrightarrow{t^1_{out}} s_8 \xrightarrow{1} s_9 \xrightarrow{t'_{test}} s_{10} \xrightarrow{0} s_{11} \xrightarrow{t'^1} s_{12} \xrightarrow{2^{\leq}}$$

in $T(P_{TA})$ witnesses $tr(\varphi)$. Thus, we have that

$$s \models E(((\#p_4 = 0) \vee \neg(\#\ell_{stable} = 1))U_{[0,\infty)}((\#p_4 = 1) \wedge (\#\ell_{stable} = 1))).$$

Chapter 10

Timed-Arc Petri Nets with Integers

In this section, we will extend TAPN model by letting tokens carry integer values besides their age. This feature is often referred to as a color. Our approach is rather simple compared to other notions of colors in Petri Nets [30, 31, 41], however our approach preserves decidability of reachability for bounded models. Before we formally define this model, we need to introduce some notation.

We define arbitrary intervals in the form $\llbracket a, b \rrbracket$ where $\llbracket \in \{[, (\}$, $\rrbracket \in \{],)\}$ and $a, b \in \mathbb{Z}$. We let the set of all arbitrary intervals be denoted by $\mathcal{I}_{all}$. Note that some intervals are not well-formed, e.g. $[2, 2)$ or $[3, -1]$. Such intervals are interpreted as the empty interval. The predicate $r \in I$ is defined for $I \in \mathcal{I}_{all}$ and $r \in \mathbb{R}_{\geq 0}$ in the expected way.

In the TAPN with integers model, we can use the value of a token as a parameter to the time interval on input arcs. Therefore we introduce *age guards* as extended time intervals with linear functions as bounds, which are defined according to the abstract syntax

$$I_{ext} ::= \llbracket a \cdot \mathsf{val} \oplus b, c \cdot \mathsf{val} \oplus d \rrbracket,$$

where $\llbracket \in \{[, (\}$, $\rrbracket \in \{],)\}$, $\oplus \in \{+, -\}$, $a, b, c, d \in \mathbb{N}_0$ and val is the special keyword referring to the value of the token used. We denote the set of all age guards by $\mathcal{I}_{ext}$. Further, we let $\mathcal{I}_{ext}^{inv} \subset \mathcal{I}_{ext}$ be the set of all intervals of the form $[0, c \cdot \mathsf{val} \oplus d]$. We will write expressions of the form $0 \cdot \mathsf{val} + b$ simply as b. We define a function $eval : \mathcal{I}_{ext} \times \mathbb{N}_0 \longrightarrow \mathcal{I}_{all}$ that evaluates extended intervals according to a specific value. For example, $eval([1 \cdot \mathsf{val} - 4, 3 \cdot \mathsf{val} + 5), 3)$ evaluates to the interval $[-1, 14)$.

We let $VG = \{\langle I_1, I_2, \ldots, I_n \rangle \mid n \in \mathbb{N}$ and $I_i \in \mathcal{I}$ for $1 \leq i \leq n\}$ be the set of all finite sets of *value guards*. For some $\langle I_1, I_2, \ldots, I_n \rangle \in VG$ and some $v \in \mathbb{N}_0$ we write $v \in \langle I_1, I_2, \ldots, I_n \rangle$ if $v \in I_i$ for some $1 \leq i \leq n$. Note that even though elements of VG are finite, they can represent infinite sets of values, e.g. $\langle [1, 4], [7, 7], [10, \infty) \rangle \in VG$ represents the infinite set $\{1, 2, 3, 4\} \cup \{7\} \cup \{v \mid v \geq 10\}$.

We further define the following sets:

- $AU = \{0, preserve\}$ is the set of possible age updates on output arcs, and
- $VU = \{\mathbb{N}_0\} \cup \{preserve\}$ is the set of possible value updates on output arcs.

Output arcs will always update the age of a token to 0, except in the case of transport arcs which may preserve the age. Further, output arcs can set the value of the newly produced token. If an output arc is part of a transport arc it may preserve the value or the age of the consumed token. This is the reason for the inclusion of the keyword *preserve* in the age and value update sets. This further means that we have four slightly different types of transport arcs: they can preserve either nothing, the age, the value or both.

From this notation we can now formally define TAPN with integers.

Definition 56 A *TAPN with integers* is a 7-tuple $(P, T, IA, OA, Transport, Inhib, \iota)$, where

- P is a finite set of places,
- T is a finite set of transitions s.t. $P \cup T = \emptyset$,
- $IA \subseteq P \times \mathcal{I}_{ext} \times VG \times T$ is a finite set of input arcs s.t.

$$((p, I, vg, t) \in IA \wedge (p, I', vg', t) \in IA) \Rightarrow (I = I' \wedge vg = vg') \quad ,$$

- $OA \subseteq T \times AU \times VU \times P$ is a finite set of output arcs s.t.

$$((t, au, vu, p) \in OA \wedge (t, au', vu', p) \in OA) \Rightarrow (au = au' \wedge vu = vu') \quad ,$$

- $Transport : IA \times OA \rightarrow \{true, false\}$ is a function defining transport arcs s.t.for all $(p, I, vg, t) \in IA$ and $(t', au, vu, p') \in OA$ if $Transport((p, I, vg, t),(t', au, vu, p'))$ then $t = t'$ and for all $\alpha \in IA$ and all $\beta \in OA$

$$\begin{aligned} &(Transport(\alpha, (t', au, vu, p')) \Rightarrow \alpha = (p, I, vg, t)) \wedge \\ &(Transport((p, I, vg, t), \beta) \Rightarrow \beta = (t', au, vu, p')) \quad , \end{aligned}$$

- $Inhib : IA \longrightarrow \{true, false\}$ is a function defining inhibitor arcs, s.t. if $Inhib(\alpha)$ for some $\alpha \in IA$ then for all $\beta \in OA$ we have $\neg Transport(\alpha, \beta)$,
- $\iota : P \rightarrow \mathcal{I}_{ext}^{inv} \times VG$ is a function assigning *age invariants* and *value invariants* to places, and
- for all $\alpha \in IA$ and all $\beta \in OA$ if $\neg Transport(\alpha, \beta)$ then $\beta = (t, 0, vu, p)$ for some $t \in T$, $vu \in \mathbb{N}_0$ and $p \in P$. ♢

The preset of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p, I, vg, t) \in IA\}$. Similarly, the postset of t is defined as $t^\bullet = \{p \in P \mid (t, au, vu, p) \in OA\}$. Like the TAPN defined in Chapter 3, we do not allow multiple input arcs (resp. output arcs) between the same place and transition (resp. transition and place).

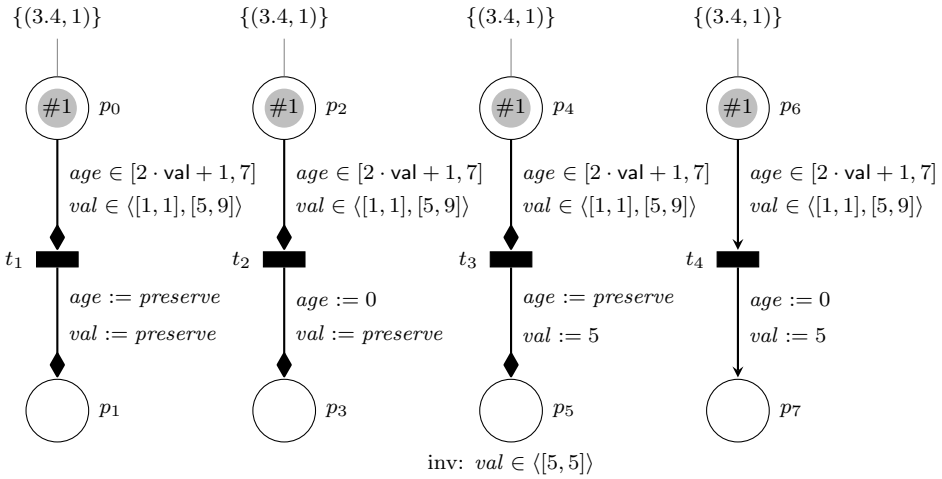


Figure 10.1: Four small TAPN with integers demonstrating the semantics of integers.

10.1 Example

Figure 10.1 shows four small TAPN with integers, each with a single transition. As evident, tokens are now pairs (x, v) where $x \in \mathbb{R}_{\geq 0}$ indicates the age of the token and $v \in \mathbb{N}_0$ indicates the value of the token. In all four models, the age of the token must lie between $2 \cdot \mathsf{val} + 1$ (in this case 3 since the value of all tokens is 1) and 7 and moreover the value of the token must belong to the set $\langle [1,1], [5,9] \rangle$. For transport arcs, the information on the output arc indicates which part of the tokens data is preserved. Thus, the leftmost TAPN has a transport arc that preserves both the age and value of a token, whereas the second and third TAPN from the left preserves only the value and the age of the token, respectively. Notice that when only the age of the token is preserved (the third TAPN from the left), we assign a specific value to the output token, in this case 5. The invariant on p_5 ensures that tokens may only have the value 5 in this place. The fourth TAPN demonstrates the semantics of normal arcs. Like a TAPN without integers, the age of the token is reset by the normal arcs. However, we must assign a value to the output token, as demonstrated on the output arc (in this case, the value 5). Figure 10.2 shows the same four TAPN after firing transition t_1, t_2, t_3 and t_4.

Finally, let us demonstrate the integer version of inhibitor arcs. Figure 10.3 shows two simple TAPN. In the left most TAPN, both the age and value of the token in p_0 satisfy their respective guards on the inhibitor arc. Thus, transition t_1 is blocked. Looking at the rightmost TAPN, we see that the age of the token in p_3 satisfies the age guard on the inhibitor arc whereas the value does not satisfy the value guard. Thus, transition t_2 is not blocked by the inhibitor arc because not all guards are satisfied. Similarly, if the value guard was satisfied but the age guard was not, then t_2 would not be blocked. This concludes our introductory example to the various features of TAPN with integers.

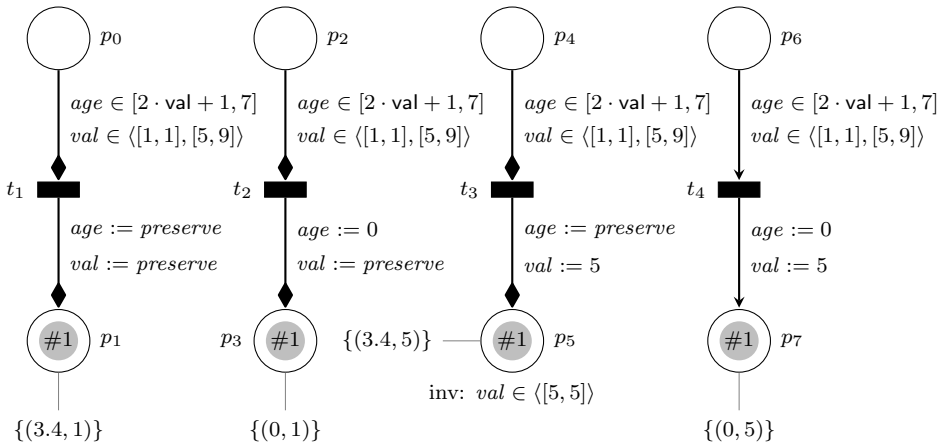


Figure 10.2: The models from Figure 10.1 after firing transitions t_1, t_2, t_3 and t_4.

10.2 Semantics

We will now define the semantics of the TAPN with integers model. This entails redefining the concepts from Section 3.3 with support for integers.

Definition 57 (Marking) Let $N = (P, T, IA, OA, Transport, Inhib, \iota)$ be a TAPN with integers. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}_{\geq 0} \times \mathbb{N}_0)$, such that for every place $p \in P$, every token $(x, v) \in M(p)$ and the invariant $\iota(p) = (I_{inv}, vi)$ it holds that $x \in eval(I_{inv}, v)$ and $v \in vi$. The set of all markings over N is denoted $\mathcal{M}(N)$. ♢

Note that we shall also sometimes use the convention (p, x, v) to refer to a token in the place p with age $x \in \mathbb{R}_{\geq 0}$ and value $v \in \mathbb{N}_0$. Likewise, we shall sometimes write $M = \{(p_1, x_1, v_1), (p_2, x_2, v_2), \ldots, (p_n, x_n, v_n)\}$ for a marking M with n tokens located in places p_i and with age x_i and value v_i for $1 \leq i \leq n$. Further, we shall sometimes index tokens by a place, such as (p_i, x_{p_i}, v_{p_i}). We shall use this notation only when there is a unique token in the set from any particular place.

A *marked* TAPN with integers is defined as a pair (N, M_0) where N is a TAPN with integers and M_0 is the initial marking on N. Note that we only allow initial markings where all tokens have age 0. There are no restrictions on the values of tokens in the initial marking.

Definition 58 (Enabledness) Let $N = (P, T, IA, OA, Transport, Inhib, \iota)$ be a TAPN with integers. We say that a transition $t \in T$ is *enabled* in a marking M by tokens $In = \{(p, x_p, v_p) \mid p \in {}^\bullet t\}$ and $Out = \{(p', x_{p'}, v_{p'}) \mid p' \in t^\bullet\}$ if

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

$$\forall (p, I, vg, t) \in IA \,.\, \neg Inhib((p, I, vg, t)) \Rightarrow (x_p \in eval(I, v_p) \wedge v_p \in vg) \quad ,$$

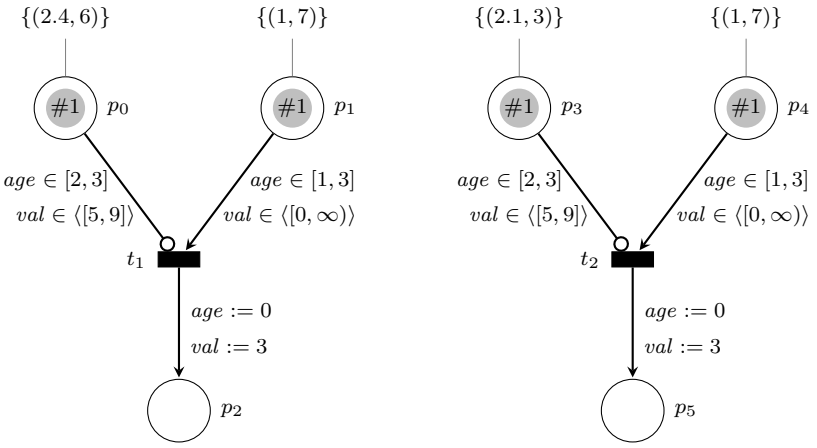


Figure 10.3: Two small TAPN demonstrating the semantics of integers and inhibitor arcs.

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

$$\begin{aligned}&\forall (p, I, vg, t) \in IA \,.\, \mathit{Inhib}((p, I, vg, t)) \Rightarrow \\ &(\neg \exists (x, v) \in M(p) \,.\, x \in \mathit{eval}(I, v) \wedge v \in vg) \quad ,\end{aligned}$$

- for all input arcs and output arcs which constitute a transport arc, if the arc is age-preserving, then the age of the output token must match that of the input token and likewise for the value if the arc is value-preserving, i.e.

$$\begin{aligned}&\forall (p, I, vg, t) \in IA \,.\, \forall (t, au, vu, p') \in OA \,. \\ &\mathit{Transport}((p, I, vg, t), (t, au, vu, p')) \Rightarrow \\ &\big(au = \mathit{preserve} \Rightarrow x_p = x_{p'}\big) \wedge (vu = \mathit{preserve} \Rightarrow v_p = v_{p'})\big)\end{aligned}$$

- for all output arcs, the age and the value must satisfy the invariants of the output locations, i.e.

$$\begin{aligned}&\forall (t, au, vu, p') \in OA \,. \\ &(au \neq \mathit{preserve} \Rightarrow x_{p'} = 0) \wedge (vu \neq \mathit{preserve} \Rightarrow v_{p'} = vu) \wedge \\ &x_{p'} \in \mathit{eval}(I_{inv}, v_{p'}) \wedge v_{p'} \in vi\end{aligned}$$

where $\iota(p') = (I_{inv}, vi)$. ♢

Definition 59 (Firing Rule) Let $N = (P, T, IA, OA, \mathit{Transport}, \mathit{Inhib}, \iota)$ be a TAPN with integers, M some marking on N and $t \in T$ some transition of N. If

t is enabled in the marking M by tokens $In = \{(p, x_p, v_p) \mid p \in {}^\bullet t\}$ and $Out = \{(p', x_{p'}, v_{p'}) \mid p' \in t^\bullet\}$ then it can be *fired*, whereby we reach a marking M' defined as

$$M' = (M \setminus In) \cup Out$$

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

Let us now define the notion of a time delay in our model. To do so, we shall introduce an addition operator on multisets. For a multiset $B = (\mathbb{R}_{\geq 0} \times \mathbb{N}_0, f)$ and a non-negative real $d \in \mathbb{R}_{\geq 0}$, we define the addition operator in the following manner,

- $(B + d)(x, v) = \begin{cases} f(x - d, v) & \text{if } x - d \geq 0 \\ 0 & \text{otherwise} \end{cases}$

where $x \in \mathbb{R}_{\geq 0}$ and $v \in \mathbb{N}_0$.

Definition 60 (Time Delay) Let $N = (P, T, IA, OA, Transport, Inhib, \iota)$ be a TAPN with integers and M some marking on N. A *time delay* $d \in \mathbb{R}_{\geq 0}$ is allowed if $(x + d) \in eval(I_{inv}, v)$ where $\iota(p) = (I_{inv}, vi)$ for all $p \in P$ and all $(x, v) \in M(p)$, i.e. by delaying d time units no token violates the invariants. By delaying d time units we reach a marking M' defined as $M'(p) = M(p) + d$ for all $p \in P$. ♢

The semantics of a TAPN with integers is given by a timed transition system. Specifically, a TAPN with integers $N = (P, T, IA, OA, Transport, Inhib, \iota)$ generates a TTS $T(N) = (\mathcal{M}(N), \longrightarrow, \mathcal{AP}, \mu)$ where the set of states are the markings on N, and the transition relation $\longrightarrow$ is defined such that $M \longrightarrow M'$ if by firing some transition t in marking M we get to marking M' and $M \xrightarrow{d} M'$ if by delaying d time units in marking M we get to marking M'. The set of atomic propositions $\mathcal{AP}$ is the same as for TAPN without integers. Thus, $\mathcal{AP} \stackrel{def}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$ and the labeling function is defined as $\mu(M) \stackrel{def}{=} \{(p \bowtie n) \mid |M(p)| \bowtie n, \bowtie \in \{<, \leq, =, \geq, >\}\}$.

10.3 Translation and Correctness

We will now extend the translations from Chapter 8 and Chapter 9 to support TAPN with integers. However, as the extension is not substantial, we will only argue informally that the addition of the integers is a straightforward extension of the translations. We will do this in terms of the Broadcast translation in Chapter 8. The extension lies solely in changing the guards, updates and invariants of the constructed NTA.

Recall that our notion of an NTA already supports integer variables as these are used as counters in the two translations. Further, since the translation adds a TA for each token in the net, we can simply add an integer variable for each token automaton which will serve as the token's value. Note that in the syntax and semantics we have used unbounded integers, which will obviously not work for verification purposes. However, since we can only update the value of tokens by constants on output arcs (e.g. $\mathsf{val} := 5$), we are using integers in a bounded way.

Value guards

Value guards are represented as a set of integer intervals. This can easily be translated into a disjunction of the elements. For instance, the value guard $\langle [1,5],[7,7],[10,\infty) \rangle$ is translated to $1 \leq x \leq 5 \vee x = 7 \vee x \geq 10$ where x is the integer variable representing the value of the token. This is added to the guard of the corresponding t_{test} edge in the NTA.

Value updates

Value updates are also straightforward to translate. For example, a value update of 4 on some output arc will be translated to the variable update $x := 4$ on the corresponding t_{fire} edge in the NTA where x is the integer variable representing the value of the token. If the value update is *preserve*, then no value update is added to the corresponding edge in the NTA.

Invariants

For every output arc which has an update of the form $\mathsf{val} := n$ for some $n \in \mathbb{N}_0$, we must check whether n belongs to the value invariant of the target place. If this is not the case, the transition can never fire in the TAPN model, and we must ensure that the simulation of the transition is always disabled in the NTA. We achieve this by setting the invariant on the $\ell(t)$ location to $x < 1 \wedge x > 1$, which is always false.

Intersection of guards and invariants

In order to avoid deadlocks, we must intersect the guards and the target invariants when creating the guard for the t_{test} edge, whenever the paired input and output places are connected by a transport arc. However, since we now have four different types of transport arcs, we must be careful to handle them correctly. Let us first describe how to intersect the guards and invariants and then explain how to handle the different types.

When intersecting value guards and value invariants, we must intersect the individual intervals. For example, given the value guard $\langle [1,1],[5,7],[10,\infty) \rangle$ and the value invariant $\langle [1,3],[6,20] \rangle$ we get the intersected guard $\{[1,1],[6,7],[10,20]\}$ containing precisely the elements that appear in both sets.

Age guards are more tricky since bounds are given as linear functions, which may have different slopes. Because of this, we cannot intersect them in the same way as in Chapter 8 (because it is not well-defined which is larger, since it depends on the value of the token). Instead, we make a conjunction of the functions. For instance, assume that an arc contains an age guard of the form $[1 \cdot \mathsf{val} + 1, 3 \cdot \mathsf{val} + 3)$ and the target place contains an age invariant of the form $[0, 2 \cdot \mathsf{val} + 2]$. The guard we create in the NTA will look like the following (assuming c_1 is the name of the clock): $c_1 \geq 1 \cdot \mathsf{val} + 1$ && $c_1 < 3 \cdot \mathsf{val} + 3$ && $c_1 \leq 2 \cdot \mathsf{val} + 2$.

Now, let us handle the four types of transport arcs.

- *Preserve nothing*: equivalent to normal arcs.
- *Preserve age*: intersect the age guard and the age invariant of the target place only. Check the value update and the value invariant of the target place, as described above.
- *Preserve value*: intersect only the value guard and the value invariant of the target place.
- *Preserve age and value*: intersect age guard and the age invariant of the target place, as well as the value guard and the value invariant of the target place.

Summary

Thus, the small addition of integers to TAPN only requires a straightforward extension of the translation in Chapter 8. Further, the definition of the complete one-by-many correspondence for these translations could be extended such that it also requires that the value of tokens in a marking matches the values of the corresponding variables in a configuration of the NTA. As there is nothing in the addition of the integers which would break the properties of a complete one-by-many correspondence, it would still preserve the full TCTL. Note the Degree 2 Broadcast translation from Chapter 9 can be extended in a similar fashion to support integers.

10.4 Decidability

In this section we will look into the decidability of reachability for bounded TAPN with integers.

We have already sketched a translation to timed automata in Section 10.3. The use of integers should not cause problems as we are using the integers in a bounded way, since we only allow value updates using constant values.

Note that the expressions used in age guards may become negative as they are of the form $age \in [\![a \cdot \mathsf{val} \oplus b,\ c \cdot \mathsf{val} \oplus d]\!]$, where $[\![\in \{[, (\}$, $]\!] \in \{],)\}$, $\oplus \in \{+, -\}$ and $a, b, c, d \in \mathbb{N}_0$. However, tokens can only carry non-negative integers and thus some age guards may not be satisfiable. For instance, we could have an arc with a guard $age \in [-6, -2]$ which can never be satisfied by any token. However, this should not be a problem as it just prevents certain transitions from firing. Thus, we get the following theorem.

Theorem 61 Reachability is decidable for bounded TAPN with integers. ♢

Let us now consider a variant of the TAPN with integers model in which we allow value updates to be of the form $val := a \cdot \mathsf{val} \oplus b$ where $\oplus = \{+, -\}$. Let us denote this model by TAPN($\pm updates$). By allowing these types of value updates we can simulate of a 2-CM. Although the TAPN model is bounded in terms of the number of tokens, there is no bound on the values of tokens. Thus, we can use a single token to simulate

a single register, where the value of the token represents the value of the register. This is possible because we can use value updates to both increase and decrease the values of tokens. Further, it is possible to test for zero on the value guards by using the value guard $\langle [0,0] \rangle$. Thus, with two tokens we are able to simulate the two registers of a 2-CM and we get the following theorem.

Theorem 62 Reachability is undecidable for bounded TAPN($\pm$*updates*). ♢

Chapter 11

TAPAAL

In this chapter, we will present an overview of the verification tool TAPAAL [1], which can be used to perform modeling, simulation and verification of TAPN models. As already mentioned, TAPAAL translates TAPN models into networks of timed automata, thereby leveraging the UPPAAL verification engine for the actual verification. Note that this section discusses a prototype of TAPAAL (not publicly available) which include additional features not found in the current version 1.3 release.

11.1 Overview

We will now give a short overview of the features of the tool.

Main Window

Figure 11.1 shows a screenshot of the application. In TAPAAL, it is possible to draw TAPN models using tools found in the toolbar. Each tab contains a drawing surface on which TAPN models can be drawn, hence each tab contains a model. Models in different tabs cannot communicate in any way. For each model, TAPAAL maintains a list of queries specified by the user and a list of integer constants in the left-hand pane. It is possible to save any number of queries for each model. Verification of a query is possible via the *Verify* button. It is possible to define integer constants which can be used in invariants and guards in the model. This is useful if we need to use the same integer in multiple places. For example, we might specify a delay constant which need to be included in multiple guards. Without integer constants, it would be necessary to change the value of the constant in multiple places if we were to, say, change the delay from 5 to 3. With an integer constant, we can simply change the value of the constant, and it is automatically updated in every place where the constant is used. TAPAAL also allows for simulation of TAPN models. Clicking the flag in the toolbar, puts TAPAAL into simulation mode, in which changes to the model cannot be made. In this mode, it is possible to simulate the net by performing time delays and transition firings. This is useful for debugging models during the modeling process.

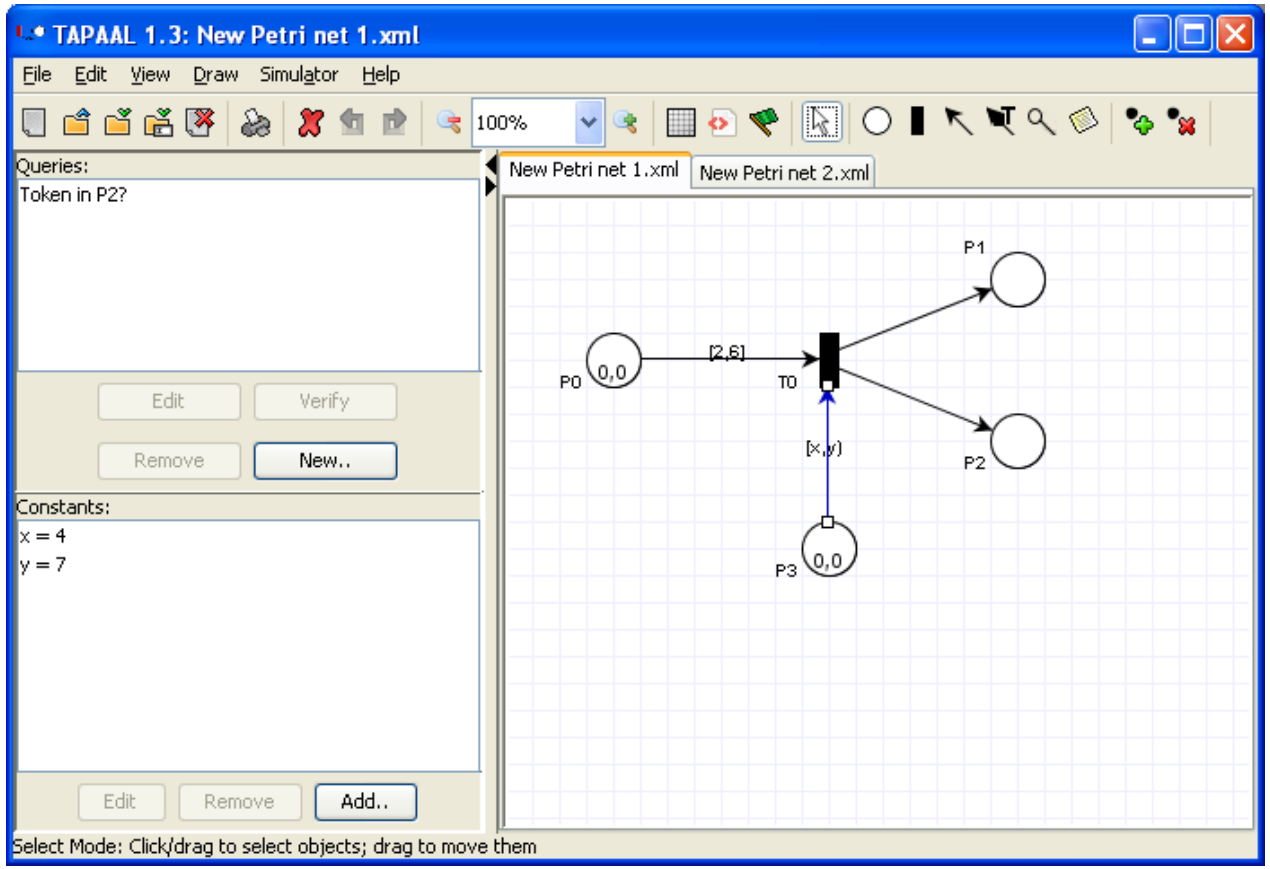


Figure 11.1: The modeling, simulation and verification tool TAPAAL.

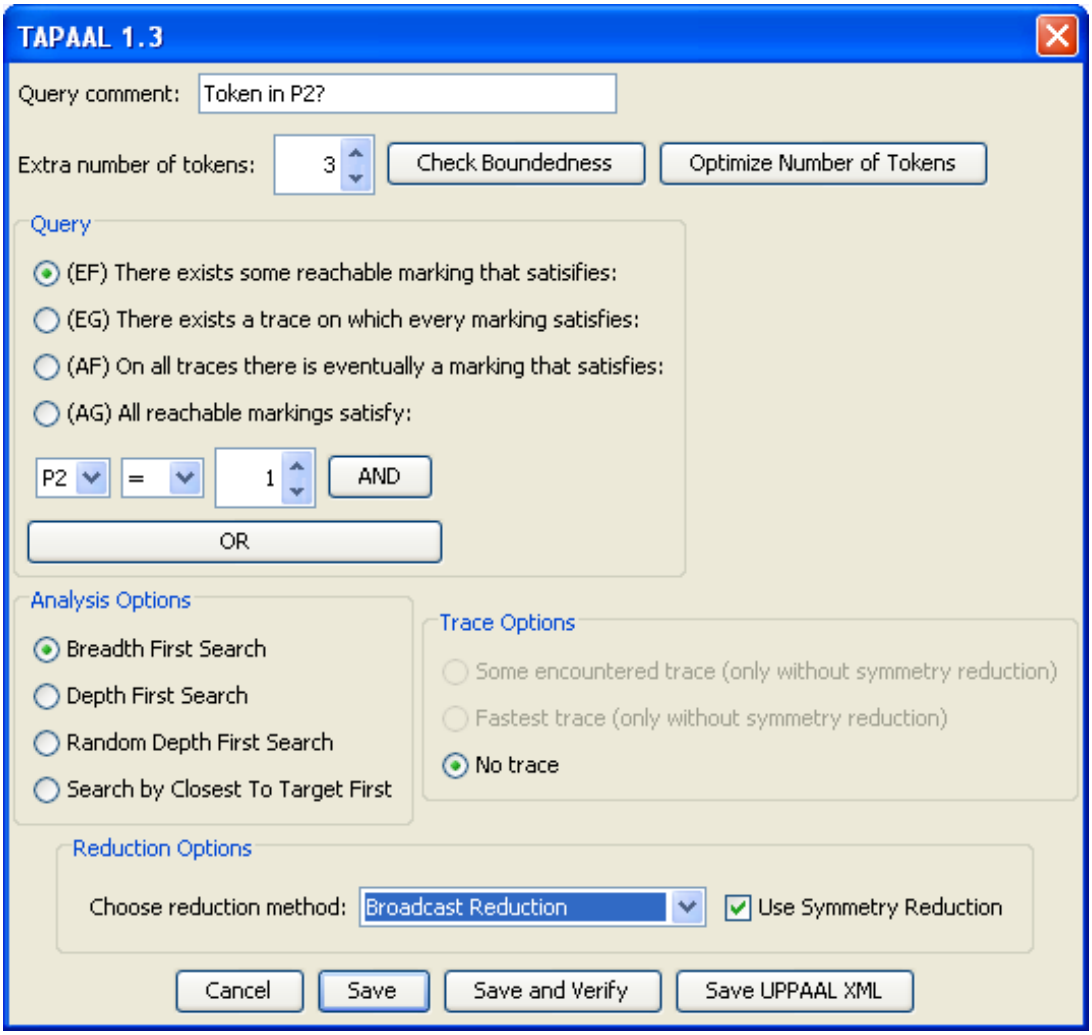


Figure 11.2: The query dialog from TAPAAL.

Query Dialog

Figure 11.2 shows the query dialog which is used when creating or editing queries. In this dialog it is, among other things, possible to check whether the net is k-bounded. One can specify the extra number of tokens to use (besides the tokens already in the model), and then check, whether the net is bounded for this number of tokens. If the net is bounded for the specified number of extra tokens, it is possible to ask TAPAAL to optimize the number of extra tokens needed. TAPAAL will then use UPPAAL to find the supremum (least upper bound) on the number of extra tokens needed for a faithful verification.

Not all reduction options in TAPAAL can verify liveness properties (EG and AF). Specifically, the translation by Byg et al. [17] introduces additional deadlocks into the

system which means it does not work for liveness in general (only when there are no inhibitor arcs and the model is already degree 2). As proved in the previous chapters, the two translations relying on broadcast transitions work for arbitrary bounded models for the full TCTL (and hence for all for operators supported by TAPAAL).

The predicate of a formula is expressed as a series of conjunctions and disjunctions of propositions. In the propositions, we can talk only about the number of tokens in a place, not their age. For example, we can formulate a query asking whether we can place more than 1 token in $P2$ in the following manner: $\mathsf{EF}(P2 > 1)$. Similarly, we can ask whether it always holds that there is a token in $P1$ and at most 3 tokens in $P3$ by using the query: $\mathsf{AG}(P1 == 1 \,\&\&\, P3 <= 3)$. The query dialog allows us to express such properties, and it will automatically generate a corresponding logic property to verify, when translating the TAPN model to a network of timed automata.

When TAPAAL translates a model to a network of timed automata, there are basically two reduction strategies, standard and symmetry reduction (with various optimizations). If we use standard reduction, we can ask UPPAAL for a trace in the positive case, showing us how the property is satisfied. A current limitation of UPPAAL means that sometimes it is not possible to obtain a timed trace (e.g. a trace with both transition firings and time delays). TAPAAL will in these cases obtain an untimed trace from UPPAAL, in which we have to fill in time delays ourselves. In either case, TAPAAL allows us to simulate this trace in the simulation mode. If we use symmetry reduction, we cannot obtain a trace from UPPAAL, but verification will in many cases be much faster.

11.2 Implementation

Since TAPAAL was mainly developed by Byg et al. [16], we shall list our contributions to the tool in this section. Our work has mainly revolved around:

1. Optimization of extra tokens needed in case the net is bounded,
2. Integer constants,
3. Export to TikZ,
4. Redesigned query list,
5. Inhibitor arcs (modeling, simulation and verification),
6. Implementation of translations detailed in Chapter 8 and Chapter 9, and
7. TAPN with integers as detailed in Chapter 10 (modeling, simulation and verification).

The first four are available in version 1.3 of TAPAAL, whereas the rest is not yet publicly available. We shall only briefly comment on item 6 and 7.

11.2.1 Translations

The algorithms in Chapter 8 and Chapter 9 have been implemented in TAPAAL and are available from the query dialog. The following optimizations apply to both translations.

Degree 2 Optimization

Our implementation contains optimizations, as presented in [17]. Specifically, if we encounter transitions in the original net, which are already degree 2 and contains no inhibitor arcs, then we simulate the transition in a single handshake synchronization in the timed automata, instead of using broadcast synchronizations. To ensure interleaving cannot occur when such transitions are present, we include a boolean variable lock which we set to true whenever the control automaton moves out of ℓ_{stable}, and we set it to false whenever the control automaton returns to ℓ_{stable}. Any transition which is already degree 2 (and has no inhibitor arcs) will then simply be simulated by a single handshake synchronization in the timed automata with the additional guard that lock == false. This prohibits interleaving of simulations of different transitions.

Symmetry reduction

Symmetry reduction is a way to combat the state space explosion problem (see e.g. [26] for an explanation of how to apply symmetry reduction to timed automata). As a simple example, imagine two identical communicating servers A and B. The state where A wants to send a message to B and B wants to receive a message is equivalent to the state in which B wants to send a message to A and A wants to receive a message. Thus, if we know how the system behaves in the case where A is the sender, then we also know how the system behaves when B is the sender. Therefore, we can simply explore one of these states during verification and disregard the others. We say that A and B are in the same equivalence class. We only need to explore one state from each equivalence class.

In UPPAAL, symmetry reduction is enabled via a special datatype called scalarset (denoted scalar in the UPPAAL specification language) [25]. Essentially, a scalarset of size n is a subrange $[0, n-1]$ of natural numbers, which supports only assignment, equality/inequality testing and array indexing [26]. Templates instantiated with a scalarset type is considered by UPPAAL to be symmetric.

Symmetry reduction is particularly useful for our purposes, since we generate a timed automaton for each token. Each of these automata are structurally equivalent except for the initial states. In order to exploit symmetry reduction, these automata have to be identical. Therefore, we shall introduce additional initialization synchronizations in our timed automata that puts token in the initial states (according to the initial marking).

For instance, assume we have the model from Figure 9.1 on page 67 and that degree 2 optimizations as described above are turned on. We shall show how to extend the

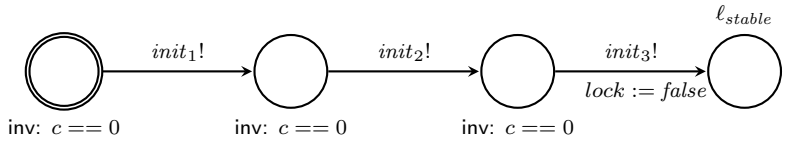


Figure 11.3: The additional initialization edges for the control automaton.

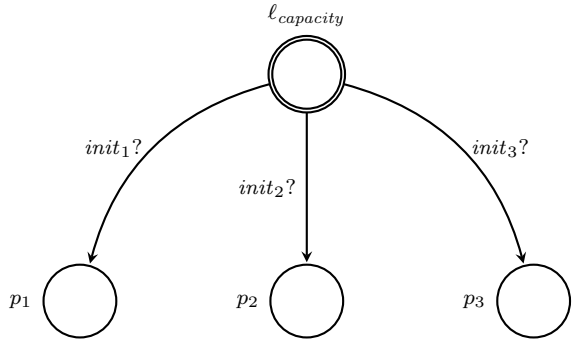


Figure 11.4: The additional initialization edges for the token automaton.

NTA to make it eligible for symmetry reduction. Specifically, we shall create a single token automaton template, and use the scalarset datatype to instantiate this template k times, where k is the number of tokens. Further, we initially set the lock boolean to true (meaning degree 2 transitions cannot fire).

Because we need a common initial location for the token automata, we shall use the location $\ell_{capacity}$ for this purpose. We augment the control automaton with additional initialization edges, which put the tokens in the correct initial locations. For the NTA in Figure 9.1, the additional edges are illustrated in Figure 11.3. Similarly, we augment the token automaton template with additional initialization edges that move the automaton out of $\ell_{capacity}$, as illustrated in Figure 11.4. Note that for both these figures, the entire TA is not shown and that the final initialization edge will set the lock boolean to false.

11.2.2 TAPN with Integers

We shall briefly comment on the implementation of integers in TAPAAL. The capabilities are derived from those outlined in Chapter 10. Each place may contain an age invariant of the form $[0, a{\cdot}\mathsf{val}{+}b]$ where a, b are non-negative integers and a value invariant. The value invariant is given as a set of ranges, e.g. $\{1, 3, 5-8, 10-\}$ which means that tokens in that place must have a value included in the set $\{1, 3, 5, 6, 7, 8\} \cup \{v \mid v \geq 10\}$. Input arcs (that is, arcs from places to transitions), regardless of type, can contain an age guard and a value guard. Age guards are of the form $[\![a{\cdot}\mathsf{val}{+}b, c{\cdot}\mathsf{val}{+}d]\!]$ where a, b, c, d are non-negative integers and value guards follow the same style as the value invariants, that is, they are given as a set of integers to which the token's value

must belong.

Normal output arcs contain an update expression indicating the value to assign to the produced token. For transport arcs, there are three possibilities: preserve both the age and value, preserve only the value and preserve only the age. If only the age is preserved, a transport arc also carries an update expression to indicate the value to assign to the consumed token upon firing the transition.

The GUI of TAPAAL has been extended to allow the manipulation of all these features and the simulator has been extended to function correctly when the model is a TAPN with integers. Naturally, save/load support has also been added to TAPAAL for these models.

Finally, the broadcast translations detailed in the previous chapters have both been extended to support integers, thereby allowing for verification using UPPAAL.

Chapter 12

Experiments

In this chapter, we present a series of experiments comparing the verification time of TAPAAL using the translations from Chapter 8 and Chapter 9 and the translation by Byg et al. [17] on some small case studies. Further, we will explore how the addition of integers to the TAPN model, as described in Chapter 10, impacts the verification time. We shall use the following conventions.

- Unless stated otherwise, a dash $(-)$ means that the verification time exceeded five minutes.
- We will refer to the translation by Byg et al. [17] as the Standard translation, the translation in Chapter 8 as Broadcast and the translation from Chapter 9 as Deg-2-Broadcast.
- When we say that optimizations (resp. symmetry) are turned on or off we are referring to the degree 2 optimizations (resp. symmetry reduction) mentioned in Section 11.2.1.

Note that there might be slight inaccuracies in the verification time measurements when the verification times are small (i.e. around 0.1 seconds) and that all experiments have been run only once using breadth-first search (except liveness experiments which are always done with depth-first search in UPPAAL).

All experiments in this chapter were performed on a Dell Optiplex 755 with an Intel Core2 Duo 2.66 GHz processor and 8 GB RAM running Ubuntu 10.04. Note that UPPAAL only utilizes one of the cores and it uses at most 4 GB of RAM.

12.1 TAPN

We will now present the experiments for TAPN using the translations from Chapter 8 and Chapter 9. Two small case studies have been selected for these experiments. The first case study will be Alternating Bit Protocol [8], which is modeled as a TAPN of degree 2 [17]. We will use this to demonstrate how the performance of our translations

Alternating Bit Protocol: Safety, Symmetry off, optimizations off			
# Messages	**Standard**	**Broadcast**	**Deg-2-Broadcast**
2	0.25s	0.12s	0.13s
3	3.91s	1.42s	1.39s
4	54.15s	18.78s	16.7s
5	-	3m 36.53s	2m 54.42s
6	-	-	-

Table 12.1: Comparison of the verification times for ABP with both symmetry and optimizations turned off.

compares to that of the Standard translation when optimizations are turned off. Recall that when optimizations are turned on transitions of degree 2 will be translated to a handshake synchronization in the NTA. Thus, if optimizations were turned on in a degree 2 net we would not use the broadcast features of our translations at all (as all transitions would be translated to handshake synchronizations).

The second case study is called Soccer Field Grass Cutting which we have invented for these experiments. The case study involves three workers tasked with keeping a number of soccer fields in shape. The grass on each field must be cut within a certain period after it was last cut to prevent the grass from becoming too long. One, two or three workers can cut a field, however, the less workers the longer it takes to cut the field. This case study will be used to compare the verification times of the three translations when optimizations are turned on. The net contains transitions with a degree greater than 2. Because of this, we know that some transitions will use broadcasts, even with optimizations turned on. Our experiments for this case study involve both safety and liveness properties.

12.1.1 Alternating Bit Protocol

Our first experiment is the verification of Alternating Bit Protocol (ABP) [8]. ABP is a network protocol for communication between a sender and a receiver over a lossy communication channel. In the event that messages are lost, the protocol will retransmit messages. The TAPN model for this protocol was taken from [17] and it is included as an example in TAPAAL. We use a query that checks whether the synchronization between the sender and the receiver can be violated. Since the protocol is correct, the query is not satisfied.

Table 12.1 presents the verification times for ABP with both symmetry and optimizations turned off. The parameter "number of messages" represents the total number of messages in transit. Observe that although verification is only possible for small instances of the problem (due to the absence of symmetry), both our translations outperform the Standard translation. Since the query is not satisfied, we are

Alternating Bit Protocol: Safety, Symmetry on, optimizations off			
# Messages	**Standard**	**Broadcast**	**Deg-2-Broadcast**
12	2.22s	9.06s	4.51s
14	4.75s	21.96s	9.53s
16	9.43s	48.56s	18.77s
18	17.58s	1m 38.85s	34.28s
20	30.96s	3m 10.87	59.74s
22	52.39s	-	1m 39.88s
24	1m 24.86s	-	2m 40.29s
26	2m 14.09s	-	4m 9.57s

Table 12.2: Comparison of the verification times for ABP with symmetry turned on and optimizations turned off.

essentially searching the whole state space during the verification. The reason that our translations are faster seems to be that the state space is smaller for our translations. This follows from the fact that the translated system cannot get stuck while simulating a single transition of the original system.

Table 12.2 presents the verification times for ABP with symmetry turned on and optimizations turned off. Interestingly, we observe that now the standard translation is faster than both our translations. We believe the reason is connected to the updating of shared integer variables as mentioned by Bengtsson and Yi [9]. Specifically, Bengtsson and Yi notes that because integer variables are updated in a specific order (sender first and then receivers in the order they are defined), the symmetry of input and output actions is essentially destroyed in the event that one of the participating TA updates a value which is used by another participating TA. This is exactly what we are doing with the counter variables, when synchronizing on the t_{test} broadcast channel.

Further, it is interesting to note that the Deg-2-Broadcast translation performs much better than the Broadcast translation in this experiment. This seems to be connected to the size of the state space. Specifically, for 12 messages the number of states explored for Broadcast is roughly two times that of Deg-2-Broadcast which fits well with the fact that Deg-2-Broadcast is roughly two times faster than Broadcast. Likewise for 20 messages the number of states explored for Broadcast is roughly three times higher than for Deg-2-Broadcast and the Deg-2-Broadcast is also roughly three times faster than Broadcast. Note that in this experiment Deg-2-Broadcast is roughly 50% slower than the Standard translation although this number seems to be increasing slowly. This slowdown is the price we pay to support liveness checking.

Soccer Field Grass Cutting: Safety, Symmetry off, optimizations on				
# Fields	**Standard**	**Standard Opt.**	**Broadcast**	**Deg-2-Broadcast**
3	6.07s	17.29s	0.21s	0.21s
4	13.22s	1m 27.93s	0.45s	0.48s
5	25.41s	-	0.98s	1.10s
6	43.77s	-	2.75s	2.64s
7	1m 11.58s	-	7.82s	7.55s
8	1m 54.89s	-	29.59s	24.45s
9	3m 12.08s	-	1m 44.18s	1m 19.98s
10	-	-	-	4m 25.72s

Table 12.3: Comparison of the verification times for the soccer field grass cutting case study with symmetry turned off and optimizations turned on.

12.1.2 Soccer Field Grass Cutting

We will now present the results for the soccer field grass cutting case study. The model can be found in Appendix B. We will explore verification times for both safety and liveness queries for this case study.

Safety

For safety, we will use a query that explores the whole state space. Table 12.3 presents the verification times for the soccer field grass cutting problem with symmetry turned off and optimizations turned on.

Note that we have included Standard translation both with and without optimization. The reason for this is that the unoptimized one performs much better than with optimizations turned on. This is surprising and counter-intuitive, however, the same phenomenon was noticed by [17]. Further, observe that just as for ABP, our translations outperform the Standard translation when symmetry is turned off. The reason also appears to be the same as before, namely that the state space is smaller for our translations because we cannot get stuck while simulating a transition.

Table 12.4 presents the verification times for the soccer field grass cutting problem with both symmetry and optimizations turned on (again, Standard translation is included both with and without optimizations).

Observe that for Broadcast, this experiment gives the same picture as ABP, namely that when symmetry is turned on, it performs worse than the Standard translation. However, this is not true for Deg-2-Broadcast which is still a little faster than the Standard translation (although the verification times are rather similar). The reason is that for Standard, Standard Opt. and Deg-2-Broadcast, the number of states explored are comparable and generally smaller than for Broadcast.

Soccer Field Grass Cutting: Safety, Symmetry on, optimizations on				
# Fields	**Standard**	**Standard Opt.**	**Broadcast**	**Deg-2-Broadcast**
20	0.33s	0.27s	0.56s	0.28s
30	0.99s	0.84s	2.95s	0.85s
40	2.73s	2.45s	12.18s	2.40s
50	6.57s	6.07s	39.41s	5.94s
60	14.86s	13.27s	1m 46.80s	13.01s
70	29.05s	27.37s	3m 11.12s	20.02s

Table 12.4: Comparison of the verification times for the soccer field grass cutting case study with both symmetry and optimizations turned on.

Soccer Field Grass Cutting: Liveness, Symmetry on, optimizations on		
# Fields	**Broadcast**	**Deg-2-Broadcast**
4	5.99s	5.81s
5	26.08s	24.83s
6	1m 43.88s	1m 33.22s
7	6m 17.09s	5m 20.49s
8	-	-

Table 12.5: Comparison of the verification times for the soccer field grass cutting case study with both symmetry and optimizations turned on.

Liveness

For liveness, we will use a query that checks whether the three workers can keep all soccer fields in perfect shape. The workers are at most able to keep three fields in shape. Thus, the query is not satisfied in the experiments below. The verification times are presented in Table 12.5. Note that we allow the liveness tests to use up to ten minutes. Thus, a dash $(-)$ means the verification did not finish in ten minutes. This only applies to Table 12.5. Note that we have not compared these results to the Standard translation, as the net contains transitions greater degree 2 and for such nets, their translation does not support liveness. As before, the results indicate that the Deg-2-Broadcast translation outperforms the Broadcast translation.

In all our experiments, the Deg-2-Broadcast translation has been faster than the Broadcast translation. It seems to be more expensive to simulate a transition in Broadcast. When the t_{test} transition has been executed, we must move additional token automata back to their original location via τ-transition, before executing the t_{fire} transition. This is not necessary in the Deg-2-Broadcast translation, because executing the t_{test} transition does not move any token automata out of their respective

locations. Moreover, the sequence of τ-transitions and the subsequent t_{fire} broadcast synchronization in the Broadcast translation appears to be more expensive than the sequence of handshake synchronizations in Deg-2-Broadcast.

12.2 TAPN with Integers

We now demonstrate how the use of integers can model the Viking case study in a more efficient way. This case study is included as an example in UPPAAL. We shall compare verification of a TAPN model, a TAPN with integers model and a UPPAAL model. The idea is that a number of vikings need to cross a damaged bridge in the middle of the night (the number of vikings can be scaled). They only have a single torch, which they need to cross the bridge safely. Further, the bridge can support at most two of them at once and the vikings move at different speed and thus take different amounts of time to cross the bridge. The goal is to get everyone across the bridge safely. We can imagine one viking carrying the torch back and forth while taking the others across the bridge. The used query checks whether it is possible to get all the vikings across the bridge safely.

We performed verification on different instances of the problem. Two parameters are used to define these. The first parameter is the number of viking types, where each viking type takes a different amount of time to cross the bridge (i.e. the first type takes 5 time units, the second type takes 10 time units, etc.). The second parameter is the number of vikings of each type. Note that for this experiment, we are interested in exploring how the addition of integers to the TAPN model impacts the verification times. Thus, we will only use the Broadcast translation. The models presented in Appendix A.

The verification times are presented in Table 12.6. Table 12.7 presents the ratio between the verification times of the Broadcast translation with and without integers for each instance of the problem. This ratio is calculated as $\frac{\text{TAPN}}{\text{TAPN with integers}}$ on each instance of the problem. Thus, a ratio greater than one means the verification of the TAPN with integers model was faster (higher means faster). The light gray cells indicate instances that could only be solved within five minutes on the TAPN with integers model.

As indicated by the ratios, the verification times is consistently faster when using integers. We believe this is connected to the fact that the these models are much smaller compared to the TAPN models. One of the contributing reasons for this is that adding a new type of viking when using integers is only a matter of adding a new token to the net with a distinguished initial value representing the time it takes to cross the bridge, whereas without integers, we need to change the static structure of the net and add a new token to accommodate a new viking type.

Another contributor to the slower verification of the TAPN models is the use of inhibitor arcs. The transitions with inhibitor arcs are always translated to the structure using t_{test} and t_{fire} transitions, regardless of whether the transition is degree 2 or not. There are no inhibitor arcs in the TAPN with integers models.

TAPN

# types \ # each type	**2**	**3**	**4**	**5**	**6**	**7**
2	0.02s	0.04s	0.05s	0.07s	0.08s	0.13s
3	0.05s	0.11s	0.23s	0.45s	0.86s	1.54s
4	0.12s	0.48s	1.71s	4.79s	11.62s	25.33s
5	0.47s	3.16s	14.78s	51.98s	2m 31.97s	-
6	1.97s	20.14s	2m 3.09s	-	-	-
7	8.48s	2m 3.95s	-	-	-	-
8	36.51s	-	-	-	-	-

TAPN with integers

# types \ # each type	**2**	**3**	**4**	**5**	**6**	**7**
2	0.02s	0.02s	0.03s	0.04s	0.05s	0.06s
3	0.02s	0.04s	0.09s	0.20s	0.38s	0.71s
4	0.05s	0.19s	0.71s	2.10s	5.40s	12.22s
5	0.16s	1.23s	6.35s	23.70s	1m 12.82s	3m 9.96s
6	0.72s	8.37s	54.26s	4m 9.50s	-	-
7	3.19s	52.01s	-	-	-	-
8	14.08s	-	-	-	-	-

UPPAAL

# types \ # each type	**2**	**3**	**4**	**5**	**6**	**7**
2	0.01s	0.01s	0.02s	0.03s	0.04s	0.06s
3	0.01s	0.05s	0.18s	0.67s	2.13s	6.32s
4	0.04s	0.50s	5.08s	35.93s	3m 26.84s	-
5	0.23s	7.39s	2m 23.80s	-	-	-
6	1.41s	1m 37.67s	-	-	-	-
7	8.83s	-	-	-	-	-
8	51.81s	-	-	-	-	-

Table 12.6: Comparison of the verification times for the Vikings example modeled with TAPN and TAPN with integers (using Broadcast translation with symmetry and optimizations turned on) and UPPAAL.

# types \ # each type	**2**	**3**	**4**	**5**	**6**	**7**
2	1.00	2.00	1.67	1.75	1.60	2.17
3	2.50	2.75	2.56	2.25	2.26	2.17
4	2.40	2.53	2.41	2.28	2.15	2.07
5	2.94	2.57	2.33	2.19	2.09	-
6	2.74	2.41	2.27	-	-	-
7	2.66	2.38	-	-	-	-
8	2.59	-	-	-	-	-

Table 12.7: The ratio $\frac{\text{TAPN}}{\text{TAPN with integers}}$ of the verification times in Table 12.6. The higher the number, the faster the TAPN with integers model was in terms of verification time. The light gray cells indicate instances which could only be solved on TAPN with integers.

# types \ # each type	**2**	**3**	**4**	**5**	**6**	**7**
2	0.50	0.50	0.67	0.63	0.80	1.05
3	0.65	1.18	1.97	3.34	5.61	8.89
4	0.80	2.64	7.15	17.11	38.30	-
5	1.45	6.01	22.65	-	-	-
6	1.95	11.67	-	-	-	-
7	2.77	-	-	-	-	-
8	3.68	-	-	-	-	-

Table 12.8: The ratio $\frac{\text{UPPAAL}}{\text{TAPN with integers}}$ of the verification times in Table 12.6. The higher the number, the faster the TAPN with integers model was in terms of verification time. The light gray cells indicate instances which could only be solved on TAPN with integers.

Table 12.8 presents the ratio between the verification times of UPPAAL and TAPN with integers for each instance of the problem. This ratio is calculated as $\frac{\text{UPPAAL}}{\text{TAPN with integers}}$. Thus, the higher the number, the faster the TAPN with integer model was in terms of verification time. As before, the light gray cell indicate instances that could only be solved within five minutes on the TAPN with integers model.

Interestingly, the results show that on a large set of the experiments, TAPAAL is actually faster than UPPAAL and there are even some instances which can only be solved by TAPAAL within five minutes. It seems the NTA constructed by TAPAAL for verification is a more efficient encoding of the Vikings case study compared to the one included in UPPAAL.

Chapter 13

Conclusion

In this thesis, we have investigated the timed-arc Petri net model with invariants, inhibitor arcs and transport arcs, abbreviated TAPN. The semantics of such a model is a natural extension of ordinary timed-arc Petri nets.

We have defined the timed computation tree logic (TCTL) and its safety fragment. In much of the work on TCTL, maximal runs are not handled completely. Often, only infinite maximal runs are considered (see e.g. [36]). However, in tools like UPPAAL, maximal runs can also be finite (e.g. a run ending in a deadlock is maximal because it cannot be extended further). We have shown how to handle maximal runs in its entirety, including both infinite and finite maximal runs.

By introducing a one-by-many correspondence, which is a relation between the states of timed transition systems (TTS) A and B, we provide a generalized framework which can be used to show that a translation from A to B preserves the full TCTL (or only the safety fragment depending on the requirements fulfilled by the relation). Intuitively, when A and B are related by a one-by-many correspondence, we can simulate one step in A by a number of steps in B.

We have presented two novel translations from TAPN models to NTA, both of which preserve the full TCTL. To the best of our knowledge, these are the first efficient translations from bounded TAPN to NTA that preserve the full TCTL. The translations allow us to perform verification of TAPN models using the UPPAAL verification engine.

The first translation uses broadcast channels to first check the enabledness condition of the transition and then use another broadcast channel to simulate the firing of the transition. Due to the semantics of broadcast synchronizations, there are additional steps that must be taken in between checking enabledness and simulating the firing of a transition. Since each token is simulated by a timed automaton, too many automata may have participated in the enabledness check, and any extra timed automata must be moved back to their original location before simulating the firing of the transition.

Our second translation incorporates the idea of using broadcast synchronizations to check the enabledness conditions into the translation by Byg et al. [17]. Again each

token is simulated by a timed automaton. The translation works by converting the TAPN model to a degree 2 net, meaning that every transition has exactly two input arcs and two output arcs. The idea is that since each transition is degree 2, every transition firing corresponds to two tokens moving to new places, which is exactly what happens when two automata (each representing a token) performs a handshake synchronization on a channel. While their translation introduced additional deadlocks into the model and hence only preserve safety properties, we overcome this problem by checking the enabledness of the transition using broadcast synchronization.

We have also extended the TAPN model with integers, in the sense that each token is allowed to carry an integer beside its age. This integer can then be used in age guards. Further, we allow value guards and value invariants on arcs and places, respectively. We argue about the decidability of reachability for bounded TAPN with various forms of integer extensions.

In order to compare the performance of our two translations and the translation by Byg et al. [17], we ran a number of experiments on some small case studies. In these experiments, the translations by Byg et al. [17] was generally faster on safety queries when symmetry reduction was turned on. However, our translations were faster when symmetry reduction was turned off. The reason seemed to be that the state space was smaller for our translations. Further, we found that the degree 2 broadcast translation was faster than the broadcast translation. Additionally, we also ran some experiments with liveness queries using our translations (on the chosen model, the translation by Byg et al. [17] did not support liveness). Lastly, we explored how the addition of integers to the TAPN model impacted the verification times and found that in certain cases the integers allows us to create smaller models which improved performance.

All translations in this thesis have been implemented in a prototype of the verification tool TAPAAL.

Finally, we provided an overview of known undecidability results for various extensions of TAPN. We answered two open problems and provided conjectures about the decidability of other open problems.

13.1 Future Work

Going forward, there are several interesting directions we could take.

Our TCTL-preserving framework is not general enough to handle certain translations (see e.g. [10, 11, 14, 22]). Common for these translations seems to be the fact that the translated system must perform additional internal transitions when simulating a time delay. Thus, we would like to extend our framework to allow for this type of behavior also. Doing so would improve on the generality of our framework, making it more widely applicable.

To improve TAPAAL, we have discussed the possibility of creating our own verification engine for TAPN models. There are several benefits to this. First of all, it would make TAPAAL independent as it would no longer rely on UPPAAL for the verification. Second, it would allow us to decide coverability for unbounded TAPN

models (without inhibitor arcs and invariants), using the techniques described by Abdulla and Nylén [3]. Because the UPPAAL verification engine is hard to extend, e.g. for multicore or distributed architectures, it would make it easier to experiment with such things if we made our own engine.

Another possible direction is to add more features akin to what is seen in colored Petri nets. Our extension of integers provide a very small step towards including the power of colored Petri nets. Allowing the definition of types (as opposed to forcing integers) would be an interesting approach, as well as allowing arithmetic expressions in guards and updates. However, such extensions often make reachability undecidable, which means compromises would have to be made. Either to focus on extensions that preserves decidability (of bounded nets) or allow all forms of extensions and investigate possible over-approximation techniques.

Bibliography

[1] TAPAAL. www.tapaal.net. Accessed: 28.05.2010.

[2] UPPAAL. www.uppaal.com. Accessed: 28.05.2010.

[3] P. A. Abdulla and A. Nylén. Timed Petri Nets and BQOs. In *Proc. of ICATPN'01*, volume 2075 of *LNCS*, pages 53–70. Springer, 2001.

[4] P. A. Abdulla, P. Mahata, and R. Mayr. Dense-Timed Petri Nets: Checking Zenoness, Token Liveness and Boundedness. *Logical Methods in Computer Science*, 3(1):1–61, 2007.

[5] R. Alur and D. L. Dill. Automata for Modeling Real-Time Systems. In *Proc. of ICALP'90*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[6] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[7] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking in Dense Real-Time. *Information and Computation*, 104(1):2 – 34, 1993.

[8] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *Communications of the ACM*, 12(5):260–261, 1969.

[9] J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2004.

[10] 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.

[11] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. When are Timed Automata Weakly Timed Bisimilar to Time Petri Nets? *Theoretical Computer Science*, 403(2-3): 202–220, 2008.

[12] 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.

[13] T. Bolognesi, F. Lucidi, and S. Trigila. From Timed Petri Nets to Timed LOTOS. In *Proc. of PSTV'90*, pages 395–408. North-Holland, 1990. ISBN 0-444-88810-1.

[14] P. Bouyer, P. A. Reynier, and S. Haddad. Extended Timed Automata and Time Petri Nets. In *Proc. of ACSD'06*, pages 91–100. IEEE Computer Society, 2006. ISBN 0-7695-2556-3.

[15] 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.

[16] 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.

[17] 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 5799 of *LNCS*. Springer, 2009.

[18] 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.

[19] L. A. Cortés, P. Eles, and Z. Peng. Modeling and Formal Verification of Embedded Systems Based on a Petri Net Representation. *Journal of Systems Architecture*, 49(12-15):571–598, 2003.

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

[21] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In *Proc. of ICALP'98*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.

[22] R. Gómez. A Compositional Translation of Timed Automata with Deadlines to Uppaal Timed Automata. In *Proc. of FORMATS'09*, volume 5813 of *LNCS*, pages 179–194. Springer, 2009.

[23] M. Hack. Petri Net Language. Technical Report MIT-LCS-TR-159, Massachusetts Institute of Technology, Cambridge, MA, USA, 1976.

[24] H. Hanisch. Analysis of Place/Transition Nets with Timed Arcs and its Application to Batch Process Control. In *Proc. of Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 282–299. Springer, 1993.

[25] M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding Symmetry Reduction to Uppaal. In *Proc. of FORMATS'03*, volume 2791 of *LNCS*, pages 46–59. Springer, 2003.

[26] C. N. Ip and D. L. Dill. Better Verification Through Symmetry. *Formal Methods in System Design*, 9(1-2):41–75, 1996.

[27] L. Jacobsen, M. Jacobsen, and M. H. Møller. Extending Timed-Arc Petri Nets. DAT5 Project Report, 2009. URL http://www.cs.aau.dk/~mortenja/dat5-report.pdf.

[28] L. Jacobsen, M. Jacobsen, and M. H. Møller. Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants. In *Proc. of MEMICS'09*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009. ISBN 978-3-939897-15-6.

[29] 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.

[30] K. Jensen. Coloured Petri Nets and the Invariant-Method. *Theoretical Computer Science*, 14:317–336, 1981.

[31] K. Jensen. High-Level Petri Nets. In *Proc. of Applications and Theory of Petri Nets*, volume 66 of *Informatik-Fachberichte*, pages 166–180. Springer, 1982.

[32] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[33] P. Merlin and D. Farber. Recoverability of Communication Protocols – Implications of a Theoretical Study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.

[34] P. M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, 1974.

[35] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967. ISBN 0-13-165563-9.

[36] 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. ISBN 978-3-540-32869-8.

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

[38] V. V. Ruiz, F. C. Gomez, and D. de Frutos-Escrig. On Non-Decidability of Reachability for Timed-Arc Petri Nets. In *Proc. of PNPM'99*, pages 188–196. IEEE Computer Society, 1999.

[39] J. Sifakis and S. Yovine. Compositional Specification of Timed Systems. In *Proc. of STACS'96*, volume 1046 of *LNCS*, pages 347–359. Springer, 1996.

[40] 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.

[41] W. M. P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In *Proc. of Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 453–472. Springer, 1993.

Appendix A

Vikings Case Study Models

The Vikings case study is one of the small examples inluded in UPPAAL. The idea is that a number of vikings (4 in the UPPAAL example, but this can be scaled) need to cross a damaged bridge in the middle of the night. The catch is that they only have 1 torch which they need, in order to find their way across the bridge. Since the bridge is damaged, it can support at most two of them at once. Further, the vikings are moving at different speeds so they take different amounts of time to cross the bridge. Thus, in order to get everyone across the bridge one can imagine a viking carrying the torch back and forth while the others are taken across the bridge one by one. Let us now look at how this is modeled in TAPAAL and UPPAAL. For this discussion, we will present a small instance of the example (i.e. with a small number of vikings), whereas in the experiments we will use instances with varying size.

A.1 UPPAAL Model

We will start by introducing the UPPAAL model which we have modified slightly to accomodate symmetry reduction. The idea is that we have (up to) eight different viking types. The difference lies in the time it takes to cross the bridge. This delay is not a parameter of the template anymore, but rather a locally declared variable. The delay is 5 for the first type, 10 for the second type, 15 for the third type and so on. The viking template is presented in Figure A.1. Each of the eight viking types are identical to this template, except for the value of the local variable delay. This allows us to declare a global type typedef scalar[N] id_t where N denotes the number of vikings of each type. A viking type template simply takes a parameter of type id_t when it is instantiated (which allows symmetry reduction). For example, if we have eight different types and set N to 2, then we would get a total of 16 vikings, two of each type.

A viking starts in the state *unsafe* indicating he is at the wrong side of the bridge and thus needs to cross over to the safe side. He starts by synchronizing with the torch on the *take* channel indicating that he picks up the torch. Note that this transition is guarded by a variable L which denotes which side of the bridge the torch is on (0 for

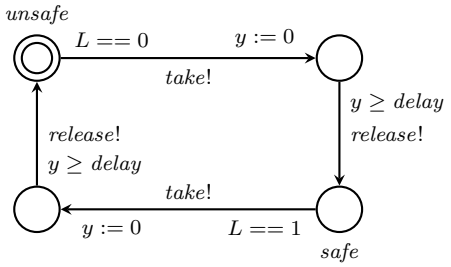


Figure A.1: The UPPAAL template for the vikings.

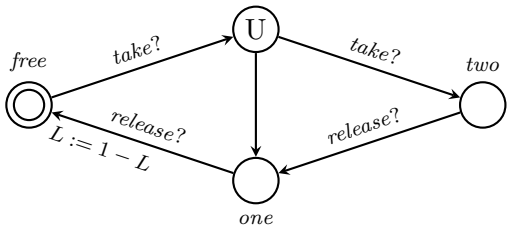


Figure A.2: The UPPAAL template for the torch.

unsafe side, 1 for the safe side). Then the viking starts walking and after delay time units he reaches the other side. Once a viking reaches the safe side, he will release the torch by synchronizing on the *release* channel with the torch. He can then travel back by taking the torch again, i.e. taking the *take* transition in the location *safe*. Note that this transition is guarded by the expression L == 1 meaning the torch must be on the safe side for the viking to take it. Finally, the viking can release the torch once he has crossed the bridge back to the unsafe side.

Let us now look at the template for the torch. The template is illustrated in Figure A.2. As mentioned above, at most two vikings can be crossing the bridge at any time. Thus, the torch allows up to two synchronizations on the *take* channel. After synchronizing on the *take* channel once, the torch moves to an urgent state which must be exited immediately (i.e. no time can pass). From here it is possible to synchronize with another viking on the *take* channel or just move on alone. Likewise, the torch allows up to two synchronizations on the *release* channel, depending on the number of vikings who moved across the bridge.

A.2 TAPN Model

We will now look at the TAPAAL model which is more complicated since we have to represent both the vikings and the torch in one big TAPN. Since this model is more complicated, we will start with a general overview of the model (see Figure A.3).

In this simple overview there are two types of vikings, represented as the two gray boxes in the middle of the figure. The two gray boxes at the left and right side of the

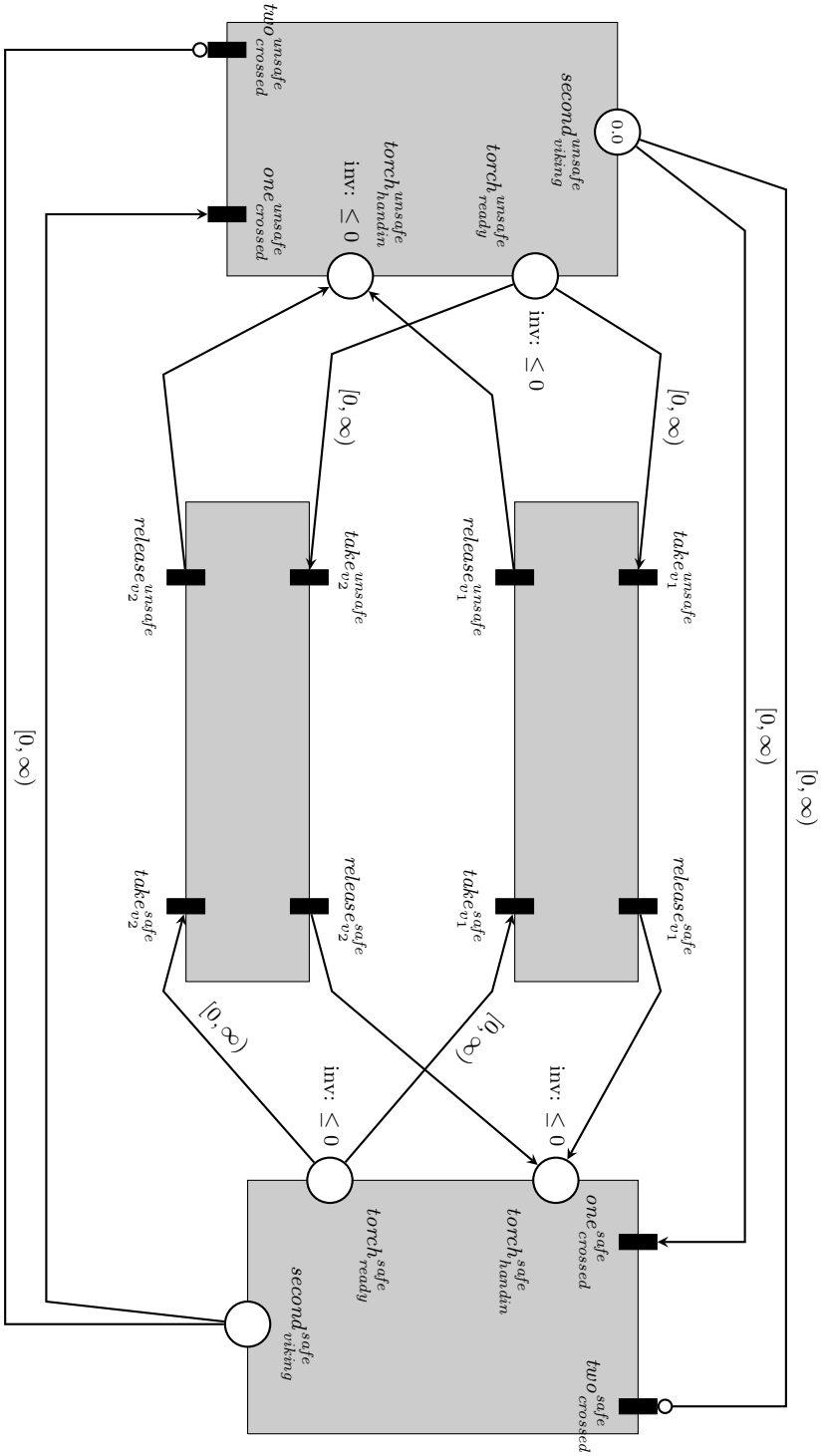


Figure A.3: An overview of the TAPN model of the vikings case study.

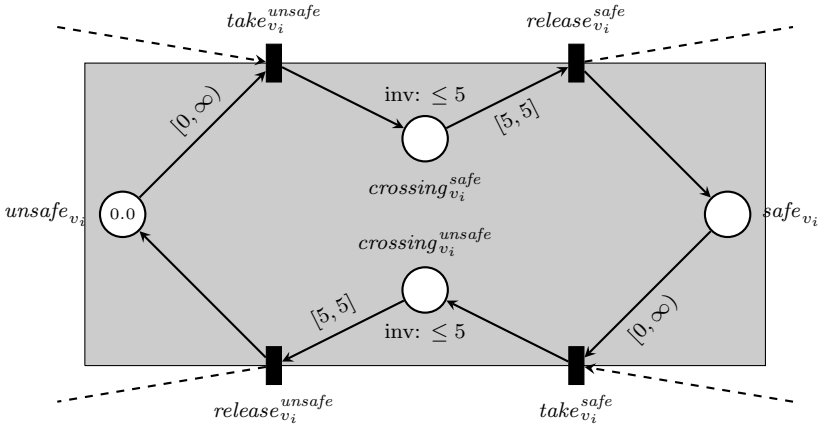


Figure A.4: The part of the TAPN model simulating a viking with delay 5. The dashed arcs are the ones which are shown in Figure A.3.

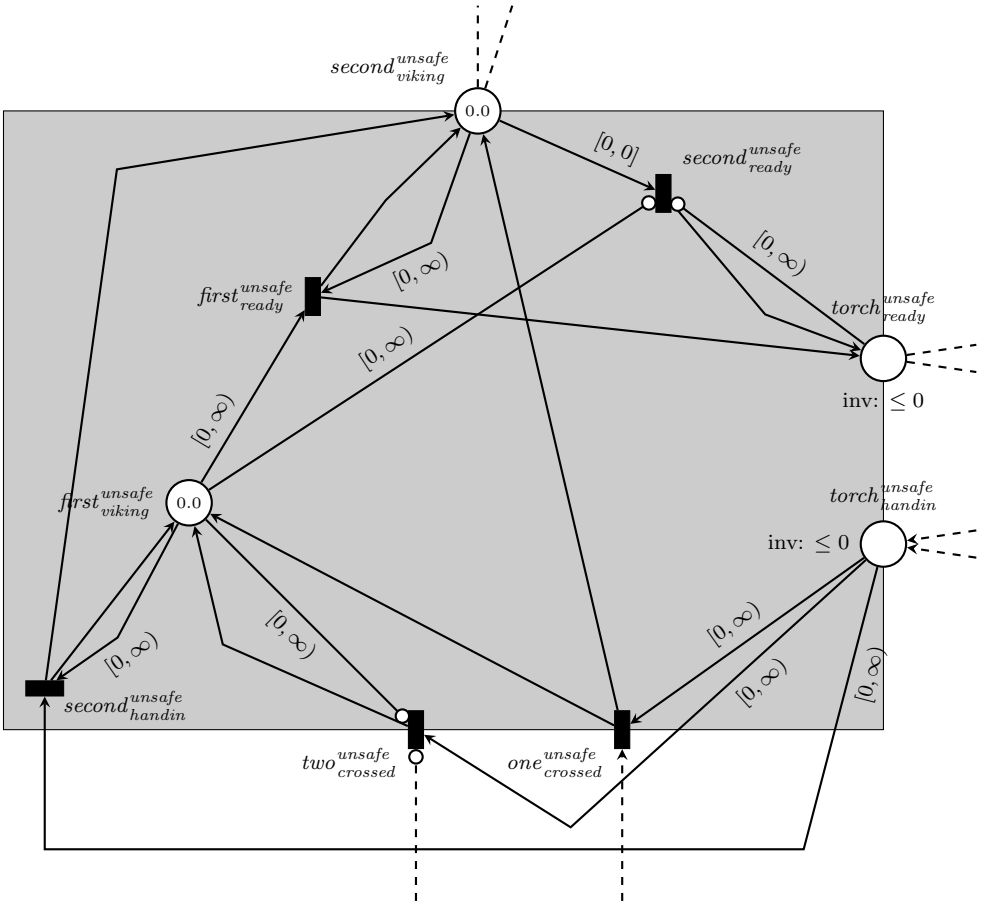


Figure A.5: The part of the TAPN model simulating the torch on the unsafe side. The dashed arcs are the ones which are shown in Figure A.3.

figure represent the torch on the unsafe and safe side, respectively. The idea is that the part of the TAPN which is within these boxes (we will see in a bit how these look) will simulate the vikings and the torch, respectively. Let us look at how these different components of the TAPN are connected.

Consider the first viking in the figure. We can see that this viking has a $take_{v_1}^{unsafe}$ transition which denotes that the viking takes the torch on the unsafe side and starts walking across the bridge. When he gets to the end of the bridge, the $release_{v_1}^{safe}$ transition can be fired which signifies that he puts down the torch on the safe side. Similarly, he can now choose to pick-up the torch and go back to the unsafe side by the $take_{v_1}^{safe}$ and $release_{v_1}^{unsafe}$ transitions.

The torches are also connected. Consider the place $second_{viking}^{unsafe}$, which denotes if one or two vikings crossed the bridge. Imagine that two vikings cross the bridge from the unsafe side. When the torch is released on the unsafe side, the place $second_{viking}^{unsafe}$ would be empty, signifying that two vikings were crossing the bridge. We need to remember how many were crossing the bridge, since the vikings moves at different speeds across the bridge and thus we have to wait for the second viking to get across before anyone can move back. Thus, the transition $one_{crossed}^{safe}$ denotes that only one viking crossed the bridge from the unsafe side and the transition $two_{crossed}^{safe}$ denotes that two vikings crossed the bridge from the unsafe side. Similar connections exist for the case where the vikings are walking back.

Now that we have an idea about the overall picture of the TAPN model of this case study, let us look at what is inside the gray boxes in Figure A.3. We will start with the part of the net which models a viking, since this is pretty straightforward. This part is illustrated in Figure A.4.

For this particular type of viking, it takes 5 time units to cross the bridge. We could make a similar structure for a viking type which takes 10 time units to cross by copying the structure and replacing each 5 with a 10 in the intervals and invariants. If we want, say, two vikings both of which takes 5 time units then we can simply put two tokens in the place $unsafe_{v_i}$ in the initial marking. If we want all vikings to use a different amount of time for crossing the bridge then we have to create such a structure for each viking. Initially, there is a token in the place $unsafe_{v_i}$ indicating the viking is on the unsafe side. The viking can pick up the torch and start walking by firing the $take_{v_i}^{unsafe}$ transition provided that the torch is ready (the dashed arrow going into the transition is connected to the $torch_{ready}^{unsafe}$ place, see Figure A.3). Now we have to wait for 5 time units before we can fire the $release_{v_i}^{safe}$ transition indicating the viking has reached the safe side (the dashed arc going out of the transition is connected to the place $torch_{handin}^{safe}$). The construction works similarly when the viking is moving back to the unsafe side.

Now we will look at the part of the net simulating the torch on the unsafe side. The construction for the torch is illustrated in Figure A.5.

The basic idea of this construction is that we will initially have a token in each of the places $first_{viking}^{unsafe}$ and $second_{viking}^{unsafe}$ (this is only on the unsafe side, these are empty

initially on the safe side). We can fire the transition $\mathit{first}^{unsafe}_{ready}$ to put a token in the place $\mathit{torch}^{unsafe}_{ready}$ and thus make the torch ready for any viking wanting to cross the bridge. Once the first viking picks up the torch, we have the opportunity to fire the $\mathit{second}^{unsafe}_{ready}$ transition to put another token in $\mathit{torch}^{unsafe}_{ready}$, signifying that two vikings can now cross the bridge (note that no time can pass before this transition is fired). However, the first viking can also cross the bridge alone. When the bridge has been crossed and the torch released, a token will be put in the $\mathit{torch}^{unsafe}_{handin}$ place (here we assume that vikings are crossing from the safe side). If two vikings crossed the bridge, then when the first one releases the torch, we can fire the transition $\mathit{two}^{unsafe}_{crossed}$ to put a token in the place $\mathit{first}^{unsafe}_{viking}$. Note that we cannot make the torch ready before the second viking has crossed the bridge also. When this happens, we can fire the $\mathit{second}^{unsafe}_{handin}$ transition, after which we can make the torch ready again for one or two vikings. If only one viking crossed the bridge, we can fire the transition $\mathit{one}^{unsafe}_{crossed}$ after which the torch can be made ready again. The part of the net simulating the torch on the safe side is identical, except for the fact that initially there are no tokens in the places $\mathit{first}^{safe}_{viking}$ and $\mathit{second}^{safe}_{viking}$.

A.3 TAPN with Integers Model

The viking example, modeled as a TAPN with integers, is illustrated in Figure A.6. The token in the place *torch* represents the torch and the value of the torch token represents the state of the torch as follows.

1. torch is on the unsafe side and no vikings are moving,
2. one viking is moving from the unsafe to the safe side,
3. two vikings are moving from the unsafe side to the safe side,
4. one viking has reached the safe side,
5. the torch is on the safe side and no vikings are moving,
6. one viking is moving from the safe side to the unsafe side,
7. two vikings are moving from the safe side to the unsafe side, and
8. one viking has reached the unsafe side.

Each viking will be represented by a token, which is initially in the *unsafe* place, where the value of a the token represents the time it takes for this viking to cross the bridge. Thus, to scale the number of vikings we just add tokens to or remove tokens from the *unsafe* place.

When a viking wants to start moving across the bridge (from unsafe to safe side), he picks up the torch by firing the $\mathit{first}^{unsafe}_{viking}$ transition. It is now possible for a second

Meaning of torch token values:

1: torch is on the unsafe side and no vikings are moving.
2: one viking is moving from the unsafe side to the safe side.
3: two vikings are moving from the unsafe side to the safe side.
4: one viking has reached the safe side.
5: the torch is on the safe side and no vikings are moving.
6: one viking is moving from the safe side to the unsafe side.
7: two vikings are moving from the safe side to the unsafe side.
8: one viking has reached the unsafe side.

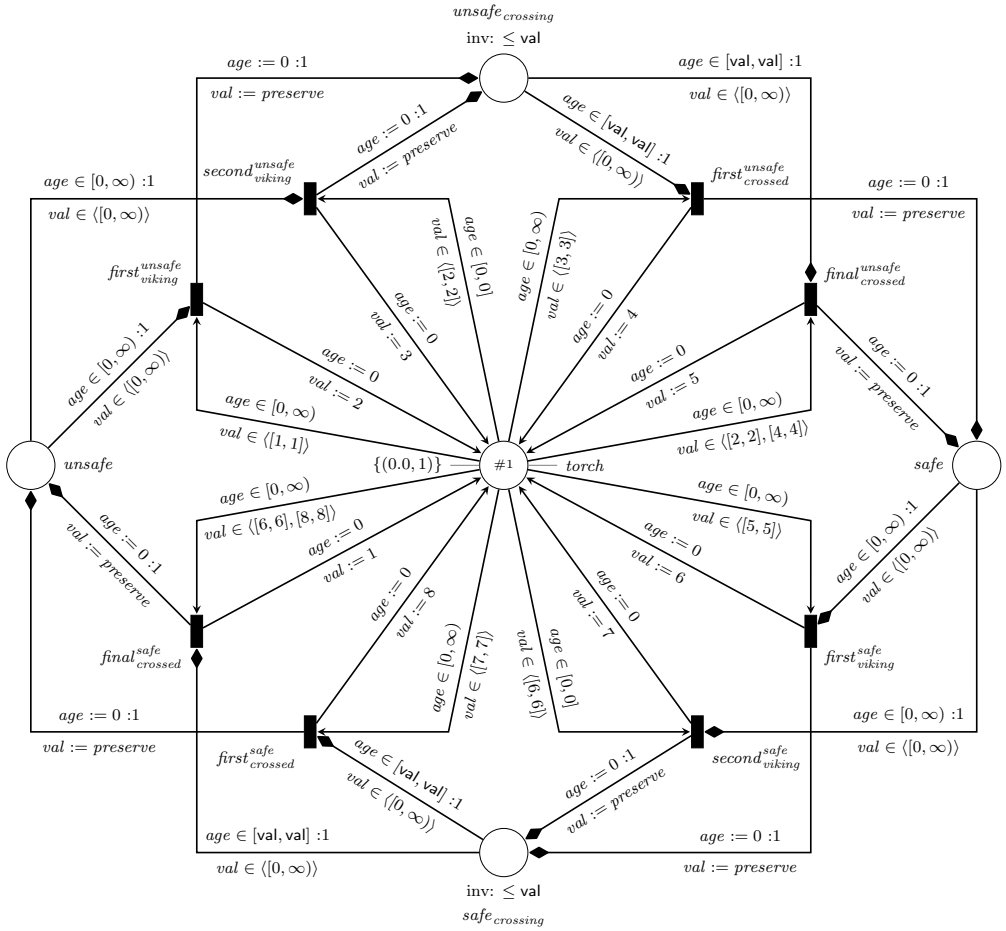


Figure A.6: A TAPN with integers model of the vikings example.

viking to tag along by firing the $second_{viking}^{unsafe}$ transition or the viking can move across the bridge alone by doing time delays. Once the time it takes a viking to cross the bridge has passed, he can reach the safe side by firing either the $first_{crossed}^{unsafe}$ transition (if he is the first of two vikings currently crossing the bridge) or the $final_{crossed}^{unsafe}$ transition (if he either crossed the bridge alone or is the second of two vikings currently crossing the bridge to arrive). The net works similarly for vikings crossing from the safe to the unsafe side.

Appendix B

Soccer Field Grass Cutting Case Study

The soccer field grass cutting case study is illustrated in Figure B.1. The model depicts three workers who are tasked with keeping a number of soccer fields in shape. The number of tokens in the *in_shape* place denotes the number of soccer fields. The workers must cut the grass of the soccer fields within a certain period since they were last cut, in order to keep them in shape. Otherwise, they will degrade into an unacceptable state. For the purpose of this model, we will assume that degraded soccer fields cannot be restored. The workers can divide the work between them or help each other out. Thus, one, two or three workers can cut a soccer field at once. However, the more workers helping out, the quicker they can cut the field. It takes a single worker 4 time units to cut a soccer field, whereas two workers can cut a soccer field in 3 time units and all three of them can cut it in 2 time units. The *cut* transitions denote the various possibilites for cutting a soccer field. For instance, the cut_{w1} transition denote that worker 1 cuts a soccer field alone, while cut_{w1}^{w2} denotes that worker 1 and 2 cuts a soccer field together. Naturally, the cut_{all} transition denotes that all three workers cut a soccer field.

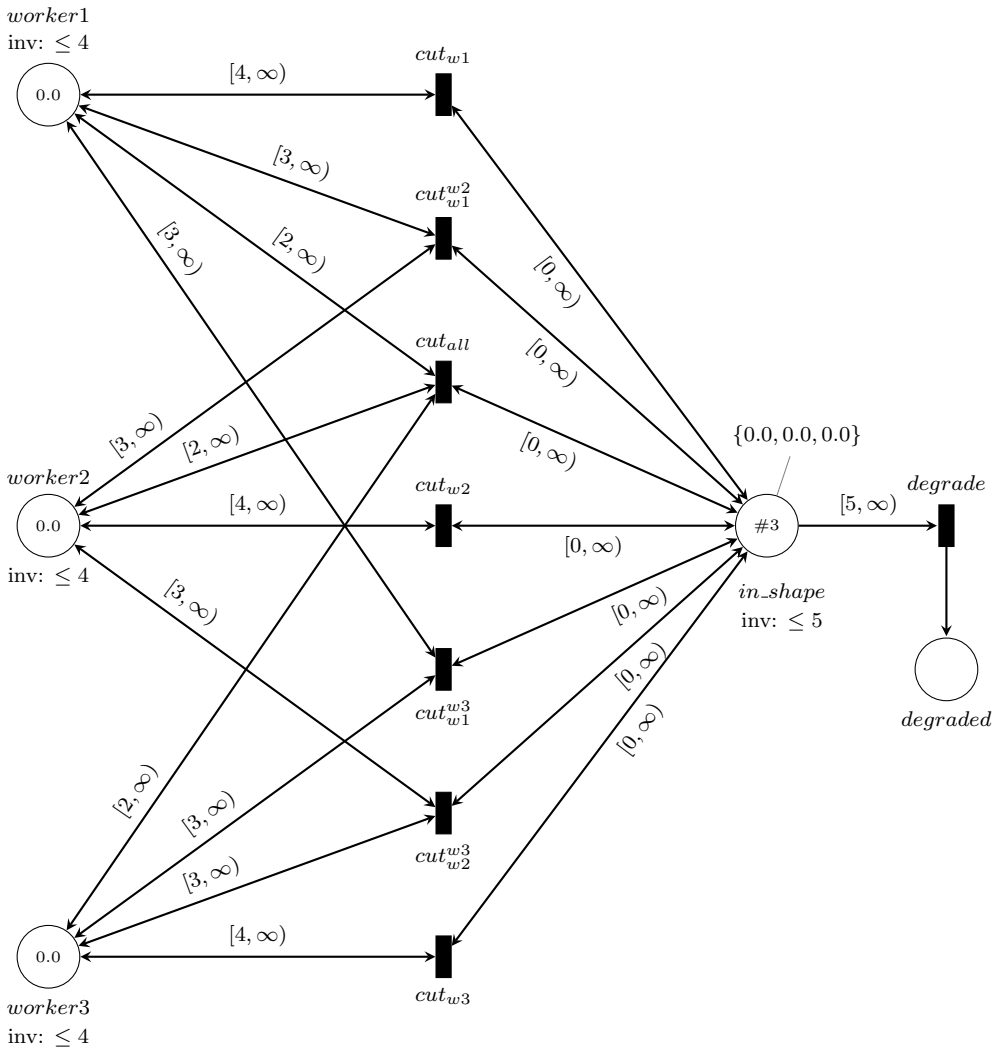


Figure B.1: The soccer field grass cutting model.

Appendix C

Summary

Time-dependent formalisms have been extensively studied due to increasing demands on the reliability and safety of embedded software systems. *Timed automata* [6] and various time-extensions of *Petri nets* (e.g.[13, 34]) are among the most studied time-dependent formalisms. Over the years, significant effort has been devoted to establishing formal relationships between different formalisms. In this respect, many translations between different formalisms (see e.g. [15, 17, 18, 20, 29]) have been proposed. Usually, the goal is to establish some equivalence (e.g. bisimulation) between the original system and the translated system, or show that the translation preserves some logic.

In this thesis, we study formal verification, specifically model checking, of *Timed-Arc Petri Nets* (TAPN), a time extension of Petri nets in which tokens are assigned a real number indicating its age and arcs from place to transitions are guarded by time intervals restricting which tokens can be used to fire a transition. We extend the basic TAPN model with transport arcs, inhibitor arcs and invariants. Further, we prove that invariants alone make the coverability and boundedness problems undecidable.

In respect to model checking, formal queries are expressed in the Timed Computation Tree Logic (TCTL), which is a popular temporal branching-time logic. Much of the work on TCTL, however, do not handle maximal runs in all details (e.g. [18, 36]). We treat the semantics of TCTL in its full generality, taking into account finite maximal runs that appear in the presence of deadlocks and invariants (strict and non-strict).

We identify a class of translations that preserve TCTL and propose a framework for proving correctness of translations with respect to TCTL formulae. We achieve this via a one-by-many correspondence. Intuitively, when systems A and B are related by a one-by-many correspondence we can simulate one step in A by a number of steps in B. If we can establish such a relation between the states of A and B, then that will allow us to conclude that the translation from A to B preserves the full TCTL (or only the safety fragment depending on the requirements fulfilled by the relation). Our framework can be seen as a generalization of the ideas in [18]. While their idea solely concerns a translation from time Petri nets to networks of timed automata, our framework works at the level of timed transition systems, making it independent of the

modeling formalisms used. Since our framework handles TCTL in its full generality, as used in state-of-the-art verification tools such as UPPAAL (www.uppaal.com), it is directly applicable to tool developers. We give examples of translations from the literature that fit our framework and argue that there are also translations which do not fit our framework.

To illustrate the applicability of our framework, we develop two novel translations from TAPN to networks of timed automata. Using our framework, we prove that both of them preserves the full TCTL. Our translations are motivated by the recent work on the verification tool TAPAAL [17], which could only handle liveness properties on a subclass of TAPN models where all transitions have two incoming and two outgoing arcs. To the best of our knowledge, our translations are the first efficient translation to preserve the full TCTL for arbitrary bounded TAPN models. Experiments show that the translations are rather efficient in practice.

We also add integer variables to the extended TAPN model. This allows tokens to carry an integer value which can be used in time intervals on the arcs and time invariants on places. Moreover, we introduce value guards and value invariants in the model. We sketch how to extend our translations to handle integers and conclude that the correctness of the translations still apply.

We have implemented our translations as well as support for inhibitor arcs and integers in a prototype of the verification tool TAPAAL (www.tapaal.net), as well as a number of improvements.